MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Digital Material

DATA SCIENCE-TOOLS & TECHNIQUES

R20A6704

Prepared by

Dr M V Kamal

Assoc. Prof & HoD

Department of

COMPUTER SCIENCE & ENGINEERING-DATA SCIENCE

(EMERGING TECHNOLOGIES)

M R C E T CAMPUS

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to JNTU, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Kompally), Secunderabad – 500100, Telangana State, India.

Contact Number: 040-23792146/64634237, E-Mail ID: mrcet2004@gmail.com, website: www.mrcet.ac.in

(R20A6704) DATA SCIENCE TOOLS AND TECHNIQUES

COURSE OBJECTIVES:

1. Study basic tools available for data science and analytics
2. Study usage of Excel tool, R and KNIME tool
3. Student will study usage of various data sources with Excel, R and Knime
4. Student will study working with various Charts
5. Student will learn working with various data type

UNIT I (Data Science and Various Data Science Tools)

Introduction to Data Science-Introduction- Definition - Data Science in various fields - Examples – Data Pre-Processing and Data Wrangling with Techniques. Impact of Data Science - Data Analytics Life Cycle

Data Science Toolkit.: Brief Introduction to data science tools: SaS, Apache Spark, BigML, Excel, R-Programming, TensorFlow, KNIME, Tableau, PowerBI etc with advantages and disadvantages.

UNIT-II (R – Programming - I)

Introduction to R- Features of R – Environment, How to run R, R Sessions and Functions, Basic Math, Variables, Data Types, Vectors, Conclusion, Advanced Data Structures, Data Frames, Lists, Matrices, Arrays, Classes, R Programming Structures, Control Statements, Loops, - Looping Over Nonvector Sets,- If-Else, Arithmetic and Boolean Operators and values, Default Values for Argument, Return Values, Functions are Objects, Recursion,

Basic Functions - R help functions - R Data Structures. Vectors: Definition- Declaration - Generating - Indexing - Naming - Adding & Removing elements - Operations on Vectors - Recycling - Special Operators - Vectorized if- then else- Vector Equality – Functions for vectors - Missing values - NULL values - Filtering & Subsetting.

UNIT-III (Working With Excel)

Introduction: Data Analysis, Excel Data analysis. Working with range names. Tables. Cleaning Data. Conditional formatting, Sorting, Advanced Filtering, Lookup functions, Pivot tables, Data Visualization, Data Validation. Understanding Analysis tool pack: Anova, correlation, covariance, moving average, descriptive statistics, exponential smoothing, fourier Analysis, Random number generation, sampling, t-test, f-test, and regression.

UNIT-IV

KNIME : Organizing your work, Nodes, Meta nodes, Ports, Flow variables, Node views. User Interface. **Data Preparation: Importing Data**-Database, tabular files, web services. **Transforming the Shape**- Filtering rows, Appending tables ,Less columns, More columns, Group By, Pivoting and Unpivoting, One2Many and Many2One,Cosmetic transformations. **Transforming values:** Generic transformations , Conversion between types, Binning, Normalization, Multiple columns, XML transformation, Time transformation, Smoothing, Data generation, Constraints ,Loops, Workflow customization.

UNIT-V

Data Exploration:

Computing statistics, Overview of visualizations, Visual guide for the views ,Distance matrix, Color , Size ,Shape ,KNIME views, HiLite , Use cases for HiLite, Row IDs, Extreme values. Basic KNIME views, The Box plots ,Hierarchical clustering, Histograms, Interactive Table, The Lift chart, Lines, Pie charts ,The Scatter plots, JFree Chart ,The Bar charts, The Bubble chart, Heatmap , The Histogram chart, The Interval chart, The Line chart, The Pie chart, The Scatter plot

Text Books:

1. Data Analysis with Excel by Manish Nigam. bpb Publications
2. R for Data Science, O'Reilly by Hadley Wickham 2016.
3. KNIME Essentials, by Gábor Bakos,2013
4. Data Science Tools by Christopher Greco,2020
5. Learn TensorFlow2.0, by Pramod Singh, Apress Publication (1st Edition)

Reference Books:

1. Introduction to Data Science a Python approach to concepts, Techniques and Applications, Igual, L;Seghi', S. Springer, ISBN:978-3-319-50016-4.
2. ALL-IN-ONE-EXCEL 2022 BIBLE FOR DUMMIES BY Bryant Shelton
3. Excel® 2019 BIBLE BY Michael Alexander ,Dick Kusleika

COURSE OUTCOMES:

1. Student will gain ability to use Excel
2. Student will gain ability to use R
3. Student will gain ability to use Knime
4. Student will be able to use various nodes available in knime
5. Student will be able to use various data sources with Knime, R
6. Student will be able to draw various Charts
6. Student will be able to explore data & data preparation.

Websites:

1. About R: https://www.r-project.org/about.html
2. About Excel: https://support.microsoft.com/en-us/office/excel-video-training-9bc05390-e94c-46af-a5b3-d7c22f6990bb
3. https://www.knime.com/learning

DATA SCIENCE TOOLS AND TECHNIQUES LAB

B.Tech. III Year I Sem **L T P C**

0 0 3 1.5

COURSE OBJECTIVES:

1. To learn Installation of R, Knime
2. To learn usage of EXCEL, R, KNIME for various data sources
3. To Perform various operations on tables of data sourse
4. To Create various visualizations
5. To learn various charts and plotting techniques

OUTCOMES:

After successfully studying this course, students will:

1. Learn Installation of R, KNIME
2. Learn usage of Excel,R, for various data sources
3. Perform various operations on tables for different data source
4. Create various visualizations
5. Various charts and plotting techniques

Week 1: (Using, Excel) Working with fundamental formulas, text, date, math and statistic functions.

Week 2: Working with conditional formatting, chats, what if analysis.

Week 3: Working with Data Analysis. Case study.

Week 4: Installation of R and KNIME

Week 5: Exploring the data using box plot, bar chart

Week 6: Implementation and use of data frames in R

Week 7: Study and implementation of Data Visualization with ggplot2

Week 8: Data Manipulation with dplyr package

Week 9: Study and implementation of various control structures in R

Week 10: Importing the different types of datasets in knime

Week 11: Download the VGsales data from kaggle and apply different types of filters in knime

Week 12: Download the bank dataset from UCI Repository and explore using knime.

Week 13: CASE Study: KNIME Testing the Model.

Text Books:

1. R for Data Science, O'Reilly by Hadley Wickham 2016.
2. Introduction to Data Science a Python approach to concepts, Techniques and Applications, Igual, L;Seghi', S. Springer, ISBN:978-3-319-50016-4.
3. Data Analysis with Excel by Manish Nigam. bpb Publications
4. KNIME Essentials, by Gábor Bakos, 2013
5. Data Science Tools by Christopher Greco, 2020
6. ALL-IN-ONE-EXCEL 2022 BIBLE FOR DUMMIES BY Bryant Shelton
7. Excel® 2019 BIBLE BY Michael Alexander, Dick Kusleika

UNIT - I

Data Science
Tools & Techniques
(R20A6704) – III B.Tech II Sem

By

Dr. M V Kamal

Professor & HoD

Dept. of Emerging Technologies (Domain Courses)

Data Science-Tools & Techniques (*DSTT)

Syllabus

UNIT I (Data Science and Various Data Science Tools)

Introduction to Data Science-Introduction- Definition - Data Science in various fields - Examples – Data Preparation - Data Pre-Processing and Data Wrangling with Techniques. Impact of Data Science - Data Analytics Life Cycle

Data Science Toolkit.: Brief Introduction to data science tools: SaS, Apache Spark, BigML, Excel, R-Programming, TensorFlow, KNIME, Tableau, PowerBI etc with advantages and disadvantages.

Syllabus

UNIT II (R Programming)

Introduction to R- Features of R – Environment, How to run R, R Sessions and Functions, Basic Math, Variables, Data Types, Vectors, Conclusion, Advanced Data Structures, Data Frames, Lists, Matrices, Arrays, Classes, R Programming Structures, Control Statements, Loops, - Looping Over Nonvector Sets,- If-Else, Arithmetic and Boolean Operators and values, Default Values for Argument, Return Values, Functions are Objects, Recursion,

Basic Functions - R help functions - R Data Structures. Vectors: Definition- Declaration - Generating - Indexing - Naming - Adding & Removing elements - Operations on Vectors - Recycling - Special Operators - Vectorized if- then else-Vector Equality – Functions for vectors - Missing values - NULL values - Filtering & Subsetting.

Syllabus

UNIT III (Excel & Advance Excel)

Introduction: Data Analysis, Excel Data analysis. Working with range names. Tables. Cleaning Data. Conditional formatting, Sorting, Advanced Filtering, Lookup functions, Pivot tables, Data Visualization, Data Validation. Understanding Analysis tool pack: Anova, correlation, covariance, moving average, descriptive statistics, exponential smoothing, fourier Analysis, Random number generation, sampling, t-test, f-test, and regression.

Syllabus

UNIT IV (KNIME) KNIME : Organizing your work, Nodes, Meta nodes, Ports, Flow variables, Node views. User Interface. Data Preparation: Importing Data-Database, tabular files, web services. Transforming the Shape- Filtering rows, Appending tables ,Less columns, More columns, Group By, Pivoting and Unpivoting, One2Many and Many2One,Cosmetic transformations.

Transforming values: Generic transformations , Conversion between types, Binning, Normalization, Multiple columns, XML transformation, Time transformation, Smoothing, Data generation, Constraints ,Loops, Workflow customization.

Syllabus

UNIT V (Data Exploration)

Computing statistics, Overview of visualizations, Visual guide for the views ,Distance matrix, Color , Size ,Shape ,KNIME views, HiLite , Use cases for HiLite, Row IDs, Extreme values. Basic KNIME views, The Box plots ,Hierarchical clustering, Histograms, Interactive Table, The Lift chart, Lines, Pie charts ,The Scatter plots, JFree Chart ,The Bar charts, The Bubble chart, Heatmap , The Histogram chart, The Interval chart, The Line chart, The Pie chart, The Scatter plot

Let's Start what is **DST&T**...??

Data Science-Tools & Techniques (*DSTT)

Syllabus

UNIT I (Data Science and Various Data Science Tools)

Introduction to Data Science-Introduction- Definition - Data Science in various fields - Examples – Data Preparation: Data Pre-Processing and Data Wrangling with Techniques. Impact of Data Science - Data Analytics Life Cycle

Data Science Toolkit.: Brief Introduction to data science tools: SaS, Apache Spark, BigML, Excel, R-Programming, TensorFlow, KNIME, Tableau, PowerBI etc with advantages and disadvantages.

What is **Data Science**..?

What is Data Science..?

- **Data science** provides meaningful information based on large amounts of complex **data** or big **data**. **Data science**, or **data**-driven **science**, combines different fields of work in statistics and computation to interpret **data** for decision-making purposes.

Data Science..

- Data science is the field **of study** that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data.

- Data Science is a blend of various tools, algorithms, and machine learning principles with the goal to discover hidden patterns from the raw data.

- In Simple, "An automated machine based insights extraction from huge volume of data".

What is Data Science..?(Contd..)

- Dealing with unstructured and structured data, Data Science is a field that comprises everything that related to data cleansing, preparation, and analysis.

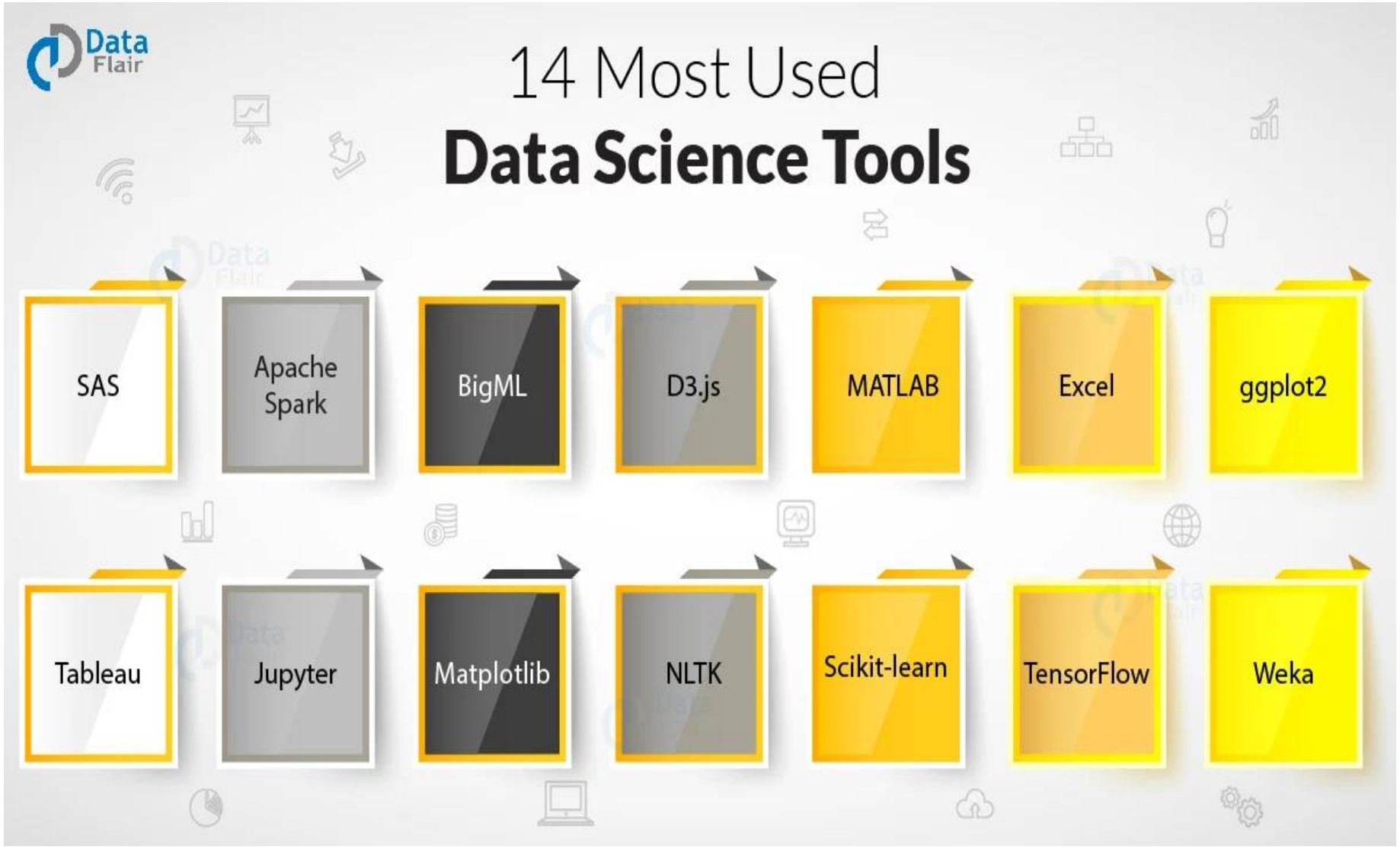
- Data Science is the combination of statistics, mathematics, programming, problem-solving, capturing data in ingenious ways, the ability to look at things differently, and the activity of cleansing, preparing, and aligning the data.

What are the popular

Data Science

Tools & Techniques

Data Science – Some of the Popular Tools



Contd…

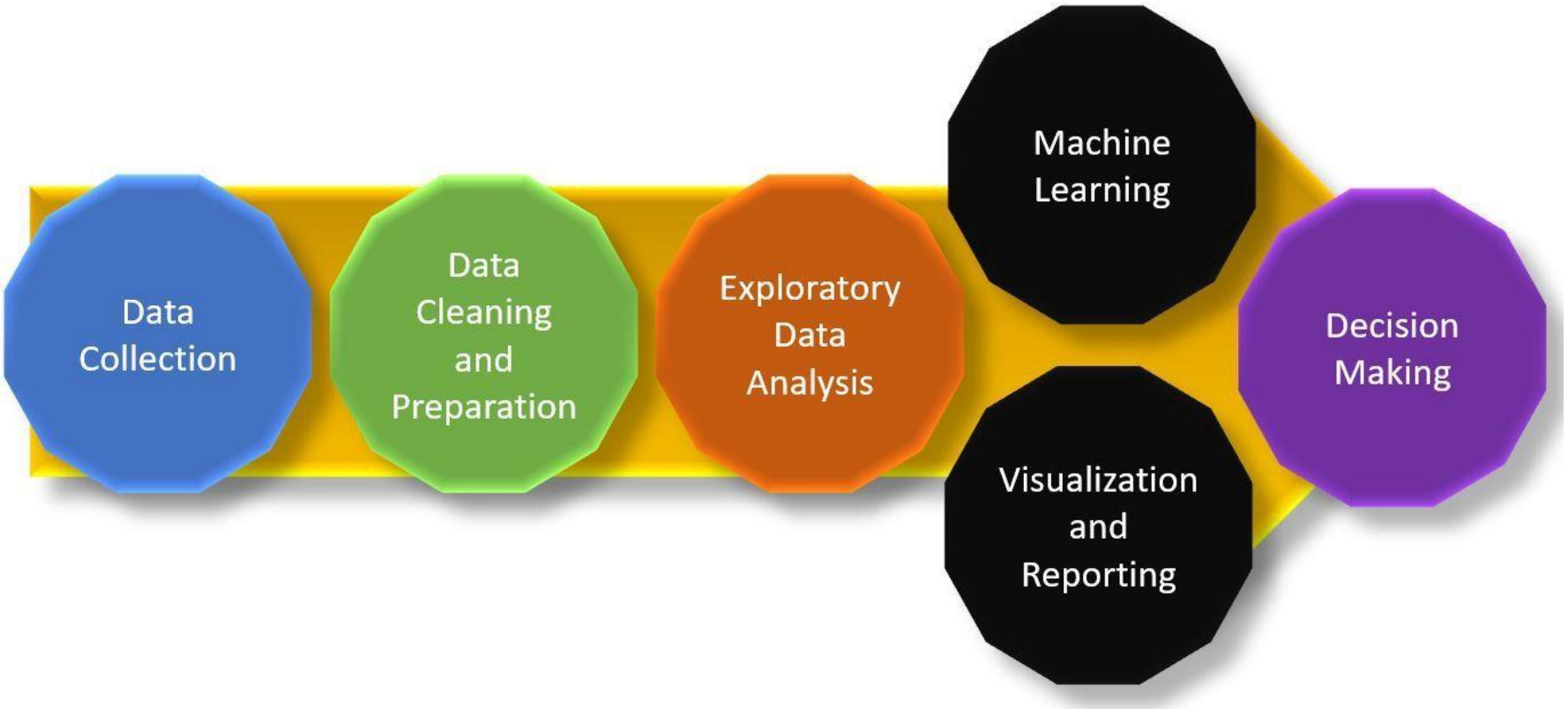
R
Microsoft Power BI
KNIME

Data Pre-Processing

Instructor: D

Data Preprocessing…

- Overall Process of Data Science…



Data Preprocessing…

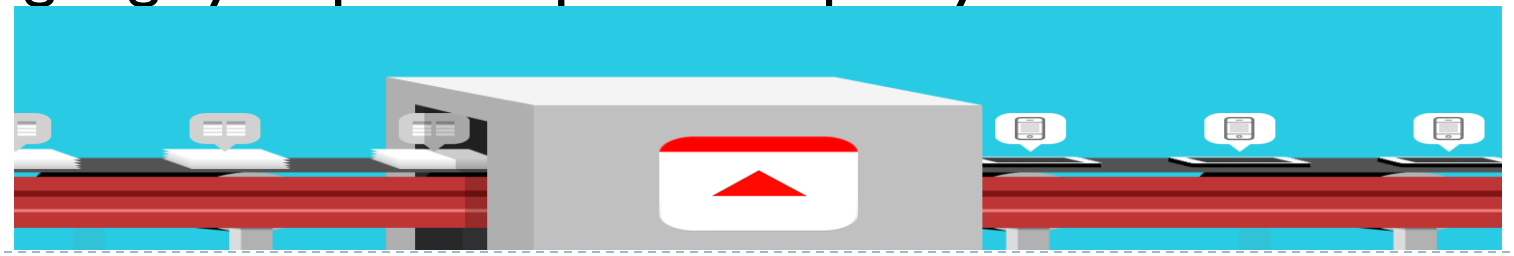
- Data preprocessing, a component of data preparation, describes any type of processing performed on raw data to prepare it for another data processing procedure.

- In any Machine Learning process, Data Preprocessing is that step in which the data gets transformed, or Encoded, to bring it to such a state that now the machine can easily parse it.

- In other words, the *features* of the data can now be easily interpreted by the algorithm.

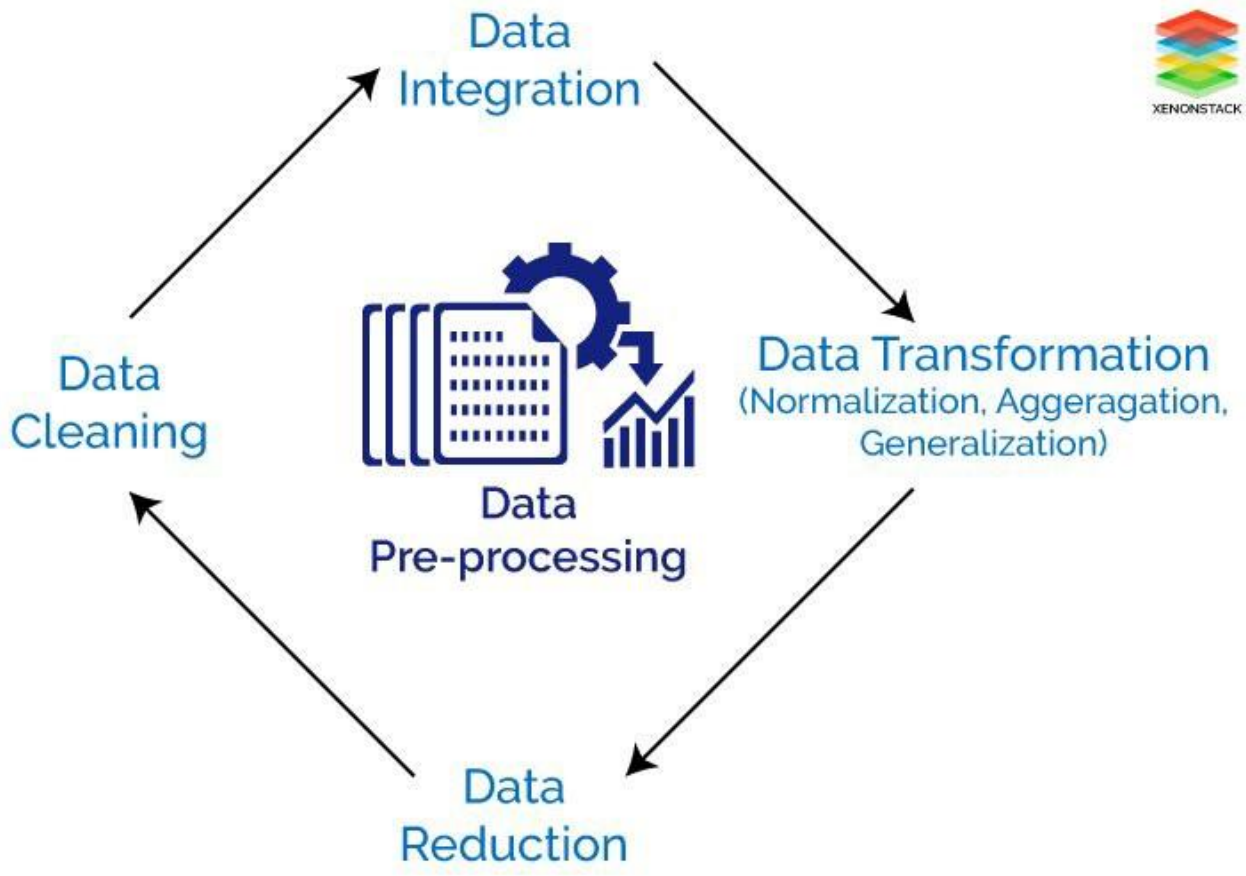
Why Data Pre-Processing is **required ?**

- Data pre-processing also knows as data wrangling is the technique of transforming the raw data i.e. an incomplete, inconsistent, data with lots of error, and data that lack certain behavior, into understandable format carefully using the different steps (i.e. from importing libraries, data to checking of missing values, categorical followed by validation and feature scaling) so that proper interpretations can be made from it and negative results can be avoided, as the quality of the model in machine learning highly depends upon the quality of data we train it on.



Tasks of Data Preprocessing

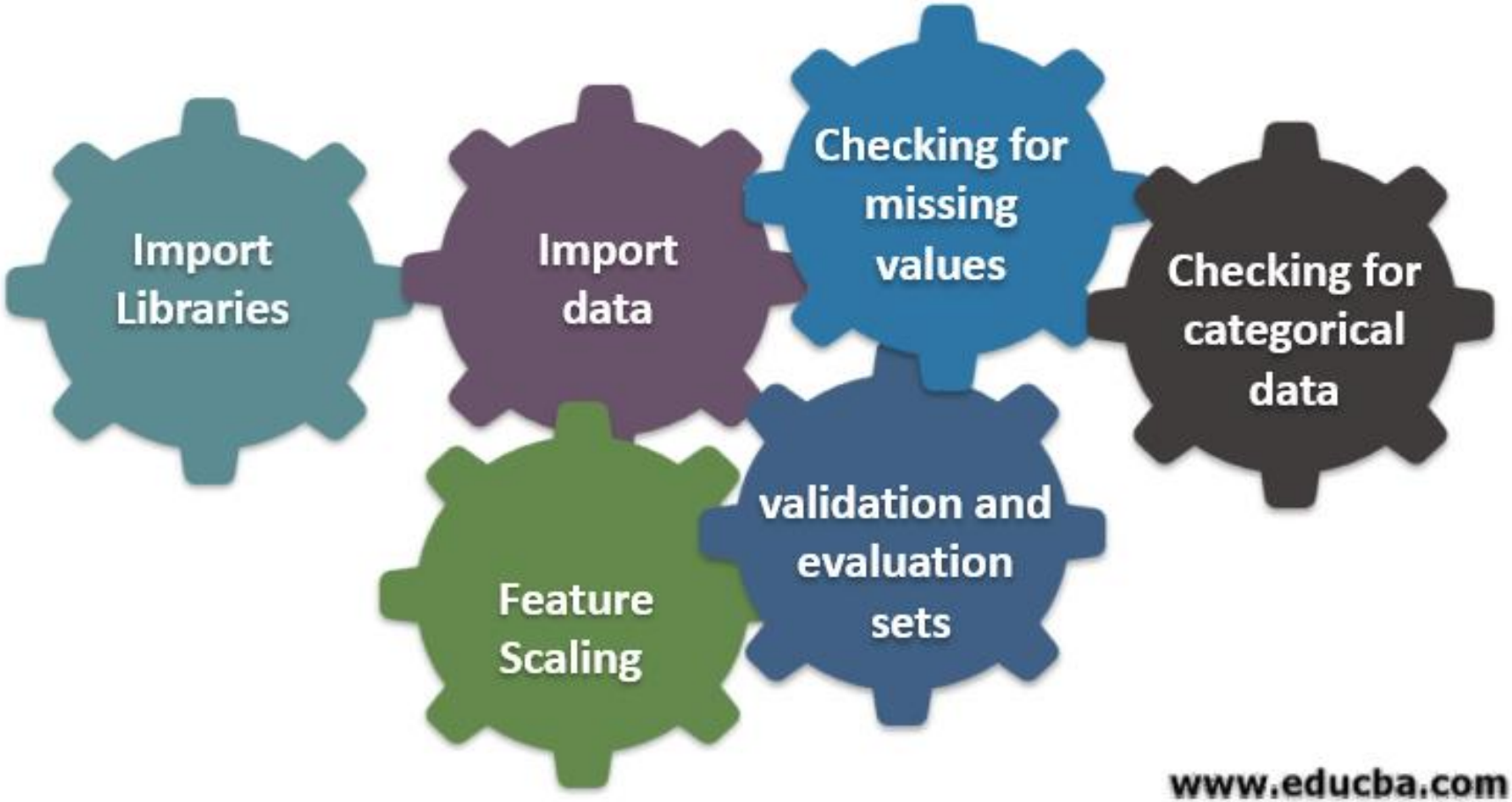
- Data Cleansing
- Data Editing
- Data Reduction
- Data Wrangling



What are the benefits of Data Preprocessing important?

- Inaccurate data (missing data)
- The presence of noisy data (erroneous data and outliers)
- Inconsistent data

Stages / Steps..



What Is Data Wrangling?

What Is Data Wrangling?

- **Data Wrangling** is a technique that is executed at the time of making an interactive model.

- In other words, it is used to convert the raw data into the format that is convenient for the consumption of data. This technique is also known as Data Munging.

- This method also follows certain steps such as after extracting the data from different data sources, sorting of data using the certain algorithms are performed, decompose the data into a different structured format and finally store the data into another database.

Data Mining & Data Wrangling

- Data mining is the process of finding patterns and relationships hidden in large data sets.
- Data mining helps businesses to decipher meaningful patterns in their data, whether it is open-source data or not.
- It is a superset of data mining and requires multiple other decision-making processes, such as data cleaning, transforming, integrating, etc.
- In this regard, wrangled data is important for accurate reporting and business intelligence insights.

Tasks of Data Wrangling

The tasks of Data wrangling are described below -

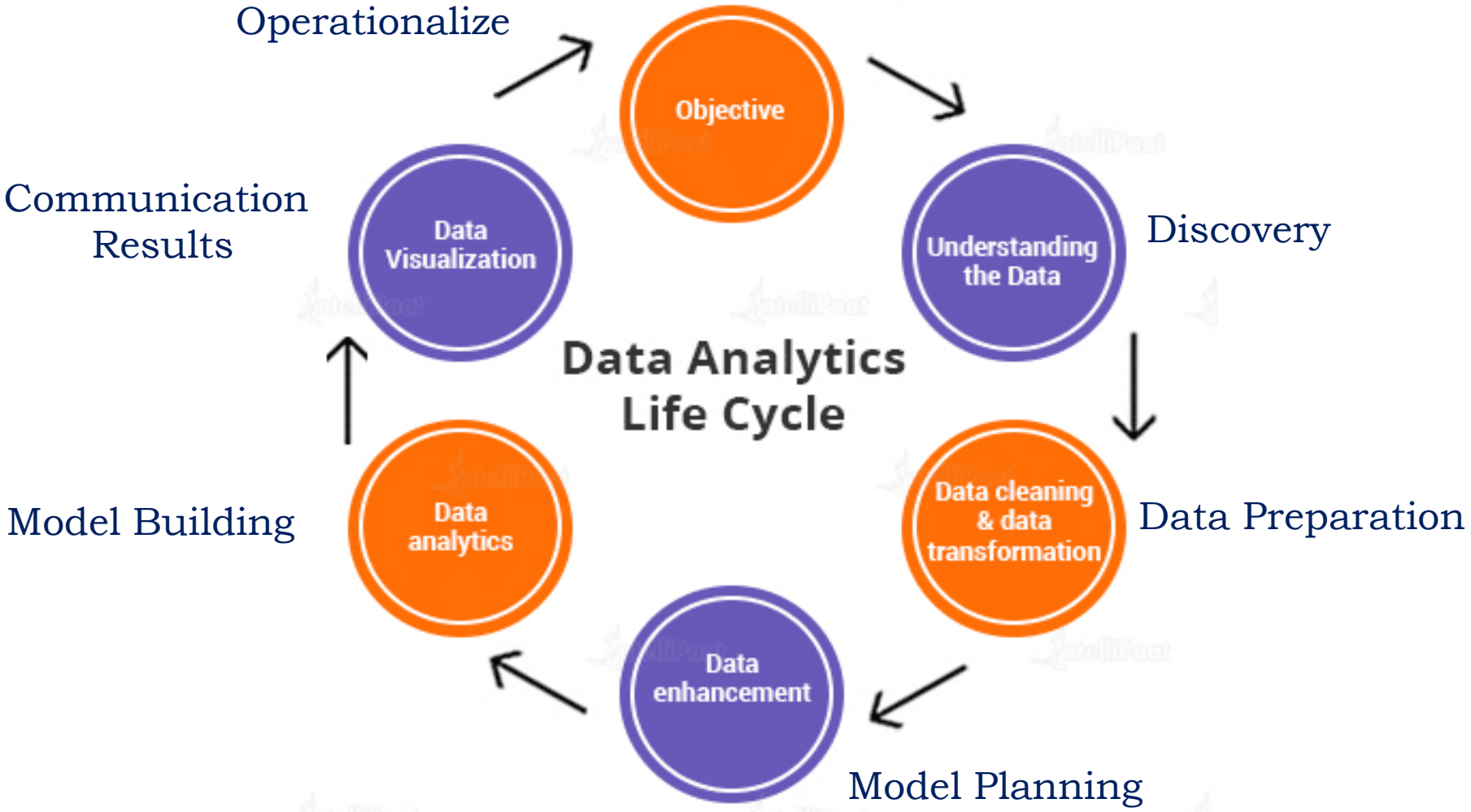
- Discovering
- Structuring
- Cleaning
- Enrichment
- Validating
- Publishing

Here are a few more amazing advantages with Data Wrangling

- Data Wrangling helps to improve **Data Usability** by converting data into a format that is compatible with the end system.
- It aids in the quick and easy creation of data flows in an **Intuitive User Interface** where the data flow process can be easily scheduled and automated.
- Data Wrangling also **integrates Different Types of Information**, as well as the sources, such as databases, files, web services, etc.
- Data wrangling allows users to **process Massive Volumes of Data** and share data flow techniques easily.
- **Reduces Variable Expenses** related to using external APIs or paying for software platforms that aren't considered business-critical.

Data Analytics Life Cycle

Data Analytics Life Cycle



Objective

- First and foremost, Understand the Objective, the purpose. Why are you doing this? What outcome you want from it?.

- If these questions are not clear, the rest is in vain. It is the same way that we do in SDLC (Software Development Life Cycle) model, If the requirement is not clear, then you might develop or test the software wrongly.

Phase 1: Discovery

Understand the Data:

- Data discovery is the process of collecting and analyzing disparate data sources to form a coherent picture of your company's data.
- The data science team learn and investigate the problem.
- Develop context and understanding.
- Come to know about data sources needed and available for the project.
- The team formulates initial hypothesis that can be later tested with data.

Phase 2: Data Preparation

Data Cleaning and Transformation

- Methods to investigate the possibilities of pre-processing, analyzing, and preparing data before analysis and modeling.
- It is required to have an analytic sandbox. The team performs, loads, and transforms to bring information to the data sandbox.
- Data preparation tasks can be repeated and not in a predetermined sequence.
- Some of the tools used commonly for this process include – Matlab, Hadoop, Alpine Miner, Open Refine, etc.

Data **Sandbox**..?

- What Is Data Sandbox?
- The goal of an analytical sandbox is to enable business people to conduct discovery and situational analytics.
- A data sandbox is **a secure environment that lets you test and learn with real-world data**.
- Data sandboxes help teams make more informed decisions by giving them access to valuable insights in large datasets. A data sandbox is a place where you can test and experiment with data.

Phase 3: Model Planning

- In this phase, the team will analyze the quality of the data and find an appropriate model for the project.
- An analytic sandbox is used to work with the data and to perform analytics throughout the project duration.

- Data can be loaded into the sandbox in three ways:
- **Extract, Transform, Load (ETL)** — The data is transformed based on a set of business rules and then loaded into the sandbox.
- **Extract, Load, Transform (ELT)** — The data is loaded into the sandbox and then transformed according to a set of business rules.
- **Extract, Transform, Load, Transform (ETLT)** — It has two transformation levels and is a combination of ETL and ELT.
- Common tools used are — R, SAS/ACCESS, SQL Analysis services,etc.

Phase 4: Model Building

- The team creates datasets for training, testing as well as production use.
- The team is also evaluating whether its current tools are sufficient to run the models or if they require an even more robust environment to run models.

- Common commercial tools used here are — Matlab, STATISTICA, Alpine Miner, etc.
- Common Free tools used are — R, Octave, WEKA, Python, etc.

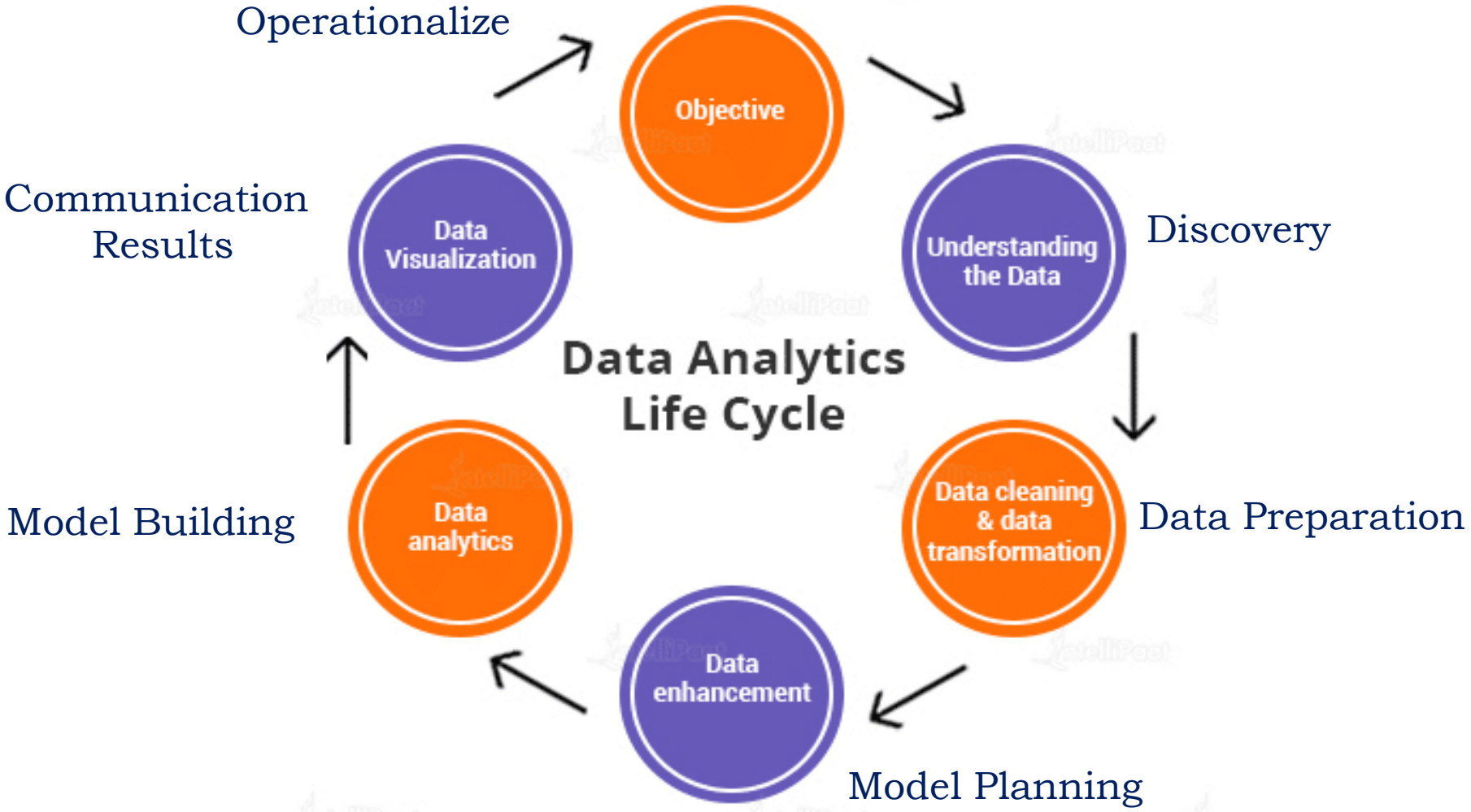
Phase 5: Communicate Results

- After executing model team need to compare outcomes of modeling to criteria established for success and failure.
- Team considers how best to articulate findings and outcomes to various team members and stakeholders, taking into account warning, assumptions.
- Team should identify key findings, quantify business value, and develop narrative to summarize and convey findings to stakeholders.

Phase 6: Operationalize

- The team communicates benefits of project more broadly and sets up pilot project to deploy work in controlled way before broadening the work to full enterprise of users.
- This approach enables team to learn about performance and related constraints of the model in production environment on small scale , and make adjustments before full deployment.
- The team delivers final reports, briefings, codes.
- Free or open source tools – Octave, WEKA, SQL, MADlib.

Data Analytics Life Cycle



Data Science-Tools & Techniques (*DSTT)

Syllabus

UNIT I (Data Science and Various Data Science Tools)

Introduction to Data Science-Introduction- Definition - Data Science in various fields - Examples – Data Preparation - Data Pre-Processing and Data Wrangling with Techniques. Impact of Data Science - Data Analytics Life Cycle

Data Science Toolkit.: Brief Introduction to data science tools: SaS, Apache Spark, BigML, Excel, R-Programming, TensorFlow, KNIME, Tableau, PowerBI etc with advantages and disadvantages.

Data Science Toolkit

Data Science Toolkit

- Brief Introduction to data science tools:
  - SaS,
  - Apache Spark,
  - BigML,
  - Excel,
  - R-Programming,
  - TensorFlow,
  - KNIME,
  - Tableau,
  - PowerBI

Data Science Toolkit

Refer the word document about Data Science Toolkit Topic

Click here

Data Science
Tools & Techniques
(R20A6704) – III B.Tech II Sem

By

Dr. M V Kamal

Professor & HoD

Dept. of Emerging Technologies (Domain Courses)

Data Science-Tools & Techniques (*DSTT)

Syllabus

UNIT I – PART-B

Data Science Toolkit.: Brief Introduction to data science tools: SaS, Apache Spark, BigML, Excel, R-Programming, TensorFlow, KNIME, Tableau, PowerBI etc with advantages and disadvantages.

SAS

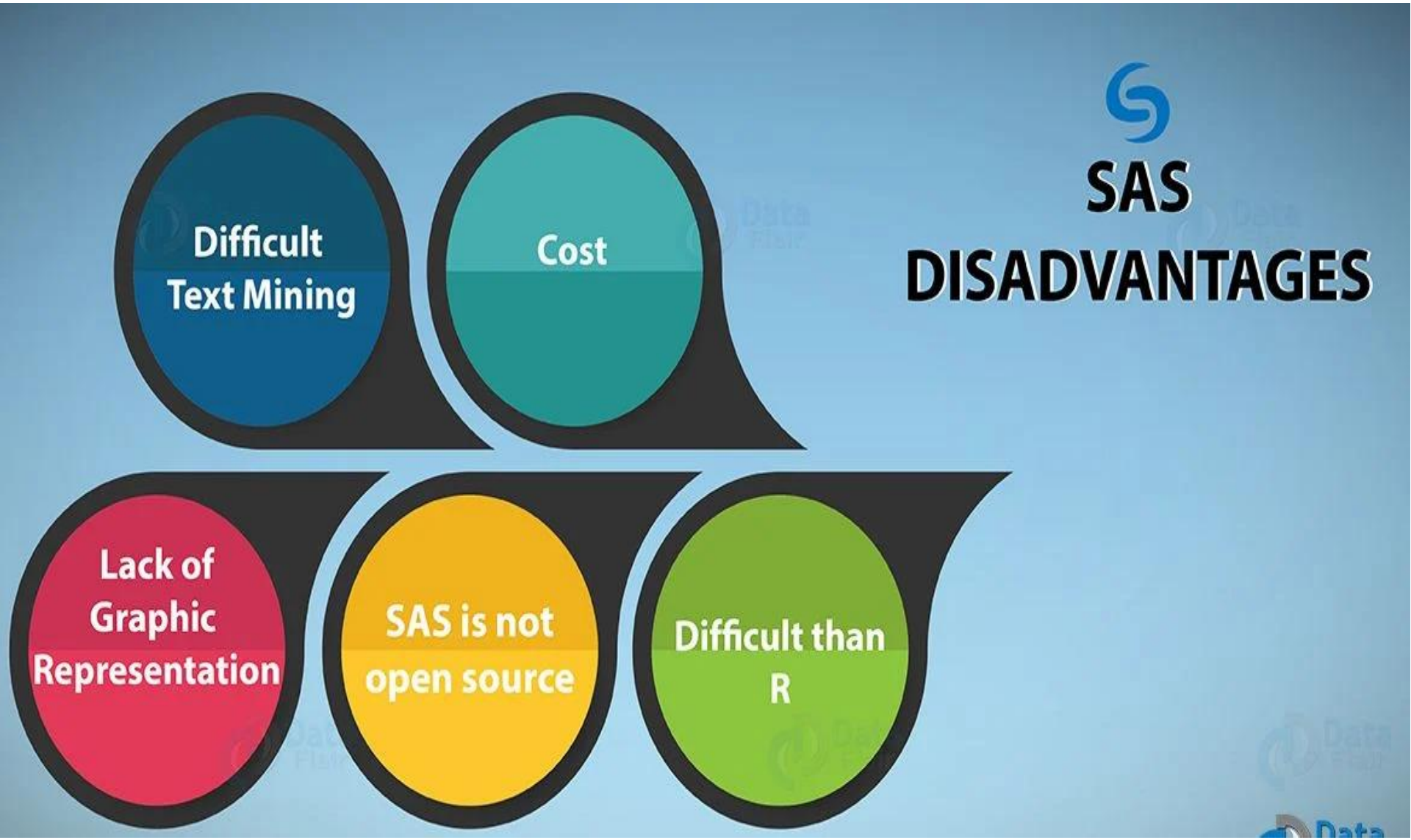
- Statistical Analysis System (SAS) is software that enables you to perform extensive data analysis. The full form of SAS is Statistical Analysis System. SAS was developed in the early 1970s at North Carolina State University.

- It is Used for data management, advanced analytics, multivariate analysis, business intelligence, criminal investigation, and predictive analytics.

- SAS Programming is a language used for analytical use. It is used for a very long time. All major companies use SAS as their official language for analysis. It's due to its features and edge points that SAS is a very preferred tool. It has a huge job market too.
- There are many reasons for which SAS is preferred over **R programming language** and Python. But it is also true that there are some limitations of SAS that are overcome by R and Python.

SaS Advantages

SaS Disadvantages

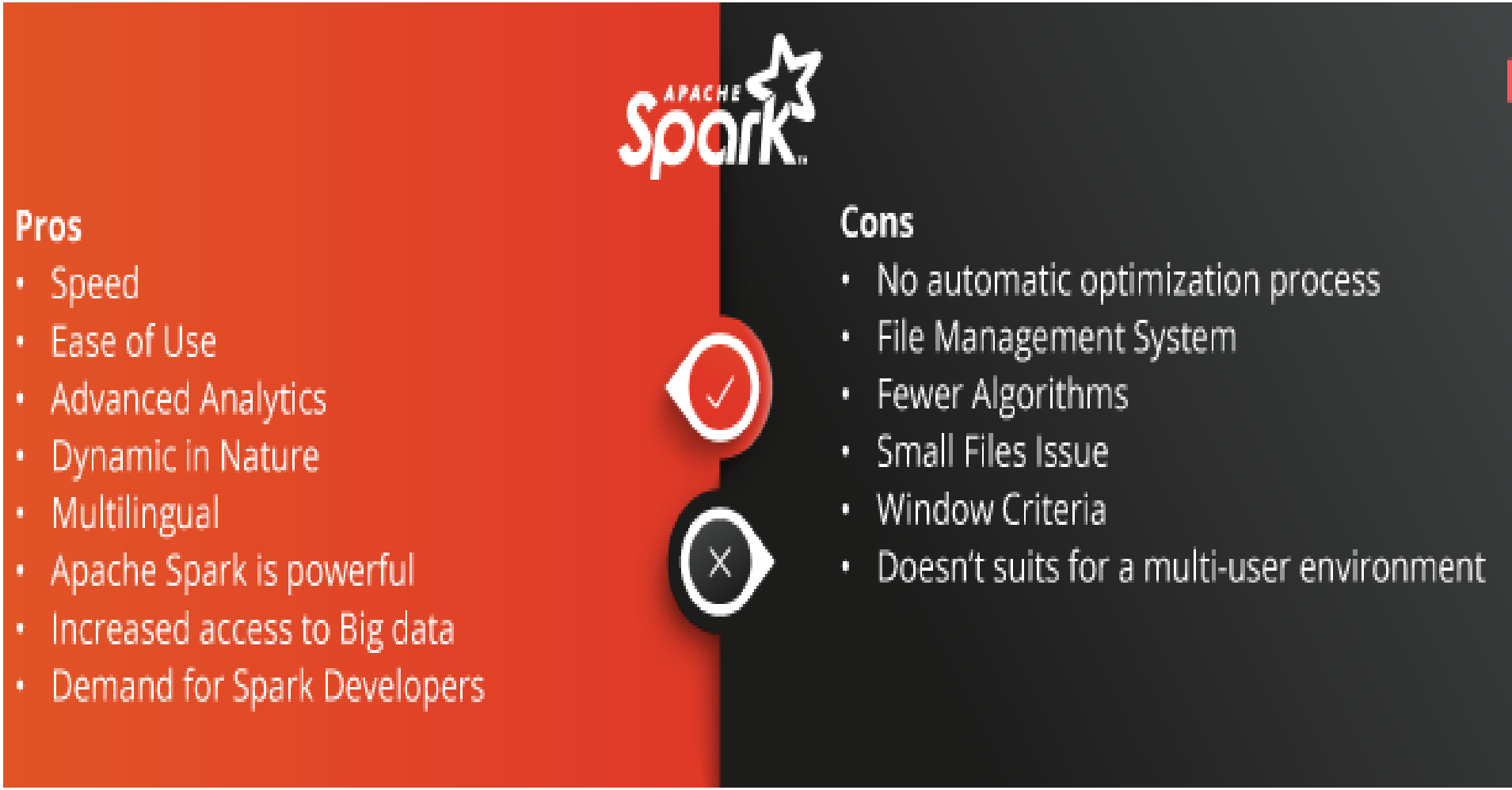


Apache Spark

- **Apache Spark** is a lightning-fast unified analytics engine for big data and machine learning. It is the largest open-source project in data processing.
- Since its release, it has met the enterprise's expectations in a better way in regards to querying, data processing and moreover generating analytics reports in a better and faster way.

- Internet substations like Yahoo, Netflix, and eBay, etc have used Spark at large scale. Apache Spark is considered as the future of Big Data Platform.

Advantages and Disadvantage of Apache Spark



bigml®

BigML

- BigML is a machine learning platform designed to enable developers to build enterprise-level applications that are capable of real-time predictions.

- It has a user-friendly and visually engaging interface that allows even beginners and amateurs to program models and execute actions.

BigML Advantage

- The main benefits of BigML are its integration and automation capabilities, its flexibility to your preferred programming language, and its ability to serve real-time predictions.
- And even..
  - **Scalability**
  - **Bindings & Adaptability**
  - **One-Line Automation**

Disadvantage of BigML

- Some advanced features like ensemble methods and anomaly detection.

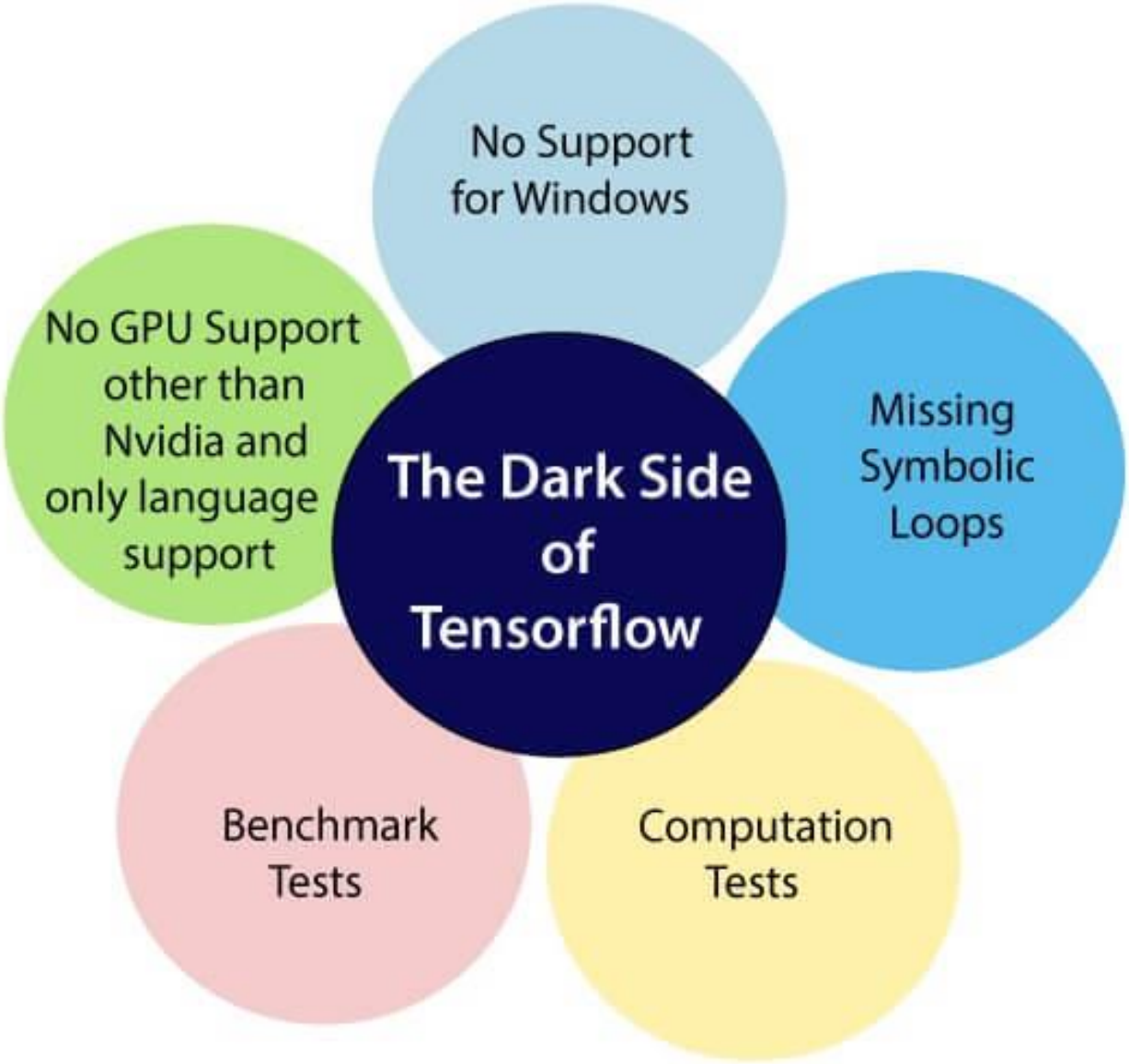

TensorFlow

TensorFlow

- TensorFlow is an open-source machine learning concept which is designed and developed by Google.
- It offers a very high level and abstract approach to organizing low-level numerical programming.
- TensorFlow stands as the trending and competition among its associates.
- With all its capabilities, it eases the computations of the machine and deep learning.

The Good Stuff about Tensorflow

- Graphs
- Library Management
- Debugging
- Scalability
- Pipelining


No Support for Windows
No GPU Support other than Nvidia and only language support
The Dark Side of Tensorflow
Missing Symbolic Loops
Benchmark Tests
Computation Tests

Power BI

Power BI

- Microsoft Power BI is an interactive data visualization software product that helps users find insights within an organization's data.

- It's part of the Microsoft Power Platform and is designed for business intelligence

Power BI

- Power BI is a collection of software services, apps, and connectors that work together to turn your unrelated sources of data into coherent, visually immersive, and interactive insights.

- Your data might be an Excel spreadsheet, or a collection of cloud-based and on-premises hybrid data warehouses. Power BI lets you easily connect to your data sources, visualize and discover what's important, and share that with anyone or everyone you want.

-Source: Microsoft

Power BI

- **Power BI can help users:**
- Connect data sets
- Transform and clean data into a data model
- Create charts or graphs to provide visuals of the data
- Create reports and dashboards that present data sets in multiple ways using visuals
- See data through a single pane of glass
- Offers a consolidated view across a business

Power BI

PRO

- Cost- Free to Use
- Learning Curve
- Constant Updates and Innovations
- Data Sources
- Excel Integration
- Custom Visualizations

CONS

- The User Interface
- Rigid Formulas
- Visuals Configurability
- Table Relationships

18 DATA SCIENCE TOOLS TO CONSIDER USING IN 2024

-Source: https://www.techtarget.com

Numerous tools are available for data science applications. Read about 18, including their features, capabilities and uses, to see if they fit your analytics needs.

The increasing volume and complexity of enterprise data as well as its central role in decision-making and strategic planning are driving organizations to invest in the people, processes and technologies they need to gain valuable business insights from their data assets. That includes a variety of tools commonly used in data science applications.

In an annual survey conducted by consulting firm Wavestone, 87.9% of chief data officers and other IT and business executives from 102 large organizations said investments in data and analytics are a top priority. Looking ahead, 82.2% expect spending increases this year, according to a report on the Data and AI Executive Leadership Survey that was published in December 2023.

The survey also found that 87% of the responding organizations got measurable business value from their data and analytics investments in 2023 -- slightly down from 91.9% in last year's survey. But strategic analytics goals improved. Half of the respondents said they're competing on data and analytics, up about 10% from 2022. Also, 48.1% feel they have created a data-driven organization -- more than double last year's 23.9%.

As data science teams build their portfolios of enabling technologies to achieve those analytics goals, they can choose from a wide selection of tools and platforms. Here's a rundown of 18 top data science tools that might aid you in the analytics process, listed in alphabetical order, with details on their features and capabilities as well as some potential limitations.

1. Apache Spark

Apache Spark is an open source data processing and analytics engine that can handle large amounts of data -- upward of several petabytes, according to proponents. Spark's ability to rapidly process data has fueled significant growth in the use of the platform since it was created in 2009, making the Spark project one of the largest open source communities among big data technologies.

Due to its speed, Spark is well suited for continuous intelligence applications powered by near-real-time processing of streaming data. However, as a general-purpose distributed processing engine, Spark is equally suited for extract, transform and load uses as well as other SQL batch jobs. In fact, Spark initially was touted as a faster alternative to the MapReduce engine for batch processing in Hadoop clusters.

Spark is still often used with Hadoop but can also run standalone against other file systems and data stores. It features an extensive set of developer libraries and APIs, including a machine learning library and support for key programming languages, making it easier for data scientists to quickly put the platform to work.

2. D3.js

Another open source tool, D3.js is a JavaScript library for creating custom data visualizations in a web browser. Commonly known as D3, which stands for Data-Driven Documents, it uses web standards such as HTML, Scalable Vector Graphics and CSS instead of its own graphical vocabulary. D3's developers describe it as a dynamic and flexible tool that requires a minimum amount of effort to generate visual representations of data.

D3.js lets visualization designers bind data to documents via the Document Object Model and then use DOM manipulation methods to make data-driven transformations to the documents. First released in 2011, it can be used to design various types of data visualizations and supports features such as interaction, animation, annotation and quantitative analysis.

D3 includes more than 30 modules and 1,000 visualization methods, making it complicated to learn. In addition, many data scientists don't have JavaScript skills.

As a result, they might be more comfortable with a commercial visualization tool such as Tableau, leaving D3 to be used more by data visualization developers and specialists who are also members of data science teams.

3. IBM SPSS

IBM SPSS is a family of software for managing and analyzing complex statistical data. It includes two primary products: SPSS Statistics, a statistical analysis, data visualization and reporting tool, and SPSS Modeler, a data science and predictive analytics platform with a drag-and-drop UI and machine learning capabilities.

SPSS Statistics covers every step of the analytics process, from planning to model deployment, and enables users to clarify relationships between variables, create clusters of data points, identify trends and make predictions, among other capabilities. It can access common structured data types and offers a combination of a menu-driven UI, its own command syntax, and the ability to integrate R and Python extensions. It also has features for automating procedures and import-export ties to SPSS Modeler.

Created by SPSS Inc. in 1968, initially with the name Statistical Package for the Social Sciences, the statistical analysis software was acquired by IBM in 2009, along with the predictive modeling platform, which SPSS had previously bought. While the product family is officially called IBM SPSS, the software is still usually known simply as SPSS.

4. Julia

Julia is an open source programming language used for numerical computing as well as machine learning and other kinds of data science applications. In a 2012 blog post announcing Julia, its four creators said they set out to design one language that addressed all their needs. A big goal was to avoid having to write programs in one language and convert them to another for execution.

To that end, Julia combines the convenience of a high-level dynamic language with performance that's comparable to statically typed languages, such as C and Java. Users don't have to define data types in programs, but an option allows them to do

so. The use of a multiple dispatch approach at runtime also helps to boost execution speed.

Julia 1.0 became available in 2018, nine years after work began on the language; the latest version is 1.9.4, with a 1.10 update now available for release candidate testing. The documentation for Julia notes that because its compiler differs from the interpreters in data science languages like Python and R, new users "may find that Julia's performance is unintuitive at first." But, it claims, "once you understand how Julia works, it's easy to write code that's nearly as fast as C."

5. Jupyter Notebook

An open source web application, Jupyter Notebook enables interactive collaboration among data scientists, data engineers, mathematicians, researchers and other users. It's a computational notebook tool that can be used to create, edit and share code as well as explanatory text, images and other information. For example, Jupyter users can add software code, computations, comments, data visualizations and rich media representations of computation results to a single document, known as a *notebook*, which can then be shared with and revised by colleagues.

As a result, notebooks "can serve as a complete computational record" of interactive sessions among the members of data science teams, according to Jupyter Notebook's documentation. The notebook documents are JSON files that have version control capabilities. In addition, a Notebook Viewer service lets users render notebooks as static webpages for viewing by users who don't have Jupyter installed on their systems.

Jupyter Notebook's roots are in the programming language Python. It was originally part of the open source IPython interactive toolkit project before being split off in 2014. The loose combination of Julia, Python and R gave Jupyter its name. Along with supporting those three languages, Jupyter has modular kernels for dozens of others. The open source project also includes JupyterLab, a newer web-based UI that's more flexible and extensible than the original one.

6. Keras

Keras is a programming interface that enables data scientists to access and use the TensorFlow machine learning platform more easily. It's an open source deep learning API and framework written in Python that runs on top of TensorFlow. It is now integrated into that platform. Keras previously supported multiple back ends but was tied exclusively to TensorFlow starting with its 2.4.0 release in June 2020.

As a high-level API, Keras was designed to drive easy and fast experimentation that requires less coding than other deep learning options. The goal is to accelerate the implementation of machine learning models -- in particular, deep learning neural networks -- through a development process with "high iteration velocity," as the Keras documentation puts it.

The Keras framework includes a sequential interface for creating relatively simple linear stacks of *layers* with inputs and outputs as well as a functional API for building more complex graphs of layers or writing deep learning models from scratch. Keras models can run on CPUs or GPUs and be deployed across multiple platforms, including web browsers as well as Android and iOS mobile devices.

7. Matlab

Developed and sold by software vendor MathWorks since 1984, Matlab is a high-level programming language and analytics environment for numerical computing, mathematical modeling and data visualization. It's primarily used by conventional engineers and scientists to analyze data; design algorithms; and develop embedded systems for wireless communications, industrial control, signal processing and other applications. This is often in concert with a companion Simulink tool that offers model-based design and simulation capabilities.

While Matlab isn't as widely used in data science applications as languages such as Python, R and Julia, it does support machine learning and deep learning, predictive modeling, big data analytics, computer vision, and other work done by data scientists. Data types and high-level functions built into the platform are designed to speed up exploratory data analysis and data preparation in analytics applications.

Considered relatively easy to learn and use, Matlab -- short for "*matrix laboratory*" -- includes prebuilt applications but also lets users build their own. It also has a library of add-on toolboxes with discipline-specific software and hundreds of built-in functions, including the ability to visualize data in 2D and 3D plots.

8. Matplotlib

Matplotlib is an open source Python plotting library that's used to read, import and visualize data in analytics applications. Data scientists and other users can create static, animated and interactive data visualizations with Matplotlib, using it in Python scripts, the Python and IPython shells, Jupyter Notebook, web application servers, and various GUI toolkits.

The library's large code base can be challenging to master, but it's organized in a hierarchical structure that's designed to enable users to build visualizations mostly with high-level commands. The top component in the hierarchy is pyplot, a module that provides a "state-machine environment" and a set of simple plotting functions like those in Matlab.

First released in 2003, Matplotlib also includes an object-oriented interface that can be used together with pyplot or on its own. It supports low-level commands for more complex data plotting. The library is primarily focused on creating 2D visualizations but offers an add-on toolkit with 3D plotting features.

9. NumPy

Short for "Numerical Python," NumPy is an open source Python library that's used widely in scientific computing, engineering, and data science and machine learning applications. The library consists of multidimensional array objects and routines for processing those arrays to enable various mathematical and logic functions. It also supports linear algebra, random number generation and other operations.

One of NumPy's core components is the N-dimensional array (ndarray) which represents a collection of items that are the same type and size. An associated data-type object describes the format of the data elements in an array. The same data

can be shared by multiple ndarrays, and data changes made in one can be viewed in another.

NumPy was created in 2006 by combining and modifying elements of two earlier libraries. The NumPy website touts it as "the universal standard for working with numerical data in Python." It is generally considered one of the most useful libraries for Python because of its numerous built-in functions. It's also known for its speed, partly resulting from the use of optimized C code at its core. In addition, various other Python libraries are built on top of NumPy.

10. Pandas

Another popular open source Python library, pandas typically is used for data analysis and manipulation. Built on top of NumPy, it features two primary data structures: the Series one-dimensional array and the DataFrame, a two-dimensional structure for data manipulation with integrated indexing. Both can accept data from NumPy ndarrays and other inputs. A DataFrame can also incorporate multiple Series objects.

Created in 2008, pandas has built-in data visualization capabilities; exploratory data analysis functions; and support for file formats and languages that include CSV, SQL, HTML and JSON. Additionally, it provides features such as intelligent data alignment, integrated handling of missing data, flexible reshaping and pivoting of data sets, data aggregation and transformation, and the ability to quickly merge and join data sets, according to the pandas website.

The developers of pandas say their goal is to make it "the fundamental high-level building block for doing practical, real-world data analysis in Python." Key code paths in pandas are written in C or the Cython superset of Python to optimize its performance. The library can be used with various kinds of analytical and statistical data, including tabular, time series and labeled matrix data sets.

11. Python

Python is the most widely used programming language for data science and machine learning and one of the most popular languages overall. The Python open source project's website describes it as "an interpreted, object-oriented, high-level

programming language with dynamic semantics," built-in data structures, and dynamic typing and binding capabilities. The site also touts Python's simple syntax, saying it's easy to learn and its emphasis on readability reduces the cost of program maintenance.

The multipurpose language can be used for a wide range of tasks, including data analysis, data visualization, AI, natural language processing and robotic process automation. Developers can create web, mobile and desktop applications in Python too. In addition to object-oriented programming, it supports procedural, functional and other types plus extensions written in C or C++.

Python is used not only by data scientists, programmers and network engineers but also by workers outside of computing disciplines, from accountants to mathematicians and scientists. They are often drawn to its user-friendly nature. Python 2.x and 3.x are both production-ready versions of the language, although support for the 2.x line ended in 2020.

12. PyTorch

An open source framework used to build and train deep learning models based on neural networks, PyTorch is touted by its proponents for supporting fast and flexible experimentation as well as a seamless transition to production deployment. The Python library was designed to be easier to use than Torch, a precursor machine learning framework that's based on the Lua programming language. PyTorch also provides more flexibility and speed than Torch, according to its creators.

First released publicly in 2017, PyTorch uses arraylike tensors to encode model inputs, outputs and parameters. Its tensors are similar to the multidimensional arrays supported by NumPy, but PyTorch adds built-in support for running models on GPUs. NumPy arrays can be converted into tensors for processing in PyTorch and vice versa.

The library includes various functions and techniques, including an automatic differentiation package named torch.autograd, a module for building neural networks, a TorchServe tool for deploying PyTorch models, and deployment support for iOS and Android devices. In addition to the primary Python API, PyTorch offers a C++

API that can be used as a separate front-end interface or to create extensions for Python applications.

13. R Programming

The R programming language is an open source environment designed for statistical computing and graphics applications as well as data manipulation, analysis and visualization. Many data scientists, academic researchers and statisticians use R to retrieve, cleanse, analyze and present data, making it one of the most popular languages for data science and advanced analytics.

The open source project is supported by The R Foundation, and thousands of user-created packages with libraries of code that enhance R's functionality are available. One example is ggplot2, a well-known package for creating graphics that's part of a collection of R-based data science tools named tidyverse. In addition, multiple vendors offer integrated development environments and commercial code libraries for R.

R is an interpreted language like Python and has a reputation for being relatively intuitive. It was created in the 1990s as an alternative version of S, a statistical programming language that was developed in the 1970s. R's name is both a play on S and a reference to the first letter of the names of its two creators.

14. SAS

SAS is an integrated software suite for statistical analysis, advanced analytics, BI and data management. Developed and sold by software vendor SAS Institute Inc., the platform helps users integrate, cleanse, prepare and manipulate data. They can then analyze it using different statistical and data science techniques. SAS can be used for various tasks from basic BI and data visualization to risk management, operational analytics, data mining, predictive analytics and machine learning.

The development of SAS started in 1966 at North Carolina State University. Use of the technology began to grow in the early 1970s, and SAS Institute was founded in 1976 as an independent company. The software was initially built for use by statisticians -- SAS was short for Statistical Analysis System. But over time, it was

expanded to include a broad set of functionality and became one of the most widely used analytics suites in both commercial enterprises and academia.

Development and marketing are now focused primarily on SAS Viya, a cloud-based version of the platform that was launched in 2016 and redesigned to be cloud-native in 2020.

15. Scikit-learn

Scikit-learn is an open source machine learning library for Python that's built on the SciPy and NumPy scientific computing libraries as well as Matplotlib for plotting data. It supports both supervised and unsupervised machine learning and includes numerous algorithms and models, called *estimators* in scikit-learn parlance. Additionally, it provides functionality for model fitting, selection and evaluation, and data preprocessing and transformation.

Initially called scikits.learn, the library started as a Google Summer of Code project in 2007, and the first public release became available in 2010. The first part of its name is short for "*SciPy toolkit*" and is also used by other SciPy add-on packages. Scikit-learn primarily works on numeric data that's stored in NumPy arrays or SciPy sparse matrices.

The library's suite of tools also enables various other tasks, such as data set loading and the creation of workflow pipelines that combine data transformer objects and estimators. But scikit-learn has some limits due to design constraints. For example, it doesn't support deep learning, reinforcement learning or GPUs. The library's website also says its developers "only consider well-established algorithms for inclusion."

16. SciPy

SciPy is another open source Python library that supports scientific computing uses. Short for Scientific Python, it features a set of mathematical algorithms and high-level commands and classes for data manipulation and visualization. It includes more than a dozen subpackages that contain algorithms and utilities for functions such as data optimization, integration and interpolation as well as algebraic equations, differential equations, image processing and statistics.

The SciPy library is built on top of NumPy and can operate on NumPy arrays. But SciPy delivers additional array computing tools and provides specialized data structures, including sparse matrices and K-dimensional trees, to extend beyond NumPy's capabilities.

SciPy predates NumPy; it was created in 2001 by combining different add-on modules built for the Numeric library that was one of NumPy's predecessors. Like NumPy, SciPy uses compiled code to optimize performance. In its case, most of the performance-critical parts of the library are written in C, C++ or Fortran.

17. TensorFlow

TensorFlow is an open source machine learning platform developed by Google that's particularly popular for implementing deep learning neural networks. The platform takes inputs in the form of tensors that are akin to NumPy multidimensional arrays and then uses a graph structure to flow the data through a list of computational operations specified by developers. It also offers an eager execution programming environment that runs operations individually without graphs, which provides more flexibility for research and debugging machine learning models.

Google made TensorFlow open source in 2015, and Release 1.0.0 became available in 2017. TensorFlow uses Python as its core programming language and now incorporates the Keras high-level API for building and training models. Alternatively, a TensorFlow.js library enables model development in JavaScript, and custom operations -- *ops*, for short -- can be built in C++.

The platform also includes a TensorFlow Extended module for end-to-end deployment of production machine learning pipelines as well as a TensorFlow Lite module for mobile and IoT devices. TensorFlow models can be trained and run on CPUs, GPUs and Google's special-purpose Tensor Processing Units.

18. Weka

Weka is an open source workbench that provides a collection of machine learning algorithms for use in data mining tasks. Weka's algorithms, called *classifiers*, can be applied directly to data sets without any programming via a GUI or a command-line

interface that offers additional functionality. They can also be implemented through a Java API.

The workbench can be used for classification, clustering, regression, and association rule mining applications. It also includes a set of data preprocessing and visualization tools. In addition, Weka supports integration with R, Python, Spark and other libraries like scikit-learn. For deep learning uses, an add-on package combines it with the Eclipse Deeplearning4j library.

Weka is free software licensed under the GNU General Public License. It was developed at the University of Waikato in New Zealand starting in 1992. An initial version was rewritten in Java to create the current workbench, which was first released in 1999. Weka stands for the Waikato Environment for Knowledge Analysis. It is also the name of a flightless bird native to New Zealand that the technology's developers say has "an inquisitive nature."

Data science and machine learning platforms

Commercially licensed platforms that provide integrated functionality for machine learning, AI and other data science applications are also available from numerous software vendors. The product offerings are diverse. They include machine learning operations hubs, automated machine learning platforms and full-function analytics suites, with some combining MLOps, AutoML and analytics capabilities. Many platforms incorporate some of the data science tools listed above.

Matlab and SAS can also be counted among the data science platforms. Other prominent platform options for data science teams include the following technologies:

- Anaconda.
- Alteryx Analytics Automation Platform.
- Amazon SageMaker.
- Azure Machine Learning.
- BigML.

- Databricks Lakehouse Platform.
- Dataiku.
- DataRobot AI Cloud.
- Domino Enterprise MLOps Platform.
- Google Cloud Vertex AI.
- H2O AI Cloud.
- IBM Watson Studio.
- Knime.
- Qubole.
- RapidMiner.
- Saturn Cloud.
- Tibco Data Science.

Some platforms are also available in free open source or community editions. Examples include Dataiku and H2O. Knime combines an open source analytics platform with a commercial Knime Hub software package that supports team-based collaboration and workflow automation, deployment and management.

For More details: https://www.techtarget.com/searchbusinessanalytics/feature/15-data-science-tools-to-consider-using

UNIT - II

Data Science Tools and Techniques

Unit-II

R and R Studio Installation

By: M V Kamal | Associate Professor | CSE Dept. | MRCET

What is

& Purpose of

About R Programming…

- **R** is a programming language and free software environment for **statistical computing** and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.
- R is an implementation of the S programming language combined with lexical scoping semantics, inspired by Scheme. S was created by John Chambers in 1976, while at Bell Labs.
- R was created by **Ross Ihaka** and **Robert Gentleman** at the University of Auckland in 1993.
- It is now available under GNU General Public License

Features of R Programming

1. Open Source

R is an open-source programming language. This means that anyone can work with R without any need for a license or a fee. Furthermore, you can contribute towards the development of R by *customizing its packages, developing new ones and resolving issues*.

2. Exemplary Support for Data Wrangling

R provides exemplary support for data wrangling. The packages like *dplyr, readr* are capable of transforming messy data into a structured form.

3. The Array of Packages

R has a vast array of packages. With over 10,000 packages in the **CRAN repository**, the number is constantly growing. These packages appeal to all the areas of industry.

4. Quality Plotting and Graphing

R facilitates quality plotting and graphing. The popular libraries like *ggplot2* and *plotly* advocate for aesthetic and visually appealing graphs that set R apart from other programming languages.

Benefits of R Programming…Contd..

5. Highly Compatible

R is highly compatible and can be paired with many other programming languages like C, C++, Java, and Python. It can also be integrated with technologies like Hadoop and various other database management systems as well.

6. Platform Independent

R is a platform-independent language. It is a cross-platform programming language, meaning that it can be run quite easily on Windows, Linux, and Mac.

7. Eye-Catching Reports

With packages like Shiny and Markdown, reporting the results of an analysis is extremely easy with R. You can make reports with the data, plots and R scripts embedded in them. You can even make interactive web apps that allow the user to play with the results and the data.

Benefits of R Programming…Contd..

8. Machine Learning Operations

R provides various facilities for carrying out machine learning operations like *classification, regression and also provides features for developing artificial neural networks.*

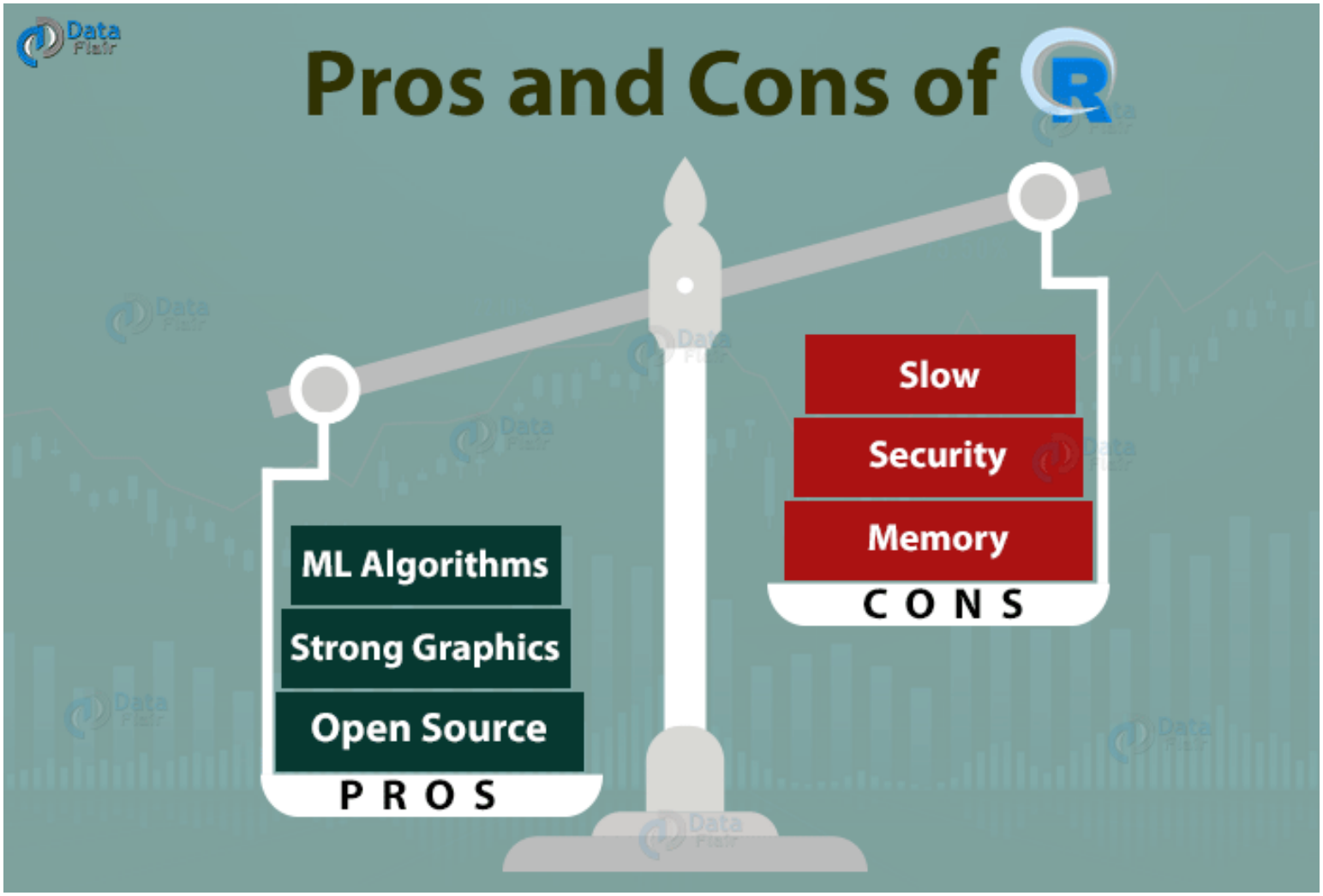
9. Statistics

R is prominently known as the lingua franca of statistics. This is the main reason as to why R is dominant among other programming languages for developing statistical tools.

10. Continuously Growing

R is a constantly evolving programming language. It is a state of the art technology that provides updates whenever any new feature is added.

Pros and Cons of R Programming



Install of ‘R’

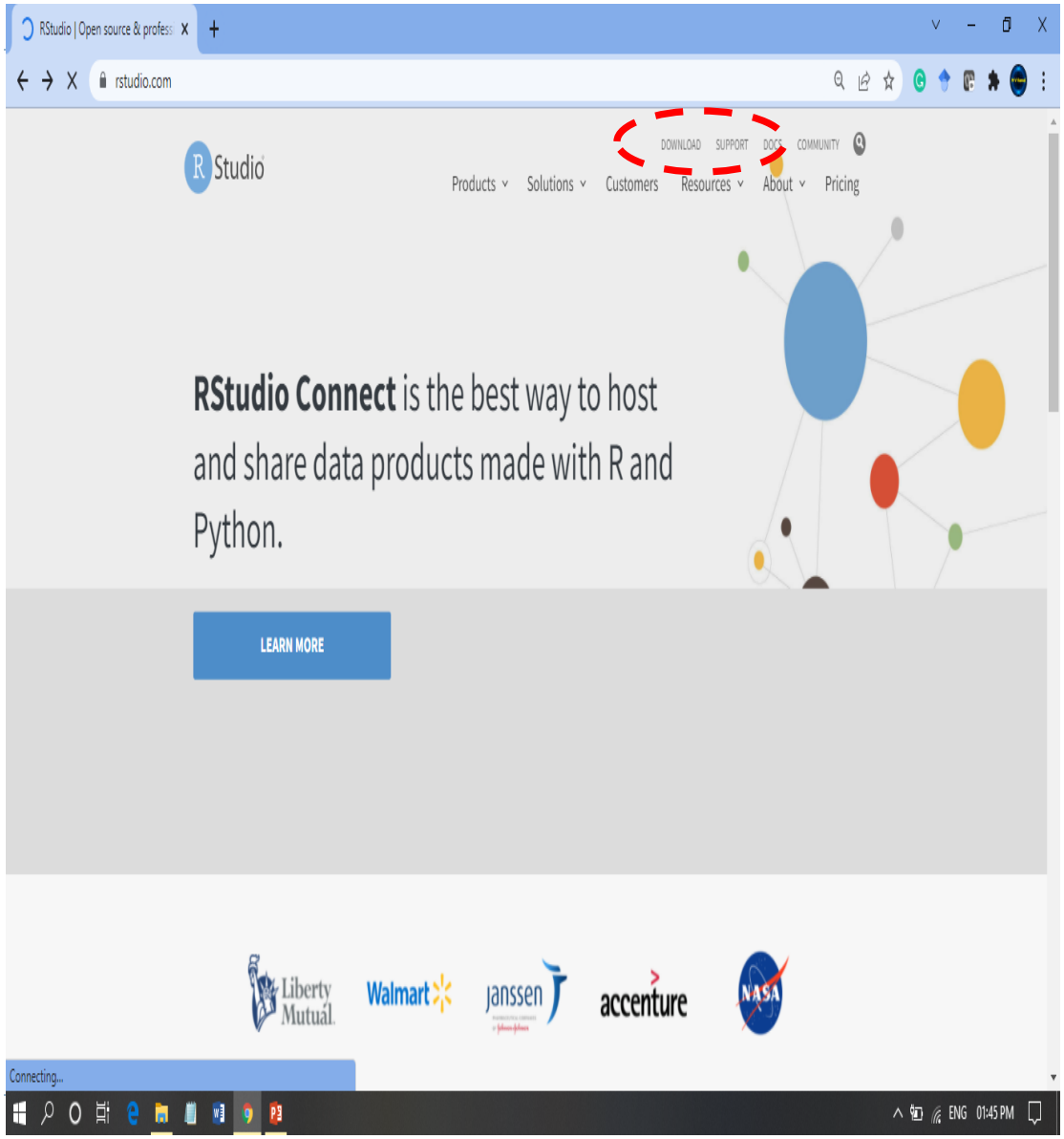
R and R Studio Download & Installation

- R-Studio

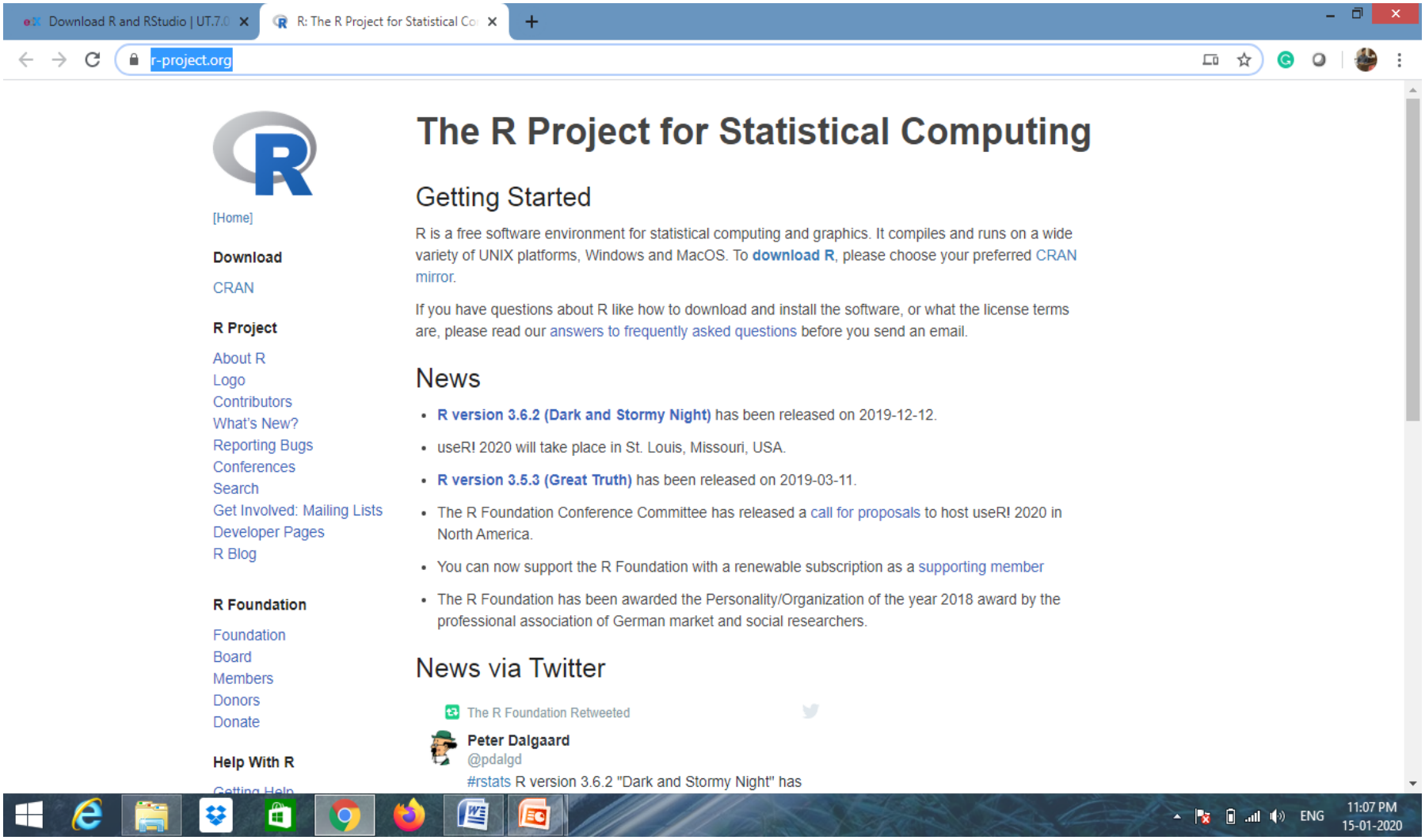
https://rstudio.com/

- R

https://www.r-project.org/

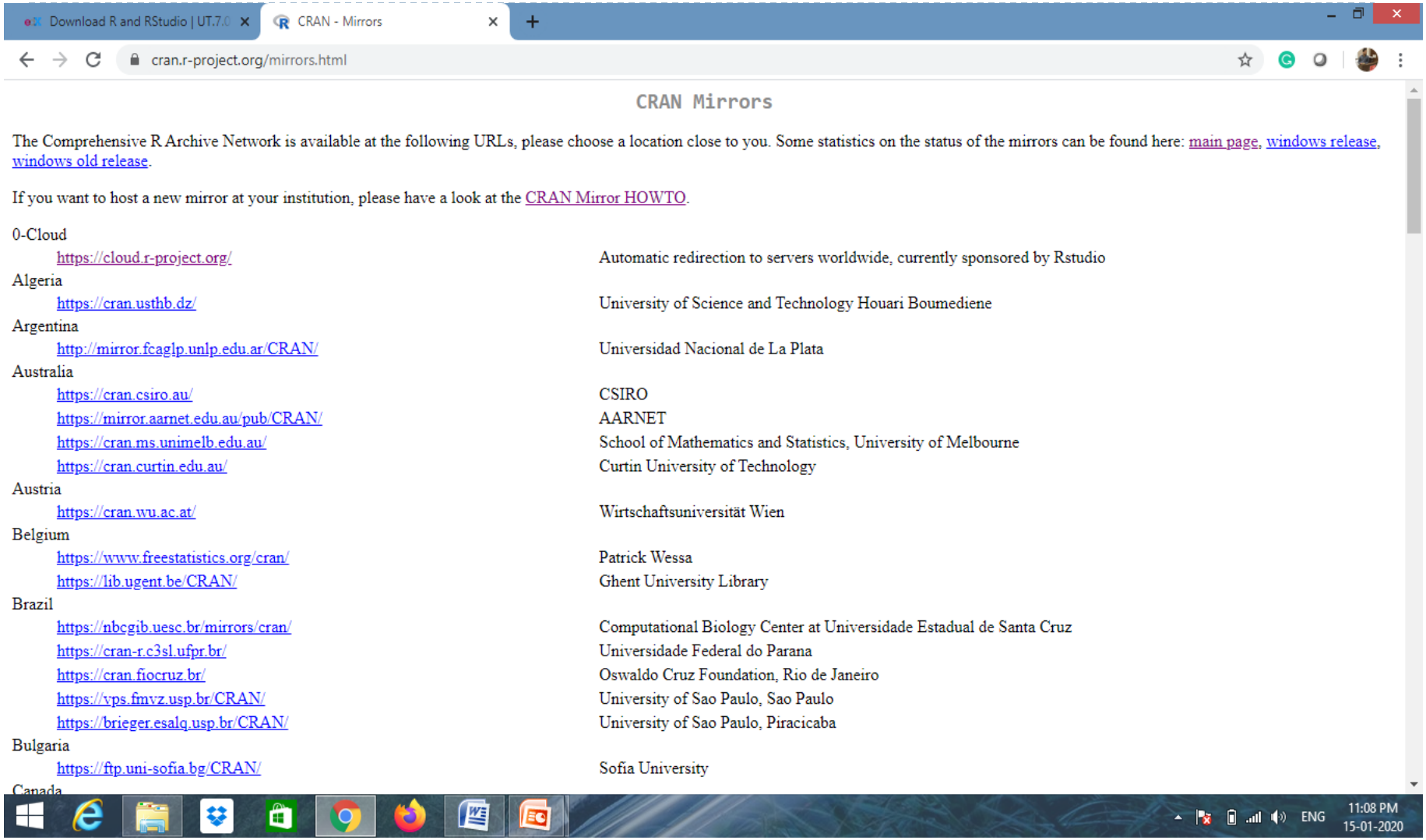


https://www.r-project.org/



- Click on CRAN (Comprehensive R Archive Network)

R's CRAN



CRAN Mirrors

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: main page, windows release, windows old release.

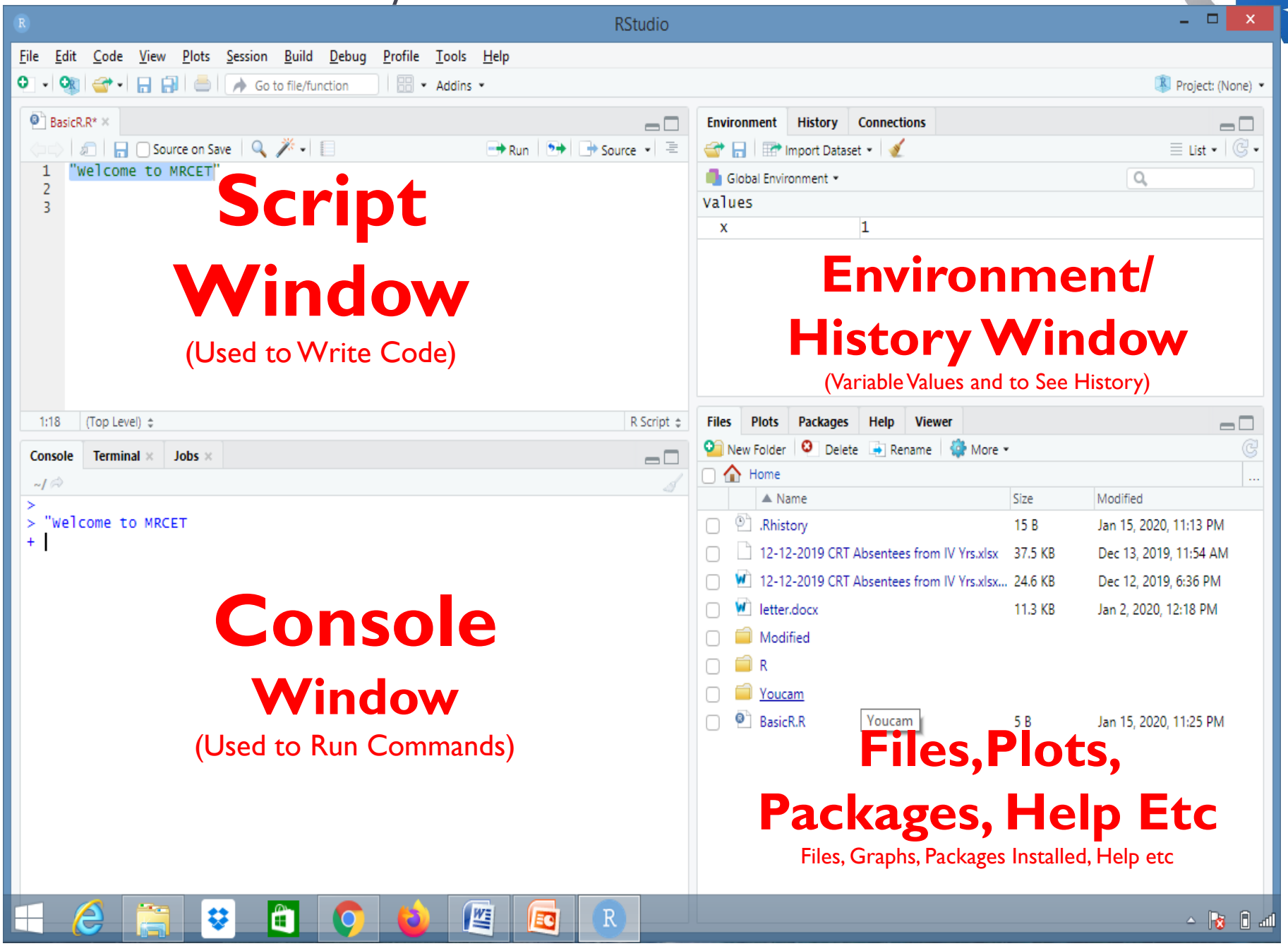
If you want to host a new mirror at your institution, please have a look at the CRAN Mirror HOWTO.

0-Cloud	
https://cloud.r-project.org/	Automatic redirection to servers worldwide, currently sponsored by Rstudio
Algeria	
https://cran.usthb.dz/	University of Science and Technology Houari Boumediene
Argentina	
http://mirror.fcaglp.unlp.edu.ar/CRAN/	Universidad Nacional de La Plata
Australia	
https://cran.csiro.au/	CSIRO
https://mirror.aarnet.edu.au/pub/CRAN/	AARNET
https://cran.ms.unimelb.edu.au/	School of Mathematics and Statistics, University of Melbourne
https://cran.curtin.edu.au/	Curtin University of Technology
Austria	
https://cran.wu.ac.at/	Wirtschaftsuniversität Wien
Belgium	
https://www.freestatistics.org/cran/	Patrick Wessa
https://lib.ugent.be/CRAN/	Ghent University Library
Brazil	
https://nbcgib.uesc.br/mirrors/cran/	Computational Biology Center at Universidade Estadual de Santa Cruz
https://cran-r.c3sl.ufpr.br/	Universidade Federal do Parana
https://cran.fiocruz.br/	Oswaldo Cruz Foundation, Rio de Janeiro
https://vps.fmvz.usp.br/CRAN/	University of Sao Paulo, Sao Paulo
https://brieger.esalq.usp.br/CRAN/	University of Sao Paulo, Piracicaba
Bulgaria	
https://ftp.uni-sofia.bg/CRAN/	Sofia University
Canada	

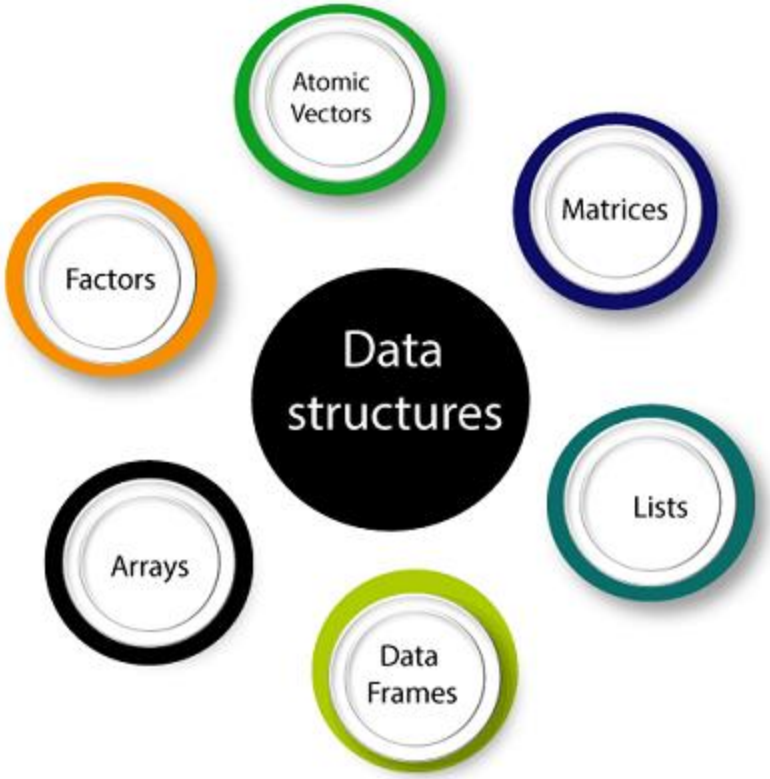
Installation of **R** and **RStudio**

- **To Install R:**
  - Open an internet browser and go to www.r-project.org.
  - Click the "download R" link in the middle of the page under "Getting Started."
  - Select a CRAN location (a mirror site) and click the corresponding link.
  - Click on the "Download R for Windows" link at the top of the page.
  - Click on the "install R for the first time" link at the top of the page.
  - Click "Download R for Windows" and save the executable file somewhere on your computer. Run the .exe file and follow the installation instructions.
  - Now that R is installed, you need to download and install RStudio.
- **To Install RStudio**
  - Go to www.rstudio.com and click on the "Download RStudio" button.
  - Click on "Download RStudio Desktop."
  - Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.

RStudio Editor / Interface



Data Types in **R**



Data Types in R..

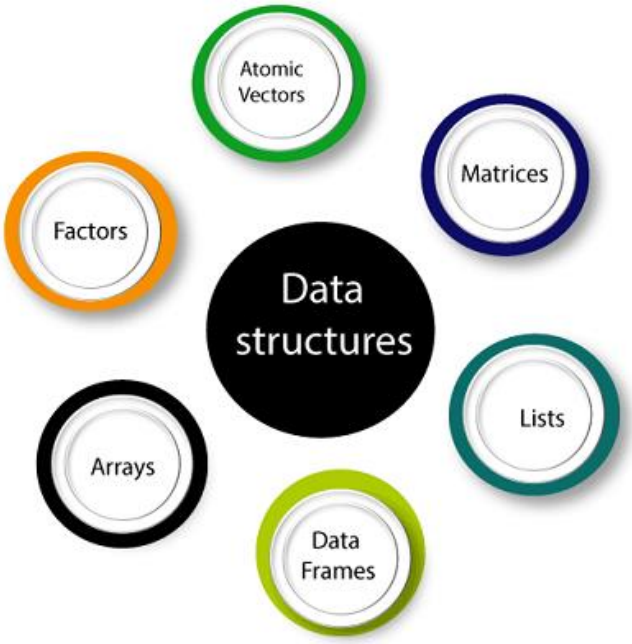
- You need to use various variables to store various information.
- Variables are nothing but reserved memory locations to store values.
- This means that, when you create a variable you reserve some space in memory.
- We can store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc.
- Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

Data Types in R..

- In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable.
- There are many types of **R-objects.**

The frequently used ones are –

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames



Data Types in R..

Data Type	Out Comes	Example
Logical	TRUE or FALSE	V <- TRUE print (class (V)) Output: [1] "Logical"
Numerical	1,2,3,912, 22.8	v <- 33.82 print(class(v)) Output: [1] "numeric"
Integer	2L , 3L, 4L	v <- 2L print(class(v)) Output: [1]: Integer

Data Types in R..

DATA TYPE	OUT COMES	Example
Complex	3 + 2i	v <- 2+5i print(class(v)) Output: [1] "complex"
Character	'a' , '"good", "TRUE", '23.4'	v<-"TRUE" print(class(v)) Output: [1] "character"

Data Types in R..

Data Type	Out Comes	Example
Raw	"Hello" It is stored as 48 65 6c 6c 6f	v<-charToRaw("Hello") print(class(v)) Output: [1] "raw"
		print(charToRaw('hello')) Output: [1] 68 65 6c 6c 6f

Vectors in R

Instructor: M V Kamal | Associate Professor | CSE Dept. | MRCET

Vectors in R

- When you want to create vector with more than one element, we have to use **c()** function which means to combine the elements into a vector.

Vectors in R

Example

```
# Creating a VECTOR IN R.
cars <- c('maruthi','Honda',"Hyundai")
print(cars)

# Get the class of the vector.
print(class(cars))
```

OUTPUT:

```
[1] "maruthi" "Honda" "Hyundai"
[1] "character"
```

List

- A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

LIST

```
# Create a LIST in R.
list1 <- list(c(5,6,8),21.3,sin)

# Print the list.
print(list1)
```

```
[[1]]
[1] 5 6 8
[[2]]
[1] 21.3
[[3]]
function (x) .Primitive("sin")
```

Matrices

- Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout.
- They contain elements of the same atomic types.
- Though we can create a matrix containing only characters or only logical values, they are not of much use.
- We use matrices containing numeric elements to be used in mathematical calculations.
- A Matrix is created using the **matrix()** function.

```
Syntax:
matrix(data, nrow, ncol, byrow, dimnames)
```

Matrices & Parameters

Following are parameters used in Matrices with Description:

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

Matrices- Example1

```
#Simple Matrix in R
M1 <- matrix(c(1:12), nrow = 4, byrow = TRUE)
print(M1)
```

Output:

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

Matrices- Example2

```
#Simple Matrix in R
M1 <- matrix(c(1:12), nrow = 4, byrow = FALSE)
print(M1)
```

Output:

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Matrices- Example3

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

m4 <- matrix(c(1:12), nrow = 4, byrow = TRUE,
             dimnames = list(rownames, colnames))

print(m4)
```

output

```
     [col1] [col2] [col3]
[row1] 1      2     3
[row2] 4      5     6
[row3] 7      8     9
[row4] 10     11    12
```

Guess the outputs for these statements..

```
print(P[1,3])
print(P[4,2])
print(P[2,])
print(P[,3])
```

Matrices – Example4

```
# Create a matrix in R.
M1 = matrix(
c('a','a','b','c','b','a'),
        nrow = 2, ncol = 3, byrow=TRUE)
print(M1)
```

```
     [,1] [,2] [,3]
[1,] "a"  "a"  "b"
[2,] "c"  "b"  "a"
```

Additional and Subtraction of Matrices

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(4, 2, -1, 6, 3, 5), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(2, 3, 0, 4, 9, 7), nrow = 2)
print(matrix2)

# Add the two matrices.
result <- matrix1 + matrix2
cat("Result of addition","\n")
print(result)


# Subtract the two matrices
result <- matrix1 - matrix2
cat("Result of subtraction","\n")
print(result)
```

Here **cat** is useful for producing output in user-defined functions

Multiplication and Division of Matrices

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(4, 2, -1, 6, 3, 5), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(2, 3, 0, 4, 9, 7), nrow = 2)
print(matrix2)

# Multiplication the matrices.
result <- matrix1 * matrix2
cat("Result of Multiplication","\n")
print(result)


# Division the matrices
result <- matrix1 / matrix2
cat("Result of Division","\n")
print(result)
```

Arrays in R

- Arrays are the R data objects which can store data in more than two dimensions.
- Example − If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns.
- Arrays can store only data type.
- An array is created using the **array()** function. It takes vectors as input and uses the values in the **dim** parameter to create an array.

Example for Array in R

```
# Create an array.
a <- array(c('CSE','ECE'),dim = c(3,3,2))
print(a)
```

```
, , 1
     [,1] [,2] [,3]
[1,] "CSE" "ECE" "CSE"
[2,] "ECE" "CSE" "ECE"
[3,] "CSE" "ECE" "CSE"
, , 2
[,1] [,2] [,3]
[1,] "ECE" "CSE" "ECE"
[2,] "CSE" "ECE" "CSE"
[3,] "ECE" "CSE" "ECE"
```

About Array and Matrices

- While matrices are confined to two dimensions, arrays can be of any number of dimensions.
- The array function takes a dim attribute which creates the required number of dimension.

Another Example of Array in R

```
# Create two vectors of different lengths.
v1 <- c(5,9,3)
v2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
result <- array(c(v1,v2),dim = c(3,3,2)) print(result)
```

```
, , 1
[,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15
, , 2
[,1] [,2] [,3]
[1,] 5 10 13
[2,] 9 11 14
[3,] 3 12 15
```

Factors in R

- Factors are the r-objects which are created using a vector.
- It stores the vector along with the distinct values of the elements in the vector as labels.
- The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector.
- They are useful in statistical modeling.
- Factors are created using the **factor()** function.
- The **nlevels** functions gives the count of levels.

Factors - Example

```
# Create a vector.
car_models <- c('benz','M800','M800','benz','rolls','M800','HONDA')
# Create a factor object.

factor_cars <- factor(car_models)

# Print the factor.
print(factor_cars)

print(nlevels(factor_cars)
```

```
[1] benz M800 M800 benz rolls M800  HONDA
Levels: benz  M800 rolls HONDA
[1] 4
```

Data Frames in R

- Data frames are tabular data objects.
- Unlike a matrix in data frame each column can contain different modes of data.
- The first column can be numeric while the second column can be character and third column can be logical.
- It is a list of vectors of equal length.
- Data Frames are created using the **data.frame()** function.

Data Frame Example

```
# Create a data Frame
MRCET_domains <- data.frame (
    branch = c("CSE", "ECE", "EEE", "IT","MECH"),
    gender = c("Male", "Female")
    courses= c("B.Tech", "M.Tech")
    duration = c(2, 4),
  )
print(MRCET_domains)
```

Assigning Operators in R

R Supports the following ways of Assigning the values to ta Variable

```
x <- value
x <<- value
value -> x
value ->> x
x = value
```

- Note: The operators <- and = assign into the environment in which they are evaluated.
- The operator <- can be used anywhere, whereas the operator = is only allowed at the top level.
- The operators <<- and ->> are normally only used in functions

Example..

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)
# Assignment using leftward operator.
var.2 <- c("mrcet","autonomous college")
# Assignment using rightward operator.
c(TRUE,1) -> var.3
```

Comment in R

Comments starts with a #

Single Line Comment

```
(1) # This is a comment
    "Hello World! "
(2) "Hello World!" # This is a comment
(3) # "Good morning!"
    "Good night!"
```

Multiple Comment

```
# This is a comment
# written in
# more than just one line
"Hello World!"
```

Creating **Variables** in **R**

Creating Variables in R

- ✓Variables are containers for storing data values.
- ✓R does not have a command for declaring a variable.
- ✓A variable is created the moment you first assign a value to it.
- ✓To assign a value to a variable, use the <- sign.
- ✓To output (or print) the variable value, just type the variable name.

Example:

```
a <- 8
age <- 22
college_name <- “mrcet”
```

Print / Output Variables

- Compared to many other progamming languages, you do not have to use a function to print/output variables in R. You can just type the name of the variable.

Example:

>> college_name <- “mrcet”

>> college_name

output:

[1] mrcet

print( ) in R

- However, R does have a print() function available if you want to use it.
- This might be useful if you are familiar with other programming languages, such as Python, which often use a print() function to output variables.

Example-1:

>> college_name <- "mrcet"

>> print(college_name)

output:

[1] mrcet

Example-2:

```
for (x in 1:10) {
  print(x)
}
```

Concatenate Elements

You can also concatenate, or join, two or more elements, by using the `paste()` function.
To combine both text and a variable, R uses comma (`,`).

Example-1:

```
name1 <- "mrcet"

paste("Our College Name is", name1)
```

Example-2:

```
FName <-  "M V"
LName <-  "kamal"

paste(FName, LName)
Output: M V kamal
```

Few more examples…

- >>a <- 5
  >>b <- 10
  >>a + b
- [1]15

- ----------
- >>a <- 5
  >>text <- "Some text"
  >>a + text

Multiple Variables

- R allows you to assign the same value to multiple variables in one line:

Example:

```
>>a <- b <- c <- d <-10
>>a
>>b
>>c
>>d
```

Output:

```
[1]10
[2]10
[3]10
[4]10
```

Rules for Variable Names..

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for R variables are:

- A variable name must start with a letter and can be a combination of letters, digits, period(.) and underscore(_).
- If it starts with period(.), it cannot be followed by a digit.
- A variable name cannot start with a number or underscore (_)
- Variable names are case-sensitive

  (Ex: name, Name and NAME are three different variables)
- Reserved words cannot be used as variables (TRUE, FALSE, NULL, if...)

Rules for usage of Variables in R

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

Examples (usage of variable names)

```
# Legal variable names:
myvar <- “MRCET"
my_var <- "MRCET"
myVar <- "MRCET"
MYVAR <- “MRCET"
myvar2 <- “MRCET"
.myvar <- MRCET
```

```
# Illegal variable names:
2myvar <- “MRCET"
my-var <- “MRCET"
my var <- “MRCET"
_my_var <- “MRCET"
my_v@ar <- “MRCET"
TRUE <- “MRCET"
```

Using CAT( )

```
a<- "Hello"
cat("The class of var_x is ",class(a),"\n")
b<- 34.5
cat(" Now the class of var_x is ",class(b),"\n")
c<- 27L
cat(" Next the class of var_x becomes ",class(c),"\n")
```

Finding Variables in **R**

- To know all the variables currently available in the workspace we use the **ls()** function. Also the ls() function can use patterns to match the variable names.

- Syntax:

```
>>print(ls())
```

- *The output will be a sample output and depending on what variables are declared in your environment.*

Finding Variables in R contd..

- The ls() function can use patterns to match the variable names.

```
# List the variables starting with the pattern "var".
print(ls(pattern = "var"))
```

```
print(ls(all.name = TRUE))
```

How to delete a variable in R

- Deleting Variables

```
rm(var1)
print(var1)
```

- All the variables can be deleted by using the **rm()** and **ls()** function together.

```
rm(list = ls())
print(ls())
```

R for Basic Math

- All common arithmetic operations and mathematical functionality are ready to use at the console prompt.
- You can perform addition, subtraction, multiplication, and division with the symbols +, -, *, and /, respectively.
- You can create exponents (also referred to as *powers* or *indices*) using ^, and you control the order of the calculations in a single command using parentheses, ().

Arithmetic

- R> 2+3
- [1] 5
- R> 14/6
- [1] 2.333333
- R> 14/6+5
- [1] 7.333333
- R> 14/(6+5)
- [1] 1.272727
- R> 3^2
- [1] 9
- R> 2^3
- [1] 8

R> sqrt(x=9)
[1] 3
R> sqrt(x=5.311)
[1] 2.304561

When using R, you'll often find that you need to translate a complicated arithmetic formula into code for evaluation

$$10^2 + \frac{3 \times 60}{8} - 3$$

```
R> 10^2+3*60/8-3
[1] 119.5
```

$$\frac{5^3 \times (6-2)}{61-3+4}$$

```
R> 5^3*(6-2)/(61-3+4)
[1] 8.064516
```

$$2^{2+1} - 4 + 64^{-2^{2.25-\frac{1}{4}}}$$

```
R> 2^(2+1)-4+64^((-2)^(2.25-1/4))
[1] 16777220
```

$$\left(\frac{0.44 \times (1-0.44)}{34}\right)^{\frac{1}{2}}$$

```
R> (0.44*(1-0.44)/34)^(1/2)
[1] 0.08512966
```

Logarithms and Exponentials

- R> log(x=243,base=3)
- [1] 5

- R> log(x=20.08554)
- [1] 3

- R> exp(x=3)
- [1] 20.08554

E-Notation

- R> 2342151012900
- [1] 2.342151e+12

- R> 0.0000002533
- [1] 2.533e-07

Operators in R

Instructor: M V Kamal | Associate Professor | CSE Dept. | MRCET

Operators in R

- Operators are used to perform operations on variables and values..
- R divides the operators in the following groups:
  - ✓ Arithmetic operators
  - ✓ Assignment operators
  - ✓ Comparison operators
  - ✓ Logical operators
  - ✓ Miscellaneous operators

Arithmetic Operators in R

Operator	Name	Example1	Example2
+	Addition	x + y	2+3
-	Subtraction	x - y	5-2
*	Multiplication	x * y	2*3
/	Division	x / y	12/4
^	Exponent	x ^ y	2^2
%%	Modulus (Remainder from division)	x %% y	24%%4
%/%	Integer Division	x%/%y	25%/%2

Assignment Operators in R

- Assignment operators are used to assign values to variables:

Example: a<-2

b<<-3

“mrcet”->college_name

“m800”->>carmodel

Comparison Operators in R

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

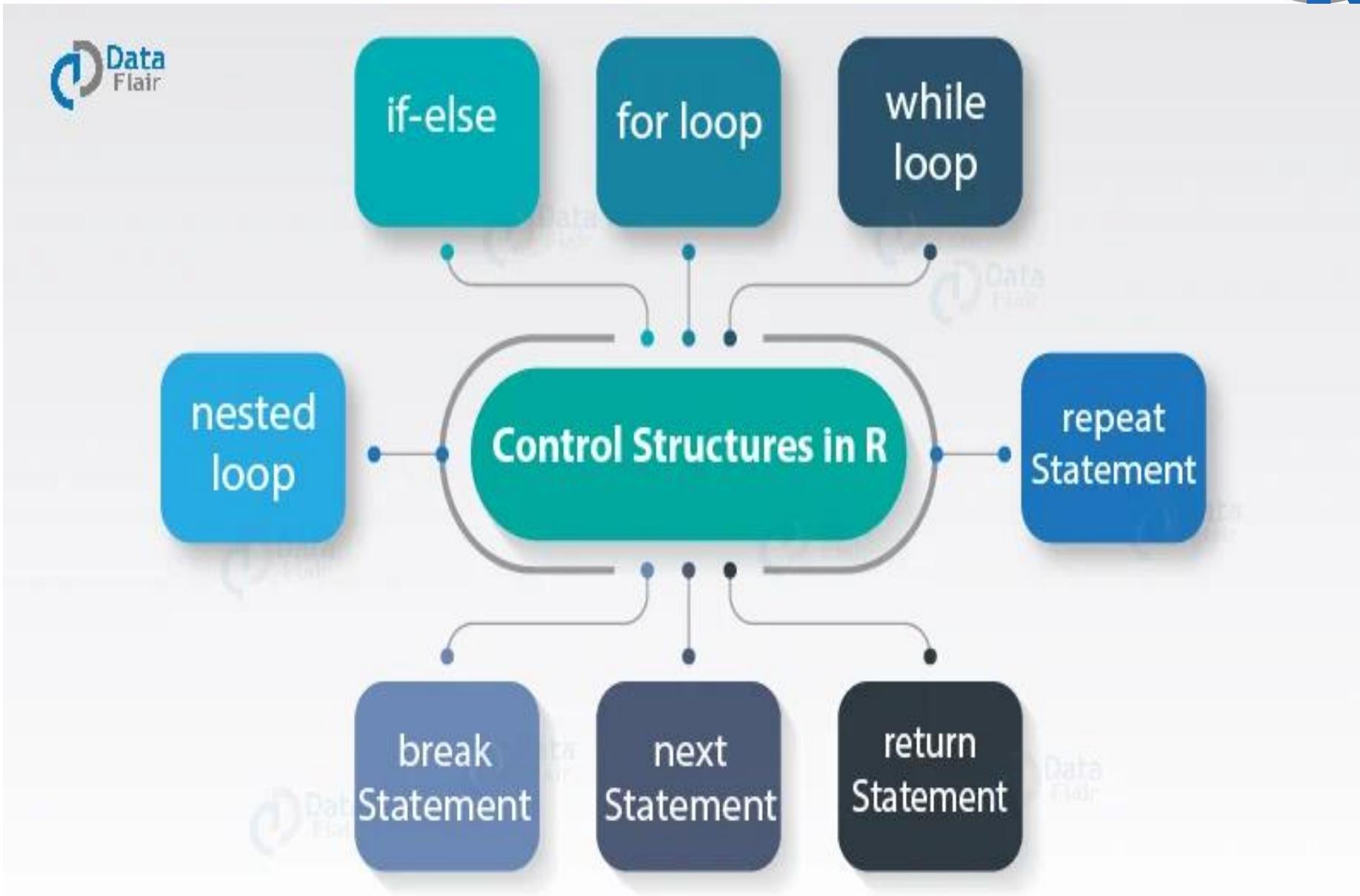
Logical Operators in R

Operator	Description
&	Element-wise Logical AND operator. It returns **TRUE** if both elements are TRUE
&&	Logical AND operator Returns **TRUE** if both statements are TRUE
|	Elementwise- Logical OR operator. It returns **TRUE** if one of the statement is TRUE
||	Logical OR operator. It returns **TRUE** if one of the statement is TRUE.
!	Logical NOT Returns **FALSE** if statement is TRUE

Miscellaneous Operators in R

- Miscellaneous operators are used to manipulate data:

Operator	Description	Example
:	Creates a series of numbers in a sequence	x <- 1:10
%in%	Find out if an element belongs to a vector	x %in% y
%*%	Matrix Multiplication	x <- Matrix1 %*% Matrix2


Data Flair
if-else
for loop
while loop
nested loop
Control Structures in R
repeat Statement
break Statement
next Statement
return Statement

LOOPS in R

- Loops can execute a block of code as long as a specified condition is reached. In general, statements are executed sequentially.
- Loops are handy because they save time, reduce errors, and they make code more readable.
- The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- R programming language provides the following kinds of loop to handle looping requirements.

LOOPS in R

- R Supports the following loops..
  - **while** loop
  - **for** loop
  - **repeat** loop

LOOPS in R

while loop

- Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

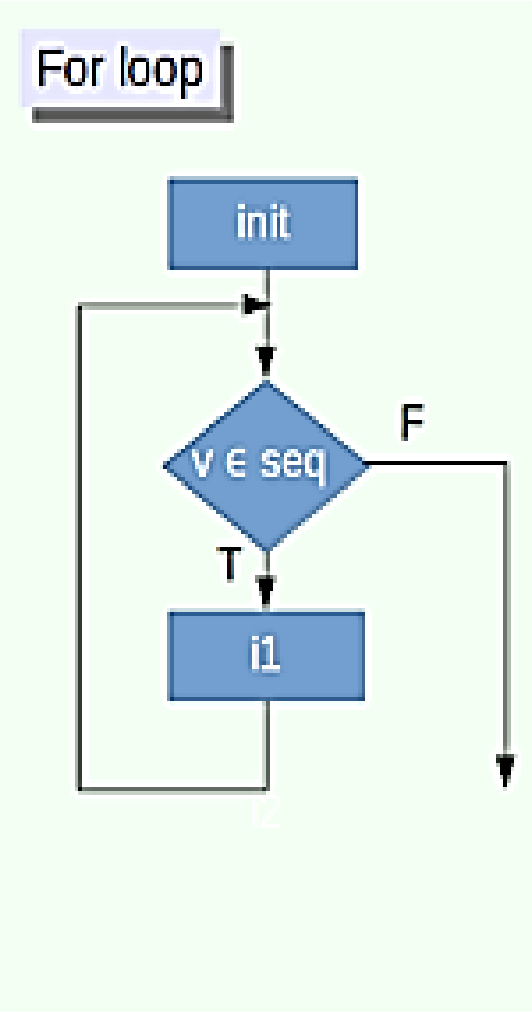
for loop

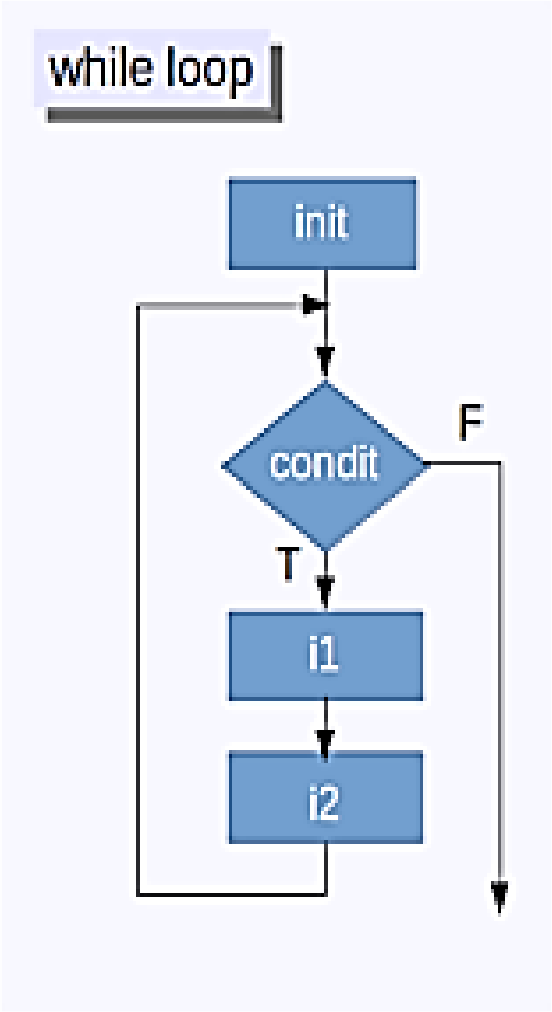
- Like a while statement, except that it tests the condition at the end of the loop body.

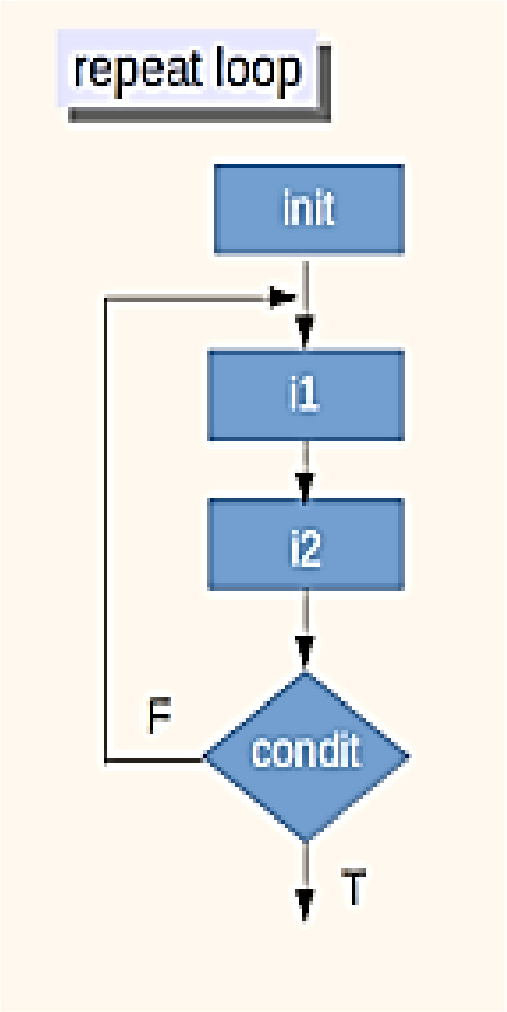
repeat loop

- The Repeat loop executes the same code again and again until a stop condition is met.
- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

for – while – repeat Loops







for Loop in R

- Syntax:

```
for (value in vector) {
statements
}
```

- Example-1

```
for (x in 1:10) {
print(x)
}
```

With the for loop we can execute a set of statements, once for each item in a vector, array, list, etc..

for Loop in R

- Example-2 (list)

```
car_model <- list("benz", "hyundai", "maruthi")
for (x in car_model) {
  print(x)
}
```

- Example-3 (vector)

```
dice <- c(1, 2, 3, 4, 5, 6)
for (x in dice) {
print(x)
}
```

With the for loop we can execute a set of statements, once for each item in a vector, array, list, etc..

for Loop in R

Break

With the break statement, we can stop the loop before it has looped through all the items:

Example-4:

```
car_model <- list (“benz", “hyundai", “maruthi")

for (x in car_model) {
  if (x == “maruthi") {
    break
  }
  print(x)
}
```

for Loop in R

Next

With the `next` statement, we can skip an iteration without terminating the loop:

Example-5:

```
car_model <- list (“benz", “hyundai", “maruthi")

for (x in car_model) {
  if (x == “hyundai") {
    next
  }
  print(x)
}
```

When the loop passes “hyundai", it will skip it and continue to loop.

for Loop in R

If .. Else Combined with a For Loop

Example-6:

```
days <- 1:7

for (x in days) {
  if (x == 7) {
    print(paste("Today is", x, "Its Weekend!"))
  } else {
    print(paste("Today is", x, "Not Weekend"))
  }
}
```

for Loop in R

Nested Loops

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

Example-7:

```
colors1 <- list("red", "blue", "while")

cars <- list("maruthi", "benz", "hyundai")
  for (x in colors1) {
    for (y in cars) {
      print(paste(x, y))
  }
}
```

while Loop in R

while Loop in R

With the while loop we can execute a set of statements as long as a condition is **TRUE**

Syntax:

```
while (test_expression) {
statement
}
```

Example-1

#Print i as long as i is less than 6:

```
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

while Loop in R

Break

With the `break` statement, we can stop the loop even if the while condition is TRUE:

Example-2

```
# using Break in while loop
# Exit the loop if i is equal to 4.

i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
  if (i == 4) {
    break
  }
}
```

```
Output
[1] 1
[1] 2
[1] 3
```

The loop will stop at 3 because we have chosen to finish the loop by using the `break` statement when `i` is equal to 4 (`i == 4`).

while Loop in R

Example-3

```
#using vector

v <- c("Hello","while loop")
cnt <- 2
while (cnt < 7) {
   print(v)
   cnt = cnt + 1
}
```

```
[1] "Hello" "while loop"
[1] "Hello" "while loop"
[1] "Hello" "while loop"
[1] "Hello" "while loop"
[1] "Hello" "while loop"
```

while Loop in R

Next

With the `next` statement, we can skip an iteration without terminating the loop:

Example-4

```
# using next in while loop
# Skip the value of 3
i <- 0
while (i < 6) {
  i <- i + 1
  if (i == 3) {
    next
  }
  print(i)
}
```

```
Output
[1] 1
[1] 2
[1] 3
```

The loop will stop at 3 because we have chosen to finish the loop by using the `break` statement when `i` is equal to 4 (`i == 4`).

while Loop in R

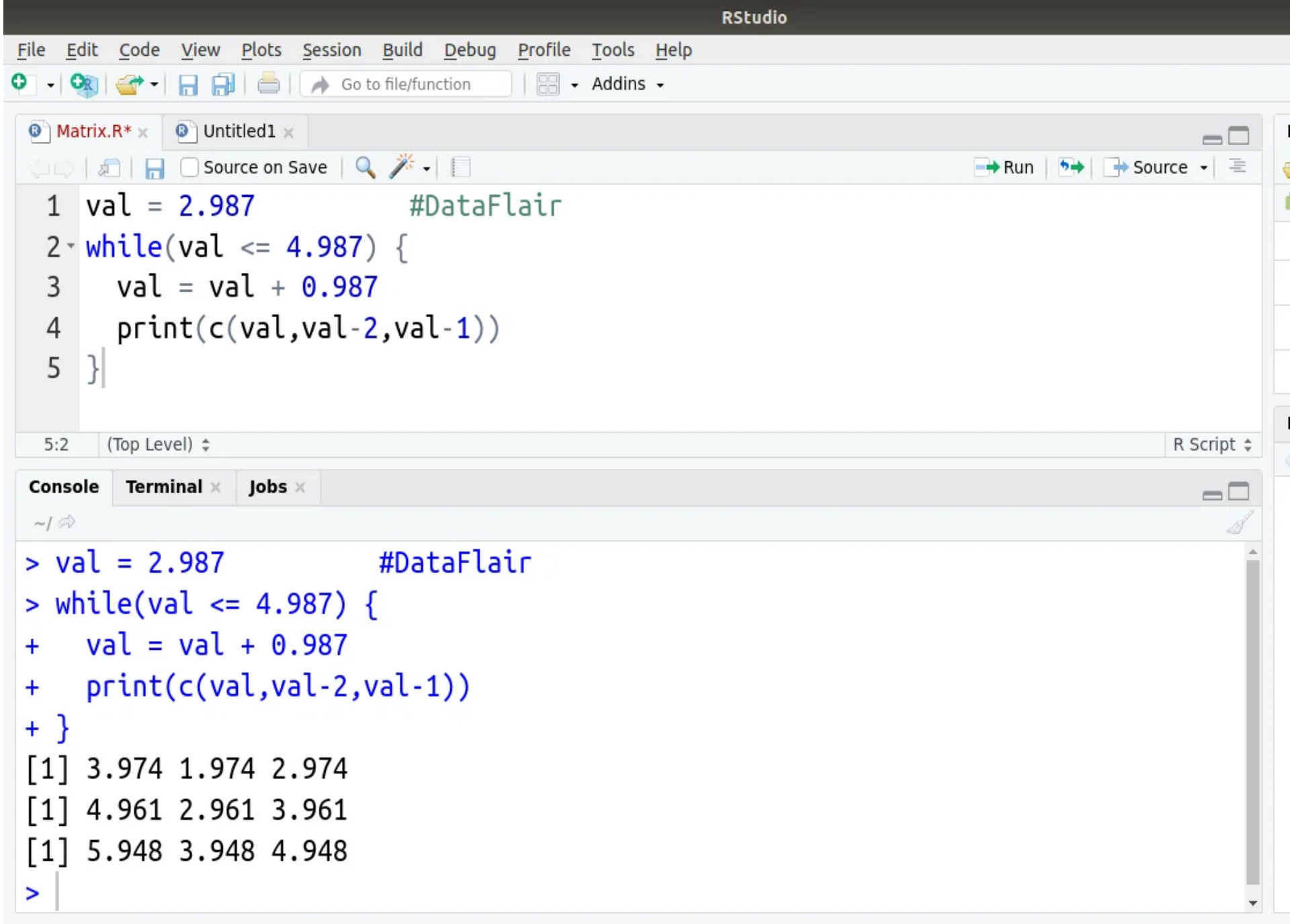
if.. Else Combined with while loop

Example-5

```
days <- 1
while (days <= 6) {
  if (days < 6) {
    print("Not a Saturday")
  } else {
    print("its Saturday and weekend!")
  }
  days <- days + 1
}
```

```
Output
[1] "Not a Saturday"
[1] "Not a Saturday"
[1] "Not a Saturday"
[1] "Not a Saturday"
[1] "Not a Saturday"
[1] "its Saturday and
weekend!"
```

If the loop passes the values ranging from 1 to 5, it prints "Not a Saturday". Whenever it passes the value 6, it prints "its Saturday and weekend!"


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Matrix.R*
Untitled1
Source on Save
Run
Source
1 val = 2.987 #DataFlair
2 while(val <= 4.987) {
3 val = val + 0.987
4 print(c(val,val-2,val-1))
5 }
5:2
(Top Level)
R Script
Console
Terminal
Jobs
~/
> val = 2.987 #DataFlair
> while(val <= 4.987) {
+ val = val + 0.987
+ print(c(val,val-2,val-1))
+ }
[1] 3.974 1.974 2.974
[1] 4.961 2.961 3.961
[1] 5.948 3.948 4.948
>

Repeat & Break Loop in R

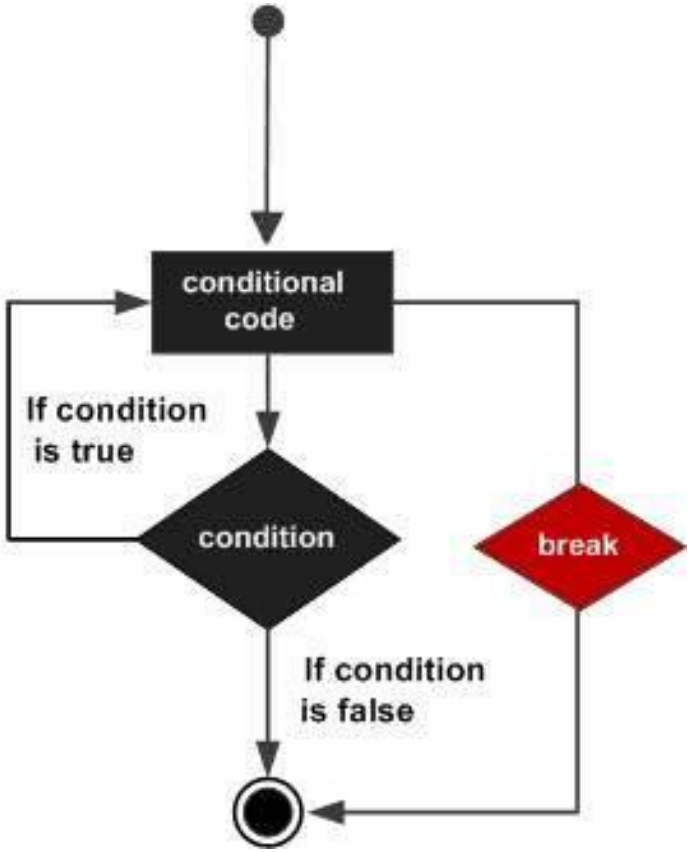
Instructor: M V Kamal | Associate Professor | CSE Dept. | MRCET

repeat Loop in R

The **Repeat loop** executes the same code again and again until a stop condition is met.

Syntax:

```
repeat {
    commands
    if(condition) {
        break
    }
}
```



repeat Loop in R

```
v <- c("Hello","loop")
cnt <- 2
repeat {
      print(v)
      cnt <- cnt+1
      if(cnt > 5) {
         break
    }
}
```

Output

```
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
```

next Statement in R

```
x = 1:4
for (i in x) {
if (i == 2) {
next
}
print(i)
}
```

return Statement in R

```
check <- function(x) {
  if (x > 0) {
    result <- "Positive"
  } else if (x < 0) {
   result <- "Negative"
  } else {
    result <- "Zero"
   }
  return(result)
  }
```

Strings in R

Instructor: M V Kamal | Associate Professor | CSE Dept. | MRCET

Strings in R

- Any value written within a pair of single quote (‘abc’) or double quotes (“abc”) in R is treated as a string.
- Internally R stores every string within double quotes, even when you create them with single quote.

Strings in R

- **Rules Applied in String Construction**
- The quotes at the beginning and end of a string should be both double quotes or both single quote. They can not be mixed.
- Double quotes can be inserted into a string starting and ending with single quote.
- Single quote can be inserted into a string starting and ending with double quotes.
- Double quotes can not be inserted into a string starting and ending with double quotes.
- Single quote can not be inserted into a string starting and ending with single quote.

Strings in R

- Assign a String to a Variable

```
str <- “Welcome to MRCET"
str # print the value of str
```

- Multiline Strings

```
str <- “Hello Welcome to MRCET. I am currently
pursuing II Yr B.Tech CSE and I am attending
Data Visualization subject which is advanced
subject in my curriculum."
str # print the value of str
cat(str) #check the output for this
```

Strings in R

String Length

#To find the number of characters in a string

```
str <- “Welcome to MRCET!"
nchar(str)
```

Strings in R

- Check a String
- str <- "Welcome to MRCET!"
- grepl("Welcome", str)
- grepl("Hello", str)
- grepl("WELCOME", str)
- grepl("to", str)
- grepl("M", str)

Strings in R

- Combine Two Strings

```
str1 <- “MRCET"
str2 <- “Autonomous college"
paste(str1, str2)
```

If, If-Else in R

If in R

```
a <- 150
b <- 200

if (b > a) {
  print("b is greater than a")
}
```

If using AND in R

```
a <- 200
b <- 33
c <- 500

if (a > b & c > a){
  print("Both conditions are true")
}
```

If using OR in R

```
a <- 500
b <- 300
c <- 400

if (a > b | a > c){
  print("At least one of the conditions is true")
}
```

If … Else in R

An "if statement" is written with the if keyword, and it is used to specify a block of code to be executed if a condition is TRUE.

Syntax:

```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true.
    } else {
   // statement(s) will execute if the boolean expression is false.
}
```

Example:

```
a <- 200
b <- 33

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print("a and b are equal")
} else {
  print("a is greater than b")
}
```

Else..If in R

The `else if` keyword is R's way of saying "if the previous conditions were not true, then try this condition"

Syntax:

```
if (condition1) {
      expr1
    } else if (condition2) {
      expr2
    } else if (condition3) {
    expr3 }
    else {
    expr4
}
```

Example

```
a <- 33
b <- 33

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print ("a and b are equal")
}
```

Nested If Statements in R

```
x <- 40
if (x > 10) {
  print("Above ten")
  if (x > 20) {
    print("and also above 20!")
  } else {
    print("but not above 20.")
  }
} else {
  print("below 10.")
}
```

Syllabus – Covered Topies

- Introduction to R- Features of R – Environment, How to run R, R Sessions and Functions, Basic Math, Variables, Data Types, Vectors, Conclusion, Advanced Data Structures, Data Frames, Lists, Matrices, Arrays, Classes, R Programming Structures, Control Statements, Loops, - Looping Over Nonvector Sets,- If-Else, Arithmetic and Boolean Operators and values, Default Values for Argument, Return Values, Functions are Objects, Recursion,
- Basic Functions - R help functions - R Data Structures. Vectors: Definition- Declaration - Generating - Indexing - Naming - Adding & Removing elements - Operations on Vectors - Recycling - Special Operators - Vectorized if- then else-Vector Equality – Functions for vectors - Missing values - NULL values - Filtering & Subsetting.

Repetition

- R> rep(x=1,times=4)

  [1] 1 1 1 1
- R> rep(x=c(3,62,8.3),times=3)

  [1] 3.0 62.0 8.3 3.0 62.0 8.3 3.0 62.0 8.3
- R> rep(x=c(3,62,8.3),each=2)

  [1] 3.0 3.0 62.0 62.0 8.3 8.3
- R> rep(x=c(3,62,8.3),times=3,each=2)

  [1] 3.0 3.0 62.0 62.0 8.3 8.3 3.0 3.0 62.0 62.0 8.3 8.3
  3.0 3.0 62.0 62.0 8.3 8.3

Sorting

Execute & Check Result…

- R> sort(x=c(2.5,-1,-10,3.44), decreasing=FALSE)
- R> sort(x=c(2.5,-1,-10,3.44), decreasing=TRUE)
- R> A← seq(from=4.3,to=5.5,length.out=8)
- R> A
- R> bar <- sort(x=A,decreasing=TRUE)
- R> bar
- R> sort(x=c(foo,bar),decreasing=FALSE)

Finding a Vector Length with length

- R> length(x=c(3,2,8,1))
- [1] 4
- R> length(x=5:13)
- [1] 9

Function in R

Instructor: M V Kamal | Associate Professor | CSE Dept. | MRCET

Functions in R Programming

- A function in R is a set of statements or commands that organzied together to perform a specific task.
- R provides a wide range of functions for obtaining summary statistics.
- R Supports built-in and User-Defined Functions
- In R, a function which is treated as object and able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.
- **KEYWORD:**

  function(arg1,agr2,arg3.....)

- **Syntax:**

```
Function_name ← function (arg1, arg2, agr3..) {
                    body_of_function
                }
```

Function Example:

```
my_function <- function() {
  # create a function with the name my_function
  print("Hello MRCET!")
}
```

Functions in R Programming

- Components in "Function" are
  - Function_Name
  - Arguments
  - Function_body
  - Return_Value

Built-In Function in R

- Many built in functions are available in R-Programming.
  - Example

```
print(sum(20:30))
print(mean(5:30))
print(sqrt(2))
print(abs(-5))
print(seq(1:10))
```

User-Defined Function in R

- As like "C", User-Defined function is nothing but we can create our own function in R.
- Example

```
#crate a function to print squares of each number in sequence
A ← function(a) {
   for(i in 1 : a) {
    b← i ^ 2
    print(b)
   }
}
# argument calling to function
A(6)
```

Function in R

Calling a Function with Argument Values (by position and by name)

```
# Create a function with arguments
   A←function(a,b,c) {
   result ← a * b + c
   print(result)
   }
# Call the function by position of arguments
         A(2,3,5)
# Call the function by names of the arguments
         A(a = 3, b = 2, c = 5)
```

Function in R

Calling a Function with Default Argument

Create a function with arguments.

Example

```
A← function(a = 3, b = 6) {
   result ← a * b
   print(result)
}
# Call the function without giving any argument.
A()
# Call the function with giving new values of the argument.
   A(4 , 8)
```

Function in R

- **Lazy Evaluation of Function in "R"**
- Arguments to functions are evaluated lazily, which means so they are evaluated only when needed by the function body

Example

- # Create a function with arguments.

```
A← function(a, b) {
        print(a^2)
        print(a)
        print(b)
   }
# Evaluate the function without supplying one of the arguments.
         A(4)
```

Descriptive Statistics in R

Instructor: M V Kamal | Associate Professor | CSE Dept. | MRCET

Descriptive Statistics in R

- All the data which is gathered for any analysis is useful when it is properly represented so that it is easily understandable by everyone and helps in proper decision making.
- After we carry out the data analysis, we delineate its summary so as to understand it in a much better way.
- This is known as **summarizing the data**.

Descriptive Statistics in R

We can summarize our data in R as follows:

- **Descriptive/Summary Statistics –** With the help of descriptive statistics, we can represent the information about our datasets. They also form the platform for carrying out complex computations as well as analysis. Therefore, even though they are developed with simple methods, they play a crucial role in the process of analysis.
- **Tabulation –** Representing the data analyzed in tabular form for easy understanding.
- **Graphical –** It is a way to represent data graphically.

Descriptive Statistics in R

Statistics is the science of analyzing, reviewing and conclude data.

Some basic statistical methods:

- ✓ Mean, Median and Mode Method
- ✓ Finding Minimum, Maximum & Range Values
- ✓ Percentiles Calculation
- ✓ Variance and Standard Deviation (SD)
- ✓ Finding Covariance and Correlation
- ✓ Interquartile Range (IQR)
- ✓ Probability Distributions Method

Descriptive Statistics in R

Some R functions for computing descriptive statistics:

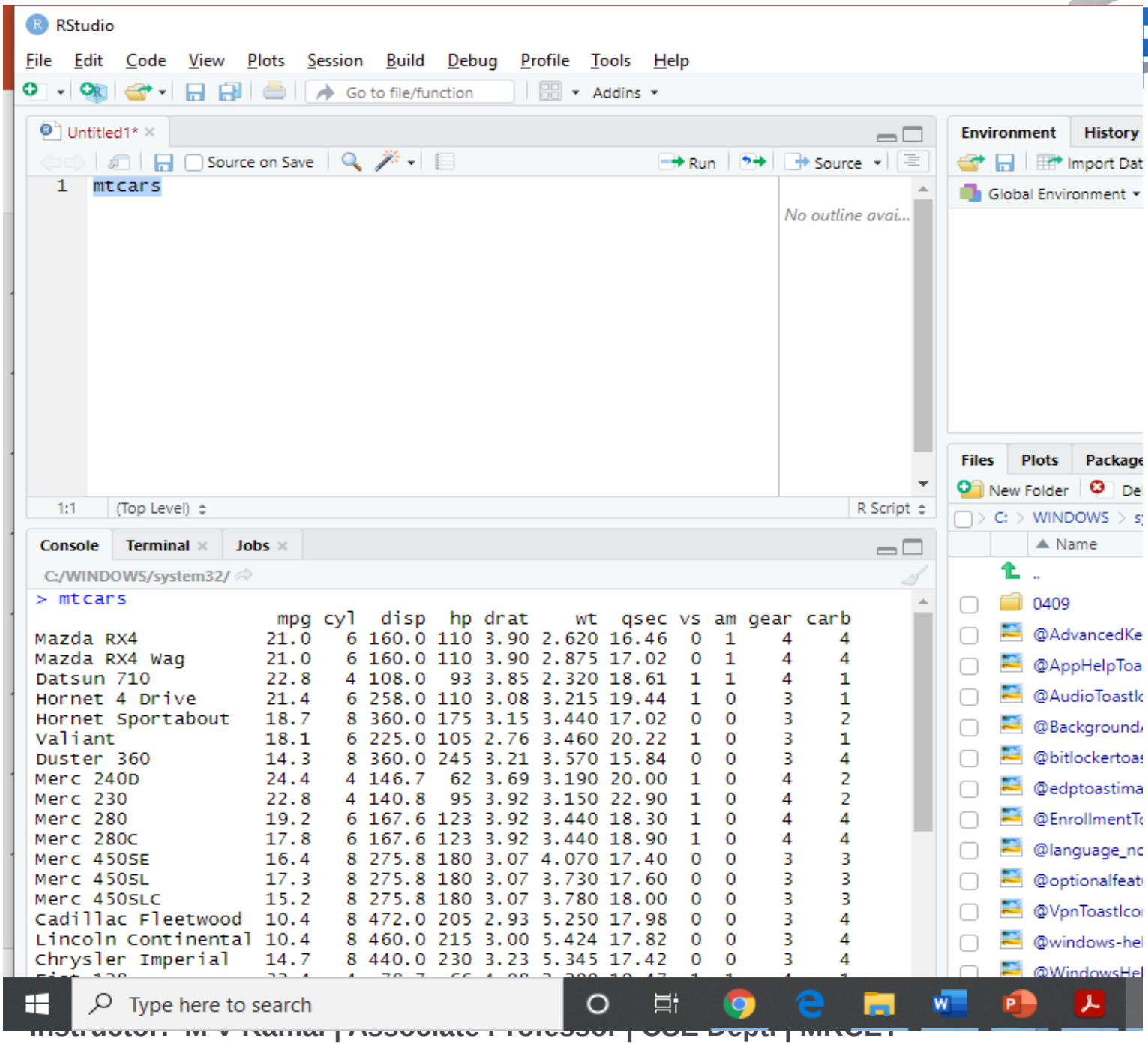
Description		R function
Mean	→	mean()
Standard deviation	→	sd()
Variance	→	var()
Minimum	→	min()
Maximum	→	max()
Median	→	median()
Range of values (minimum and maximum)	→	range()
Sample quantiles	→	quantile()
Generic function	→	summary()
Interquartile range	→	IQR()

Let's..do...

- Consider the Popular Built-in Dataset of R
- "mtcars" (Motor Trend Car Road Tests)

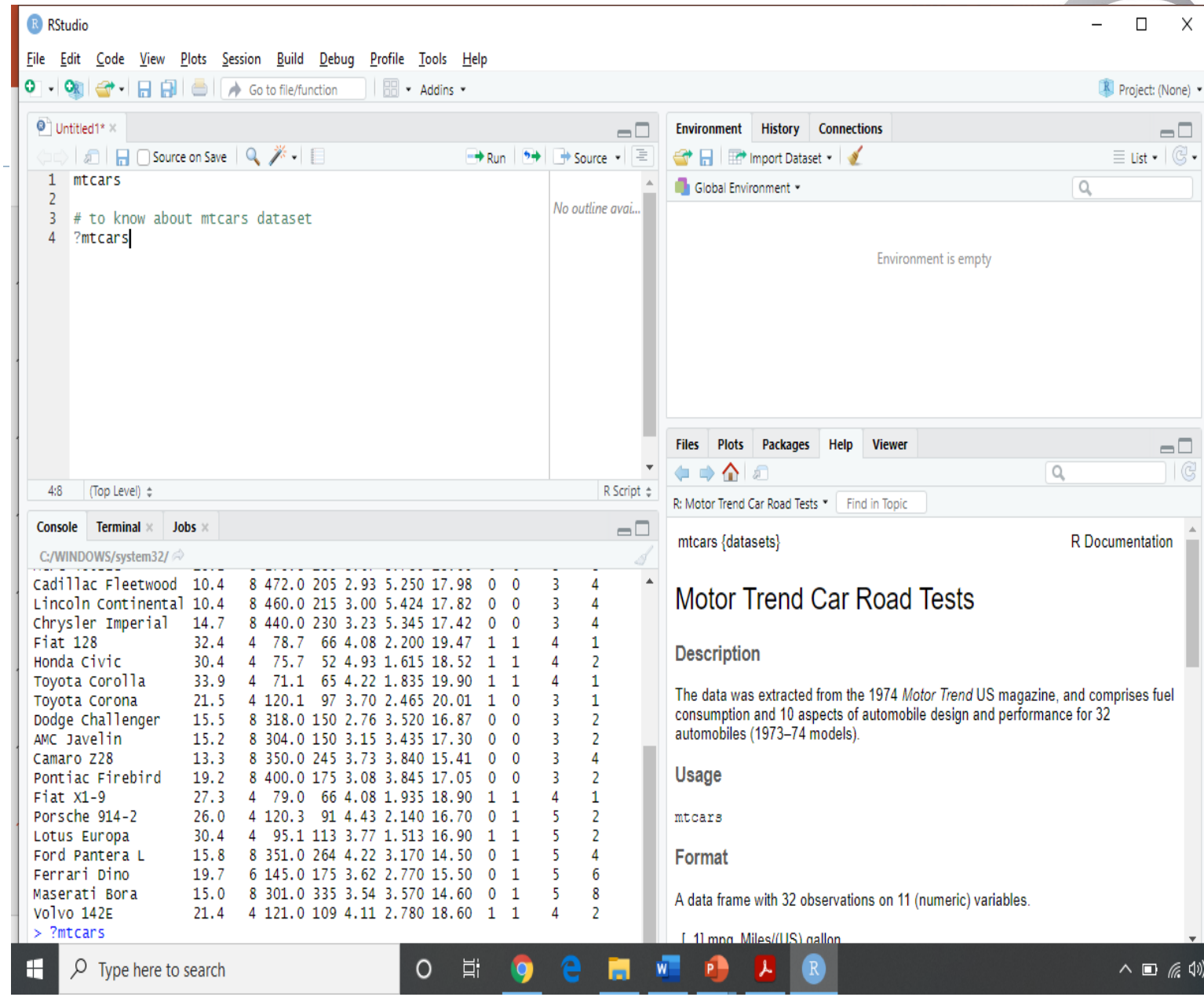
```
# Print the mtcars data set
>>mtcars
```

mtcars

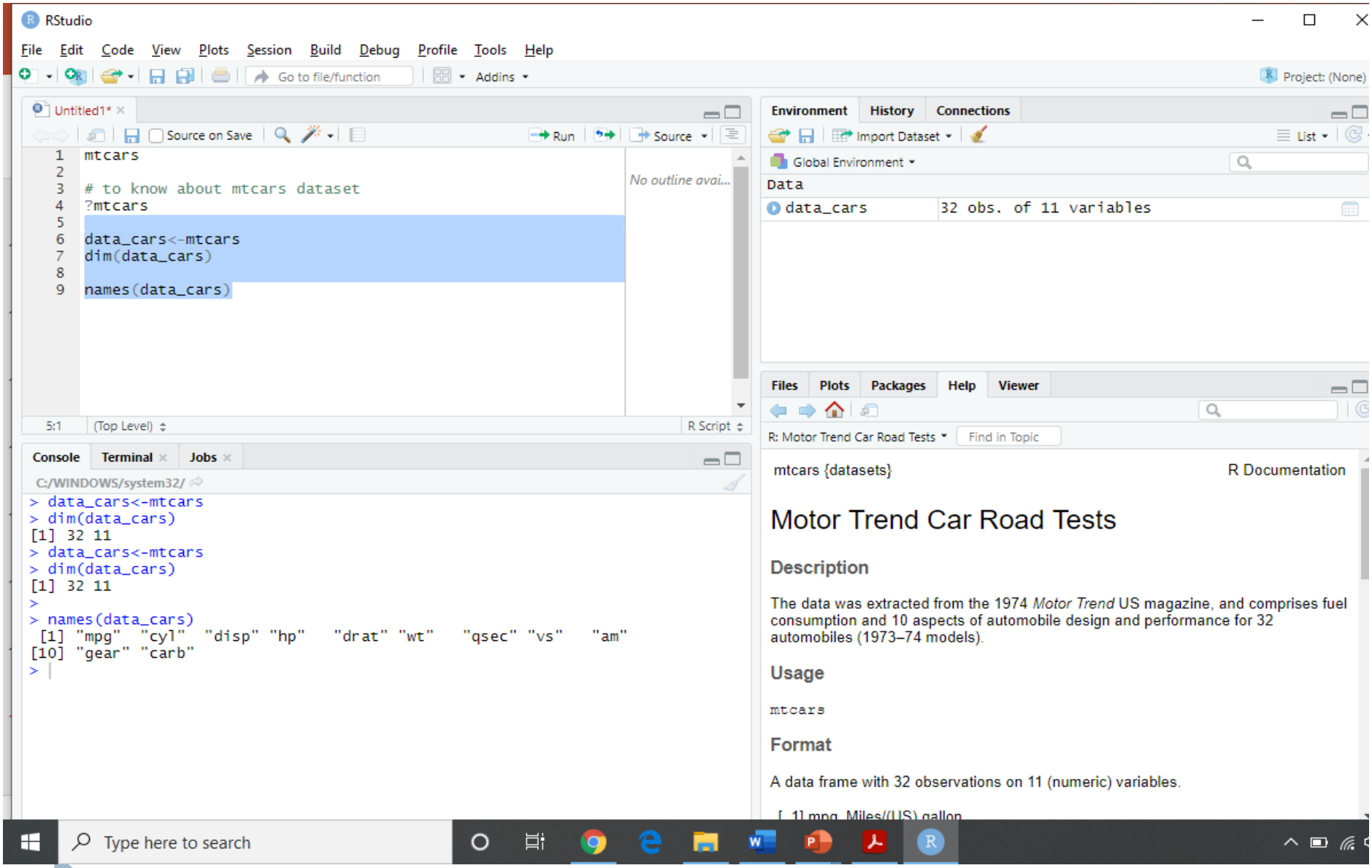


	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4

?mtcars


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Untitled1*
Source on Save
Run
Source
1 mtcars
2
3 # to know about mtcars dataset
4 ?mtcars
No outline avai...
4:8
(Top Level)
R Script
Console
Terminal
Jobs
C:/WINDOWS/system32/
Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4
Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82 0 0 3 4
Chrysler Imperial 14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4
Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1
Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2
Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1
Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1
Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0 0 3 2
AMC Javelin 15.2 8 304.0 150 3.15 3.435 17.30 0 0 3 2
Camaro Z28 13.3 8 350.0 245 3.73 3.840 15.41 0 0 3 4
Pontiac Firebird 19.2 8 400.0 175 3.08 3.845 17.05 0 0 3 2
Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1
Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2
Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2
Ford Pantera L 15.8 8 351.0 264 4.22 3.170 14.50 0 1 5 4
Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6
Maserati Bora 15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8
Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2
> ?mtcars
Environment
History
Connections
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
R: Motor Trend Car Road Tests
Find in Topic
mtcars {datasets}
R Documentation
Motor Trend Car Road Tests
Description
The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
Usage
mtcars
Format
A data frame with 32 observations on 11 (numeric) variables.
[, 1] mpg Miles/(US) gallon
Type here to search

Use the `dim()` function to find the dimensions of the dataset.
Use `names()` function to view the names of the variables:



Some Statistical Commands with Examples…

To see the structure of dataset

- str(data_set)
- Example: `str(mtcars)` or `str(iris)`

To see the minimum values of the data element

- min(data_set$attribute_name)
- Example: `min(mtcars$disp)`

To see the maximum values of the data element

- `max(mtcars$disp)`

To see the Range values of the data element

- `range(mtcars$mpg)`

Few Experiments in Descriptive Statistics using R..

```
#Range
➢ range1<-range(mtcars$mpg)
➢ range1
  [1] 10.4 33.9
➢ range1[1]
  [1] 10.4

#find the Range
max(mtcars$mpg) - min(mtcars$mpg)

#Using Function to find Range
range2 <- function(x) {
  range <- max(x) - min(x)
  return(range)
  }
> > range2(mtcars + $mpg)
```

```
#To display the only row names in a dataset
>rownames(mtcars)
#To display the only column names in a dataset
>colnames(mtcars)

#To display...
>which.max(mtcars$mpg)
#To display...
>Which.min(mtcars$cyl)
```

```
#To Find MEAN for an attribute in a dataset
>mean(mtcars$mpg)
#if there is at least one missing value in your dat
Use the following way to compute the mean with the
> mean(mtcars$mpg,na.rm = TRUE)
#To truncate the mean value for an attribute in a d
>mean(mtcars$mpg,trim = 0.10)
>mean(mtcars$mpg,rrim =0.10)
```

Few Experiments in Descriptive Statistics using R..

```
#To Find MEDIAN for an attribute in a dataset
>median(mtcars$mpg)
or
>median(mtcars$mpg, na.rm = FALSE)
Or
>quantile(mtcars$mpg,0.5)
#First and third quartile
```

???

```
# How to calculate IQR
-----------------------------------------
>rownames(mtcars)[which.max(mtcars$mpg)]
>sort(mtcars$cyl)
>table(mtcars$cyl)
```

How to calculate MODE

```
> unique(mtcars$cyl)

#Alternative Method to find the MODE for a data_element

>get_mode <- function(a) {
   unique1 <- unique(a)
   unique1[which.max(tabulate(match(a,unique1)))]
 }


>a<-mtcars$cyl
>mode_result<-get_mode(a)
> print(mode_result)
```

What is the solution for this…

(1)

```
>head(mtcars$)
```

(2)

```
>names(sort(table(mtcars$cyl)))
```

(3)

```
>data <- data.frame(x1 = 1:10,
                    x2 = letters[1:10],
                    x3 = "x")
```

Percentiles Calculation in R

Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than.

```
> quantile(mtcars$cyl,c(0.5))
```

If you run the quantile() function without specifying the c() parameter, you will get the percentiles of 0, 25, 50, 75 and 100:

```
> quantile(mtcars$cyl)
```

```
Output:
0% 25% 50% 75% 100%
4    4    6    8     8
```

Quartiles

- Quartiles are data divided into four parts, when sorted in an ascending order:
- ✓ The value of the first quartile cuts off the first 25% of the data
- ✓ The value of the second quartile cuts off the first 50% of the data
- ✓ The value of the third quartile cuts off the first 75% of the data
- ✓ The value of the fourth quartile cuts off the 100% of the data

Standard Deviation and Variance

The standard deviation is computed with the sd() function:

```
#To Calculate the Standard Deviation
>sd(mtcars$mpg)
```

The variance is computed with the var() function:

```
#To Calculate the Variance
>var(mtcars$mpg)
```

Few Built-In Function in R

By: M V Kamal | Associate Professor | CSE Dept. | MRCET

About Function in **R**

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- To create a function, use the function() keyword:

Function Components

The different parts of a function are −

- **Function Name** − This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments** − An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body** − The function body contains a collection of statements that defines what the function does.
- **Return Value** − The return value of a function is the last expression in the function body to be evaluated.

Syntax:

- function_name <- function(arg_1, arg_2, ...) {

  Function body

  }

Example:

```
my_function <- function() {
# create a function with the name my_function
  print("Welcome to MRCET!")
}
my_function
# To call the function and to print the result (Function calling)
```

User-Defined Function

- # Create a function to print squares of numbers in sequence.
- ```
  new.function <- function(a) {
      for(i in 1:a) {
      b <- i^2
      print(b)
    }
  }
  ```
  ```

Number of **Arguments**

- By default, a function must be called with the correct number of arguments…

Example

- my_function <- function(fname, lname) {
  paste(fname, lname)
  }

  my_function("Emerging","Technologies")

- **Note:** If you try to call the function with 1 or 3 arguments, you will get an error:
- *Check out the above statement is correct or not

Default Parameter Value

```
my_function <- function(branch = "CSE") {
  paste("I am from", branch)
}
```

- my_function("Cyber_Security")
- my_function("Data Science")
- my_function() # will get the default value, which is CSE
- my_function("IoT")

Return Values

Example for Return Values

```
my_function <- function(x) {
  return (3 * x)
}

print(my_function(2))
print(my_function(3))
print(my_function(4))
```

Built-in Functions in R

- `min()`, `max()`, `mean()`, `median()` – return the minimum / maximum / mean / median value of a numeric vector, correspondingly
- `sum()` – returns the sum of a numeric vector
- `range()` – returns the minimum and maximum values of a numeric vector
- `abs()` – returns the absulute value of a number
- `str()` – shows the structure of an R object
- `print()` – displays an R object on the console
- `ncol()` – returns the number of columns of a matrix or a dataframe
- `length()` – returns the number of items in an R object (a vector, a list, etc.)
- `nchar()` – returns the number of characters in a character object
- `sort()` – sorts a vector in ascending or descending (`decreasing=TRUE`) order
- `exists()` – returns `TRUE` or `FALSE` depending on whether or not a variable is defined in the R environment

Built-in Functions in R (Contd)

Example

- vector <- c(3, 5, 2, 3, 1, 4)
- print(min(vector))
- print(mean(vector)) or print(mean(25:82))
- print(median(vector))
- print(sum(vector)) or print(sum(41:68))
- print(range(vector))
- print(str(vector))
- print(length(vector))
- print(sort(vector, decreasing=TRUE)) print(exists('vector'))
- print(seq(32,44))

Functions for obtaining summary statistics.

Functions in R Programming

- R provides a wide range of functions for obtaining summary statistics.
- **apply(), lapply(), sapply(), tapply() Function**

- **apply()** takes Data frame or matrix as an input and gives output in vector, list or array.
- apply() Function is primarily used to avoid explicit uses of loop constructs. It is the most basic of all collections can be used over a matrice.

- Syntax: **apply(X, MARGIN, FUN)**
- **Here** MARGIN is here the manipulation is performed on rows or column or both..

Where…

→**X**: an array or matrix

→**MARGIN**: take a value or range between 1 and 2 to define where to apply the function:

→MARGIN=1: The manipulation is performed on rows

→MARGIN=2`: the manipulation is performed on columns

→MARGIN=c(1,2)` the manipulation is performed on rows and columns

→**FUN:** tells which function to apply. Built functions like mean, median, sum, min, max and even user-defined functions can be applied>

- apply()
- sapply()

The above two will be called as "matrix function" in R

apply( ) function

- apply ( ) function

Example

matrixM ← matrix(C←(1:10),nrow=5, ncol=6)

print(matrixM)

a ← apply(matrixM, 2, sum)

print(a)

```
[1] 15 15 15 15 15 15
> matrixM <- matrix(C<-(1:10),nrow=5, ncol=6)
> matrixM
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    6    1    6    1    6
[2,]    2    7    2    7    2    7
[3,]    3    8    3    8    3    8
[4,]    4    9    4    9    4    9
[5,]    5   10    5   10    5   10
> a<-apply(Matri,3,sum)
Error in apply(Matri, 3, sum) : object 'Matri' not found
> a<-apply(matrixM,3,sum)
Error in apply(matrixM, 3, sum) : 'MARGIN' does not match dim(X)
> a<-apply(matrixM,2,sum)
> print(a)
[1] 15 40 15 40 15 40
> a<-apply(matrixM,1,sum)
> a
[1] 21 27 33 39 45
> |
```

lapply() function

- **lapply()** function is useful for performing operations on list objects and returns a list object of same length of original set.

- lappy() returns a list of the similar length as input list object, each element of which is the result of applying FUN to the corresponding element of list.

- lapply() takes list, vector or data frame as input and gives output in list.

lapply() function

- lapply(X, FUN)
- Arguments:

  Here 'X' is a vector or an object

  and 'FUN' is a Function applied to each element of X

*

Note: lapply() function does not need MARGIN.

lapply() function

```
>car_models ←c("MARUTHI","HONDA","HYUNDAI","BENZ")
>car_models_lower← lapply(car_models, tolower)
>car_models_lower
[[1]]
[1] "maruthi"
[[2]]
[1] "honda"
[[3]]
[1] "hyundai"
[[4]]
[1] "benz"
>str(car_models_lower)
    List of 4
    $ : chr "maruthi"
    $ : chr "honda"
    $ : chr "hyundai"
    $ : chr "benz"
```

```
> car_models <- c("MARUTHI","HONDA","HYUNDAI","BENZ")
> car_models_lower<-lapply(car_models, tolower)
> car_models_lower
[[1]]
[1] "maruthi"

[[2]]
[1] "honda"

[[3]]
[1] "hyundai"

[[4]]
[1] "benz"

> str(car_models_lower)
List of 4
 $ : chr "maruthi"
 $ : chr "honda"
 $ : chr "hyundai"
 $ : chr "benz"
>
```

unlist()

- We can use unlist() to convert the list into a vector.

Example:

```
> str(car_models_lower)
List of 4
 $ : chr "maruthi"
 $ : chr "honda"
 $ : chr "hyundai"
 $ : chr "benz"
> car_models_lower <-unlist(lapply(car_models,tolower))
> str(car_models_lower)
 chr [1:4] "maruthi" "honda" "hyundai" "benz"
> car_models_lower <-unlist(lapply(car_models,toupper))
> str(car_models_lower)
 chr [1:4] "MARUTHI" "HONDA" "HYUNDAI" "BENZ"
> 
```

sapply( ) function

- **sapply()** function takes list, vector or data frame as input and gives output in vector or matrix.
- It is useful for operations on list objects and returns a list object of same length of original set.
- sapply() function does the same job as lapply() function but returns a vector.

sapply( ) function

- sapply(X, FUN)
- Arguments:

  Here 'X' is a vector or an object

  Here 'FUN' is a Function applied to each element of X

Example

Consider "cars" dataset.

Check their attribute and values of that "cars" dataset and we can measure the minimum speed and stopping distances..

```
A ← cars
lcars ←lapply(A, min)
scars ←sapply(A,min)
print(lcars)
print(scars)
```

sapply()

We can use a user built-in function into lapply() or sapply(). We create a function named avg to compute the average of the minimum and maximum of the vector.

```
Console   Terminal   Jobs
C:/WINDOWS/system32/
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
> dt<-cars
> min_cars<-lapply(dt,min)
> min_cars
$speed
[1] 4

$dist
[1] 2

> stopping_dist<-sapply(dt,min)
> stopping_dist
speed  dist
    4     2
> avg <- function(x) {
+     ( min(x) + max(x) ) / 2}
>
> fcars <- sapply(dt, avg)
> fcars
speed  dist
 14.5  61.0
> |
```

Summary – apply, lapply() and sapply()

Function	Arguments	Objective	Input	Output
apply	apply(x, MARGIN, FUN)	Apply a function to the rows or columns or both	Data frame or matrix	vector, list, array
lapply	lapply(X, FUN)	Apply a function to all the elements of the input	List, vector or data frame	list
sapply	sappy(X FUN)	Apply a function to all the elements of the input	List, vector or data frame	vector or matrix

tapply() function

- **tapply()** computes a measure (mean, median, min, max, etc..) or a function for each factor variable in a vector.
- It is a very useful function that lets you create a subset of a vector and then **apply some functions to each of the subset.**

tapply() function

- Syntax:

tapply(X, INDEX, FUN = NULL)

Arguments:

'X' is an object, usually a vector

'INDEX' is a list containing factor

'FUN' is function applied to each element of x

tapply() function

```
> ?cars
> data(cars)
> str(cars)
'data.frame':   50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
> tapply(cars$speed,cars$dist, mean)
       2        4       10       14       16       17       18       20       22
 4.00000  7.00000  6.50000 12.00000  8.00000 11.00000 10.00000 13.50000  7.00000
      24       26       28       32       34       36       40       42       46
12.00000 13.00000 11.50000 17.66667 12.00000 16.50000 16.50000 18.00000 16.00000
      48       50       52       54       56       60       64       66       68
20.00000 17.00000 20.00000 19.00000 19.00000 14.00000 20.00000 22.00000 19.00000
      70       76       80       84       85       92       93      120
24.00000 18.00000 14.00000 18.00000 25.00000 24.00000 24.00000 24.00000
>
```

Descriptive Data analysis using R

- Datasets link

https://vincentarelbundock.github.io/Rdatasets/datasets.html

Recursive Function

- In a recursive function (recursion), function calls itself. In this, to solve the problems, we break the programs into smaller sub-programs.

- **For example:**

  4! = 4*3*2*1 = 24

Recursive Function (contd..)

```
tri_recursion <- function(k) {
  if (k > 0) {
    result <- k + tri_recursion(k - 1)
    print(result)
  } else {
    result = 0
    return(result)
  }
}
tri_recursion(6)
```

```
[1] 1
[1] 3
[1] 6
[1] 10
[1] 15
[1] 21
```

Find the Factorial of a given number in R

```
Factorial <- function(N)
{
if (N == 0)
return(1)
else
return( N * Factorial (N-1))
}
```

Finding factorial of a number using the recursive function.

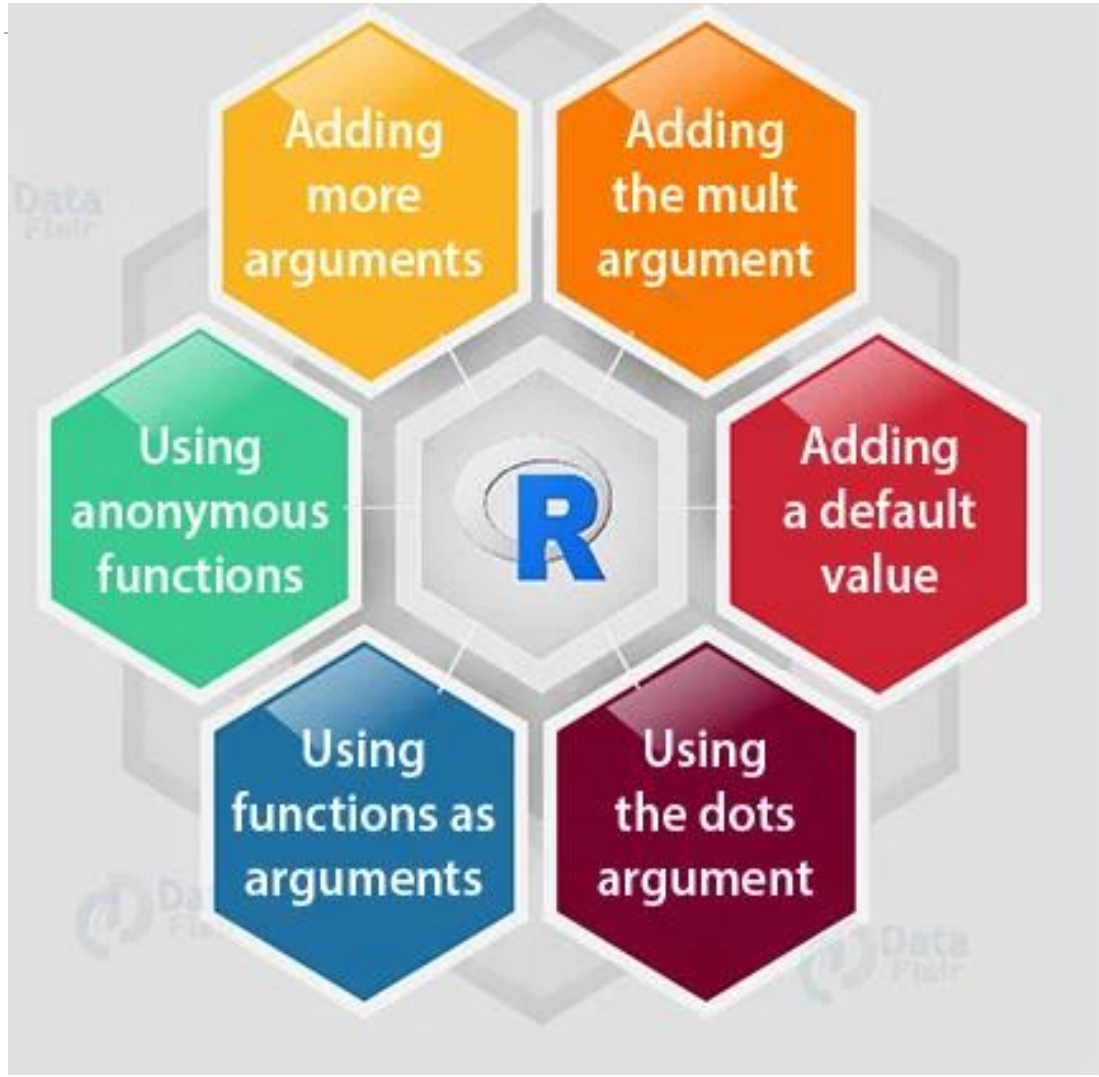
```
recurive_factorial <- function(n) {
if(n <= 1) {
return(1)
} else {
return(n * recurive_factorial(n-1))
}
}
```

- Let's find out the solution for the below program…

Finding sum of series $1^2+2^2+3^2+\ldots..+n^2$ using the recursive function

Arguments

In



Arguments in **R**

- *Arguments are always named when you define a function.*
- When you call a function, you do not have to specify the name of the argument.
- Arguments are optional; you do not have to specify a value for them.
- They can have a default value, which is used if you do not specify a value for that argument yourself.
- You can use as many arguments as you like, there is no limit to the number of arguments.
- An argument list comprises of comma-separated values that contain the various formal arguments.

```
Packages in library 'C:/Users/M V Kamal/Documents/R/win-library/3.6':

assertthat              Easy Pre and Post Assertions
BH                      Boost C++ Header Files
cli                     Helpers for Developing Command Line Interfaces
clipr                   Read and Write from the System Clipboard
colorspace              A Toolbox for Manipulating and Assessing Colors
                        and Palettes
```

Console | Terminal

```
> #To pass values to a function, you can use R arguments as many as needed. First, we w
ill create our generic function addPercent as follows:
> addPercent <- function(x, mult = 100, ...){
+     percent <- round(x*mult, ...)
+     paste(percent, "%", sep = "")
+ }
>
> #The addPercent function converts the value to a percentage. When the calculated numb
ers are in percentage format, then first you will have to divide these numbers by 100 t
o get the correct result.
> percentages <- c(58.23, 120.4, 33)
> addPercent(percentages/100)
[1] "58%"  "120%" "33%"
>
```

Syllabus – Covered Topies

- Introduction to R- Features of R – Environment, How to run R, R Sessions and Functions, Basic Math, Variables, Data Types, Vectors, Conclusion, Advanced Data Structures, Data Frames, Lists, Matrices, Arrays, Classes, R Programming Structures, Control Statements, Loops, - Looping Over Nonvector Sets,- If-Else, Arithmetic and Boolean Operators and values, Default Values for Argument, Return Values, Functions are Objects, Recursion,
- Basic Functions - R help functions - R Data Structures. Vectors: Definition- Declaration - Generating - Indexing - Naming - Adding & Removing elements - Operations on Vectors - Recycling - Special Operators - Vectorized if- then else-Vector Equality – Functions for vectors - Missing values - NULL values - Filtering & Subsetting.

End of Unit-II (Part-A)

Data Science Tools and Techniques

Unit-II

R Functions (Part-B)

By: M V Kamal | Associate Professor | CSE Dept. | MRCET

Syllabus – Covered Topics

- Introduction to R- Features of R – Environment, How to run R, R Sessions and Functions, Basic Math, Variables, Data Types, Vectors, Conclusion, Advanced Data Structures, Data Frames, Lists, Matrices, Arrays, Classes, R Programming Structures, Control Statements, Loops, - Looping Over Nonvector Sets,- If-Else, Arithmetic and Boolean Operators and values, Default Values for Argument, Return Values, Functions are Objects, Recursion.
- Basic Functions - R help functions - R Data Structures. Vectors: Definition- Declaration - Generating - Indexing - Naming - Adding & Removing elements - Operations on Vectors - Recycling - Special Operators - Vectorized if- then else-Vector Equality – Functions for vectors - Missing values - NULL values - Filtering & Subsetting.

The **help() function** in R

- The **help() function** in R is used to get help on any given R function passed to it.

Or

- The help() function and ? help operator in R provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages.

- To access documentation for the standard lm (linear model) function, for example, enter the command help(lm) or help("lm"), or ?lm or ?"lm" (i.e., the quotes are optional).

The **help() function** in R **(contd..)**

- You can learn more about help() function
- https://www.r-project.org/help.html

The **help() function** in R **(contd..)**

- To access help for a function in a package that's *not* currently loaded, specify in addition the name of the package:

  For example, to obtain documentation for the rlm() (robust linear model) function in the **MASS** package, help(rlm, package="MASS").

The **help() function** in R **(contd..)**

- The help() function and ? operator are useful only if you already know the name of the function that you wish to use.
- There are also facilities in the standard R distribution for discovering functions and other objects.
- The following functions cast a progressively wider net. Use the help system to obtain complete documentation for these functions: for example,

?apropos

Note: The apropos() function searches for objects, including functions, directly accessible in the current R session that have names that include a specified character string

help.search() and ??

- The help.search() function scans the documentation for packages installed in your library.
- The (first) argument to help.search() is a character string or regular expression.

Vignettes and Code Demonstrations:
browseVignettes(), vignette() and demo()

- Many packages include *vignettes*, which are discursive documents meant to illustrate and explain facilities in the package.

- You can discover vignettes by accessing the help page for a package, or via the browseVignettes() function.

- The command browseVignettes() opens a list of vignettes from *all* of your installed packages in your browser.

RSiteSearch()

- RSiteSearch() uses an internet search engine to search for information in function help pages and vignettes for all CRAN packages, and in CRAN task views
- Unlike the apropos() and help.search() functions, RSiteSearch() requires an active internet connection and doesn't employ regular expressions.
- Braces may be used to specify multi-word terms; otherwise matches for individual words are included.

  For example, RSiteSearch("{generalized linear model}")

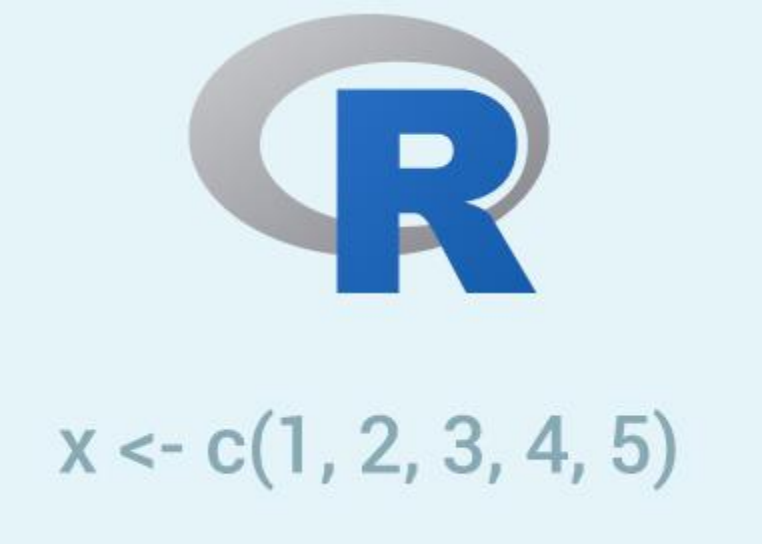
findfn() and ???

- findfn() and ??? in the **sos** package, which is *not* part of the standard R distribution but is available on CRAN, provide an alternative interface to RSiteSearch().

help.start()

- help.start() starts and displays a hypertext based version of R's online documentation in your default browser that provides links to locally installed versions of the R manuals, a listing of your currently installed packages and other documentation resources.

Vector &
Vector Indexing



Vector & Vector Indexing

- Know about Vector and Vector Indexing through below link..

- https://thomasleeper.com/Rcourse/Tutorials/vectorindexing.html

- *& Refer the PDF Material (File Name:Vector in R_Unit_II (Part-B).pdf)*

A Vectorized if-then-else: The ifelse() Function

- In addition to the usual if-then-else construct found in most languages, R also includes a vectorized version, the ifelse() function.
- ifelse(b,u,v)
- where b is a Boolean vector, and u and v are vectors.
- Example

```
> x <- 1:10
>
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> y <-(x %% 2)
> y
 [1] 1 0 1 0 1 0 1 0 1 0
> y <- ifelse(x %% 2 == 0,5,12)
> y
 [1] 12  5 12  5 12  5 12  5 12  5
> |
```

Vector Equality

Unit-II (Part-B)

Vector Equality

- Suppose we wish to test whether two vectors are equal. The naive approach, using ==, won't work.

- Example-1
- > x <- 1:3
- > y <- c(1,3,4)
- > x == y

[1] TRUE FALSE FALSE

Example-2

- i <- 2
- > "= ="(i,2)

[1] TRUE

In fact, == is a vectorized function. The expression x == y applies the function ==() to the elements of x and y. yielding a vector of Boolean values.

all()

- > x <- 1:3
- > y <- c(1,3,4)
- > x == y

[1] TRUE FALSE FALSE

- all(x == y)

[1] FALSE

Applying all() to the result of == asks whether all of the ...

Check if Two Objects are Equal in R Programming – **setequal()** Function

- **setequal()** function in R Language is used to check if two objects are equal.
- This function takes two objects like Vectors, dataframes, etc. as arguments and results in TRUE or FALSE, if the Objects are equal or not.

```
# R program to illustrate
# the use of setequal() function
x1 <- c(1, 2, 3, 4, 5, 6)          # Vector 1
x2 <- c(1:6)                       # Vector 2
x3 <- c(2, 3, 4, 5, 6)             # Vector 3
# Calling setequal() Function
setequal(x1, x2)
setequal(x1, x3)
```

Output:

```
[1] TRUE
[1] FALSE
```

Example-2 for **setequal**

```
# R program to illustrate the use of setequal() function

# Dataframe-1
data_x <- data.frame(x1 = c(5, 6, 7),
                     x2 = c(2, 2, 2))

# Dataframe-2
data_y <- data.frame(y1 = c(5, 6, 7),
                     y2 = c(2, 2, 2))

# Calling setequal() Function
setequal(data_x, data_y)
```

Output:

```
[1] TRUE
```

Functions for vectors

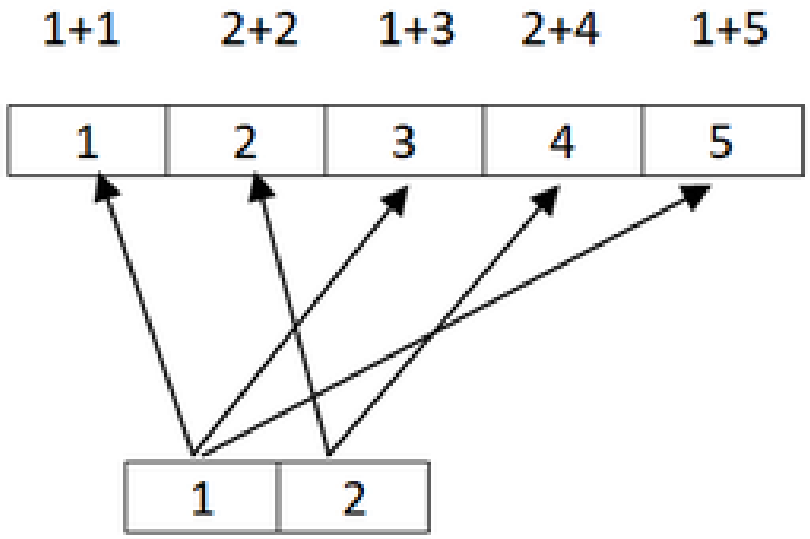
- Vector functions under R the ones that allow us to either create or manipulate the data structure called vectors.
- These functions most of the time take a vector/s as an argument to generate an output.
- rep() function
- seq() function
- is.vector() function
- as.vector() function
- any() function
- all() function
- lapply() function
- sapply() function

Vector Recycling in R

- We can see vector recycling, when we perform some kind of operations like addition, subtraction. . . .etc on two vectors of unequal length.
- The vector with a small length will be repeated as long as the operation completes on the longer vector.
- If we perform an addition operation on a vector of equal length the first value of vector1 is added with the first value of vector 2 like that.
- The repetition of small length vector as long as completion of operation on long length vector is known as **vector recycling.**

For Example

```
# creating vector with 1 to 5 values
> v1<-c(1:5)
> v1
 [1] 1 2 3 4 5
# creating vector with 1:2 values
> v2<-c(1:2)
> v2
[1] 1 2
# adding vector1 and vector2
> print(v1+v2)
[1] 2 4 4 6 6
Warning message: In v1 + v2 : longer
object length is not a multiple of
shorter object length
```



Missing values

- Dealing with Missing Values in R is important and simple.
- A common task in data analysis is dealing with missing values. In R, missing values are often represented by NA or some other value that represents missing values

Missing values (contd..)

- A common task in data analysis is dealing with missing values. In R, missing values are often represented by NA or some other value that represents missing values.

- To identify missing values use is.na() which returns a logical vector with TRUE in the element locations that contain missing values represented by NA. is.na() will work on vectors, lists, matrices, and data frames.

```
longer object length is not a multiple of shorter ob
> x <- c(1:4, NA, 6:7, NA)
> x
[1]  1  2  3  4 NA  6  7 NA
> is.na(x)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
>
> # data frame with missing data
> df <- data.frame(col1 = c(1:3, NA),
+                  col2 = c("this", NA,"is", "text"),
+                  col3 = c(TRUE, FALSE, TRUE, TRUE),
+                  col4 = c(2.5, 4.2, 3.2, NA),
+                  stringsAsFactors = FALSE)
> df
  col1 col2  col3 col4
1    1 this  TRUE  2.5
2    2 <NA> FALSE  4.2
3    3   is  TRUE  3.2
4   NA text  TRUE   NA
> is.na(df)
      col1  col2  col3  col4
[1,] FALSE FALSE FALSE FALSE
[2,] FALSE  TRUE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE
[4,]  TRUE FALSE FALSE  TRUE
> # identify NAs in specific data frame column
> is.na(df$col4)
[1] FALSE FALSE FALSE  TRUE
>
```

is.na() validates the blank position as true

which()

```
x <- c(1:4, NA, 6:7, NA)
> x
 [1] 1 2 3 4 NA 6 7 NA


> # which (is.na(x) is to identify location of 'NA' in vector
> which(is.na(x))
 [1] 5 8
```

Vector in R | DSTT_Unit_II (Part-B)

(Compiled by Dr MVK)

Vector is a basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw.
A vector's type can be checked with the `typeof()` function.
Another important property of a vector is its length. This is the number of elements in the vector and can be checked with the function `length()`.

How to Create Vector in R?

Vectors are generally created using the `c()` function.

Since, a vector must have elements of the same type, this function will try and coerce elements to the same type, if they are different.
Coercion is from lower to higher types from logical to integer to double to character.

```
> x <- c(1, 5, 4, 9, 0)
> typeof(x)
[1] "double"
> length(x)
[1] 5
> x <- c(1, 5.4, TRUE, "hello")
> x
[1] "1"     "5.4"   "TRUE"  "hello"
> typeof(x)
[1] "character"
```

If we want to create a vector of consecutive numbers, the `:` operator is very helpful.

Example 1: Creating a vector using : operator

```
> x <- 1:7; x
[1] 1 2 3 4 5 6 7
> y <- 2:-2; y
[1]  2  1  0 -1 -2
```

More complex sequences can be created using the `seq()` function, like defining number of points in an interval, or the step size.

Example 2: Creating a vector using seq() function

```
> seq(1, 3, by=0.2)           # specify step size
[1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
> seq(1, 5, length.out=4)     # specify length of the vector
[1] 1.000000 2.333333 3.666667 5.000000
```

How to access Elements of a Vector?

Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector.

Using integer vector as index

Vector index in R starts from 1, unlike most programming languages where index start from 0.
We can use a vector of integers as index to access specific elements.
We can also use negative integers to return all elements except that those specified.
But we cannot mix positive and negative integers while indexing and real numbers, if used, are truncated to integers.

```
> x
[1]  0  2  4  6  8 10
> x[3]           # access 3rd element
[1] 4
> x[c(2, 4)]     # access 2nd and 4th element
[1] 2 6
> x[-1]          # access all but 1st element
[1]  2  4  6  8 10
> x[c(2, -4)]    # cannot mix positive and negative integers
Error in x[c(2, -4)] : only 0's may be mixed with negative subscripts
> x[c(2.4, 3.54)]    # real numbers are truncated to integers
[1] 2 4
```

Using logical vector as index

When we use a logical vector for indexing, the position where the logical vector is `TRUE` is returned.
This useful feature helps us in filtering of vector as shown below.

```
> x[c(TRUE, FALSE, FALSE, TRUE)]
[1] -3  3
> x[x < 0]  # filtering vectors based on conditions
[1] -3 -1
> x[x > 0]
[1] 3
```

In the above example, the expression `x>0` will yield a logical vector `(FALSE, FALSE, FALSE, TRUE)` which is then used for indexing.

Using character vector as index

This type of indexing is useful when dealing with named vectors. We can name each elements of a vector.

```
> x <- c("first"=3, "second"=0, "third"=9)
> names(x)
[1] "first"  "second" "third"
> x["second"]
second
0
> x[c("first", "third")]
first third
3     9
```

How to modify a vector in R?

We can modify a vector using the assignment operator.
We can use the techniques discussed above to access specific elements and modify them.
If we want to truncate the elements, we can use reassignments.

```
> x
[1] -3 -2 -1  0  1  2
> x[2] <- 0; x        # modify 2nd element
[1] -3  0 -1  0  1  2
> x[x<0] <- 5; x   # modify elements less than 0
[1] 5 0 5 0 1 2
> x <- x[1:4]; x      # truncate x to first 4 elements
[1] 5 0 5 0
```

How to delete a Vector?

We can delete a vector by simply assigning a `NULL` to it.

```
> x
[1] -3 -2 -1  0  1  2
> x <- NULL
> x
NULL
> x[4]
NULL
```

How to **Sort a Vector..?**

To sort items in a vector alphabetically or numerically, use the `sort()` function:

Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")
numbers <- c(13, 3, 5, 7, 20, 2)

sort(fruits)  # Sort a string
sort(numbers) # Sort numbers
```

How to **Access** Vectors

You can access the vector items by referring to its index number inside brackets `[]`. The first item has index 1, the second item has index 2, and so on:

Example

```
fruits <- c("banana", "apple", "orange")

# Access the first item (banana)
fruits[1]
```

You can also access multiple elements by referring to different index positions with the `c()` function:

Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")

# Access the first and third item (banana and orange)
fruits[c(1, 3)]
```

You can also use negative index numbers to access all items except the ones specified:

Example

```
fruits <- c("banana", "apple", "orange", "mango", "lemon")

# Access all items except for the first item
fruits[c(-1)]
```

How to **Repeat** Vectors

To repeat vectors, use the `rep()` function:

Example

Repeat each value:

```
repeat_each <- rep(c(1,2,3), each = 3)
repeat_each
```

Example

Repeat the sequence of the vector:

```
repeat_times <- rep(c(1,2,3), times = 3)
repeat_times
```

Example

Repeat each value independently:

```
repeat_indepent <- rep(c(1,2,3), times = c(5,2,1))
repeat_indepent
```

Generating **Sequenced** Vectors

One of the examples on top, showed you how to create a vector with numerical values in a sequence with the `:` operator:

Example

```
numbers <- 1:10
numbers
```

To make bigger or smaller steps in a sequence, use the `seq()` function:

Example

```
numbers <- seq(from = 0, to = 100, by = 20)
numbers
```

Note: The `seq()` function has three parameters: `from` is where the sequence starts, `to` is where the sequence stops, and `by` is the interval of the sequence.

UNIT - III

Data Science Tools and Techniques

Unit-III

Data Analytics with Excel

By: M V Kamal | Associate Professor | CSE Dept. | MRCET

Syllabus

- Introduction: Data Analysis, Excel Data analysis. Working with range names. Tables. Cleaning Data. Conditional formatting, Sorting, Advanced Filtering, Lookup functions, Pivot tables, Data Visualization, Data Validation. Understanding Analysis tool pack: Anova, correlation, covariance, moving average, descriptive statistics, exponential smoothing, fourier Analysis, Random number generation, sampling, t-test, f-test, and regression

Data Analysis

- What….?
- Why….?

What is **Data Analysis**

- Businesses today need every edge and advantage they can get.
- Data analysis is the process of cleaning, changing, and processing raw data and extracting actionable, relevant information that helps businesses make informed decisions.
- The procedure helps reduce the risks inherent in decision-making by providing useful insights and statistics, often presented in charts, images, tables, and graphs.

What is **Data Analysis**

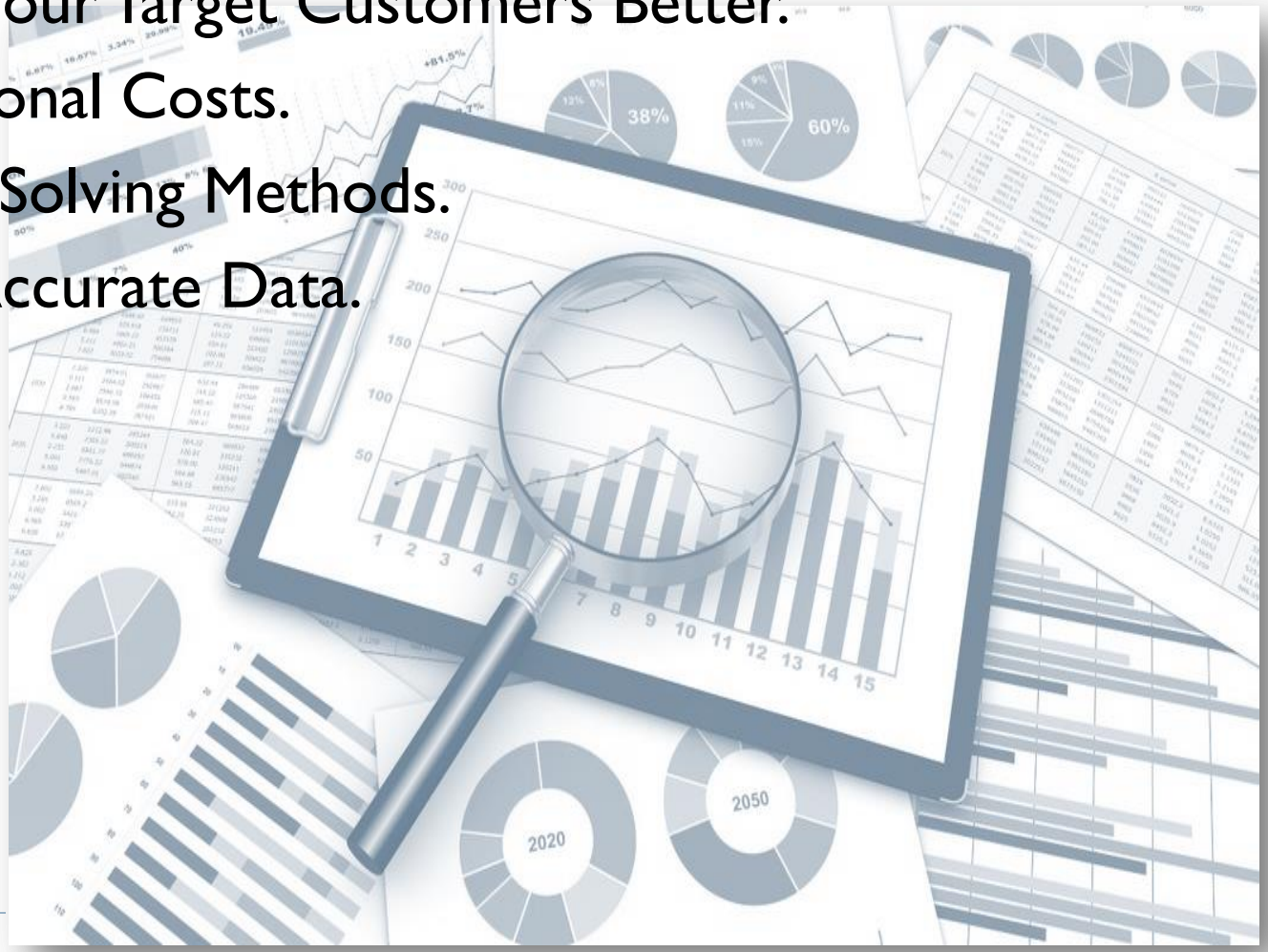
- **Data Analysis** is the process of systematically applying statistical and/or logical techniques to describe and illustrate, condense and recap, and evaluate data.

- An essential component of ensuring data integrity is the accurate and appropriate analysis of research findings.

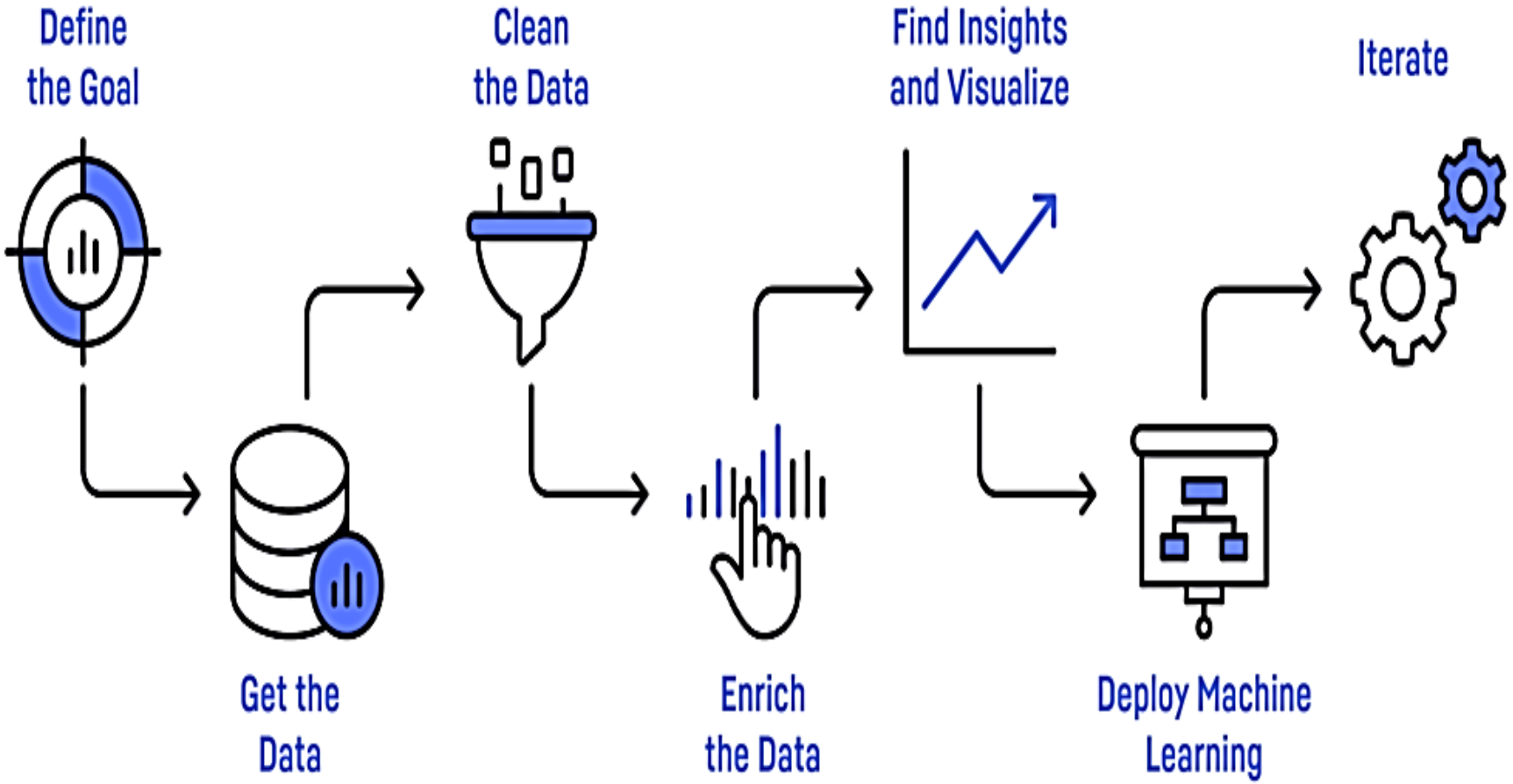
- Data analysis plays a crucial role in processing big data into useful information.

Why is Data Analysis **Important**?

- Better Customer Targeting.
- You Will Know Your Target Customers Better.
- Reduce Operational Costs.
- Better Problem-Solving Methods.
- You Get More Accurate Data.

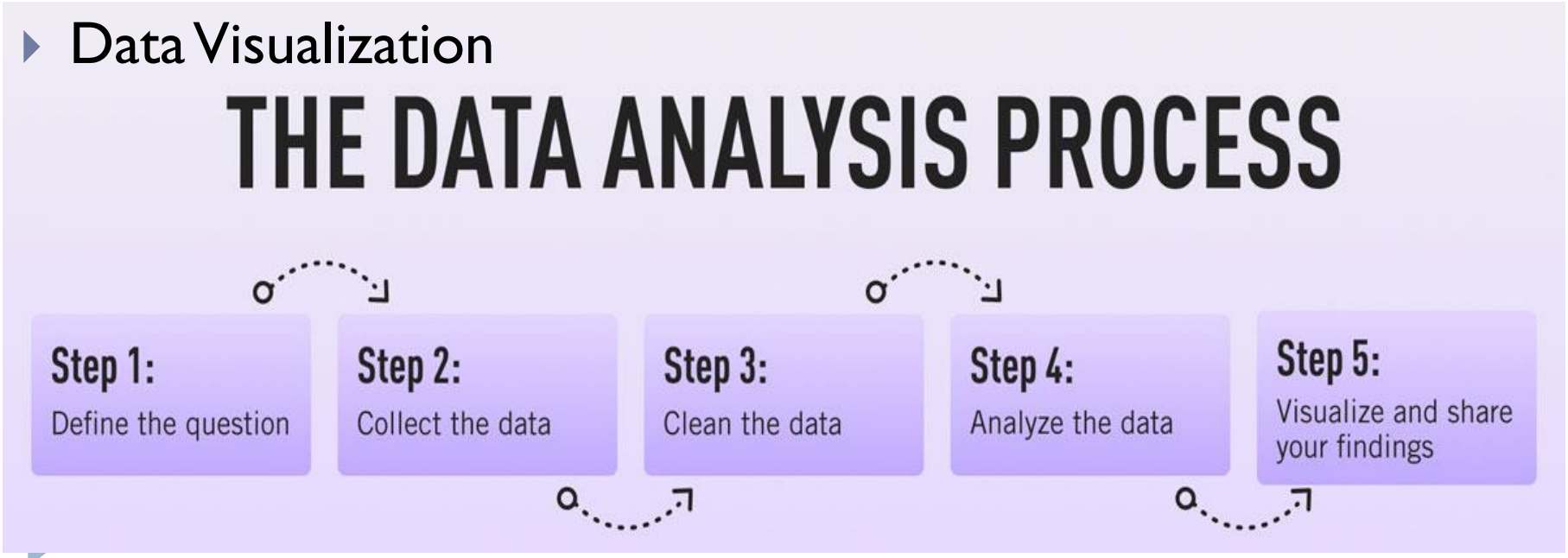


Data Analysis - **Process**

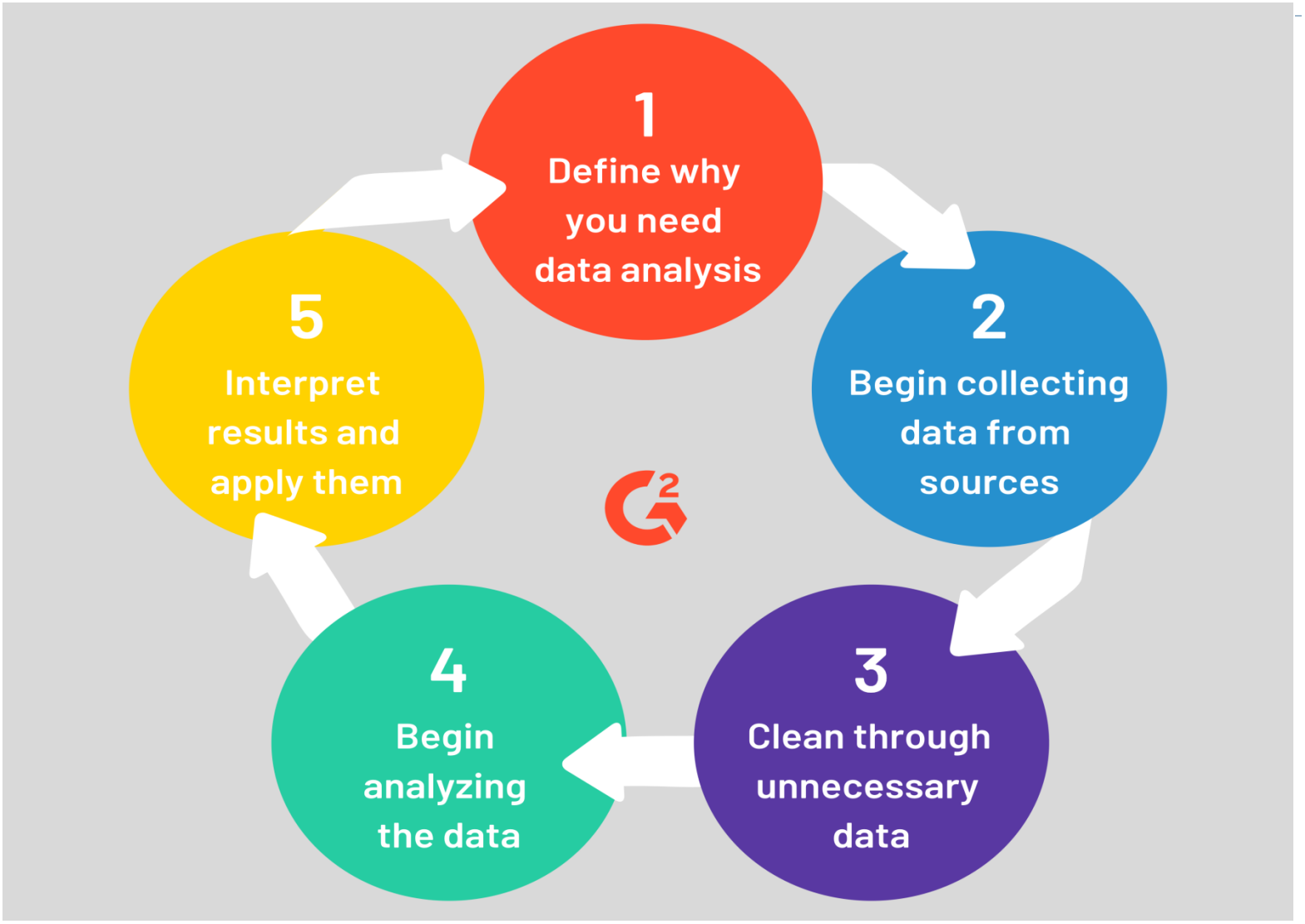


Data Analysis **Process.**

- Data Requirement Gathering
- Data Collection
- Data Cleaning
- Data Analysis
- Data Interpretation
- Data Visualization



Data Analysis **Process.**



Considerations/issues in data analysis

- There are a number of issues that researchers should be cognizant of with respect to data analysis.
- These include:
  - Having the necessary skills to analyze
  - Concurrently selecting data collection methods and appropriate analysis
  - Drawing unbiased inference
  - Inappropriate subgroup analysis
  - Following acceptable norms for disciplines
  - Determining **statistical significance**
  - Lack of clearly defined and objective **outcome measurements**

Considerations/issues in data analysis

Contd...

- Providing honest and accurate analysis
- Manner of presenting data
- Environmental/contextual issues
- Data recording method
- **Partitioning 'text'** when analyzing qualitative data
- Training of staff conducting analyses
- Reliability and Validity
- **Extent of analysis**

Excel Data Analysis

Instructor: M V Kamal | Associate Professor | CSE Dept. | MRCET

Excel Data Analysis

- Microsoft Excel is one of the most popular applications for data analysis.
- Equipped with built-in pivot tables, they are without a doubt the most sought-after analytic tool available.
- It is an all-in-one data management software that allows you to easily import, explore, clean, analyze, and visualize your data.
- In this article, we will discuss the various methods of data analysis in Excel.

How to Utilize Data Analysis in Excel

- Charts – Data Visualization
- Conditional Formats – Statistical

etc…

- Refer the PDF document
  (About Excel and Various Data Analysis Operation which can perform using Excel)

File Name: **A Comprehensive guide to Microsoft Excel for Data Analysis.pdf**

Understanding Analysis tool pack: **ANOVA**

Refer: PDF Document and Excel File for Execution of ANOVA

File Name: **ANOVA-Document.pdf**

ANOVA? (Analysis of Variance)

How to Install the ANOVA Package in the Excel

Steps:

How to load the Analysis ToolPak add-in (Windows)

1. Go to the File tab on the ribbon and click Options,
2. Click the Add-Ins category on the left. (In Excel 2007, click the Microsoft Office Button, and then click Excel Options.)
3. From the Manage drop-down list, select Excel Add-ins, then click Go.
4. In the Add-Ins dialog box, tick the Analysis ToolPak check box, then click OK.

About ANOVA-Data Analysis and its Options

Data Analysis window pops up, listing **19 analysis tools** which are linked to functions designed to analyze data using various mathematical formulas.
You may notice that Excel takes slightly longer to open when the add-in is loaded. This is to be expected, as more resources are being used to run the application.
The tools currently available are:

- Anova (Single-Factor)
- Anova (Two-Factor With Replication)
- Anova (Two-Factor Without Replication)
- Correlation
- Covariance
- Descriptive Statistics
- Exponential Smoothing
- F-Test Two-Sample for Variances
- Fourier Analysis
- Histogram
- Moving Average
- Random Number Generation
- Rank and Percentile
- Regression
- Sampling
- t-Test: Paired Sample for Means
- t-Test: Two-Sample Assuming Equal Variances
- t-Test: Two-Sample Assuming Unequal Variances
- z-Test: Two Sample for Means

The purpose of each analytical tool is shown below.

Tool	Description
Anova (Analysis of Variance): Single Factor	This tool determines if there is a relationship between two datasets by performing a simple analysis of variance.
Anova (Analysis of Variance): Two Factor with Replication	This tool determines if there is a relationship between two datasets by performing an analysis of variance when each data set has more than one observable data point.
Anova (Analysis of Variance): Two-Factor without Replication	This tool determines if there is a relationship between two data sets by performing an analysis of variance. There is only a single observable data point for each pair.
Correlation	Tells you how strongly two variables are related to each other.
Covariance	The Covariance analysis tool calculates the average of the product of deviations of values from the means of each data set.
Descriptive Statistics	Generates a report of univariate statistics for the selected data. Statistics generated include: Mean, Standard Error, Median, Mode, Standard Deviation, Sample Variance, Kurtosis, Skewness, Range, Minimum, Maximum, Sum, Count, Largest, Smallest and Confidence Level.
Exponential Smoothing	Smooths out irregularities (peaks and valleys) in data, to easily recognize trends . More recent data is weighted more heavily.
F-Test Two Sample for Variances	This analysis tool compares the variances between two groups of data.
Fourier Analysis	This tool solves problems in linear systems and analyzes periodic data by using the Fast Fourier Transform (FFT) method to transform data. The Fourier Analysis tool also supports inverse transformations, where the inverse of transformed data returns the original data.
Histogram	The Histogram analysis tool counts occurrences in each of several data bins. It calculates individual and cumulative frequencies for a cell range of data and data bins. The output is a table and column chart by the frequency of occurrences.

Tool	Description
Moving Average	Calculates a moving average to allow you to smooth out a data series that contains peaks and outliers. Used for forecasting trends in sales, inventory, call volume, etc.
Random Number Generation	Creates a number of several types of random numbers including Uniform, Normal, Bernoulli, Poisson, Patterned and Discrete. More flexible than the RAND and RANDBETWEEN functions.
Rank and Percentile	Creates a table which ranks numbers from highest to lowest and provides a percentile value of each number relative to the other numbers within the data set.
Regression	Uses the function LINEST to analyze how a single dependent variable is affected by the values of one or more independent variables. Creates a table of statistics that result from least-squares regression.
Sampling	Samples a population randomly or periodically, as desired.
t-Test: Paired Two Sample for Means	Paired two-sample student's T-Test. Each Two-Sample t-Test analysis tool tests for equality of the population means that underlie each sample. The paired two-sample form of the t-Test is used when there is a natural pairing of observations in the samples — for example, when a sample group is tested twice, before and after an experiment. There is no assumption that the variances of both populations are equal.
T-Test: Two Sample assuming equal Variances	This analysis tool performs a two-sample student's t-Test. This t-Test form is based on the assumption that the two paired data sets came from distributions with the same variances. It is also known as a “homoscedastic t-Test”. This t-Test can be used to determine if the two samples are likely to have come from distributions with equal population means.
T-Test: Two Sample assuming unequal Variances	This t-Test form assumes that the two datasets are from distributions where the variances are unequal. This is called a “heteroscedastic t-Test”.
Z-Test: Two Sample for Means	The Two Sample for Means analysis tool performs a two sample z-Test for means with known variances. This analysis tool is used to test the null hypothesis that there is no difference between two population means against either one-sided or two-sided alternative hypotheses. If mean variances are not known, use the Z.TEST function instead.

What Is ANOVA? (Analysis of Variance)

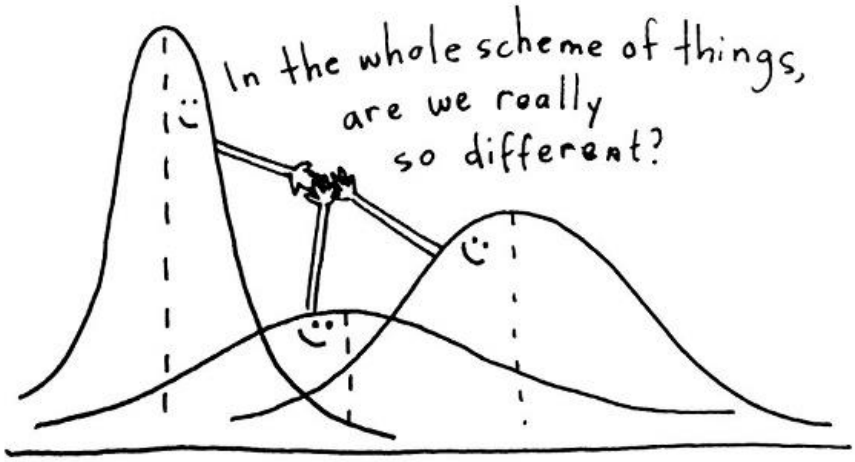
Buying a new product or testing a new technique but not sure how it stacks up against the alternatives? It's an all too familiar situation for most of us. Most options sound similar to each other, so picking the best out of the lot is a challenge.

Consider a scenario where we have three medical treatments for patients with similar diseases. Once we have the test results, one approach is to assume that the treatment which took the least time to cure the patients is the best among them. What if some of these patients had already been partially cured, or if any other medication was already working on them?

In order to make a confident and reliable decision, we will need evidence to support our approach. This is where the concept of ANOVA comes into play.

A common approach to figuring out a reliable treatment method would be to analyze the days the patients took to be cured. We can use a statistical technique to compare these three treatment samples and depict how different these samples are from one another. Such a technique, which compares the samples based on their means, is called ANOVA.

Analysis of variance (ANOVA) is a statistical technique used to check if the means of two or more groups are significantly different from each other. ANOVA checks the impact of one or more factors by comparing the means of different samples. We can use ANOVA to prove/disprove whether all the medication treatments were equally effective.



Another measure to compare the samples is called a t-test. When we have only two

samples, t-test, and ANOVA give the same results. However, using a t-test would not be reliable in cases with more than 2 samples. If we conduct multiple t-tests for comparing more than two samples, it will have a compounded effect on the error rate of the result.

Terminologies Related to ANOVA

Before we start with the ANOVA applications, I would like to introduce some common terminologies used in the technique.

Grand Mean

Mean is a simple or arithmetic average of a range of values. There are two kinds of means that we use in ANOVA calculations, which are separate sample means $(\mu_1, \mu_2 \ \& \ \mu_3)$ and the grand mean (μ). . The grand mean is the mean of sample means or the mean of all observations combined, irrespective of the sample.

Hypothesis

Considering our above medication example, we can assume that there are 2 possible cases – either the medication will have an effect on the patients or it won't. These statements are called Hypothesis. A hypothesis is an educated guess about something in the world around us. It should be testable either by experiment or observation.

Just like any other kind of hypothesis that you might have studied in statistics, ANOVA also uses a Null hypothesis and an Alternate hypothesis. The Null hypothesis in ANOVA is valid when all the sample means are equal, or they don't have any significant difference. Thus, they can be considered as a part of a larger set of the population. On the other hand, the alternate hypothesis is valid when at least one of the sample means is different from the rest of the sample means. In mathematical form, they can be represented as:

$$H_o: \ \mu_1 = \mu_2 = \cdots = \mu_L \qquad \textit{Null hypothesis}$$

$$H_1: \ \mu_l \neq \mu_m \qquad \textit{Alternate hypothesis}$$

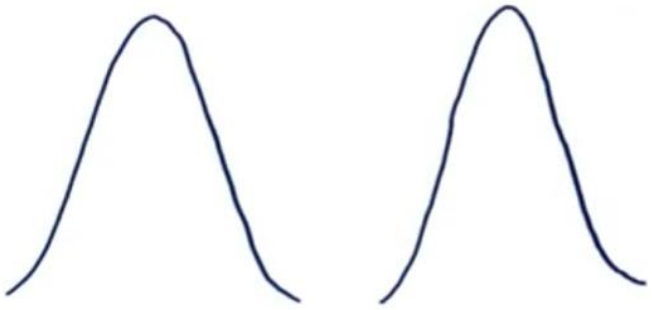
Where μ_l and μ_m belonging to any two sample means out of all the samples considered for the test. In other words, the null hypothesis states that all the sample

means are equal or the factor did not have any significant effect on the results. Whereas, the alternate hypothesis states that at least one of the sample means is different from another. But we still can't tell which one specifically. For that, we will use other methods that we will discuss later in this article.

Between Group Variability

Consider the distributions of the below two samples. As these samples overlap, their individual means won't differ by a great margin. Hence the difference between their individual and grand means won't be significant enough.

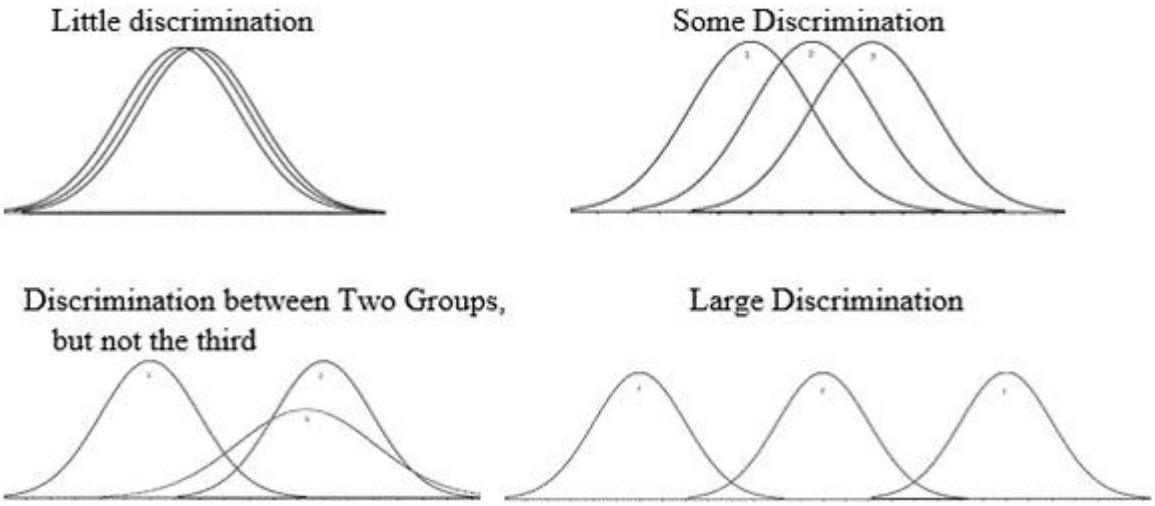
Now consider these two sample distributions. As the samples differ from each other by a big margin, their individual means would also differ. The difference between the individual means and grand mean would, therefore, also be significant.



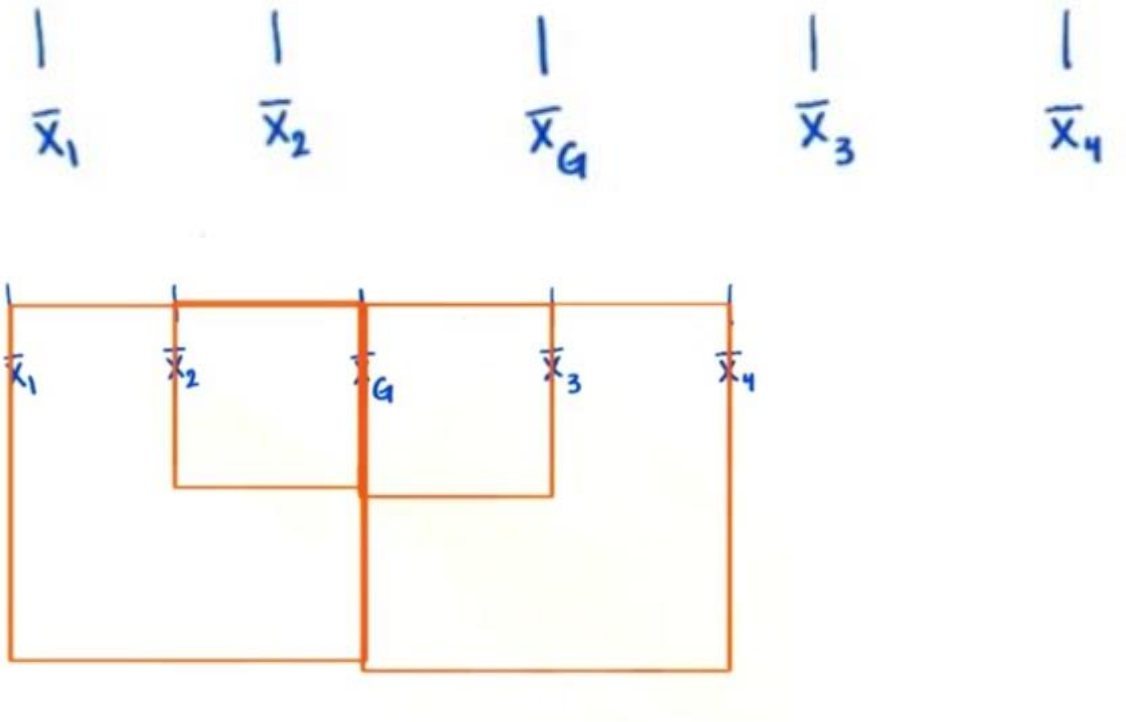
Such variability between the distributions is called **Between-group variability**. It refers to variations between the distributions of individual groups (or levels) as the values within each group differ.

Each sample is examined, and the difference between its mean and grand mean is calculated to calculate the variability. If the distributions overlap or are close, the

grand mean will be similar to the individual means, whereas if the distributions are far apart, the difference between means and grand mean would be large.



We will calculate **Between Group Variability** just as we calculate the standard deviation. Given the sample means and Grand mean, we can calculate it as follows:



We also want to weigh each squared deviation by the sample size. In other words, a deviation is given greater weight if it's from a larger sample. Hence, we'll multiply

each squared deviation by each sample size and add them. This is called the **sum-of-squares for between-group variability**$(SS_{between})$.

$$SS_{between} = n_1(\overline{x_1} - \overline{x_G})^2 + n_2(\overline{x_2} - \overline{x_G})^2 + n_3(\overline{x_3} - \overline{x_G})^2 + \ldots. n_k(\overline{x_k} - \overline{x_k})^2$$

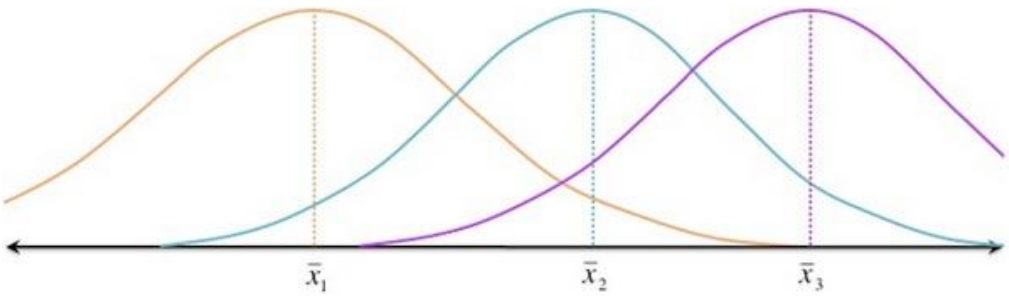
We must do one more thing to derive a good measure of between-group variability. Again, recall how we calculate the sample standard deviation.

We find the sum of each squared deviation and divide it by the degrees of freedom. For our between-group variability, we will find each squared deviation, weigh them by their sample size, sum them up, and divide by the degrees of freedom (), which in the case of between-group variability is the number of sample means (k) minus 1.

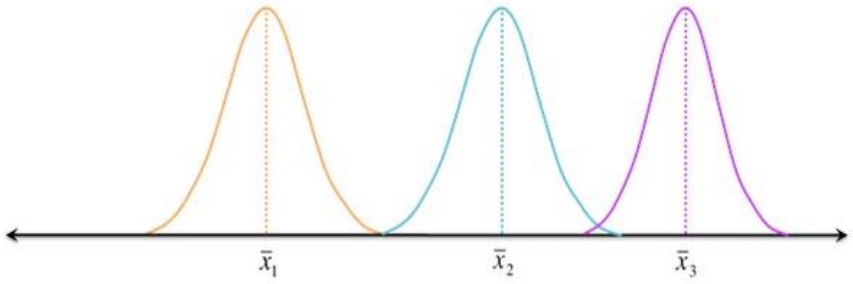
$$MS_{between} = \frac{n1(\overline{x_1} - \overline{x_G})2 + n2(\overline{x_2} - \overline{x_G})2 + n3(\overline{x_3} - \overline{x_G})2 + \ldots.nk(\overline{x_k} - \overline{x_k})2}{k-1}$$

Within Group Variability

Consider the given distributions of three samples. As the spread (variability) of each sample increases, their distributions overlap, and they become part of a big population.



Now consider another distribution of the same three samples but with less variability. Although the means of samples are similar to those in the above image, they seem to belong to different populations.

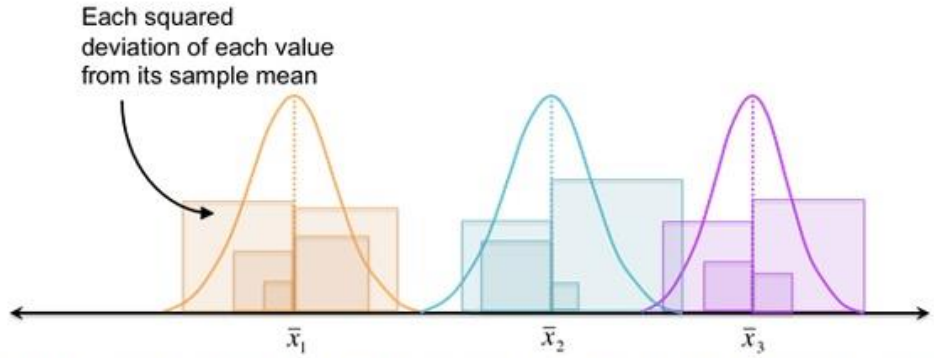


Such variations within a sample are denoted by **Within-group variation**. It refers to variations caused by differences within individual groups (or levels), as not all the values within each group are the same. Each sample is looked at on its own, and variability between the individual points in the sample is calculated. In other words, no interactions between samples are considered.

We can measure **Within-group variability** by looking at how much each value in each sample differs from its respective sample mean. So first, we'll take the squared deviation of each value from its respective sample mean and add them up. This is the **sum of squares for within-group variability**.

$$SS_{within} = \Sigma(x_{i1} - \bar{x}_1)^2 + \Sigma(x_{i2} - \bar{x}_2)^2 + \ldots + \Sigma(x_{ik} - \bar{x}_k)^2$$
$$= \Sigma(x_{ij} - \bar{x}_j)^2$$

Note: x_{i1} is the ith value from the first sample, x_{i2} is the ith value from the second sample, and so on all the way to x_{ik}, the ith value from the kth sample. x_{ij} is therefore the ith value from the jth sample.



With within-group variability, SS_{within} is the sum of each squared deviation of each value from its respective sample mean (the total area of all the squares in the figure above). MS_{within} is the average-sized square.

Like between-group variability, we then divide the sum of squared deviations by the **degrees of freedom** (df_{within}) to find a less-biased estimator for the average

squared deviation (essentially, the average-sized square from the figure above). Again, this quotient is the mean square, but for within-group variability: MS_{within}. This time, the degrees of freedom is the sum of the sample sizes (N) minus the number of samples (k). Another way to look at degrees of freedom is that have the total number of values (N) and subtract 1 for each sample:

$$df_{within} = (n_1 - 1) + (n_2 - 1) + \dots (n_k - 1) = n_1 + n_2 + n_3 + \dots + n_k - k(1) = N - k$$

$$MS_{within} = \sum(x_{ij} - \overline{x_j})^2 / (N - k)$$

F-Statistic (F-test)

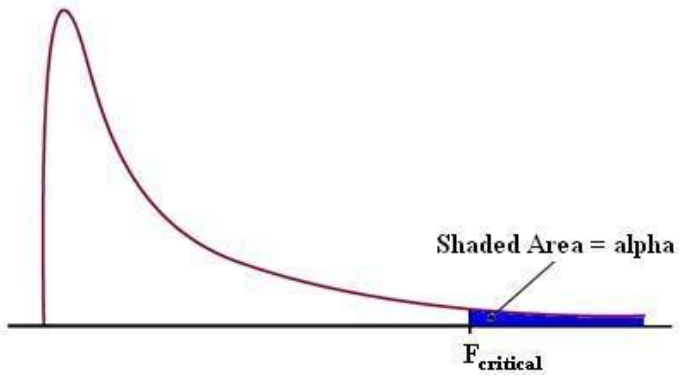
The statistic that measures whether the means of different samples are significantly different is called the F-Ratio. The lower the F-Ratio, the more similar will the sample means be. In that case, we cannot reject the null hypothesis.

F = Between-group variability / Within-group variability

This above formula is pretty intuitive. The numerator term in the F-statistic calculation defines the between-group variability. As we read earlier, the sample means to grow further apart as between-group variability increases. In other words, the samples are likelier to belong to different populations.

This F-statistic calculated here is compared with the F-critical value for concluding. In terms of our medication example, if the value of the calculated F-statistic is more than the F-critical value (for a specific α/significance level), then we reject the null hypothesis and can say that the treatment had a significant effect.

Unlike the z and t-distributions, the F-distribution has no negative values because between and within-group variability are always positive due to squaring each deviation.



Therefore, there is only one critical region in the right tail (shown as the blue-shaded region above). If the F-statistic lands in the critical region, we can conclude that the means are significantly different, and we reject the null hypothesis. Again, we must find the critical value to determine the cut-off for the critical region. We'll use the F-table for this purpose.

We need to look at different F-values for each alpha/significance level because the F-critical value is a function of two things: df_{within} and $df_{between}$

Material Source: **Analytics Vidya**

https://www.analyticsvidhya.com/blog/2018/01/anova-analysis-of-variance/

Compiled by: **Dr M V Kamal**

Other Reference Resources:

https://support.microsoft.com/en-us/office/use-the-analysis-toolpak-to-perform-complex-data-analysis-6c67ccf0-f4a9-487c-8dec-bdb5a2cefab6

Video: https://www.youtube.com/watch?v=ZvfO7-J5u34

UNIT - IV

Community Experience Distilled

KNIME Essentials

Perform accurate data analysis using the power of KNIME

Gábor Bakos

[PACKT] PUBLISHING

KNIME Essentials

Perform accurate data analysis using the power of KNIME

Gábor Bakos

[PACKT]
PUBLISHING

BIRMINGHAM - MUMBAI

KNIME Essentials

First published: October 2013

Production Reference: 1101013

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84969-921-1

www.packtpub.com

Cover Image by Abhishek Pandey (abhishek.pandey1210@gmail.com)

Credits

Author
Gábor Bakos

Reviewers
Thorsten Meinl
Takeshi Nakano

Acquisition Editors
Saleem Ahmed
Edward Gordon

Commissioning Editor
Amit Ghodake

Technical Editors
Iram Malik
Aman Preet Singh

Copy Editors
Gladson Monteiro
Kirti Pai
Mradula Hegde
Sayanee Mukherjee

Project Coordinator
Esha Thakker

Proofreader
Clyde Jenkins

Indexers
Tejal Daruwale
Priya Subramani

Graphics
Ronak Dhruv
Yuvraj Mannari

Production Coordinator
Prachali Bhiwandkar

Cover Work
Prachali Bhiwandkar

About the Author

Gábor Bakos is a programmer and a mathematician, having a few years of experience with KNIME and KNIME node development (HiTS nodes and RapidMiner integration for KNIME).

In Trinity College, Dublin, the author was helping a research group with his data analysis skills (also had the opportunity to improve those), and with the new KNIME node development. When he worked for the evopro Kft. or the Scriptum Informatika Zrt., he was also working on various data analysis software products. He currently works for his own company, Mind Eratosthenes Kft. (`www.mind-era.com`), where he develops the RapidMiner integration for KNIME (`tech.knime.org/community/rapidminer-integration`), among other things.

The author would like to thank the reviewers and Packt Publishing for their help in creating this book.

About the Reviewers

Thorsten Meinl is currently a Senior Software Developer at KNIME.com in Zurich. He holds a PhD in Computer Science from the University of Konstanz. He has been working on KNIME for over seven years. His main responsibilities are quality assurance, testing, and the continuous integration infrastructure, as well as managing the KNIME Community Contributions. Besides this, he is also interested in parallel computing and cheminformatics.

Takeshi Nakano is a Senior Research Engineer working for Recruit Technologies Co., Ltd. and leads the Advanced Technology Lab in Japan. He holds a Master's degree from the Nara Institute of Science and Technology (NAIST) in Computer Science. He is the lead author of Hadoop Hacks, a book from O'Reilly Japan, and also the author of Getting Started with Apache Solr, a book from GijutsuHyohron in Japan. He loves to find inspiration for his hobbies (reading, scuba diving, and others).

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Dear reader, welcome to an intuitive way of data analysis. Using a visual programming language based on dataflows, you can create an easy-to-understand analysis process, while it internally checks signals about some of the common problems. Obviously, any environment that does not help with proper documentation would be destined to fail, but KNIME's success is based not just on its high quality—cross-platform—code, but also on the good description about what it does and how you can use the building blocks.

This book covers the most common tasks that are required during the data preparation and visualization phase of data analysis using KNIME. Because of the size constraints—and to bring the best price/value for those who are already familiar with or not interested in modeling—we have not covered the modeling and machine learning algorithms available for KNIME. If you are already familiar with these algorithms, you will easily get familiar with the options in KNIME, and these are quite obvious to use, so you lose almost nothing. If you have not found time yet to get acquainted with these concepts, we encourage you to first learn for what these procedures are good and when you should use them. There are some good books, courses, and training available—these are the ideal options for learning—but the Wikipedia articles can also give you a basic introduction specific to the algorithm you want to use.

What this book covers

Chapter 1, *Installation and Using KNIME*, introduces the user interface, the concepts used in the first three chapters, and how you can install and configure KNIME and its extensions.

Chapter 2, *Data Preprocessing*, covers the most common tasks, so that you can analyze your data, such as loading, transforming, and generating data; it also introduces the powerful regular expressions and some case studies.

Chapter 3, *Data Exploration*, describes how you can use KNIME to get an overview about your data, how you can visualize them in different forms, or even create publication quality figures.

Chapter 4, *Reporting*, introduces the KNIME reporting extension with the specific concepts, the user interface, and the basic blocks of reports.

What you need for this book

You only need a KNIME-compatible operating system, which is either a modern Linux, Mac OS X (10.6 or above), or Windows XP or above. The Java runtime is bundled with KNIME, and the first chapter describes how you can download and install KNIME. For this reason, you will need Internet connection too.

Who this book is for

This book is designed to give a good start to the data scientists who are not familiar with KNIME yet. Others, who are not familiar with programming, but need to load and transform their data in an intuitive way might also find this book useful.

Conventions

In this book, you will find a number of styles of text that distinguish among different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: " In the first case, you have not much control about the details, for example, a `Pattern` object will be created for each call of the facade methods delegating to the `Pattern` class "

A block of code is set as follows:

```
// system imports
// Your custom imports:
import java.util.regex.*;
// system variables
// Your custom variables:
Pattern tuplePattern = Pattern.compile("\\((\\d+),\\s*(\\d+)\\)");
// expression start
```

```
// Enter your code here:
if (c_edge != null) {
  Matcher m = tuplePattern.matcher(c_edge);
  if (m.matches()) {
    out_edge = m.replaceFirst("($2, $1)");
  } else {
    out_edge = "NA";
  }
} else {
  out_edge = null;
}
// expression end
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
// system imports
// Your custom imports:
import java.util.regex.*;
// system variables
// Your custom variables:
Pattern tuplePattern = Pattern.compile("\\((\\d+),\\s*(\\d+)\\)");
// expression start
// Enter your code here:
if (c_edge != null) {
  Matcher m = tuplePattern.matcher(c_edge);
  if (m.matches()) {
    out_edge = m.replaceFirst("($2, $1)");
  } else {
    out_edge = "NA";
  }
} else {
  out_edge = null;
}
// expression end
```

Any command-line input or output is written as follows:

```
$ tar -xvzf knime_2.8.0.linux.gtk.x86_64.tar.gz -C /path/to/extract
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Eclipse's main window is the **workbench**".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic in which you have expertise, and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Installing and Using KNIME

In this chapter, we will go through the installation of KNIME, add some useful extensions, customize the settings, and find out how to use it for basic tasks. You will also be familiarized with the terminology of KNIME, so there's no misunderstanding in the later chapters.

As always, it is a good idea to read the manual of the software you get. You will find a short introduction on KNIME in the file, `quickstart.pdf`, present in the installation folder. The topics we will cover in the chapter are as follows:

- Installation of KNIME on different platforms
- Terms used in KNIME
- Introduction to the KNIME user interface

Few words about KNIME

KNIME is an open source (GNU GPL available at `http://www.gnu.org/licenses/gpl.html`) data analytics platform with a large set of building blocks and third-party tools. You can use it from loading your data to a final report or to predict new values using a previously found model.

KNIME is available in four flavors: Desktop/Professional, Team Space, Server, and Cluster Execution. Only the Desktop version is open source; with a Professional subscription, you will get support for it, and also support the future development of KNIME. We will cover only the open source version. There is also an SDK version for free, but it is intended for use by node developers. Most probably, you will not need it yet.

At the time of writing this book, KNIME Desktop 2.8.0 was the latest version available; all the information presented in this book is based on that version.

Installing KNIME

KNIME is supported by various operating systems on 32-bit and 64-bit x86 Intel-architecture-based platforms. These operating systems are: Windows (from XP to Windows 8 at the time of writing this book) and Linux (most modern Linux operating systems work well with KNIME, Mac OS X (10.6 and above); you can check the list of supported platforms for details at: `http://www.eclipse.org/eclipse/development/readme_eclipse_3.7.1.html`. It also supports Java 7 on Windows and Linux, so extensions requiring Java 7 can be used too. Unfortunately under Mac OS X, there were some problems with Java 7. So on Mac OS X, the recommended version is Java 6.

There are two ways to install KNIME: an easier way is to unpack the archive you can download from their site, and a bit more complicated way is to install KNIME to an existing Eclipse installation as a plugin. Both have use cases, but the general recommendation is to install it from an archive.

Installation using the archive

We assume you are using the open source version of KNIME, which can be downloaded from the following address (always download the latest version):

`http://www.knime.org/knime-desktop-sdk-download`

It is not necessary to subscribe to the newsletters, but if you have not done it yet, it might be worth doing it. Some of the newsletters also contain tips for KNIME usage. This is quite infrequent, usually one per month.

The supported operating system versions are 32-bit and 64-bit for Linux and Windows, and 64-bit for Mac OS X.

KNIME for Windows

KNIME is available in an executable file for Windows (in a 7-zip compressed format). You can execute it as a regular user (unless your network administrator blacklists running executable files that are downloaded from the Internet); just double-click on it and in the window that appears, select the destination folder.

On an older version of Windows (7 and older), there is a limitation to the path length; it cannot be longer than 260 characters. KNIME and some extensions can get close to this limit, so it is recommended to install it to a short path. Installing it to Program Files is not recommended.

You do not have to specify the folder name (such as knime), as a folder with the name `knime_KNIME version` (in our case `knime_2.8.0`) will be created at the destination address, and it will contain the whole installation. You can have multiple versions installed.

You can start KNIME GUI with the `knime.exe` executable file from that folder. You can create a shortcut of it on your desktop using the right-click menu by navigating to **Send to | Desktop (create shortcut)**. On its first start, KNIME might ask for permissions to connect to the Internet. This may require administrator rights, but it is usually a good idea to change the firewall settings to let KNIME through.

KNIME for Linux

This file is just a simple `tar.gz` archive. You can unzip it using a command similar to the one shown as follows:

```
$ tar -xvzf knime_2.8.0.linux.gtk.x86_64.tar.gz -C /path/to/extract
```

Alternatively, you can use your favorite archive-handling tool to achieve similar results. The executable you need is named `knime`. Your window manager's manual might help you create application launchers for this executable if you prefer to have one.

KNIME for Mac OS X

You should drag the `dmg` file to the **Applications** place, and if you have Java installed, it should just work. The executable to start is called `knime.app` from the command line, `knime.app/Contents/MacOS/knime`.

Troubleshooting

If you have problems installing KNIME, maybe others also had similar problems; please check the FAQ page of KNIME at `http://tech.knime.org/faq` first. If it does not solve your problem, you should search the forum at `http://tech.knime.org/forum`; if even that fails to help, ask the experts there.

KNIME terminologies

It is important to share your thoughts and problems using the same terms. This makes it easier to reach your goal, and others will appreciate if it is easy to understand. This section will introduce the main concepts of KNIME.

Organizing your work

In KNIME, you store your files in a **workspace**. When KNIME starts, you can specify which workspace you want to use. The workspaces are not just for files; they also contain settings and logs. It might be a good idea to set up an empty workspace, and instead of customizing a new one each time, you start a new project; you just copy (extract) it to the place you want to use, and open it with KNIME (or switch to it).

The workspace can contain **workflow group**s (sometimes referred to as **workflow set**) or **workflow**s. The groups are like folders in a filesystem that can help organize your workflows. Workflows might be your *programs* and *processes* that describe the steps which should be applied to load, analyze, visualize, or transform the data you have, something like an execution plan. Workflows contain the executable parts, which can be edited using the **workflow editor**, which in turn is similar to a canvas. Both the groups and the workflows might have metadata associated with them, such as the creation date, author, or comments (even the workspace can contain such information).

Workflows might contain *nodes*, *meta nodes*, *connections*, *workflow variables* (or just *flow variables*), *workflow credentials*, and *annotations* besides the previously introduced metadata.

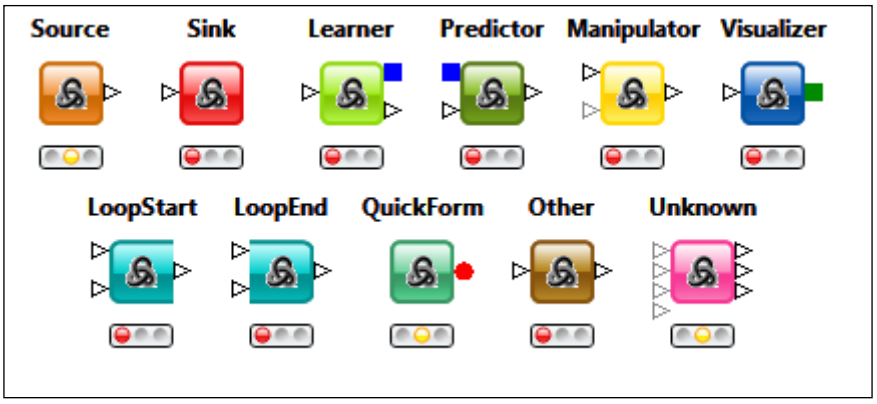
Workflow credentials is the place where you can store your *login name* and *password* for different connections. These are kept safe, but you can access them easily.

It is safe to share a workflow if you use only the workflow credentials for sensitive information (although the user name will be saved).

Nodes

Each node has a type, which identifies the algorithm associated with the node. You can think of the type as a template; it specifies how to execute for different inputs and parameters, and what should be the result. The nodes are similar to functions (or operators) in programs.

The node types are organized according to the following general types, which specify the color and the shape of the node for easier understanding of workflows. The general types are shown in the following image:



Example representation of different general types of nodes

The nodes are organized in categories; this way, it is easier to find them.

Each node has a *node documentation* that describes what can be achieved using that type of node, possibly use cases or tips. It also contains information about *parameters* and possible *input ports* and *output ports*. (Sometimes the last two are called *inports* and *outports*, or even *in-ports* and *out-ports*.)

Parameters are usually single values (for example, *filename*, *column name*, *text*, *number*, *date*, and so on) associated with an identifier; although, having an array of texts is also possible. These are the settings that influence the execution of a node. There are other things that can modify the results, such as workflow variables or any other state observable from KNIME.

Node lifecycle

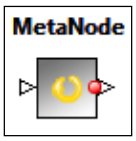
Nodes can have any of the following states:

- Misconfigured (also called IDLE)
- Configured
- Queued for execution
- Running
- Executed

There are possible warnings in most of the states, which might be important; you can read them by moving the mouse pointer over the triangle sign.

Meta nodes

Meta nodes look like normal nodes at first sight, although they contain other nodes (or meta nodes) inside them. The associated context of the node might give options for special execution. Usually they help to keep your workflow organized and less scary at first sight.



A user-defined meta node

Ports

The ports are where data in some form flows through from one node to another. The most common port type is the **data table**. These are represented by white triangles. The input ports (where data is expected to get into) are on the left-hand side of the nodes, but the output ports (where the created data comes out) are on the right-hand side of the nodes. You cannot mix and match the different kinds of ports. It is also not allowed to connect a node's output to its input or create circles in the graph of nodes; you have to create a loop if you want to achieve something similar to that.

Currently, all ports in the standard KNIME distribution are presenting the results only when they are ready; although the infrastructure already allows other strategies, such as streaming, where you can view partial results too.

The ports might contain information about the data even if their nodes are not yet executed.

Data tables

These are the most common form of port types. It is similar to an Excel sheet or a data table in the database. Sometimes these are named example set or data frame.

Each data table has a *name*, a *structure* (or schema, a table specification), and possibly *properties*. The structure describes the data present in the table by storing some properties about the *columns*. In other contexts, columns may be called attributes, variables, or features.

A column can only contain data of a single type (but the types form a hierarchy from the top and can be of any type). Each column has a *type*, a *name*, and a *position* within the table. Besides these, they might also contain further information, for example, statistics about the contained values or color/shape information for visual representation. There is always something in the data tables that looks like a column, even if it is not really a column. This is where the identifiers for the *rows* are held, that is, the *row keys*.

There can be multiple rows in the table, just like in most of the other data handling software (similar to observations or records). The row keys are unique (textual) identifiers within the table. They have multiple roles besides that; for example, usually row keys are the labels when showing the data, so always try to find user-friendly identifiers for the rows.

At the intersection of rows and columns are the (data) *cells*, similar to the data found in Excel sheets or in database tables (whereas in other contexts, it might refer to the data similar to values or fields). There is a special cell that represents the *missing values*.

The missing value is usually represented as a question mark (?).

If you have to represent more information about the missing data, you should consider adding a new column for each column, where this requirement is present, and add that information; however, in the original column, you just declare it as missing.

There are multiple cell types in KNIME, and the following table contains the most important ones:

Cell type	Symbol	Remarks
Int cell	I	This represents integral numbers in the range from -2^{31} to $2^{31}-1$ (approximately 2E9).
Long cell	L	This represents larger integral numbers, and their range is from -2^{63} to $2^{63}-1$ (approximately 9E18).
Double cell	D	This represents real numbers with double (64 bit) floating point precision.
String cell	S	This represents unstructured textual information.

Cell type	Symbol	Remarks
Date and time cell	calendar & clock	With these cells, you can store either date or time.
Boolean cell	B	This represents logical values from the Boolean algebra (true or false); note that you cannot exclude the missing value.
Xml cell	XML	This cell is ideal for structured data.
Set cell	{…}	This cell can contain multiple cells (so a collection cell type) of the same type (no duplication or order of values are preserved).
List cell	{…}	This is also a collection cell type, but this keeps the order and does not filter out the duplicates.
Unknown type cell	?	When you have different type of cells in a column (or in a collection cell), this is the generic cell type used.

There are other cell types, for example, the ones for chemical data structures (SMILES, CDK, and so on), for images (SVG cell, PNG cell, and so on), or for documents. This is extensible, so the other extension can define custom data cell types.

Note that any data cell type can contain the missing value.

Port view

The port view allows you to get information about the content of the port. Complete content is available only after the node is executed, but usually some information is available even before that. This is very handy when you are constructing the workflow. You can check the structure of the data even if you will usually use *node view* in the later stages of data exploration during workflow construction.

Flow variables

Workflows can contain flow variables, which can act as a loop counter, a column name, or even an expression for a node parameter. These are not constants, but you can introduce them to the workspace level as well.

This is a powerful feature; once you master it, you can create workflows you thought were impossible to create using KNIME. A typical use case for them is to assign roles to different columns (by assigning the column names to the role name as a flow variable) and use this information for node configurations. If your workflow has some important parameters that should be adjusted or set before each execution (for example a file name), this is an ideal option to provide these to the user; use the flow variables instead of a preset value that is hard to find. As the automatic generation of figures gets more support, the flow variables will find use there too.

Iterating a range of values or files in a folder should also be done using flow variables.

Node views

Nodes can also have *node views* associated with them. These help to visualize your data or a model, show the node's internal state, or select a subset of the data using the **HiLite** feature. An important feature exists that a node's views can be opened multiple times. This allows us to compare different options of visualization without taking screenshots or having to remember what was it like, and how you reached that state. You can export these views to image files.

HiLite

The HiLite feature of KNIME is quite unique. Its purpose is to help identify a group of data that is important or interesting for some reason. This is related to the node views, as this selection is only visible in nodes with node views (for example, it is not available in port views). Support for data *high lighting* is optional, because not all views support this feature.

The HiLite selection data is based on row keys, and this information can be lost when the row keys change. For this reason, some of the nonview nodes also have an option to keep this information propagated to the adjacent nodes. On the other hand, when the row keys remain the same, the marks in different views point to the same data rows.

It is very important that the HiLite selection is only visible in a well-connected subgraph of workflow. It can also be available for non-executed nodes (for example, the *HiLite Collector node*).

The HiLite information is *not* saved in the workflow, so you should use the *HiLite filter node* once you are satisfied with your selection to save that state, and you can reset that HiLite later.

Eclipse concepts

Because KNIME is based on the Eclipse platform (`http://eclipse.org`), it inherits some of its features too. One of them is the *workspace model* with projects (workflows in case of KNIME), and another important one is *modularity*. You can extend KNIME's functionality using plugins and features; sometimes these are named KNIME **extensions**. The extensions are distributed through *update sites*, which allow you to install updates or install new software from a local folder, a zip file, or an Internet location.

The help system, the update mechanism (with proxy settings), or the file search feature are also provided by Eclipse. Eclipse's main window is the **workbench**. The most typical features are the **perspectives** and the **views**. Perspectives are about how the parts of the UI are arranged, while these independently configurable parts are the views. These views have nothing to do with node views or port views. The Eclipse/KNIME views can be detached, closed, moved around, minimized, or maximized within the window. Usually each view can have at most one instance visible (the **Console** view is an exception). KNIME does not support alternative perspectives (arrangements of views), so it is not important for you; however, you can still reset it to its original state.

It might be important to know that Eclipse keeps the contents of files and folders in a special form. If you generate files, you should refresh the content to load it from the filesystem. You can do this from the context menu, but it can also be automated if you prefer that option.

Preferences

The preferences are associated with the workspace you use. This is where most of the Eclipse and KNIME settings are to be specified. The node parameters are stored in the workflows (which are also within the workspace), and these parameters are not considered to be preferences.

Logging

KNIME has something to tell you about almost every action. Usually, you do not care to read these logs, you do not need to do so. For this reason, KNIME dispatches these messages using different channels. There is a file in the workplace that collects all the messages by default with considerable details. There is even a KNIME/Eclipse view named **Console**, which contains only the most important details initially.

User interface

So far, you got familiar with the concepts of KNIME and also installed it. Let's run it!

Getting started

When you start the program, the first dialog asks for the location of the workspace you want to use. If the location does not exist, it will be created.

After this, a splash screen will inform you about the progress of the start, and bring you to the welcome screen.

In the background, your firewall might notify you that this program wants to connect to other computers. This is normal; it loads tips from the Internet and tests whether other services (for example, the public repository of KNIME workflows) are available or not. You can allow this if you have permission to do so, but unless you want to connect to other servers, you do not have to give that permission.

The welcome screen shows two main options: one for initializing the workbench for first use, and the other is to install new extensions.

Before we select either of them, we will introduce the most important preferences, because configuring before the first use is always useful.

Setting preferences

Navigate to the **Preferences...** menu item under **File** | **Preferences...** to gain access to the preferences dialog. In the **General** section, you will see an option to enable **Show heap status**. It is useful, because it can help you optimize the memory settings for KNIME. I suggest you to turn it on. It will be visible in the lower-right corner of the status bar.

KNIME

You can set some KNIME-related options in the preferences of the **KNIME** category.

The **KNIME GUI** subcategory contains confirmation, **Console** logging, workflow editor grid options, and some text-related options.

If you want to connect to databases, you should find a driver for your database, and register it by navigating to **KNIME** | **Database Driver**. There, you can add the archive file, and later, you will be able to use them in database connections.

Database drivers

You can find JDBC database drivers on your database provider's homepage, but you can also try the JDBC database: `http://www.databasedrivers.com/jdbc/`

With **Preferred Renderers** you can set the default renderers for the columns. This options is especially useful if you are working with chemical structures.

The main **KNIME** preference page contains the file logging detail settings, the parallelism option, and the path to the temporary files.

Other preferences

To set up the proxy, you should navigate to **General** | **Network Connections**.

In the **General** | **Keys** page, you can redefine the key bindings for KNIME commands. So, you can use the shortcuts with which you are familiar or comfortable on your keyboard.

General | **Web Browser** and the **Help** pages are especially useful when you have problems displaying help, or you want to browse local help in your browser.

You can also set some update sites by navigating to **Install/Update** | **Available Software Sites**, but usually that is also done by navigating to **Help** | **Install New Software...**.

You can uninstall extensions by navigating to **Help** | **About KNIME** behind the **Installation Details** button's dialog. The **Installed Software** tab contains the extensions; you can uninstall them with a button.

Installing extensions

For installing extensions you need some update sites. You already have the default KNIME options, which contain some useful extensions. There are community nodes that also add helpful functionality to KNIME. The stable update site is `http://tech.knime.org/update/community-contributions/2.8`, while nightly builds are available at `http://tech.knime.org/update/community-contributions/nightly`.

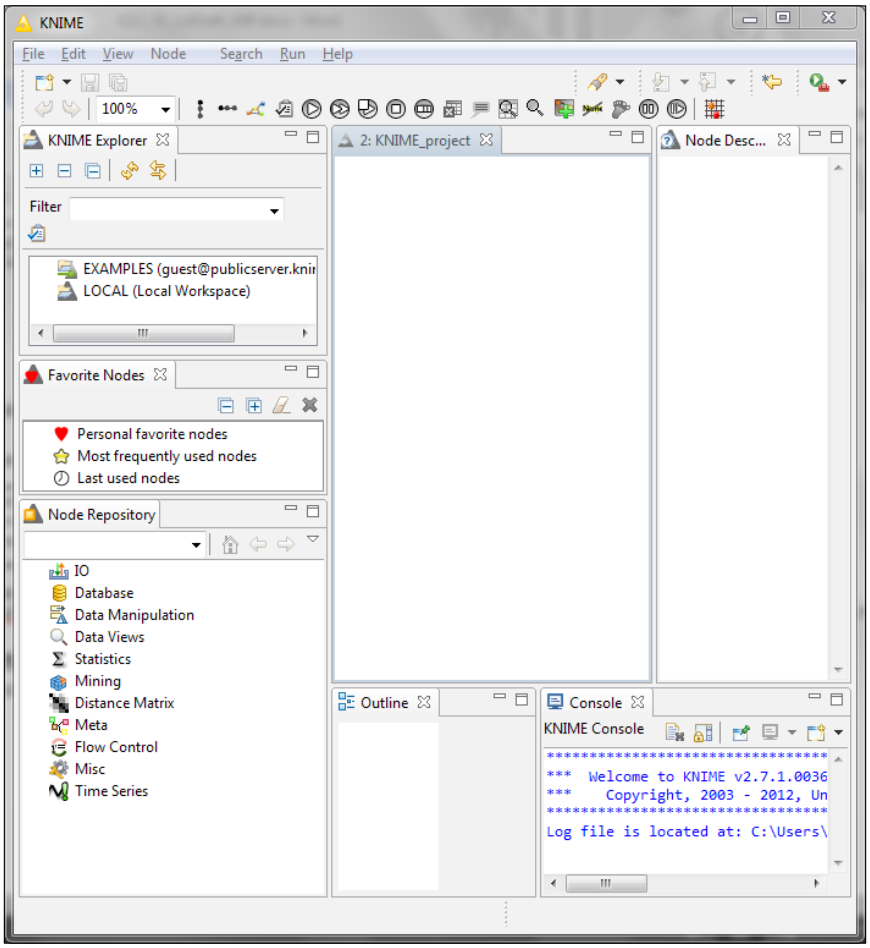
To add update sites, navigate to **Help** | **Install New Software...**. Once you have selected an update site, it will download its summary so you can select which extensions (features) you want to install. These features have short descriptions, so you can have an idea what functionality it will offer after installation. Once you have selected what you want to install from the update site, you should click **Next**.

The wizard's next page gives some details and summaries about the selected features.

On the next page, you can check the licenses and accept them if you are OK with them. After clicking **Finish**, the installation starts. During the installation, you might be asked to check whether you really want to install extensions with unsigned content, or you want to accept a signing key. Once it is ready, you will be asked to restart your workbench. After restarting it, you can use the features that were installed; however, sometimes there are some preferences to be set before using them.

Workbench

So far, we have set up the work environment and installed some extensions. Now let's select the large button named **Open KNIME Workbench**.



The initial workbench

The menu bar is similar to any other menu bar, just like the toolbars and the status bar. We will cover the menu bar and the toolbar in detail.

The **KNIME Explorer** view can be used to handle your workflows, workflow groups, or connect to KNIME servers. The **Favorite Nodes** view contains the favorite, last used, and most used nodes as a shortcut. You can specify the maximum number of items that should be there.

You should play with the view controls a bit more and get familiar with their usage.

Node Repository is one of the most important views. It contains nodes organized in categories. The search box is really helpful when it comes to the workflow design, and if you remember a part of the name but not its category. You will use this feature quite often.

The **Outline** view gives an overview on what is in the current editor window; it can also help navigating if the window is too large.

It is considered bad practice to have a single, huge workflow for your task. Using meta nodes, you can have more compact parts in every level.

The **Console** view contains messages—initially only the important ones.

The **Node Description** tab provides you with help information for the selected node. Information on how you should use it, what are the parameters, what should be its input, what is its output, and what kind of views are available are answered in that tab. When you select a category in the **Node Repository** view, the contents of the category will be displayed.

And finally, the central area of the window is for the *workflow editor*. A workflow named **KNIME_project** was created. Now, you can start working on it. Try adding the **File Reader** node from the **IO | Read** category in **Node Repository**. Drag it from the repository to the workflow or just double-click it in the repository, move it around, add another, delete it using the context menu, and that would be a good start.

The **Undo** (*Ctrl + Z*) and **Redo** (*Ctrl + Y*) commands from the **Edit** or the context menu (or from the toolbar: curved left and right arrows) can help you go back to the previous editing state.

Workflow handling

To create a workflow group, open the context menu of the **LOCAL (Local Workspace)** item in the **KNIME Explorer** view and select **New Workflow Group...** from the menu. Specify the name of the workflow group and where it should be created (once you have more groups, you can create groups inside those too). Creating a workflow can also be done using the **New Workflow...** command. These commands are also available from the **File | New...** (*Ctrl + N*) dialog.

The key bindings are not always easy to remember because there are many of them; for more information and help about them, navigate to the **Help | Key Assist...** menu item or use *Ctrl + Shift + L*.

To load a workflow, first you have to make it available locally. There are many options to do that. You can import it to the workspace using the **File | Import KNIME workflow...** dialog (also available from the context menu).

There is a file named `ExampleFlow.zip` in the installation folder; you can use that.

The **Example Flow** workflow loads the iris dataset (do not reset that node), colors the rows according to their class label, and visualizes the data in three different ways.

Another option is to download a workflow from the KNIME Server. Fortunately, the public KNIME Server is available for guests too. First you have to log in using the context menu. Select the only available option, **Login**. Once the catalog has been loaded, you can browse it similar to what you can do with the local workspace. But you cannot open the workflow from there. You have to select the one you want to import and copy it (in the context menu, use **Copy** or press *Ctrl + C*). Once you have the right place in the local workspace, insert the workflow (in the context menu use **Paste**, or press *Ctrl + V*).

The metadata information can be handy if you want to know when it was created, who the author is, or what did someone comment. The comment information is especially handy if you want to choose the workflow you want to download. To get (or set for local workflows) this information, the context menu's **Show Meta Information** (or **Edit Meta Information...**) command should be used.

Describe your dependencies

If you mention the prerequisites to your workflow, it will help the next user (who may be the future you) to set up things properly.

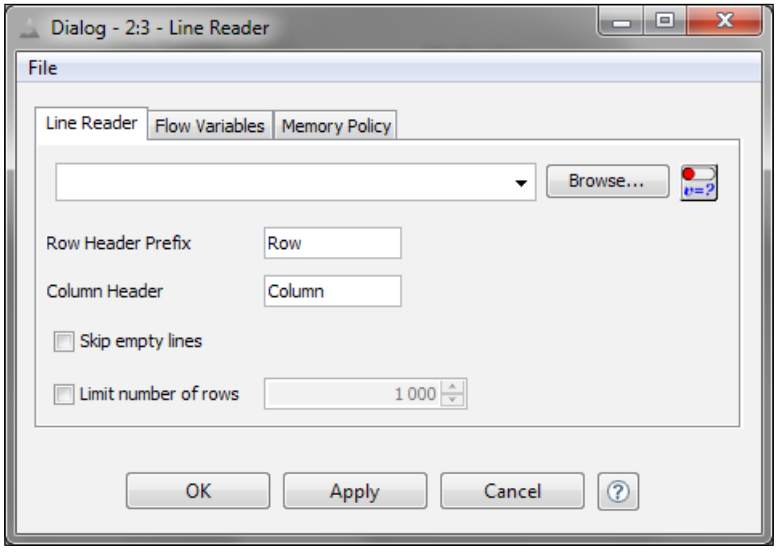
In loaded workflows, sometimes there are yellow notes about the structure of the workflow to grab your attention for customization options, and others. You can create your own notes from the context menu of the workflow editor using the **New Workflow Annotation** menu item. You can close the workflow by closing its editor.

The context menu gives options to **Rename...** (*F2*) (only available for closed workflows), **Delete...** (*Delete*), **Copy** (*Ctrl* + *C*), **Paste** (*Ctrl* + *V*), or **Cut** (*Ctrl* + *X*)—or just move using dragging—workflows or workflow groups.

The `quickstart.pdf` file describes how you can export workflows to share them with other users. The web guide for this is available at: `http://tech.knime.org/workbench#export`

Node controls

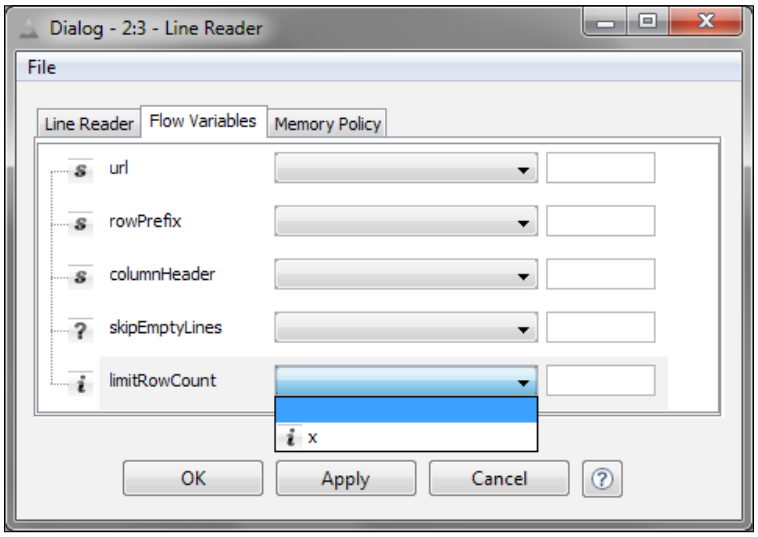
Once you have nodes in the editor, you want to configure it. To do that, you should double-click it, select it from the context menu or the **Node** menu using the **Configure...** command, or use the toolbar's checklist icon (also accessible by pressing *F6*). This opens a configuration dialog (**Line Reader** node), as shown in the following screenshot:



Example configuration dialog

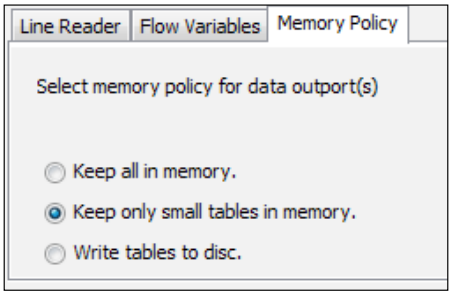
This way you can set the parameters of the node. There can be various controls, usually with helpful tooltips; you can open them in a side window, and add the node description too. You might wonder what should that **v=?** button do. It opens up the variable settings. For example, you can use the filename in subsequent nodes as a flow variable, or substitute it with a flow variable, if that is what you need.

The configurations are organized in tabs. The last two tabs are present in all the configuration dialogs. The **Flow Variables** tab allows you to assign flow variables to the parameters as values, as shown in the following screenshot:



The Flow Variables tab

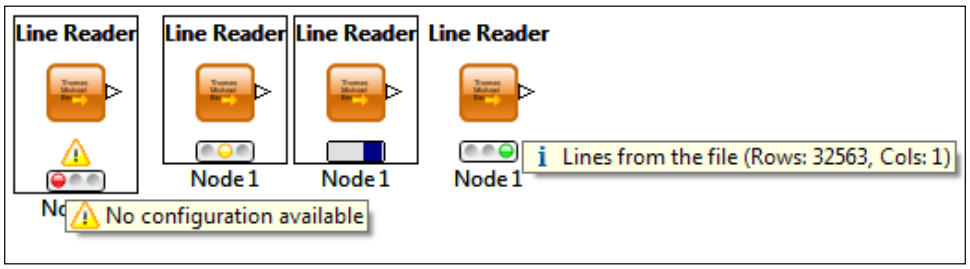
The **Memory Policy** tab is seldom needed; you can specify how the data should be handled within KNIME during execution of the node, as shown in the following screenshot:



The Memory Policy tab

It really helps to identify the nodes or their purpose if you give them meaningful names. To change the name, click on a previously selected node or press *F2*. If you want more detailed information, you might consider adding a workflow annotation around it. Alternatively, you might want to add a node description to it by navigating to the context menu item **Edit Node Description...**, or the **Node** menu **Edit Node Name and Description...** (*Alt* + *F2*), or by clicking the toolbar's yellow speech balloon. This information will be the tooltip of the node.

If you find the names distracting or if they are the default name, you can hide or enable them by navigating to **Node | Hide Node Names**, by pressing *Ctrl* + *Alt* + *Q* or the stroked through text on the toolbar.



The way from not configured to configured, and then the executing and executed states.

We want to execute the node to get the results. To achieve this, select the context menu or the **Node** menu, and select **Execute** (*F7*). On the toolbar, this is the *play* button (a white triangle on green circle). You can also schedule execution to show the first view after that (*Shift* + *F10*). You can change your mind and try to stop the execution before it is finished. For this purpose, navigate to **Node | Cancel Execution** (*F9*) of the selected nodes, or navigate to **Node | Cancel All Execution** (*Shift* + *F9*).

There might be warnings or errors even after the execution; you will be notified about those.

If the execution finishes successfully, you can check the ports by selecting one of them from the context menu; alternatively, if you want to check the first output port, navigate to **Node | Open First Out-Port View** (*Shift* + *F6*, a magnifier over a table on the toolbar). Checking views is a good idea too (it can be selected from the context menu or via **Node | Open First View**, *F10*, a magnifier on the toolbar). The node views also have some common parts: the **File** and the **HiLite** menus.

If you make changes to the configuration, your node will be reset to the configured state; it can also be achieved using **Node** or the context menu's **Reset** (*F8*) command (or the toolbar's x-table button). The reset will not delete the previously set parameters.

To connect a node's output port to another node's input port, just drag the output port to the input port; when the mouse button is released they will be connected (assuming the ports are compatible and would not create cycle in the graph of nodes). From one output port, you can connect to as many input nodes as you want (to same nodes too), but the input ports can only handle one port at the most.

There are arrangement commands available on the toolbar (horizontal, vertical, and auto layout), and you can also configure the node snapping grid properties by navigating to **Node | Grid Settings...** (*Ctrl* + *Alt* + *Shift* + *X*) from the toolbar—a grid.

HiLite

As we mentioned previously, HiLite is a view-related feature of KNIME, which allows selecting certain set of rows and making it visible across different rows. The Example Flow is a good start to get familiar with this concept and see it in action. As you can see, there are four visual type nodes available, the **Color Manager**, **Scatter Plot**, **Parallel Coordinates**, and **Interactive table**. Please open a view for the last three nodes, and also execute them in the same order.

The interactive table node shows data with different colors for different flowers. Select the first **Iris-versicolor** row, **51**. Now from the **HiLite** menu, select **HiLite selected** (also available from the context menu in this view). As you can see, a point and a path has already been highlighted on the other two views—those representing the row **51**. If you try, you can highlight another row from the **Interactive table** view; you can select some dots from the scatter plot or paths from the parallel coordinates. Highlighting them can be done similar to what you did in the first view. You also have the option to selectively unhighlight (**UnHiLite Selected**) or unhighlight all (**Clear HiLite**). You can also hide or keep only the highlighted rows (in the view, the port content will not be changed) using the **HiLite | Filter** menu items.

To store the HiLite information, you should add **HiLite Filter** (for example, add it to the **Color Manager** node), execute them, and save the workflow. With the **Interactive HiLite Collector** node, you can add custom information to the currently highlighted rows, so that later you can identify multiple subsets (if you check the **New Column** box before clicking on **Apply**). Do not forget to execute the node, and later save the workflow once you are satisfied with your selection.

Variable flows

When you bring your mouse cursor to the left and upper-right corner of the nodes (a bit outside of it), you will get a different tooltip—**Variable Inport** and **Variable Outport (Variables Connection)** respectively. Something useful is hidden there. Select a node, and from the context menu, select **Show Flow Variable Ports**. This way two circles will appear filled with the color red. You can connect them to the other node's input/output flow ports. These connections are red. This way you can make sure the proper set of variables will be available at the right time (circular dependencies are not allowed this way). The loops also use the workflow variables, and there are multiple nodes to create these or change them. You seldom need these connections as flow variables are propagated through normal connections.

You can also specify workflow variables from the context menu of the workflow (**Workflow Variables...**), or by using the QuickForm nodes.

Meta nodes

We mentioned that the meta nodes are useful for encapsulating the parts of the workflow and to hide the distracting details. The `quickstart.pdf` file gives a nice introduction to meta nodes; you can find the content on the web too at the link `http://tech.knime.org/metanodes`.

An unmentioned option to create new meta nodes is by selecting a closed subset of non-executed nodes or meta nodes and invoking the **Collapse into Meta Node** action from the context menu. The opposite process (bringing the contents of the meta node to the current level) is also possible with the **Expand Meta Node** context menu item.

Opening a meta node is possible by double-clicking on it or selecting the **Open Meta Node** context menu item. Both ways, another workflow editor tab will appear, where you can continue the workflow design.

Workflow lifecycle

Once you have a workflow, you might want to save the changes you made and the computed data and models. That is really easy; navigate to **File** | **Save** (*Ctrl* + *S*) or use the toolbar's disc icon.

You cannot save workflows with executing nodes, so you have to save them before or later, else you have to stop the execution.

Sometimes you want to execute the whole workflow. To do that, you can use the toolbar's **Execute all executable nodes** button (a fast forward icon with a green circle background, *Shift* + *F8*) or the **Node** | **Execute All** menu item.

Batch processing

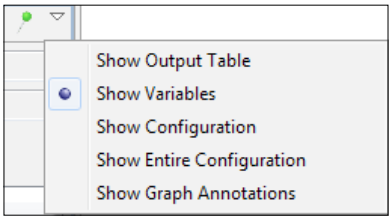
To process workflows from the command line (or from other program), the KNIME FAQ gives a good description at the following link: `http://tech.knime.org/faq#q12`

If there are multiple entry points to your workflow, it can be boring to reset all those nodes one by one, but the **Reset** command from the context menu of **KNIME Explorer** will reset all the nodes in the selected workflow.

Other views

The **Server Workflow Projects** view shows only the workflows (and groups) available on servers, but the **Workflow Projects** view shows only the local ones. If you do not need server workflows, this might be a better choice than the **KNIME Explorer** view, as this is more compact.

KNIME Node Monitor (**View** | **Other...** | **KNIME Views**) view gives you information about the selected item's state and other parameters. I think you will find this useful, especially if you explore the dropdown menu from the white triangle:



KNIME Node Monitor's possible contents

Summary

In this chapter, we have installed KNIME, set it up for its first usage, configured it, and installed a few extensions. We also went through the most important concepts you will use. We started using the workflow editor and executed our first workflow. Now it is time for you to check some of the example workflows from the KNIME public server and try to execute and modify them.

2
Data Preprocessing

Data preprocessing usually takes a lot of time to set up, because you have to take care of lot of different formats or sources. In this chapter, we will introduce the basic options to not only read and generate data but also to reshape them. Changing values is also a common task, and we will cover that too.

It is a good practice to keep checking certain constraints especially if you have volatile input sources. This is also important in KNIME. Finally, we will go through an example of workflow from import to preprocessing. In this chapter, we will cover the following topics:

- Data import
    - From database
    - From files
    - From web services
- Regular expressions
- Transforming tables
- Transforming values
- Generating data
- Constraints
- Case studies

Importing data

Your data can be from multiple sources, such as databases, Internet/intranet, or files. This section will give a short introduction to the various options.

Importing data from a database

In this section, we will use the Java DB (`http://www.oracle.com/technetwork/java/javadb/index.html`) to create a local database because it is supported by Oracle, bundled with JDKs, cross-platform, and easy to set up. The database we use is described on eclipse's **BIRT Sample Database** page (`http://www.eclipse.org/birt/phoenix/db/#schema`).

Starting Java DB

Once you have Java DB installed (unzipped the binary distribution from Derby (`http://db.apache.org/derby/derby_downloads.html`) or located your JDK), you should also download the `BirtSample.jar` file from this book's website (originally from `http://mirror-fpt-telecom.fpt.net/eclipse/birt/update-site/3.7-interim/plugins/org.eclipse.birt.report.data.oda.sampledb_3.7.2.v20120213.jar.pack.gz`). Unzip the content to the database server's install folder.

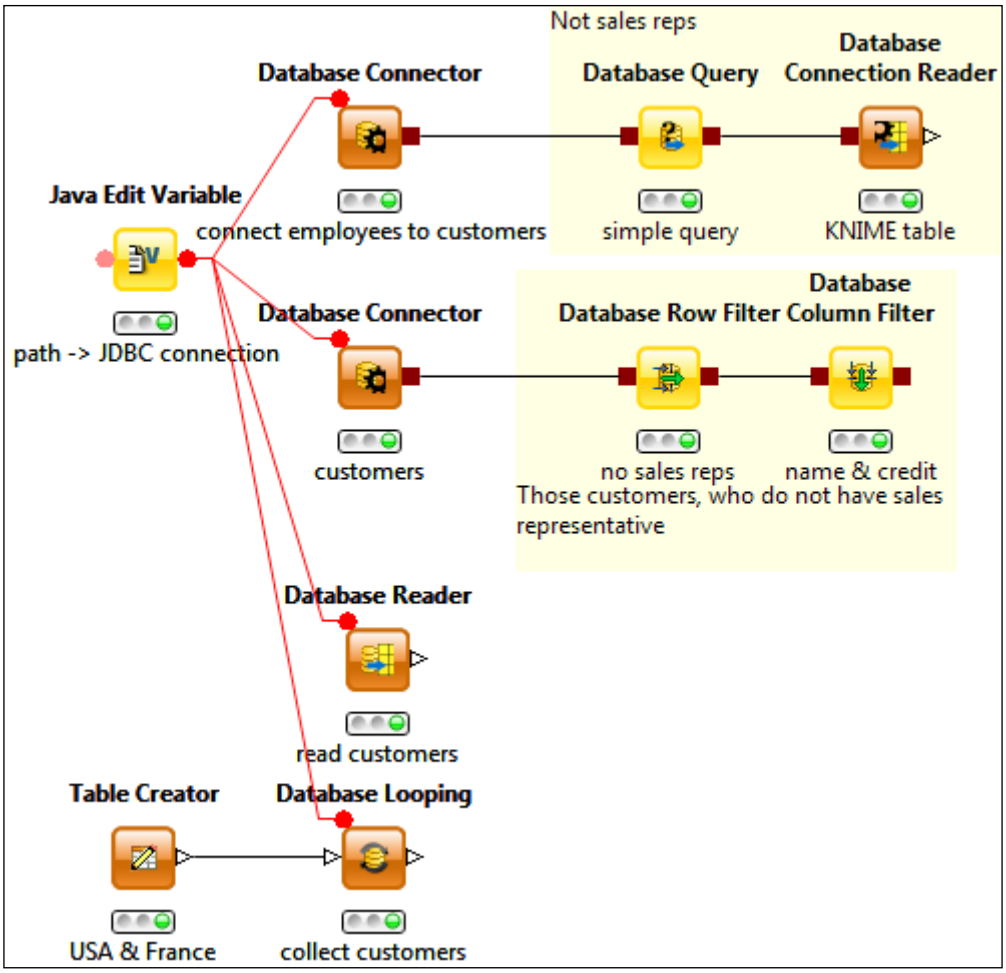
You should start a terminal from the database server's home folder, using the following command:

```
bin/startNetworkServer
```

You can stop it with the `bin/stopNetworkServer` command.

Locate the database server's `lib/derbyclient.jar` file. You should install this driver as described in the previous chapter (**File** | **Preferences** | **KNIME** | **Database Driver**).

You can import the `DatabaseConnection.zip` file, downloaded from this book's website, as a KNIME workflow. This time, we were not using workflow credentials as it would always be asked for on load, and it might be hard to remember the `ClassicModels` password.



The previous screenshot reads different tables and filter some of the results. The **Java Edit Variable** node provides the JDBC connection string as a flow variable.

There is a workflow variable named database location (default value: `BirtSample`), in case you want to specify an absolute path to the database you want to use. The **Java Edit Variable** node appends this path to the default local database connection, so you can use the Derby JDBC connection variable in the subsequent nodes. You should start with executing this node to configure the other nodes.

The **Database Connector** node can connect to the database and give a stub for further processing (you can inspect it using the port viewer, though, once you execute).

The **Database Query** can be used to express complex conditions in the table. Please be careful. You should name the #table#, like we did in the following query:

```
SELECT * FROM #table# customerTable where customernumber < 300 or
customernumber is null
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

If you have simpler (single column) conditions, you can also use the **Database Row Filter** node. Removal of a few columns (projection) can be performed with the **Database Column Filter** node.

If you want to process or visualize the data in KNIME, you have to convert the database connection port type to regular data tables using the **Database Connection Reader** node. If you do not need post-processing of the database tables, you can simply specify the connection and the query with the **Database Reader** node.

An interesting option to read data is by using the **Database Looping** node. It can read the values from one of the input table's columns and select only the values that match a subset of the column for one of the database columns' values.

Exercise

Check what happens if you uncheck the **Aggregate by row** option or increase the **No of Values per Query** parameter.

You also have the option to modify the database, such as deleting rows, updating certain rows, creating tables, and appending records. For details, check the **Database Delete**, **Database Update**, and **Database Writer** nodes. While replacing/creating a table for an existing database, the connection can be performed using the **Database Connection Writer** node.

Importing data from tabular files

This time, for example, we will load a simple comma-separated file. For this purpose, you can use the **File Reader** node and the following link:

```
http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.
data
```

KNIME will automatically set the parameters, although you have to specify the column names (the `http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.names` file gives a description of the dataset).

In the configuration dialog, you can refine the columns in the preview area by clicking on its header.

Naturally, you can open the local files too, if you specify the URL using the **Browse...** button.

If you have the data in the Excel format, you might need the **KNIME XLS Support** extension from the standard KNIME update site. This way, you will be able to read (with the **XLS Reader** node) and also write the `xls` files (with the **XLS Writer** node).

The extension can also read the `xlsx` files, but cannot write them.

Just like the **File Reader** node, **XLS Reader** can load the files from the Internet too. (If you have the data in the `ods` format, you have to convert/export it to either the `xls`(`x`) or the `csv` file to be able to load from KNIME.)

The **CSV Reader** node is less important if you prefer to use the KNIME Desktop product; however, with the batch mode, you might find this node useful (less options for configuration, but it can provide the file name as a flow variable).

Try dragging a file which can be imported on the editor area.

Attribute-Relation File Format (**ARFF**) is also tabular (`http://weka.wikispaces.com/ARFF`). You can read them with the **ARFF Reader** node. Exporting to ARFF can be done with **ARFF Writer**.

Importing data from web services

For **Web Services Description Language** (**WSDL**) web services, you can use the **KNIME Webservice Client** standard extension. It provides the **Generic Webservice Client** node.

This node gives many advanced features to access WSDL services, but you should test it to see whether or not it is compatible with your service interface before implementing a new one. It is based on Apache CXF (`http://cxf.apache.org/`), so any limitation of that project is a limitation of this node too.

Unfortunately, not much WSDL web services are available for free without registration, but you can try it out at `http://www.webservicex.com/globalweather.asmx?wsdl`. Naturally, if you are registered for another service, or you have an own in the intranet, you can give it a try.

REST services

Nowadays, the **REST** (**Representational State Transfer**) services has gathered momentum, so it is always nice if you can use it too. In this regard, I would recommend the next section where we introduce the **XML Reader** node. You can use the KREST (`http://tech.knime.org/book/krest-rest-nodes-for-knime`) nodes to handle the JSON or XML REST queries.

Importing XML files

You need the **KNIME XML-Processing** extension from the standard KNIME update site. The **XML Reader** node can parse either local or external files, which you can further analyze or transform.

Importing models

Once you have a model, you might want to save it (**Model Writer** or **PMML Writer**) to use it later in other workflows. In those workflows, you can use the **Model Reader** or **PMML Reader** nodes to bring these models to the new workflow.

Other formats

Some extensions also provide reader nodes to certain data types. The standard KNIME update site contains multiple chemical extensions supporting different formats of chemical compounds.

The **KNIME Labs Update Site** extensions support text processing, graphs, and logfile analyzing, and they contain readers for these tasks.

Public data sources

Most probably you are already familiar with the available data sources for your area of research/work, although a short list of generic data collections might interest you in order to improve the results of your data analysis.

Here are some of them:

- Open data (`http://en.wikipedia.org/wiki/Open_data`) members, such as **DATA.GOV** (`http://www.data.gov/`) and **European Union Open Data Portal** (`http://open-data.europa.eu/`)
- **Freebase** (`http://www.freebase.com/`)
- **WIKIDATA** (`http://www.wikidata.org/wiki/Wikidata:Main_Page`)
- **DBpedia** (`http://dbpedia.org/`)
- **YAGO2** (`http://www.mpi-inf.mpg.de/yago-naga/yago/`)
- **Windows Azure Marketplace** (`http://datamarket.azure.com/`)

This was just a short list; you can find many more of these, and the list of data sources for specific areas would be even longer.

Regular expressions

Regular expressions are excellent for simpler parsing tasks, replaces, or splits. We will give a short introduction on them and show some examples. These will allow you to get better idea. At the end of this section, we will suggest further reading.

Basic syntax

Usually, when you write a text as a pattern, this means that the text will be matched; for example, `apple or pear` will match the highlighted parts from the following sentence: "`Apple stores do not sell` **`apple or pear`**."

These are case sensitive by default, so if the pattern were to be simply `apple`, this will not match the first word of the sentence or the company name.

There are special characters that need to be escaped when you want to match them: `.`, `[`, `]`, `(`, `)`, `{`, `}`, `-`, `^`, `$`, `\` (Well, some of these only in certain positions). To escape them, you should prefix them with `\`, which will result in the following patterns: **`\.`**, **`\[`**, **`\]`**, **`\(`**, **`\)`**, **`\{`**, **`\}`**, **`\-`**, **`\^`**, **`\$`**, **`\\`**.

When you do not want an exact match of characters, you can use the **[**`characters`**]** brackets around the possible options, such as `[abc]`, which will match either **a**, **b**, or **c** but not `bc` (not a single character) or `d` (not among the options). You can specify the range of characters using the character within brackets, such as `[a-z]`, which will match any lower case English alphabet characters. You can have multiple ranges and values within brackets, such as `[a-zA-Z,]`, which will match either a lowercase or an uppercase character or a comma (equivalent to `[[a-z][A-Z][,]]` but not to `[a-z][A-Z][,]` because the latter would match three characters, not one).

To negate a certain character class, you can use the **^** character within brackets; for example, the `[^0-9]` pattern will match a single character except the digits (or the line separators).

It might be tedious and error prone to specify always certain groups of characters, so there are special sets/classes predefined. Here is a non-exhaustive list of the most important ones:

- `\d`: It identifies the decimal digits (`[0-9]`)
- `\s`: It identifies the whitespace characters
- `\n`: It identifies a new line character (by default, only single lines are handled so new lines cannot be matched in that mode, but you can specify a multiline match too)
- `\w`: It identifies the English alphabet (identifier) characters and decimal digits (`[a-zA-Z_`**0-9**`]`)

You can also use the groups within brackets to complement them; for example, `[^\d\s]` (a character that is neither a whitespace nor a digit).

These can be used when you know in advance how long you want to match the parts; although, usually this is not the case. You can specify a range for the number of times you want to match certain patterns using the **{**`n,m`**}** syntax, where *n* and *m* are nonnegative numbers; for example, `[ab]{1,3}` will match the following: `a`, `aa`, `aaa`, and `bab` but not `baba` or the empty string.

When you do not specify *m* in the previously mentioned syntax, it will be (right) unbounded the number of times it can appear. When you omit the comma sign too, the preceding pattern has to appear exactly *n* times to get a match.

There are shorter versions for `{0,1}` **-** **?**, `{0,}` **-** *****, `{1,}` **-** **+**.

When there is no suffix for these numeric or symbolic quantifiers, you are using the greedy match; if you append `?`, it implies the reluctant; while if you append a `+` sign, it will be possessive. Here are some examples: `[ab]+b`, `[ab]+?b`, and `[ab]++b`. The details are important, and can be shown by example. We will highlight the matches for certain patterns and texts (we will separate the matches with | if there are multiple):

Text\ pattern	[ab]+b	[ab]+?b	[ab]++b	[ab]+?	[ab]++
ababab**b**	**ababab**b	**ab**|**ab**|**ab**|**b**b	ababab**b**	**a**|**b**|**a**|**b**|**a**|**b**|**b**|**b**	**ababab**b
ab**aba**	**aba**ba	**ab**|**ab**a	**aba**ba	**a**|**b**|**a**|**b**|**a**	**aba**ba
a**bb**	**bb**a	**ab**b	a**bb**	**a**|**b**|**b**	**bb**a

The last column is a whole text match for each example, also the first column's first and third patterns, but all other examples are just partial (or no) matches.

You might want to create more complex conditions, but you need grouping of certain patterns for them. There are capturing groups and non-capturing groups. The capturing groups can be referred to with their number (there is always an implicit capturing group for each match and the whole match; that is, the *0* group), but the non-capturing groups are not available for further reference or processing, although they can be very useful when you want to separate unwanted parts. The syntax for capturing groups is `(subpattern)` and for non-capturing groups is `(?:subpattern)`.

When you want to refer back to previous groups, you should use the `\n` notation, where *n* is the index of the previous group (in the pattern, the start of the nth starting group parentheses).

There is also an option to create named groups using the `(?<name>subpattern)` syntax. (This feature is available since Java 7, so it will not work on Mac OS X until you can use KNIME with Java 7 or a later version.) Referring to named patterns can be done with the `\k<name>` syntax.

With these groups, you can express not just more kinds of quantification, but also alternatives using the | (or) construct, for example `(ab)?((?:[cd]+)|(?:xzy))`, which means that there is optionally a group of `ab` characters followed by some sequence of `c` or `d` characters or the text `xzy`. The following will match: **`abxzy`**, **`abdcdccd`**, **`xzy`**, **`c`**, and **`cd`**, but `xzyc` or `cxzy` will not.

Positionally, you do not have many options; you can specify whether the match should start at the beginning of the line (`^`), or it should match till the end of the line (`$`), or you do not care (no sign).

The `lookahead` and `lookbehind` options can be handy in certain situations too, but we will not cover them at this time.

Beware. For certain patterns, the matching might take exponentially long; see `http://en.wikipedia.org/wiki/ReDoS` for examples. This might warn you to do not accept arbitrary regular expressions as a user input in your workflows.

Partial versus whole match

The pattern can be matched by two ways. You can test whether the whole text matches the pattern or just tries to find the matching parts within the text (probably multiple times). Usually, the partial match is used, but the whole match also has some use cases; for example, when you want to be sure that no remaining parts are present in the input.

Usage from Java

If you want to use regular expressions from Java, you have basically two options:

- Use `java.lang.String` methods
- Use `java.util.regex.Pattern` and related classes

In the first case, you have not much control about the details; for example, a `Pattern` object will be created for each call of the facade methods delegating to the `Pattern` class (methods such as `split`, `matches`, or `replaceAll`, `replaceFirst`). The usage of `Pattern` and `Matcher` allows you to write efficient (using `Pattern#compile`) and complex conditions and transformations. However, in both cases, you have to be careful, because the escaping rules of Java and the syntax of regular expressions do not make them an easy match. When you use `\` in a regular expression within a string, you have to double them within the quotes, so you should write `\\d` instead of `\d` and `\\\\` instead of `\\` to match a single `\`.

Automate the escaping

The QuickREx tool (see *References, tools*) can do the escaping. You create the pattern, test it, navigate to **File** | **New...** | **Untitled Text File**, and select the **Copy RE to Java** action from the menu or the **QuickREx** toolbar. (Now you can copy the pattern to the clipboard and insert them anywhere you want and close the text editor.)

On the `Pattern` object, you can call the `matcher` method with the text as an argument and get a `Matcher` object. On the `Matcher` object, you can invoke either the `find` (for partial matches) or the `matches` (for whole matches) methods. As we described previously, you might have different results.

References and tools

- The Java tutorial about regular expressions might be a good starting point, and can be referred to at: `http://docs.oracle.com/javase/tutorial/essential/regex/index.html`
- The Javadoc of the `Pattern` class is a good summary and you can refer to it at: `http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html`
- If you prefer testing the regular expressions, QuickREx is a good choice for eclipse (KNIME) and can be referred to at: `http://www.bastian-bergerhoff.com/eclipse/features/web/QuickREx/toc.html`

There is a **Reg. Exp. Library** view that is also included in QuickREx.

Alternative pattern description

In KNIME, there is an alternative, simpler form of pattern description named **wildcard patterns**. These are similar to the DOS/Windows or UNIX shell script wildcard syntax. The `*` character matches zero or more characters (greedy match), but the `?` character matches only a single character. The star and question mark characters cannot be used in patterns to match these characters.

Transforming the shape

There are multiple ways to change the shape of the data. Usually, it is just projection or filtering, but there are more complex options too.

Filtering rows

For row filters, the usual naming convention is used; that is, the node names ending with "Filter" give only a single table as a result, while the "Splitter" nodes generate two tables: one for the matches and one for the non-matching rows.

For single-column conditions, the **Row Filter** (and **Row Splitter**) node can be used to select rows based on a column value in a range, regular expression, or missing values. It is also possible to keep only these rows or filter these out. For row IDs, you can only use the regular expressions.

The rows can also be filtered by the (one-based) row index.

The **Nominal Value Row Filter** node gives a nice user interface when the possible values of textual columns are known at configuration time; so, you do not have to create complex regular expressions to match only those exact values.

There is a splitter, especially for numeric values, named **Numeric Row Splitter**. The configuration dialog allows you to specify the range's openness and gives better support for the variable handling than the **Row Splitter** node.

When you want to filter based on a date/time column, you should use the **Extract Time Window** node, which allows you to specify which time interval should be selected in the result table.

Imagine a situation where you already have a list of values that should be used as a filter for other tables; for example, you used HiLite to select certain values of a table. In this case, you can use one of this table's column to keep or remove the matching rows based on the other table's column. This can be performed by using the **Reference Row Filter** node. The **Set Operator** node is also an option to filter based on the reference table (`Complement`, `Intersection`, `Exclusive-or`), but in this case, you get only the selected columns and not the rest of the rows.

Use the **Set Operator** node to create reference tables.

A very general option to filter rows is using either the **Java Snippet Row Filter** or the **Java Snippet Row Splitter** node. These are interpretations of Java (Boolean) expressions for each row, and based on these results the rows are included or excluded.

We have already introduced the **HiLite Filter** node in the previous chapter, which is also a row-filtering node.

Sampling

If you want to split the data for training, testing, or validation, you can use the **Partition** node that allows you to use the usual options for this purpose (such as stratified sampling). The filtering version is named **Row Sampling**. If you need sampling with replacement, you should use **Bootstrap Sampling**.

The **Equal Size Sampling** node tries to find a subset of rows that satisfies the condition of each value being represented (approximately or exactly) the same number of times as a given nominal column.

Appending tables

This node might not be so easy to find; it is named **Concatenate** or **Concatenate (Optional in)**. These nodes can be used to have two or more (up to four) tables' content in a new one. The handling of the row IDs and the different columns should be specified.

If the data you want to add is just the empty rows with the specified columns, **Add Empty Rows** will do that for you.

Less columns

Sometimes too much data can be distracting, or it might cause problems during modeling and transformation. For this reason, there are nodes to reduce the number of columns. In this section, we will introduce these nodes.

The **Column Filter** node is the most basic option to remove columns. You can specify which columns you want to keep or remove. A similar purpose node is the **Splitter** node. The only difference is that both parts will be available, but in different tables.

The **Reference Column Filter** node helps in creating similar tables, but you can also use this to remove common columns based on a reference table.

When you create a column to represent the reason for missing values, you might need to replace the original column's missing values with that reason. For this task, the **Column Merger** node can be used. It has the option to keep the original columns too.

When you want to have the values from different columns in a single collection column, you should use the **Create Collection Column** node. It can keep the original columns, but can also remove them. You can specify if you want to get the duplicate values removed, or if they should be kept in the selected columns.

Dimension reduction

Sometimes, you don't have a prior knowledge of which columns are useful and which are not. In these cases, the dimension reduction nodes are of great help.

The **Low Variance Filter** node keeps the original columns unless their variance is lower than a certain threshold (you can specify the variance threshold and the columns to check). Low variance might indicate that the column is not having an active role in identifying the samples.

When you want to select the columns based on the inter-column correlation, you should use the **Correlation Filter** node with the **Linear Correlation** node. The latter can compute the correlation between the selected columns, and the filter keeps only one of the highly correlated columns (for "high", you can specify a threshold).

The **Principal Component Analysis** (**PCA**) is a well-known dimension-reduction algorithm. KNIME's implementation allows you to invert the transformation (with errors if any information was omitted). The nodes are: **PCA** (computes and applies transformation based on threshold or number of dimensions), **PCA Compute** (computes the covariance matrix and the model), **PCA Apply** (applies the model with the settings), **PCA Inversion** (inverts the transformation).

The **multidimensional scaling** (**MDS**) operation is also a dimension-reduction algorithm. To use a fixed set of points/rows, you should use the **MDS Projection** node, but if you want to use data points automatically, the **MDS** node is your choice.

More columns

When you have columns that contain too much data in a structured form, you might want them being separated to new columns. You might also need to combine one data source with another; we will describe how to do this in this section.

The **Cell Splitter** node can create new columns from textual columns by splitting them using a delimiter text, while the **Cell Splitter By Position** node creates the new columns by the specified positions (and column names). The first node is useful when you have to do simple parsing, (for example, you read a table with tabs as separator characters, but the date field also uses a separator character, such as `/`, or `-`), but the second is better when you have a well-defined description with fixed length parts (like ISBN numbers or personal IDs).

With the **Regex Split** node, you can do more complex parsing of the data. Each capturing group can be extracted to a column. Keep in mind that for groups that have multiple matches, such as `(...)+`, only the last match will be returned, not all, or the first.

The **Column to Grid** node is used for moving data from rows to new columns in the order of the rows. It will remove the unselected columns, because those cannot be represented in this way, but the selected ones will contain the values from rows in the new columns.

A practical task is referring to previous rows. It is not impossible to achieve this with other nodes, but the **Lag Column** node makes this an easy task.

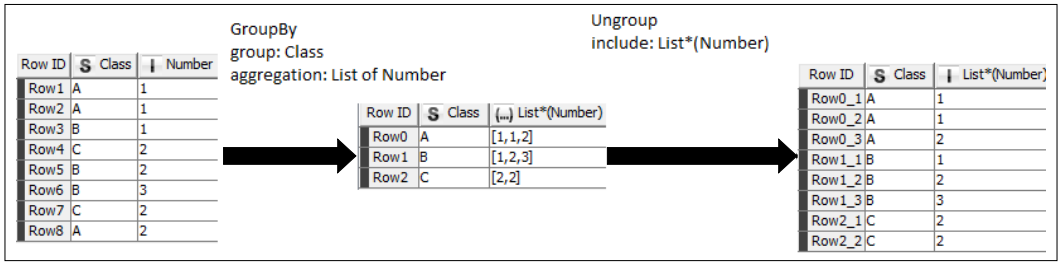
Finally, you can combine two tables using the **Joiner** node. It can perform inner, left, right, or outer joins, based on the row keys or columns. This way you can enrich your data from other data sources (or from the same data source if there are self-references). If you would like to join two tables based on the row indices (practically combine them in a new table horizontally), you should use the **Column Appender** node.

GroupBy

GroupBy is the most versatile data shaping node, even though it looks simple. You specify certain columns that should be used to group certain rows (when the values in the selected columns are the same in two rows, they will be in the same group) and compute aggregate values for the nongroup columns. These aggregation columns can be quite complex; for example, you might retain all the values if you create a list of them (almost works like pivoting). If you want to create a simple textual summary about the values, the **Unique concatenate with count** node might be a nice option for this purpose. If you want to filter out the infrequent or outlier rows/groups, you can compute the necessary statistics with this node. It is worth noting that there are special statistical nodes when you do not want to group certain rows. Check the **Statistics** category for details. However, you can also check the **Conditional Box Plot** node for robust estimates.

With the **Ungroup** node, you can reverse the effect of **GroupBy** transformations by creating collection columns; for example, if you generate the group count and the values in the first step, filtering out the infrequent rows will give you a table, which can be retransformed with the **Ungroup** node (assuming you need only a single column).

Simpler pivoting/unpivoting can be done this way.

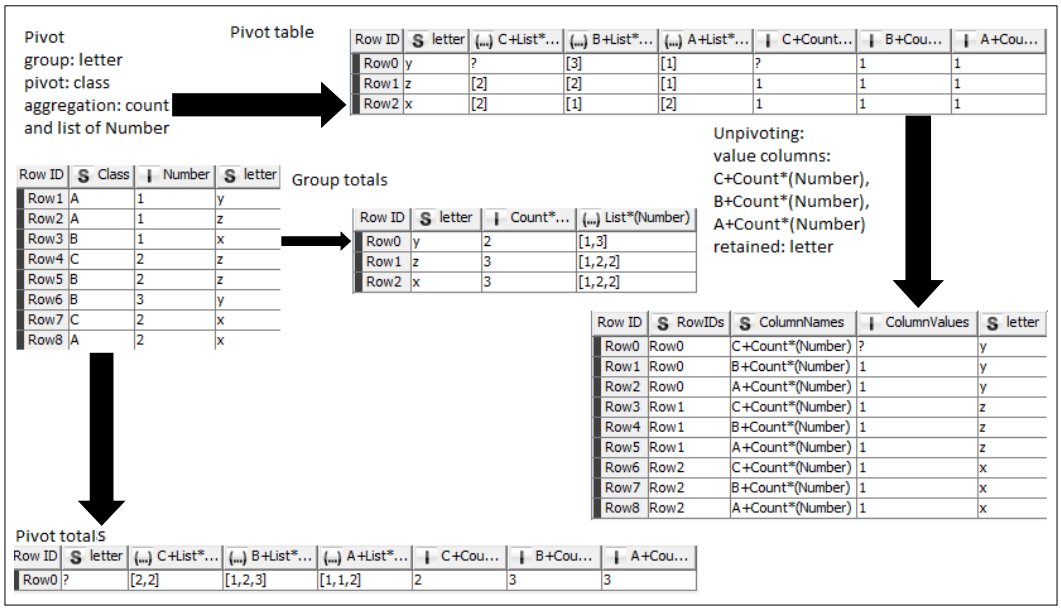


In the preceding screenshot, we start with a simple table, `GroupBy` using the `Class` column, and generate the list of values belonging to those classes, then we undo this transformation using the **Ungroup** node by specifying the collection column.

Pivoting and Unpivoting

The **Pivoting** node's basic option (when there is no actual pivoting) is the same as the **GroupBy** node. When you select the pivoting columns too, these columns will also act as grouping columns for their values; however, the values for group keys will not increase the number of rows, but multiply the number of columns for each aggregate option. The group totals and the whole table totals are also generated to separate the tables. The **Append overall totals** option has results in the `Pivot totals` table only.)

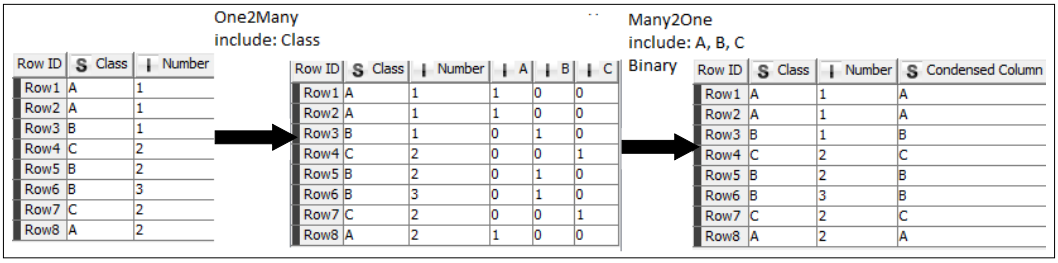
When you want to move the column headers to the rows and keep the values, **Unpivoting** will be your friend. With this node, the column names can be retrieved, and if you further process it using the **Regex Split** and **Split Collection Column** nodes, you can even reconstruct the original table to some extent.



This time the initial table is a bit more complex, it has a new column, `letter`. The **Pivot** node used with the new column (`letter`) as grouping and the `Class` as pivot column. This time not just the list, but also the count of numbers are generated (the count is the most typical usage). The three output tables represent the results, while the table with the `RowIDs` column is the result when the **Unpivoting** node is used on the top result table with the `count` columns as values and the letter column retained.

One2Many and Many2One

Many modeling techniques cannot handle multinomial variables, but you can easily transform them to binomial variables for each possible value. To perform this task, you should use the **One2Many** node. Once you have created the model and applied it to your data, you might want to see the results according to their original values. With the **Many2One** node, this can be easily done if you have only one winner class label.



Row ID	Class	Number
Row1	A	1
Row2	A	1
Row3	B	1
Row4	C	2
Row5	B	2
Row6	B	3
Row7	C	2
Row8	A	2

Row ID	Class	Number	A	B	C
Row1	A	1	1	0	0
Row2	A	1	1	0	0
Row3	B	1	0	1	0
Row4	C	2	0	0	1
Row5	B	2	0	1	0
Row6	B	3	0	1	0
Row7	C	2	0	0	1
Row8	A	2	1	0	0

Row ID	Class	Number	Condensed Column
Row1	A	1	A
Row2	A	1	A
Row3	B	1	B
Row4	C	2	C
Row5	B	2	B
Row6	B	3	B
Row7	C	2	C
Row8	A	2	A

The **One2Many** node creates new columns with binary variables, while the **Many2One** can convert them back.

Cosmetic transformations

This section will summarize some of the options that are not so important for the data mining algorithms, but are important when you want to present the results to humans.

Renames

The **Extract Column Header** and **Insert Column Header** nodes can help you if you want to make multiple renames with a pattern in your mind. This way, you can extract the header, modify it as you want (for example, using another table's header as a reference), and insert the changed header to the result. For those places where a regular expression is suitable for automatic renames, the **Column Rename (Regex)** node can be used.

When a manual rename is easier, the **Column Rename** node is the best choice; it can even change the type of columns to more generic or compatible ones.

Changing the column order

The **Column Resorter** node can do what its name suggests. You can manually select the order you would prefer, but you can also specify the alphabetical order.

Reordering the rows

Using the **Sorter** node, you can order your data by the values of the selected column. Other columns can also be selected to handle ties.

When you want the opposite, for example, get a random order of rows, the **Shuffle** node will reorder them.

The row ID

The row ID, or row key, has an important role in the views, as in the tooltips, or as axis labels, where usually the row ID is used. With the **RowID** node, you can replace the current ID of rows based on column values, or create a column with the values of row ID. You can even test for duplication with this node by creating a row ID from that column. If there are duplicates, the node can fail or append a suffix to the generated ID depending on the settings.

When you use the row IDs to help HiLiting, the **Key-Collection HiLite Translator** node is useful if you have a column with a collection of strings, which are the row keys in the other table.

Transpose

The **Transpose** node simply switches the role of rows and columns and performs the mathematical transpose function on matrices. It is not a cosmetic transformation, although it can be seldom used to get better looking results. The type of the column is the most specific type available for the original row.

Transforming values

Sometimes the data that you have requires further processing; that is, not just moving around but also changing some values.

Generic transformations

A quite flexible node is the **Rule Engine** node that creates or replaces a column based on certain rules involving other columns. It might contain logical and relational operators for texts, and it can even check for inclusion (`IN`) for a certain set of values or limited pattern matching (`LIKE` as in SQL). The result can be either a constant, a column's value, or a flow variable. It can also handle the missing values.

When you want to fill the metadata, you should use the **Domain Calculator** node. With the help of this node, you can create nominal columns from textual (`String`) columns.

UNIT - V

3
Data Exploration

In this chapter, we will go through the main functions of KNIME visualization (except reporting) and other techniques to explore the data you have. This can be helpful when you want to do the preprocessing too, but you can also check the result of visualization or see how well they fit the computed models and the test/validation data. The topics covered in this chapter are as follows:

- Statistics
- Distance matrix
- Visual properties
- KNIME views and HiLiting
- JFreeChart nodes
- Some third party visualization options
- Tips with HiLiting
- Visualizing models

Computing statistics

When you want to explore your data, it usually is a good idea to compute some statistics about them so that you can spot the obviously wrong data (for example, when some data should be positive and it appears as a negative minimal value, it is suspicious).

Most of the nodes require you to not have NaN values within the data to be analyzed. You can remove them with the value modification techniques presented in the previous chapter, or by filtering the rows, also discussed in the previous chapter.

The minimal and maximal values can be checked in the port view's **Spec Columns** tab. This can already be used to spot certain kinds of problems.

For statistics within groups, we have the good old **GroupBy** node. That allows you to aggregate using the functions described on the **Description** tab of the configuration dialog.

When you do not need the grouping, you can use the **Statistics** node with easier configuration. Just select the columns, the number of values that should be present in the view, and the number of common/rare values that should be enumerated. You might find that the median is not computed in the results. In this case, you should check the **Calculate median values (computationally expensive)** checkbox. The following is the statistics you get in the view (for the numeric columns):

- **Minimum**
- **Maximum**
- **Mean**
- **Std deviation**
- **Variance**
- **Overall sum**
- **No. missings**
- **Median**
- **Row count**

You also get the number of missing values and the most common and rarest values for the selected nominal (and also numeric) columns, with their number of occurrences.

The statistics table, which is the first output port, contains the same content as the view for the numeric columns. The second output port (occurrences table) gives a table with the number of occurrences for each numeric and nominal values in a decreasing order of frequencies (including the missing values).

Using the output tables, you can create conditions or further aggregate operations. For example, creating the flow variables from the certain mean and standard deviation and creating conditions using the **Java Edit Variable** node allows you to filter the rows with certain ranges related to the mean and standard deviation with the row filtering/splitting nodes. (Or use the **Java Snippet Row Filter** node directly with the flow variables.)

The **Value Counter** node acts in a manner similar to the **Statistics** node's second output, but in this case, only a single column is used. So, no missing values will appear in the `count` column (which is not sorted) and the values from the original column will appear as row IDs. In this form, they are better suited for visualization. Also, because this node is able to support HiLite, you can select the original rows based on the frequency values.

When you want a similar (frequency) report with two columns and a possible weight column to create crosstabs, you should use the **Crosstab** node. In the view of the node, you get the crosstab values in the usual form. You can specify which parts (**Frequency**, **Expected**, **Deviation**, **Percent**, **Row Percent**, **Column Percent**, or **Cell Chi-Square**) should be visible. (The row and column totals are always visible, and if there are too many rows or columns, you can keep only the first few.)

There is another table in the view, beneath the frequency. It is the summary of the Chi-Square statistics (degree of freedom (DF), the χ^2 Value, and the probability (Prob) of no association between the values (a p-value)), and also the Fischer test's probability, when both columns contain exactly two values.

The **Crosstab** node's first output port contains the values similar to the view's main table, but in this case, it is in a different form: the column values are in columns, while the statistics (Frequency, Expected, Deviation, Percent, Row Percent, Column Percent, Total Row Count, Total Column Count, Total Count, and Cell Chi-Square) are in other columns. You can transform it to the usual crosstab form (keeping a single statistics) using the **Pivoting** node (select one of the columns as the group column, the other as pivot, and the statistics should be used as an aggregation option). You can check the workflow from the `crosstab.zip` file available on this book's website.

The second output table of the **Crosstab** node contains the statistics just like the second part of the view, but in this case it is in a single row even if both the columns contain two values (the Fischer test's p-value is in the last column).

When you want to create a correlation matrix, you should use the **Linear Correlation** node. It will compute the correlation between the numeric-numeric and nominal-nominal pairs. Also, a model will be created for further processing. You can use this information to reduce the number of columns with the help of the **Correlation Filter** node.

The view of the **Linear Correlation** node gives an overview about the correlation values with the color codes.

There are three t-test computing nodes: **Single sample t-test**, **Independent groups t-test**, and **Paired t-test**. The **Single sample t-test** can be used to test whether the average of the selected columns is a specified value or not. The **t-value (t)**, **degree of freedom (df)**, **p-value (2-tailed)**, **Mean Difference**, and **confidence interval differences** are computed relative to the specified mean value (the **Test value**). The other output table contains some statistics about the columns, such as the computed mean, standard deviation, standard error mean, and the number of missing values in that column.

The view of **Single sample t-test** contains the same information as the two output tables.

When you want to compare the means of two measurements of the same population (or at least not independent), you can use the **Paired t-test** node. The view and the resulting tables contain the same statistics as the **Single sample t-test** node, but in this case the mean difference is replaced with the standard deviation and the standard error mean values, both in the view and the first output table. The configuration options allow you to select multiple pairs of numeric columns.

For two sample t-tests, you should use the **Independent groups t-test** node. It expects the two groups to be defined by a column; the values are grouped by that column's values. You can select the column that contains the class for grouping and the values/labels for the two groups within that column. The average of the columns will be compared, and the t-tests will be computed both for the equal variance assumption and without that assumption (first output table). The Levene test is also computed to help decide whether the equal variance can be assumed (second output table).

The descriptive statistics is augmented with the number of rows that are not in either group (`Ignored Count (Group Column)`).

The last test for hypothesis testing is the **One-way ANOVA**. It allows you to compare the means within groups defined by the values of a single column, just like the **Independent groups t-test** node does; however, it supports multiple groups.

Finally, when you need robust statistics, you can use the **Conditional Box Plot** node. It gives you the minimum and maximum values, the median, Q1, Q3, and the whisker values (can be the same as min/max, else the 1.5 times interquartile range (Q3 – Q1) below or above Q1 and Q3).

Overview of visualizations

The various options to visualize data in KNIME allow you to get an overview or even publication-quality figures from the data you have preprocessed and analyzed.

The interactive versions of a node allow you to change the column selections and probably the other extra options.

The **JFreeChart** nodes generate images from the input data, which is also available as a view with further customization options. These nodes usually do not support the HiLite feature and the different visual properties (color, size, and shape).

First, to help decide what you use to open the data, we will compare the capabilities of the different visualization nodes:

Node	Supported data types	Remarks
Box Plot	Numeric (multiple)	Provides robust stats
Conditional Box Plot	Nominal and numeric (multiple)	Also gives robust stats
Histogram	Nominal or numeric and numeric	
Histogram (interactive)	Nominal or numeric and numeric	
Interactive Table	Any	Similar to port view
Lift Chart	Nominal and probability	
Line Plot	Numeric (multiple)	
Parallel Coordinates	Nominal or numeric	
Pie chart	Nominal and numeric	
Pie chart (interactive)	Nominal and numeric	
Scatter Matrix	Nominal or numeric	Multiple scatter plots
Scatter Plot	Nominal or numeric (two)	
Bar Chart (JFreeChart)	Nominal	
Bubble Chart (JFreeChart)	Numeric (three)	
Group By Bar Chart (JFreeChart)	Nominal (unique) and numeric	Color properties supported
HeatMap (JFreeChart)	Distance or numeric	Distance between rows
Interval Chart (JFreeChart)	Date and nominal	
Line Chart (JFreeChart)	Numeric (multiple) or date	Color properties supported
Pie Chart (JFreeChart)	Nominal	Color properties supported
Scatter Plot (JFreeChart)	Numeric (two)	Color, shape used
Linear Regression (Learner)	Numeric (multiple)	Scatter + line of model
Polynomial Regression (Learner)	Numeric (multiple)	Scatter + graph of model
OSM Map View	Numeric (two)	Spatial data
OSM Map to Image	Numeric (two)	Spatial data, creates image
Hierarchical Cluster View	Distance and cluster model	Dendrogram

Node	Supported data types	Remarks
ROC Curve	Nominal and numeric (multiple)	
Enrichment Plotter	Numeric (multiple)	
Spark Line Appender	Numeric (multiple)	No view, but creates images
Radar Plot Appender	Numeric (multiple)	No view, but creates images

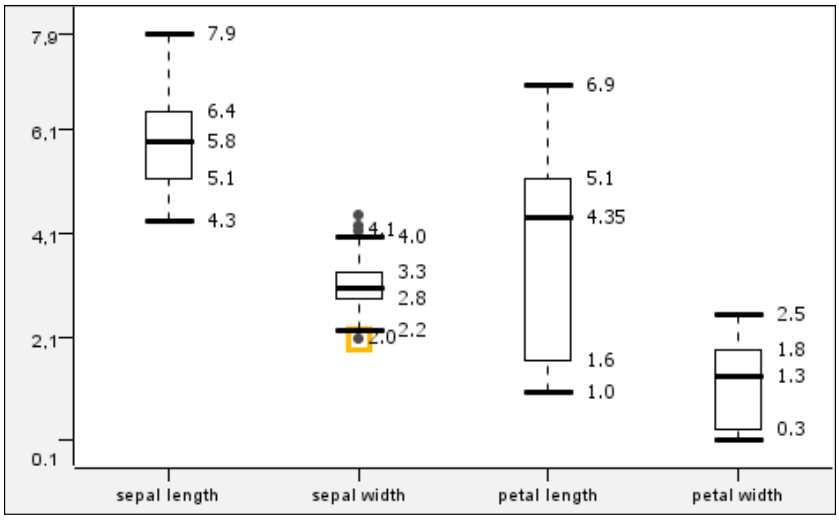
There are a few other view-related nodes in KNIME (and many more with mostly textual views). The **Image To Table** node can be useful when you want to iterate (loop) through certain parts generating images. Because the image ports (dark green filled rectangles) cannot be used with loop end nodes, you have to convert them to a table column. This is the exact purpose of the **Image To Table** node.

On the other hand, when you want an image port to hold an image (for example, to include it in a report), you should use the **Table To Image** node, which selects the first row's selected image column and returns it as an image port object.

The last notable node is the **Renderer to Image**. It simply grabs a column and the selected renderer, and creates an SVG or PNG image column with its content. You can use this later in web pages or other places, where supported. This is very handy when you want to handle a special kind of content; for example, molecules.

Visual guide for the views

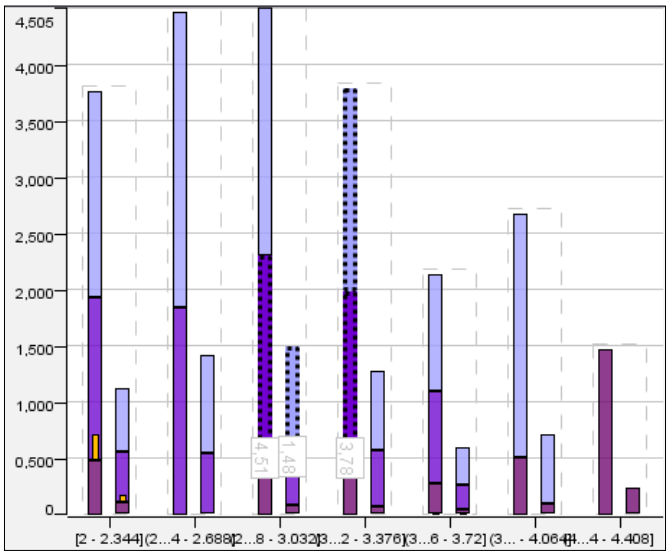
In this section, we will introduce the iris dataset (*Frank*, *A. & Asuncion*, *A.* (*2010*). *UCI Machine Learning Repository* (`http://archive.ics.uci.edu/ml`). *Irvine*, *CA*: *University of California*, *School of Information and Computer Science*. Iris dataset: `http://archive.ics.uci.edu/ml/datasets/Iris`) with some screenshots from the views (without their controls).



Box plot for the numeric columns

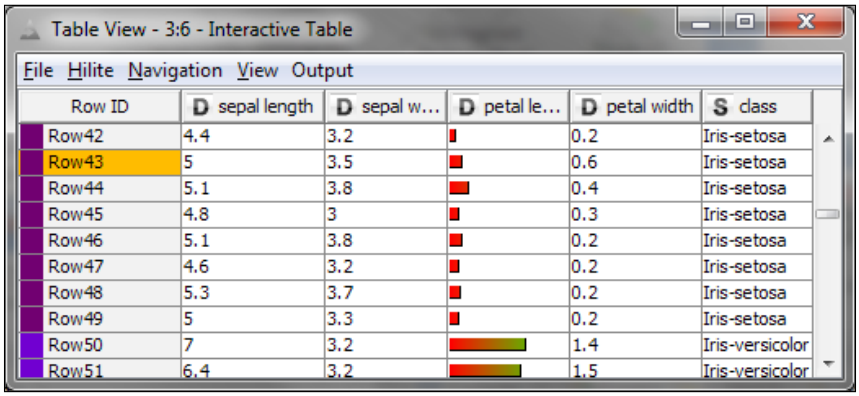
The **Conditional Box Plot** and the **Box Plot** nodes' views look similar. These are also sometimes called box-and-whisker diagrams. The **Box Plot** node visualizes the values of different columns, while the **Conditional Box Plot** view shows one column's values grouped by a nominal column's values. As you can see in the screenshot, the HiLite information is visible for the outliers (but only for those values). You can also select the outliers and HiLite them.

The shape of the outlier points is not influenced by the shape property.



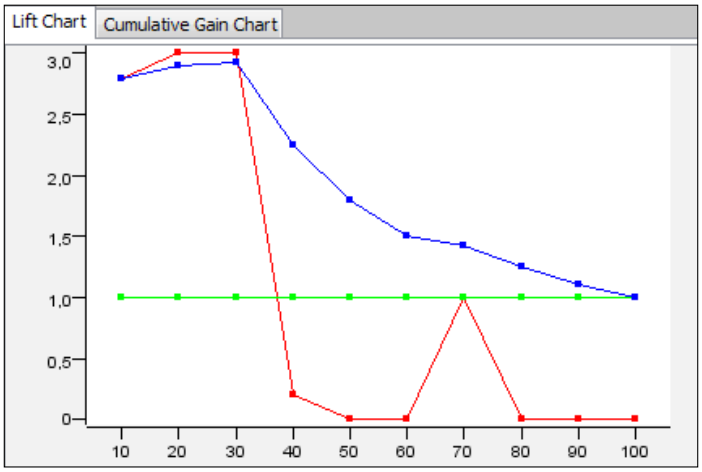
Histogram with a few columns selected, HiLited rows and colored values based on class attribute

As the screenshot shows, the **Histogram** node's view is capable of handling the color properties. It also supports the aggregation of different values, and the option to show the values for the selected (or all) columns. The adjacent columns within the dashed lines represent the different columns for each binning column value. This way, you can compare their distributions for certain aggregations. The interactive and the normal versions look quite similar, but they differ in configuration and view options.



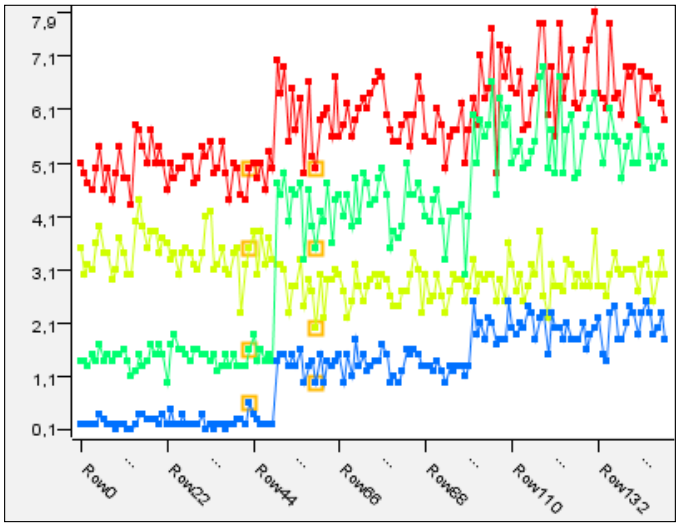
The Interactive Table view with changed renderer for petal length and color codes for class, Row43 is HiLited

The **Interactive Table** view first looks and works like a normal port view for a data table (such as the options on the context menu for the column header: **Available Renderers**, **Show Possible Values**, and sorting by *Ctrl* + clicking on the header; the latter can be done from the menu with a normal click, too), although it offers HiLiting and a few other options.



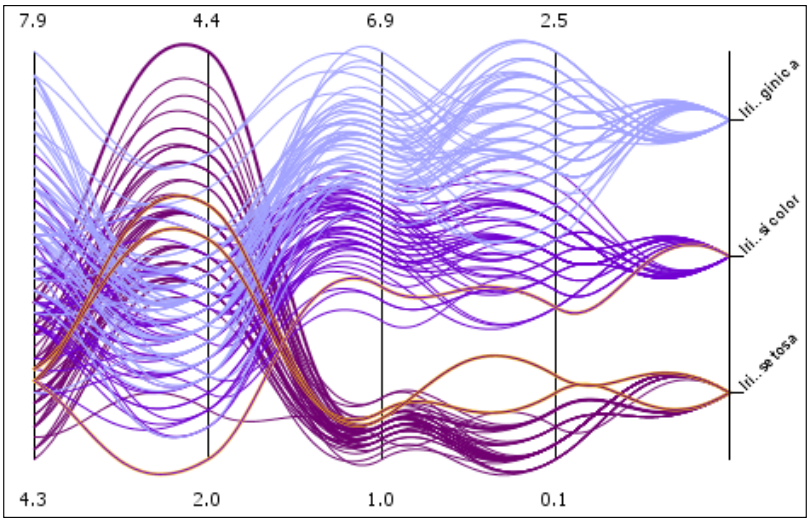
Lift chart of a model predicted by a decision tree, the colors are: red – lift, green – baseline, cumulative lift – blue

The **Lift Chart** view can help evaluate a models' performance. The **Cumulative Gain Chart** tab looks similar, although it has only two lines.



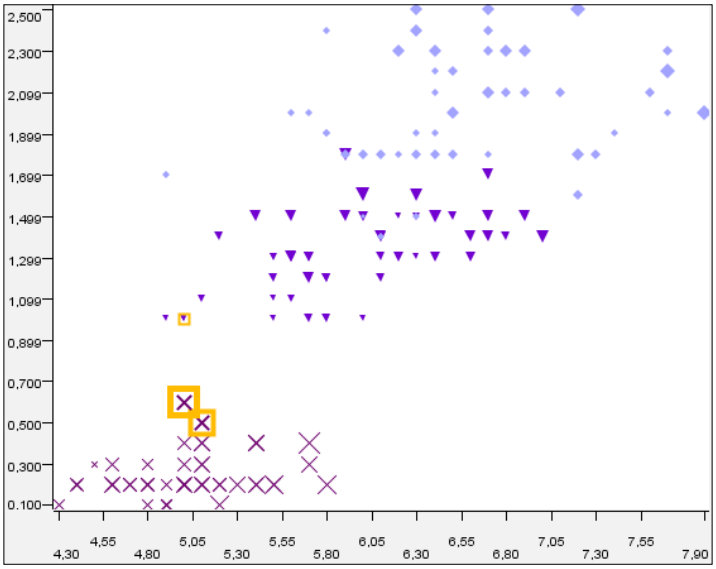
Line plot with some two HiLited rows and the four numeric columns: red – sepal length, yellowish – sepal width, green – petal length, blue – petal width

The **Line Plot** view can be used to compare the different columns of the same rows. The rows are along the x axis, while their values for different columns are along the y axis. The adjacent row's values for the same column are connected with a line.



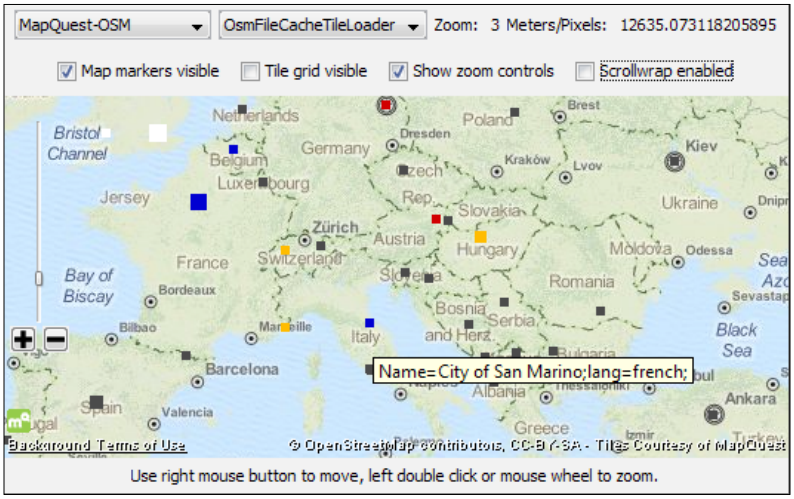
Parallel coordinates with colored curvy lines, the columns are: sepal length, sepal width, petal length, petal width and class

The **Parallel Coordinates** view can also visualize the individual rows, but in this case, the row values for the different columns are connected (with lines or with curves). In this case, the columns are along the x axis, while the values are along the y axis.



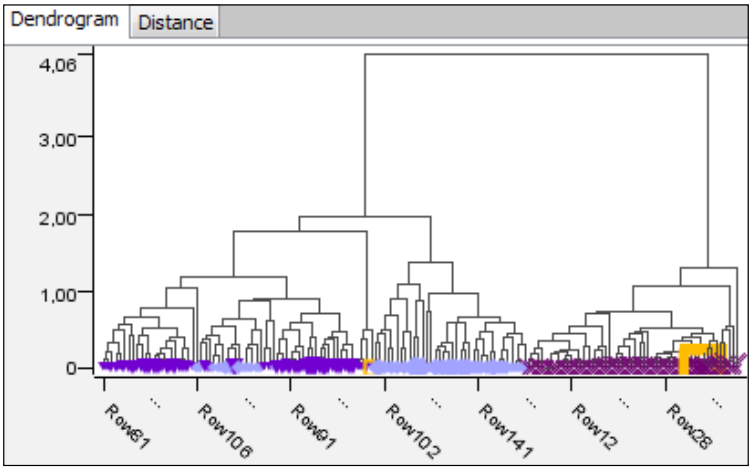
Scatterplot of sepal length vs. petal width with size information from sepal width

The **Scatter Plot** views can be used efficiently to visualize the two dimensions. Although, with the properties, the number of dimensions from which information is presented can grow to five.



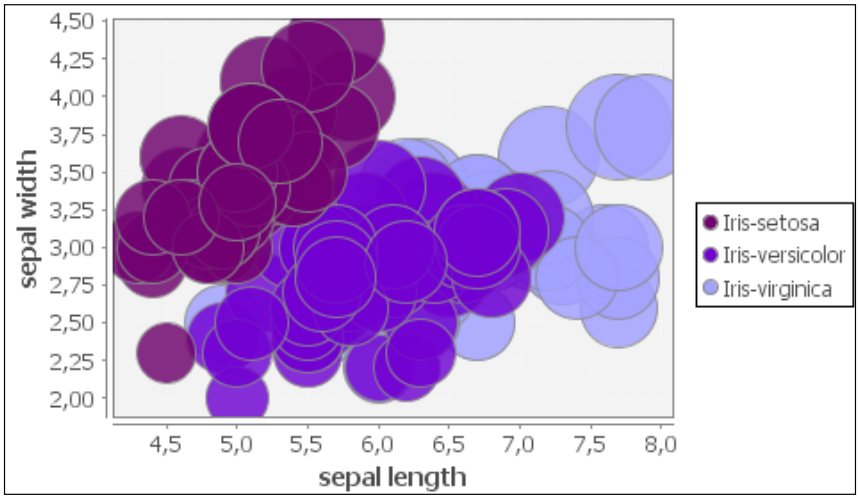
The Open Street Map integration offers many ways to visualize spatial data; it supports color, shape, and size properties and also works with HiLiting. Selected information from the input table is also available as a tooltip.

The **OSM Map View** and **OSM Map to Image** nodes are designed to show data on maps. They are very flexible, and can show many details, but they can also hide the distracting layers.



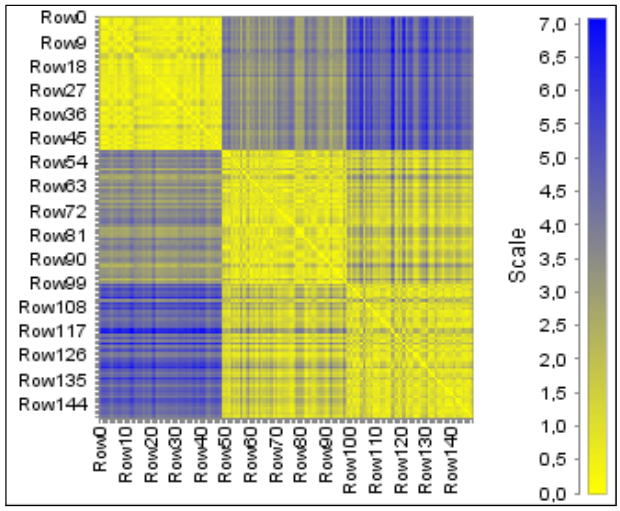
Hierarchical clustering dendrogram (average linkage with Euclidean distance using the numeric columns)

The best way to visualize a clustering is by using a dendrogram, because the distances between the clusters are visible in this way. The **Hierarchical Cluster** view offers this kind of model visualization. To show the similarity between the rows, first you have to compute the cluster model using the **Hierarchical Clustering (DistMatrix)** node from the KNIME Distance Matrix extension, available on the KNIME update site.



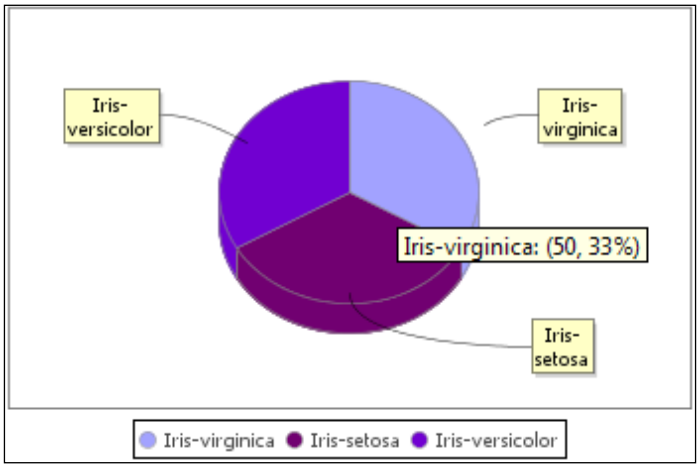
JFreeChart bubble chart

The **Bubble Chart (JFreeChart)** node can offer an alternative to the scatter plots; however, in this case, the dimension of the size is also mandatory.



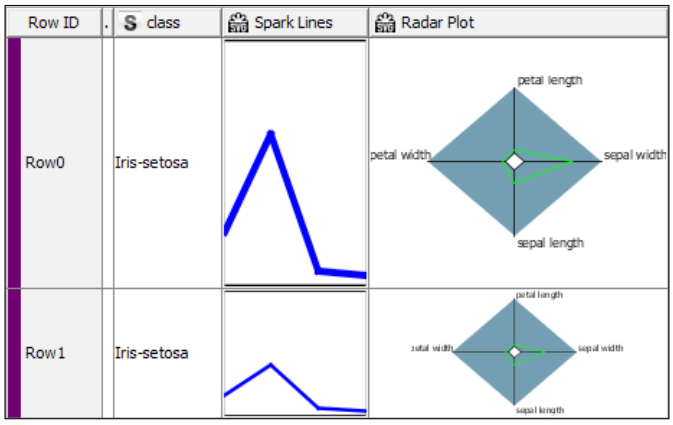
JFreeChart heatmap with Euclidean distance of numeric columns

The **HeatMap (JFreeChart)** node provides a way to visualize not just the collection columns, but also the distances, as shown in the previous screenshot. To use the regular tables, you might require a preprocessing step which uses the **Create Collection Column** or the **GroupBy** node to compute the distances, but it also works fine for displaying the values.



JFreeChart pie chart

The **Pie Chart (JFreeChart)** node also offers a visualization with a pie, and unlike the **Pie chart** and the **Pie chart (interactive**) nodes, this can create three-dimensional pies.



The spark lines and radar plot for numeric columns

The results of the **Spark Lines Appender** and the **Radar Plot Appender** nodes are not the individual views, but are the new columns with the SVG images generated for each row. We can use this in the next chapter.

Distance matrix

The distance matrix is used not just for visualization, but for learning algorithms too. You can think of them as a column of collections, where each cell contains the difference between the previous rows.

The supported distance functions are the following:

- Real distances
  - Euclidean($||v_1 - v_2||_2$)
  - Manhattan ($||v_1 - v_2||_1$)
  - Cosine ($1 - \frac{v_1 v_2}{||v_1||_2 ||v_2||_2}$)
- Bitvector distances
  - Tanimoto ($1 - \frac{|v_1 v_2|}{|v_1| + |v_2| - |v_1 v_2|}$)
  - Dice ($1 - \frac{2|v_1 v_2|}{|v_1| + |v_2|}$)
  - Bitvector cosine ($1 - \frac{|v_1 v_2|}{|v_1| |v_2|}$)
- Distance vector (assuming you already have a distance vector, you can transform it to a distance matrix when there are row order changes or filtering)
- Molecule distances (from extensions)

The distance matrix feature can be used together with the hierarchical clustering, which also provides a node to view it; this is the main reason we introduced them in this chapter.

You can generate distances using the **Distance Matrix Calculate** node (just select the function, the numeric columns, and set the name. The chunk size is just for fine tuning larger tables), but you can also load that information with the **Distance Matrix Reader** node.The HiTS extension (`http://code.google.com/p/hits`) also provides a view to show dendrograms with heatmaps.

Using visual properties

One of KNIME's great features is that it allows you to set certain properties of the views in advance. So, you need not remember how you set them in one view and how it is set in another, you just have to connect them to the same table. This is a big step towards reproducible experimental results and figures with the ease of graphical configuration. Each property is applied to the rows based on column values, so changes in column values will affect (remove) the property and each kind of property is exclusive (a new node with the same kind of properties replaces the original property). When you want to reuse the properties in another place of the workflow, you can use the appender nodes.

The three supported properties are: color, size, and shape.

Color

With the **Color Manager** node, you can set the color for different rows. The colors can be assigned either to a nominal or a numeric column.

In the case of the nominal columns, each value can have a different color. This can be useful when you want to compare the actual or the predicted labels/classes of the rows.

When you assign colors to the numeric columns, the color of the minimal and the maximal value (as it is available in the column specification: **Lower Bound**, **Upper Bound**) should be specified. The remaining shades are linearly computed.

The **Color Appender** node allows you to use the same color configuration for other tables. Be careful when there are values outside the domain. The nearest extreme value is used in case of numeric columns and the black color is used for nominal columns. It is also possible to set an incompatible format to the column, but in that case, it will not be used.

Size

The size of the points can be really a good indicator of the nonvisible attributes. It allows you to have larger or smaller dots for the different data points in views. The size is computed by the **Size Manager** node as a function of the input from the minimal value to the maximal value, similar to the numeric color property. (Based on the domain bounds, outside them the nearest extreme is used.)

Be careful not to use this node on columns where the minimum is less than zero (the logarithmic and the square root function would generate a complex number). Also, check the bounds after filtering; you might need to use the **Domain Calculator**.

The following are the supported functions:

- **LINEAR**: It is a linear function between the bounds
- **SQUARE_ROOT**: It is useful when you want a less increase in the higher values, but want more details of the lower values
- **LOGARITHMIC**: It is ideal when there is large difference between the bounds and more details near the lower bound is interesting
- **EXPONENTIAL**: The exponential function will make even small differences large

The **Size Appender** allows you to use the same size configurations in different places of the workflow, even for other columns.

Shape

The last property you can set is the shape of the points. For this purpose, you have the **Shape Manager** node, which allows you to set the shape based on a nominal column's values. Together with the **Color Manager,** you can visualize both the predicted and the original class of the training dataset. This can give you a better idea when the data is not properly learned and clustered, and might give you ideas to improve the settings.

Similar to other properties, the **Shape Appender** can bring the shape configuration to other parts of a workflow.

KNIME views

You can export the view contents to either the PNG or SVG files from the **File** | **Export as** menu. (The latter is only available when the KNIME SVG Support is installed.)

It is worth noting the other usual view controls. The **File** menu contains the **Always on top** and **Close** options, besides the previously discussed **Export as** menu. The first option allows you to compare the multiple views easily by having them side-by-side and still working with other windows.

The rest of the menus are related to HiLiting, which will be discussed soon.

The configuration of nodes usually includes an option of how many different values or how many rows should be used when you create the view. Because the views usually load all the data (or the specified amount) in the memory to have a resizable content, too many rows would require too much memory, while too many different values would make it hard to understand either the legends or the whole view in certain cases.

The mouse mode controls allow you to select certain points or set of points (for example, in the case of hierarchical clustering and the histogram nodes), to zoom in or to move around in a zoomed view. With the **Background Color** option, you can change the background of the plot. The **Use anti-aliasing** option can be used to apply subpixel rendering for fonts and lines.

HiLite

The **HiLite** menu consists of the **HiLite Selected**, **UnHiLite Selected**, and **Clear HiLite** items. With these items, you can create fine-grained HiLite rows. Once you select a few data points/rows, you can add or remove the HiLite signal using the first two options, and the third clears all the HiLite signals from this part of the workflow.

Lots of the nonview nodes also have HiLite-related options, which can be very handy when the row's IDs change and want to propagate HiLiting to the parts with different row IDs of the workflow; however, beware, as this usually requires additional memory.

The **Show/Hide** menu (or the **HiLite/Filter** menu) also helps the HiLite operations. The **Show hilited only** option hides all the non-HiLited rows/points. The default option is usually **Show all**, but the **Fade unhilited** option is a compromise between the two (shows both the kinds of data, but the non-HiLited are faded or grey).

Use cases for HiLite

You might wonder how this HiLite feature is useful.

With the **Box Plot** and the **Conditional Box Plot** nodes, you can select the rows that have extreme values in certain columns or extreme values within a class without creating complex filtering. (The extremity is defined as below $Q1 - 1.5IQR$ or as above $Q3 - 1.5IQR$

It is also useful to see the same selection of data from different perspectives. For example, you have the extremes selected based on some columns, but you are curious to know how they relate to other columns' values. The **Parallel Coordinates** or the **Line Plot** can give a visual overview of the values. The **Scatter Plot** (or the **Scatter Matrix**) node is also useful when different columns should be compared.

When you prefer the numeric/textual values of the selected rows, you should use the **Interactive Table** node. It allows you to check the HiLited and non-HiLited rows together or independently with the order of the column you want.

With the **Hierarchical Clustering View** node, you can select certain clusters (similar rows). This can also be useful to identify the outlier groups based on multiple columns (as the distances can be computed from more than one columns).

Row IDs

It is important to remember that the row IDs play an important role for most of the KNIME views. The row IDs are used as axis values; that is, tooltips. So, to create a nice, easy-to-understand figure/view, you have to provide as many useful row IDs as you can.

To use meaningful labels, you have to create a column with the proper (unique) values, and make that column a row ID with the help of the **RowID** node. This node also offers HiLite support (**Enable Hiliting**), so you do not have to make a compromise between neat figures and HiLiting.

Extreme values

The infinite values (`Double.POSITIVE_INFINITY` and `Double.NEGATIVE_INFINITY`) make the ranges meaningless, because these values are not measurable by normal real values.

The other special value is the `Double.NaN` (not a number) value, which you get, for example, when you divide zero by zero. It is not equal to any numeric value, not even to itself. It also makes comparison impossible, so it should be avoided as much as possible. The previous chapter has already introduced how to handle these cases.

The missing values are usually handled by not showing the rows containing them, but some views make it possible to use different strategies.

Basic KNIME views

The main views of KNIME give you multiple options to explore data. These nodes do not provide options to generate images for further nodes, but they give quite a good overview about the data, and you can save the files using the **File** menu.

There are different flavors for some of the nodes: the interactive and the normal. With the interactive flavor, you can modify certain parameters of the view without reconfiguring (and executing) the view. The interactive versions are better suited for data exploration, but the normal ones make it easier to check certain things with new data.

The Box plots

The **Box Plot** node has no configuration, but gives robust statistics (minimum, smallest, lower quartile, median, largest, and maximum) for numeric columns. You might wonder about the difference between the minimum and the smallest values or the largest and maximum values. The smallest is the maximum of the minimal value and the $Q1 - 1.5IQR = Q1 - 1.5(Q3 - Q1)$ value. The largest is computed analogously.

The view gives a box-and-whisker diagram, which is useful to find outliers. The **Column Selection** tab allows you to focus only on certain columns. The **Normalize** option on the **Appearance** tab will rescale the box-and-whisker diagrams to have the same length on the screen between the minimum and maximum values.

The **Conditional Box Plot** node's view is quite similar to the **Box Plot** view, although in this case, the diagram is not split by the columns, but by a preselected nominal column. The values are representing the values from a numeric column. You can also select whether the missing values should be visible or not.

The node view controls are really similar to the **Box Plot**'s. However, in this case, the **Column Selection** tab does not refer to the columns from the table, but to the columns on the diagram; you can select the class values that should be visible.

Hierarchical clustering

There is an option to visualize the result of hierarchical clustering with the **Hierarchical Cluster View** node; however, it is worth summarizing how you can reach the state when you can show the cluster model. First, you have to specify the distance between the rows using one of the options we described in the *Distance matrix* section.

In the **Hierarchical Clustering (DistMatrix)** node's configuration, the main option you have to select is the **Linkage Type**, which defines how the distance between the clusters should be measured:

- **Single**: It measures the minimal distance between the cluster points
- **Average**: It measures the average of differences between the points of the clusters
- **Complete**: It measures the maximal distance between the cluster points

You can also select between the distance matrices if you have multiple columns.

Histograms

The difference between **Histogram** and **Histogram (interactive)** is minimal in the configurations (the non-interactive version allows you to specify the number of bins configuration time). The common configuration options are the **Binning column**, **Aggregation column**, and the **No. of rows to display**. With the **Binning column** option, you can define how the main bins should be created; it can be either nominal or numeric. The coloring information splits between the bars, and the aggregation columns are available as separate, adjacent bars.

The possible aggregation options are: **Average**, **Sum**, **Row Count**, and **Row Count (w/o missing values)**. When you have multiple aggregation columns selected, **Row Count** (with missing values) is not an informative or recommended choice.

On the **Visualization settings** tab, you can further customize the view, by enabling/disabling outlines, grid lines, the orientation, width, or the labels.

The **Details** tab gives the information about the selected bars, such as the average, sum, count for each column, and colors. (You can select the monochrome part of a bar too.)

Interactive Table

The interactive table looks like a plain port view; however, it gives further options, such as the HiLiting support and the optional color information (in the port view, it is not optional). You can also save the content to the CSV file (**Output | Write CSV**), adjust the default column and row size (**View | Row Height... and Column Width...**), and find certain values (**Navigation | Find**, *Ctrl + F*).

The options for sorting by columns (*Ctrl* + click, or the menu from the regular click) and reordering (dragging) them are also available in this view, and you can select the preferred renderers for them. However, you cannot check the metadata information (column stats and the properties).

The Lift chart

The **Lift Chart** node is useful when you want to evaluate the fit of a model for a binominal class. In the configuration dialog, you can specify what is the training label and the value learned. The probabilities of the learned label should also be specified, just like the width of the bins (in percentage, you will get 100/that value points). In the view, there are two parts—**Lift Chart** and **Cumulative Chart**—both with separate configurations of color, line widths and dot sizes (with visibilities).

The **Lift Chart** node also contains the cumulative lift, but it can be made invisible if you do not want it.

Lines

The **Line Plot** node and the **Parallel Coordinates** views are similar, but they show the data in the orthogonal/transposed form with respect to each other. The **Parallel Coordinates** view contains the selected columns on the x axis and the row values flow horizontally colored by the color properties, while in **Line Plot**, the rows are on the x axis and the (numeric) columns are represented by user-defined colors.

The missing values are handled differently; in **Line Plot**, you can try to interpolate, while in the other, you can either omit or show them or their rows.

Line Plot is more suited for equidistant data, such as time series, for other data it might give misleading results (the distances between the rows are the same). The **Parallel Coordinates** view is better suited to find connections between the values of different columns, because in this case you have no ordering bias. The **Parallel Coordinates** view gives a neat option to use curves instead of straight lines. Fortunately, you can change the order of columns within the view using the extra mouse mode **Transformation**, so you can create neat figures with this view. This view is quite good to show intuitive correlations.

Pie charts

The **Pie Chart** and the **Pie Chart (interactive)** nodes have the same configuration options, although for the latter, the configuration gives only the overridable defaults in the view. These configurations include the binning column and the aggregation column, just like the aggregation function.

With *Ctrl* + click, you can select multiple pies. HiLiting works in this view, and the **Details** tab contains statistical information for each selected sections, which is split by the colors within the pies. When the binning is not consistent with the color property, no coloring is applied unless you select them (and enable the **Color selected** section).

In the **Visualization setting** tab, you can specify whether the section representing the missing values should be visible or not, show outline, explode the selection, or whether the aggregated value/percent should be visible or not (for selected, all, or no sections). The size of the diagram too can be adjusted in this tab.

The Scatter plots

The **Scatter Matrix** and the **Scatter Plot** nodes are quite similar. The **Scatter Matrix** node is a generalization of the latter. It allows you to check the scatter plots for different columns side-by-side.

A scatter plot can use all the visual properties (size, shape, and color), so you can visualize up to five different columns' values on a 2D plot.

There are not many configurations for either maximum rows or maximum distinct nominal values in a column.

In the case of **Scatter Plot**, you can only select the two columns for the x and y axes, but in case of the **Scatter Matrix** node, you can set the ranges for them. With the **Scatter Matrix,** you can select multiple columns, and when you are in the **Transformation mouse** mode, you can rearrange the rows/columns, but you cannot change their ranges.

Both the views support the jittering when one of the columns is nominal (the **Appearance** tab, **Jitter slider**). In that case, the values in the other dimension get some random noise, so the number of points at a position could be easily estimated. If you want precise positions, you might consider adding transparency to the color of the points, so when there are overlaps, they will be more visible.

The **Linear Regression (Learner)** and the **Polynomial Regression (Learner)** nodes also provide the scatter plot views, although these show the model as a line. It can be useful to have a visual view of the regression, even though these do not specify which slice of the function is shown from the many possible functions, parallel to the selected.

Spark Line Appender

The **Spark Line Appender** node does not have a view, but it generates a column with an SVG image of a line plot of the selected numeric columns, for that row. This can be useful to find interesting patterns. However, it is recommended to use **Interactive Table**, because the initial size is hard to see, and changing the row height multiple times is not so much fun (and can be avoided if you hold the *Shift* key while you resize the height of a row). But with the special view, you can do that from the menu.

Radar Plot Appender

The **Radar Plot Appender** node works quite like the previous node, although it has more configuration options. You can set many colors for the SVG cell, and also the ranges and the branches (columns) of the radar plot. The resulting table has a bit larger predefined row height, but the use of an **Interactive Table** view might still be a good idea.

The Scorer views

The **ROC Curve** (**ROC** (**Receiver Operating Characteristic**)) and **Enrichment Plotter** nodes give options to evaluate a certain model's performance visually. Because the views are not too interactive, you have to specify every parameter upfront in the configuration dialog.

In the **ROC Curve** configuration, you have to select the binominal **Class column** and the label (**Positive class value**) to which the probabilities belong. This way, you will be able to compare different kinds of models or models with different parameters. The node also provides the areas beneath the ROC curve in the result table.

The **Enrichment Plotter** node helps you decide where to set the cut-off point to select the hits. The node description gives a more detailed guide on how to use it.

JFreeChart

The **JFreeChart** nodes are not installed by default, but the extension is available from the standard KNIME update site under the name **KNIME JFreeChart**.

The common part of these nodes is that you have to specify the appearance of the result in advance, and the focus is not on the view, but on the resulting image port object.

In the **General Plot Options Configuration** tab, you can specify the type of the resulting image (PNG or SVG), the size, the title, colors, and the font size (relative to the standard font for each item printed).

You can use the port objects in the reports, but you can also use them to check certain properties if you iterate through a loop and convert the result with **Image To Table**.

It is important to note that the customizable **JFreeChart View** tab is only available in freshly executed nodes. The generated image can be visualized either using the view or the image output.

In the **JFreeChart View** tab, you can customize (from the context menu) almost every aspect of the diagram (fonts, colors, tics, ranges, orientation, and outline style). This way, the output can be of quite a high quality. It is also important to note that the export is easier: you can use the **Copy** option to copy it to the clipboard or directly use the **Save as...** option to save it as a PNG file, and because there are no visible controls, you do not have to cut them off.

These nodes do not support HiLiting, but they provide tooltips about values. The support for properties is usually not implemented.

You can zoom in on these nodes by selecting a region (left to right, top to bottom) and zoom out by selecting in the opposite direction. You can also use the context menu's zooming options. (It seems that you cannot move around using the mouse or keyboard, so you have to zoom out and select another region if you want to see the details of that region.)

The Bar charts

The **Bar Chart (JFreeChart)** node's view is similar to a usual histogram, but it does not allow any other aggregation other than the count function, and only nominal columns are accepted. The color of the first column can be specified, just like the labels for the axis. The nominal columns' values can be rotated, and the angle can be set. You can also enable/disable the legends.

The **GroupBy Bar Chart (JFreeChart)** node's configuration is similar, except in this case, the nominal column is a single column (it can also be numeric), and the rest of the numeric columns can be visualized against it. It is important to note that the binning column should contain unique values. (The numeric values are grouped by these values.)

The Bubble chart

The **Bubble Chart (JFreeChart)** node's view is analogous to the **Scatter Plot** view, but in this case, you cannot set the color and the shape, but the color is not opaque. It also cannot handle nominal columns, so you have to convert them to numbers if you want to plot them against other columns. You must specify the x and y positions of the bubbles, just like their radius.

Heatmap

The **Heatmap (JFreeChart)** node is capable of visualizing not just the values in multiple columns, but also the distances from the other color-coded rows, when a distance column is available.

The extreme colors can be specified in the **HeatMap (JFreeChart)** node's configuration for the minimal and the maximal distance, and the legend can also be visible or hidden. The labels for the axes can be specified, and the tooltip is also available on demand.

The Histogram chart

This is a bit different from the histogram views previously introduced. In this view, the histograms can be either behind or in front of other histograms. The different ranges are shown on the same scale, so some of them can be wider while the others are narrower.

The color of the bars is only adjustable for the first column. The histograms are plotted in order, the last is at the back, while the first is in the front. You cannot change the order of the histograms from the view of **Histogram (JFreeChart)**.

The Interval chart

The **Interval Chart (JFreeChart)** node's view is not so interesting when your label is not unique (or the order is not defined by its alphabetical order). But this view supports the time values without the need to transform your data with time information before visualization, focusing on that information.

You can specify the grouping nominal column (**Label**) and the start and end positions of the time intervals. Each row represents an interval.

It supports the color properties, so you can create overlapping intervals with different colors.

The Line chart

The **Line Chart (JFreeChart)** node's view is quite similar to the regular **Line Plot** view, except in this case, you cannot have dots to show the values. However, there is an extra input table to specify the colors of the series.

The other difference is that when specified, you can use the numeric or date column's values instead of the rows for the values of other columns; however, the connections are still done by the adjacent rows.

The Pie chart

The **Pie Chart (JFreeChart)** node's view is similar to the **Pie Chart** node, but it is less interactive. It still uses the color properties (as opposed to the other JFreeChart nodes) and can draw the pie in 3D.

The Scatter plot

The **Scatter Plot (JFreeChart)** node uses the shape and color properties, so it can visualize at most four columns. This is still quite static but configurable, and the result looks good (it can contain the legend, so it is practically ready to paste).

This node is quite constant too; you have to decide which columns should be there in the configuration dialog.

Open Street Map

In the **KNIME Labs Extensions** (available from the main KNIME update site) you can install the **KNIME Open Street Map Integration** in order to visualize spatial data.

This extension contains two nodes, **OSM Map View** and **OSM Map to Image**. The first one is the interactive, you can browse the map and check the data points (the tooltips can give details about them), think find the distribution of interesting points by HiLiting them. (HiLiting cannot be done using these nodes, but you can select area "blindly" if you use a **Scatter Plot** with the longitude and latitude information.)

Both nodes require coordinates to be in the range of -90 to 90 for latitude and -180 to 180 for longitude if there is an input table (which is optional). The image node's configuration includes a map to select which area should be visible on the resulting image, the configuration for the coordinates is on the **Map Marker** tab.

In the **OSM Map View**, you can browse by holding the right mouse button down and moving around. Zooming is configured for double-click and mouse wheel.

3D Scatterplot

We are highlighting a view from the many third party views because this is really neatly done, and you might not find it initially interesting if you do not work with chemical data.

In the **Erl Wood Open Source Nodes** extension (from the community update site), you can find a node called **2D/3D Scatterplot**. It allows you to plot 3D data and still use KNIME The HiLite functionality and the color, and size properties (but that can also be selected on demand). This is a very well designed and implemented view node. Its configuration is limited to column filtering and the number of rows/distinct values that should be on the screen.

This node does not support the automatic generation of a diagram. It's more focused towards exploration and not towards creating final figures.

It can also provide a regression fit line in 2D mode. It can be a good alternative to the normal **Scatter Plot** node too (unless you need the shape properties).

A right-click on the canvas gives information about the nearest point as a tooltip, which can be very useful when you need more information about the other dimensions (even the chemical structures and images are rendered nicely).

In the 3D mode, you can select points while holding down the *Ctrl* key.

Other visualization nodes

There are many options to show data, and you really do not have to limit yourself with those which are bundled with KNIME. In the community contributions (`http://tech.knime.org/community`), there are many options available. We will cherry-pick some of the more general and interesting visualization nodes.