MRCET CAMPUS

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION – UGC, GOVT. OF INDIA)

(Emerging Technologies)

DATA SCIENCE

B.TECH III YEAR – I SEM (R20 Regulation)
2023-2024

STATISTICAL FOUNDATIONS LAB MANUAL

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of CSE-Emerging Technologies

Vision

- "To be at the forefront of Emerging Technologies and to evolve as a Centre of Excellence in Research, Learning and Consultancy to foster the students into globally competent professionals useful to the Society."

Mission

The department of CSE (Emerging Technologies) is committed to:

- To offer highest Professional and Academic Standards in terms of Personal growth and satisfaction.
- Make the society as the hub of emerging technologies and thereby capture opportunities in new age technologies.
- To create a benchmark in the areas of Research, Education and Public Outreach.
- To provide students a platform where independent learning and scientific study are encouraged with emphasis on latest engineering techniques.

QUALITY POLICY

- To pursue continual improvement of teaching learning process of Undergraduate and Post Graduate programs in Engineering & Management vigorously.
- To provide state of art infrastructure and expertise to impart the quality education and research environment to students for a complete learning experiences.
- Developing students with a disciplined and integrated personality.
- To offer quality relevant and cost effective programmes to produce engineers as per requirements of the industry need.

For more information: www.mrcet.ac.in

INDEX

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

(R20A6703) Statistical Foundations of Data Science Lab

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

0 0 3 1.5

COURSE OBJECTIVES:

- The students are exposed to various experimental skills in data analytics which is very essential forData Science.
- Students are exposed to the Probability distribution using R & Python Programming.
- Students are able to use R and Python Programming and perform all types of operators andfunctions to generate the effective reports.
- To inculcate in students professional and ethical attitude, multidisciplinary approach and an ability to relate statistical analysis in data science by using various statistical methods or principles.
- Students should aware of Hypothetical Tests, Regression Analysis and Monte Carlo Integration.
- To provide student with an academic environment aware of excellence, written ethical codesand guidelines and lifelong learning needed for a successful bright and professional career.

Required Software Tools: R and Python (numpy, scipy, matplotlib)Also required: Pandas, Statsmodels and Seaborn.

COURSE OUTCOMES:

- The student learns the concept of R and Python Programming and Statistical analysis and try formulate new solutions or programs.
- Demonstrate an ability to design and develop R and Python programs with this, analysis the data and generate the related report or results.
- Demonstrate an ability to design programming on probability distribution and computed all possible outcomes or required reports.
- Able to do hypothetical analysis and transformation of data into useful manner using Python Programming
- Able to generate the linear algebra, Monte Carlo Integrations etc by using Python programming.

WEEK 1: Study R Languages, Commands, etc
Consider 50 observations (dataset), generating random data using functions provided, like rbinom, performing basic statistical computations using built-in functions of R. Discussion of R graphics. Histograms. Stem and leaf plots. Boxplots. Scatterplots. Bar graphs plotting the data using line graph, histograms, multiple graphs, etc. Generate 3D graphs or plots.

WEEK 2: Measures of Central Tendency: Given a sample of 50 Observations (from any dataset), use possible functions R or Python and calculate mean, sd, var, min, max, median, range, and quantile. Discuss the properties of this distribution. generate bell curve of a random normal distribution.

WEEK 3: Pragmatic matters. Tabulating data. Transforming a variable. Subsetting vectors anddata frames.

WEEK 4: (i) Consider 100 observations, find out Correlation "cor()" and Covariance "cov()" and programs on Frequencies and Crosstabs.
(ii) Finding and analyze the missing data.

WEEK 5: Sorting, transposing and merging data. Reshaping a data frame. Basics of text processing.Reading unusual data files. Basics of variable coercion.

WEEK 6: Hypothesis testing and t-test for any given dataset. Find out null hypothesis, alternate hypothesis, draw the picture (graph) to visualize problem. Test the value of population mean.

WEEK 7: State alpha level and rejection region, estimate the maximum likely hood and inference.

WEEK 8: Binomial simulation: Making the computer flip coins for you. Make use of rbinomfunction of R to generate samples, and other functions: counts, avgs, mean, sd, sqrt, hist (histogram).

WEEK 9: Bayesian Hypothesis testing on any given dataset or dataframe.

WEEK 10: Use seaborn and combines simple statistical fits with plotting on pandas dataframes.

WEEK 11: Working on Linear Algebra and Linear Systems

WEEK 12: Working on Monte Carlo Integration (Quasi-random numbers and find out thevariance on any dataframe)

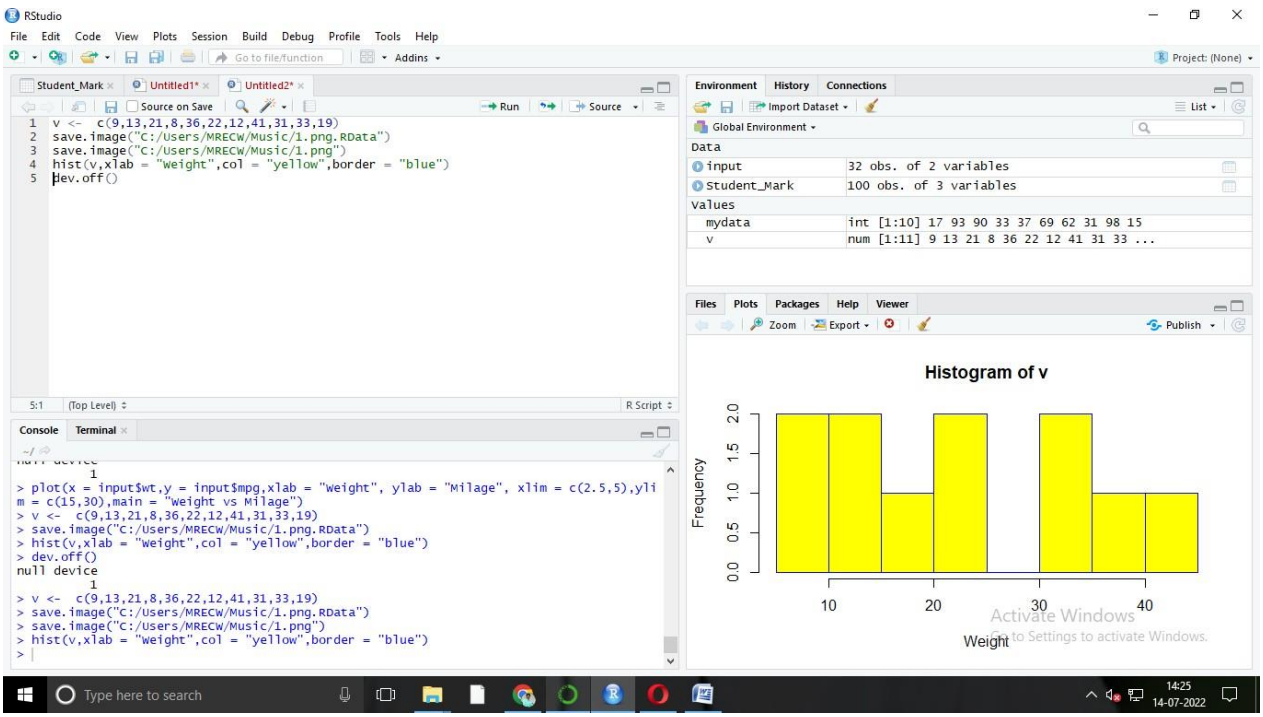
WEEK 1:Study R Languages, Commands, etc

Consider 50 observations (dataset), generating random data using functions provided, like rbinom, performing basic statistical computations using built-in functions of R.

```
mydata<- sample(1:nrow(Student_Mark), 10)

> mydata

 [1] 17 93 90 33 37 69 62 31 98 15

> Student_Mark[mydata,]

# A tibble: 10 x 3
   number_courses time_study Marks
            <dbl>      <dbl> <dbl>
 1              5     5.72  30.5
 2              4     5.03  23.9
 3              7     6.38  40.0
 4              8     0.932 15.0
 5              4     2.97  13.1
 6              4     1.40  8.92
 7              4     2.44  10.8
 8              8     3.86  24.2
 9              4     7.16  41.4
10              3     2.91  11.4
```

Discussion of R graphics. Histograms. Stem and leaf plots. Boxplots. Scatterplots. Bar graphs plotting the data using line graph, histograms, multiple graphs, etc. Generate 3D graphs or plots.

Histogram



v <- c(9,13,21,8,36,22,12,41,31,33,19)

save.image("C:/Users/MRECT/Music/1.png.RData")

save.image("C:/Users/MRECT/Music/1.png")

hist(v,xlab = "Weight",col = "yellow",border = "blue")

dev.off()

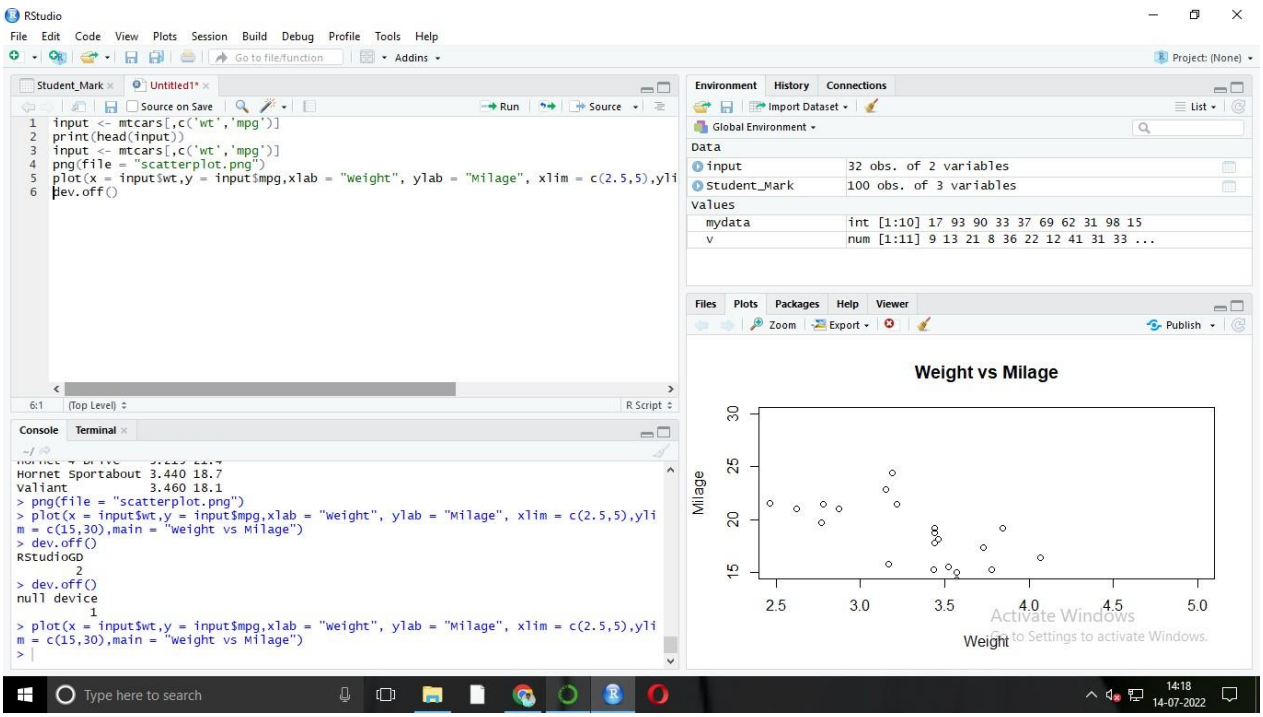
Stem and Leaf Plots

ChickWeight

	weight	Time	Chick	Diet
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1
6	93	10	1	1
7	106	12	1	1
8	125	14	1	1
9	149	16	1	1
10	171	18	1	1
11	199	20	1	1
12	205	21	1	1
13	40	0	2	1
14	49	2	2	1
15	58	4	2	1
16	72	6	2	1
17	84	8	2	1

```
18   103  10   2   1
19   122  12   2   1
20   138  14   2   1
21   162  16   2   1
22   187  18   2   1
23   209  20   2   1
24   215  21   2   1
25    43   0   3   1
26    39   2   3   1
27    55   4   3   1
28    67   6   3   1
29    84   8   3   1
30    99  10   3   1
31   115  12   3   1
32   138  14   3   1
33   163  16   3   1
34   187  18   3   1
35   198  20   3   1
36   202  21   3   1
37    42   0   4   1
38    49   2   4   1
39    56   4   4   1
[ reached 'max' / getOption("max.print") -- omitted 328 rows ]
stem(ChickWeight$weight)

  The decimal point is 1 digit(s) to the right of the |

   2 | 599999999
   4 | 000001111111111111111111122222222222222233334566788888888999999999+38
   6 | 001111111222222223333344444555556666777778888889001111112222223333 4+8
   8 | 00112223344444455555566777788999990001223333566666788888889
  10 | 000011112223333334566667778889901122223445555667789
  12 | 000022233333444455556677888901134445555667888 89
  14 | 111234444555566666777888900112344445556667777777789
  16 | 000022333344444667889000013444555578 9
  18 | 12244444555677782225677778889999
  20 | 0123444555557900245578
  22 | 0012357701123344556788
  24 | 08001699
  26 | 12344569259
  28 | 01780145
  30 | 355798
  32 | 12712
  34 | 1
  36 | 13
```

Scatter plot



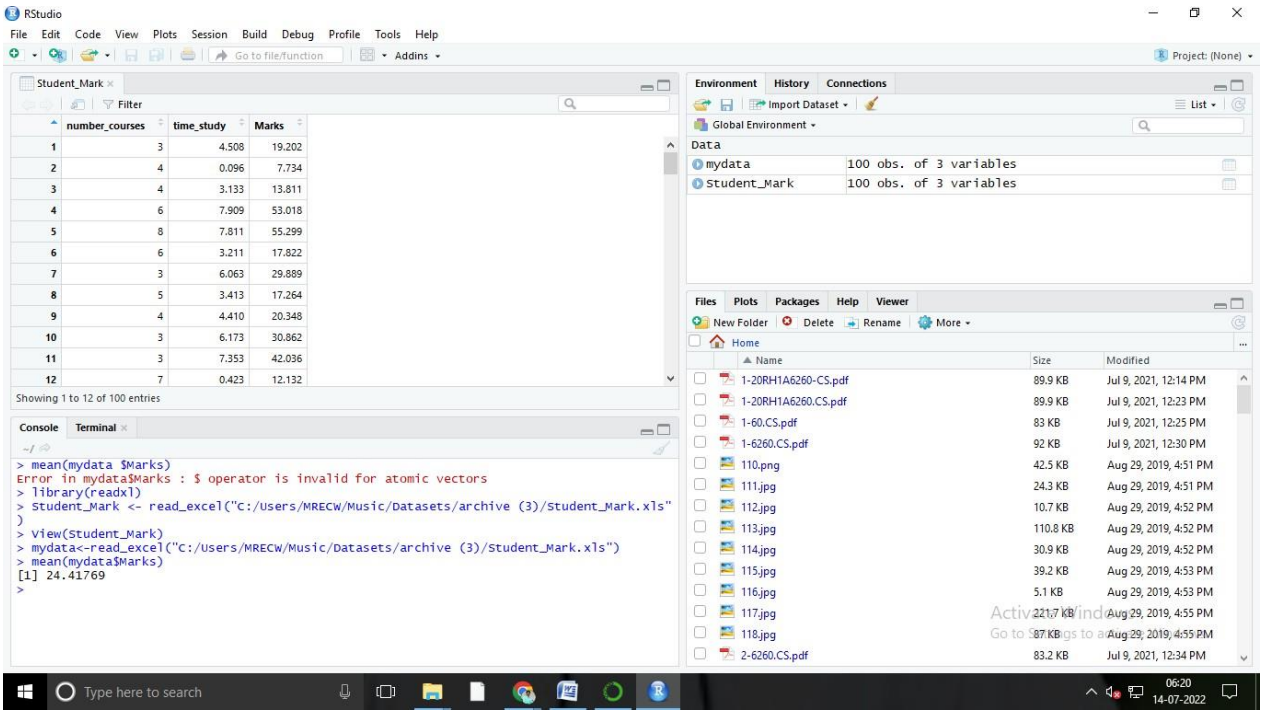

```
input <- mtcars[,c('wt','mpg')]

print(head(input))

input <- mtcars[,c('wt','mpg')]

png(file = "scatterplot.png")

plot(x = input$wt,y = input$mpg,xlab = "Weight", ylab = "Milage", xlim = c(2.5,5),ylim = c(15,30),main = "Weight vs Milage")
```

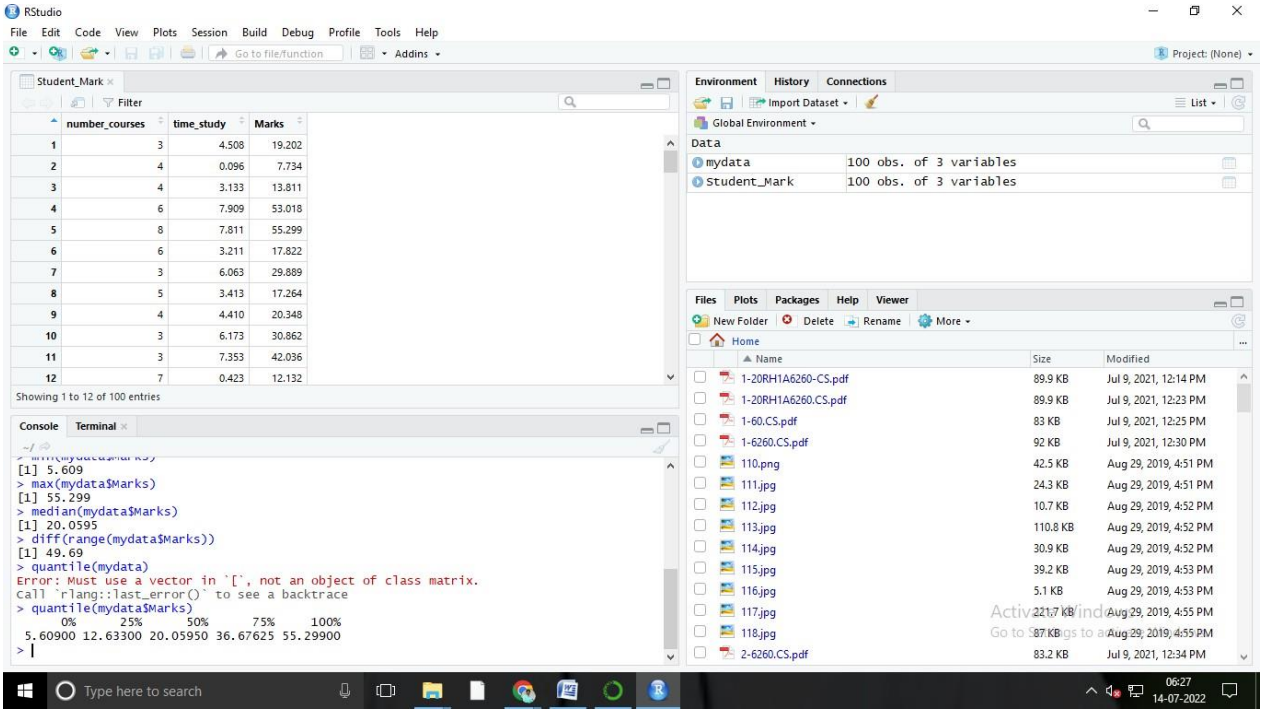
dev.off()

Bar graphs using line graphs

3D Graphs or Plots

WEEK 2: Measures of Central Tendency: Given a sample of 50 Observations (from any dataset), use possible functions R or Python and calculate mean, sd, var, min, max, median, range, and quantile. Discuss the properties of this distribution. generate bell curve of a random normal distribution.





```
Student_Mark <- read_excel("C:/Users/MRECW/Music/Datasets/archive (3)/Student_Mark.xls")
> View(Student_Mark)
> mydata<-read_excel("C:/Users/MRECW/Music/Datasets/archive (3)/Student_Mark.xls")
> mean(mydata$Marks)
[1] 24.41769
> sd(mydata$Marks)
[1] 14.3262
> var(mydata$Marks)
[1] 205.24
> min(mydata$Marks)
[1] 5.609
> max(mydata$Marks)
[1] 55.299
> median(mydata$Marks)
[1] 20.0595
> diff(range(mydata$Marks))
[1] 49.69
> quantile(mydata$Marks)
     0%      25%      50%      75%     100%
 5.60900 12.63300 20.05950 36.67625 55.29900
```

WEEK 3 :

Tabulating the Data using Python

Visualizing the data in tabular form is easier than visualizing it in a paragraph or comma-separated form. Nicely formatted tables not only provide you with a better way of looking at tables it can also help in understanding each data point clearly with its heading and value.
Tabulate is an open-source python package/module which is used to print tabular data in nicely formatted tables. It is easy to use and contains a variety of formatting functions. It has the following functionalities:

- One function call for all types of formatting
- Can be downloaded in multiple output formats
- Provides a better presentation with text and data.

In this week programs, we will see what are the different types of table formatting we can perform using Tabulate.

Implementation:

We will start by installing tabulate using **pip install tabulate.**

1. **Importing required libraries**

We will be using the tabulate function from the tabulate library so we need to import that. Other than this we do not require to import any python module.

```
from tabulate import tabulate
```

2. **Creating Formatted Tables**

Now we will start by creating different types of formatted tables.

```
data = [["Himanshu",1123, 10025], ["Rohit",1126,10029], ["Sha",111178,7355.4]]
print(tabulate(data))
```

Here we can see a plain table which is nicely formatted. Now let's see how we can add a header to this table that we just created.

```
print(tabulate(data, headers=["Name","User ID", "Roll. No."]))
```

In order to define the header along with the data, we can set that header='firstrow', let us see it through an example.

```
data = [['Name','ID'],["Himanshu",1123], ["Rohit",1126], ["Sha",111178]]
print(tabulate(data, headers='firstrow'))
```

We can also display the indices of the rows by using the show index parameter.

```
data = [['Name','ID'],["Himanshu",1123], ["Rohit",1126], ["Sha",111178]]
print(tabulate(data, headers='firstrow', showindex='always'))
```

tabular data types supported by tabulate

*The **tabulate** function can transform any of the following into an easy to read plain-text table:*

- *list of lists or another iterable of iterables*
- *list or another iterable of dicts (keys as columns)*

- *dict of iterables (keys as columns)*
- *two-dimensional NumPy array*
- *NumPy record arrays (names as columns)*
- *pandas.DataFrame*

I) list of lists

For example, if we have the following list of lists:

table = [['First Name', 'Last Name', 'Age'], ['John', 'Smith', 39], ['Mary', 'Jane', 25], ['Jennifer', 'Doe', 28]]

We can turn it into into a much more readable plain-text table using the ***tabulate*** function:

print(tabulate(**table**))

```
print(tabulate(table))
```

```
----------  ---------  ---
First Name  Last Name  Age
John        Smith      39
Mary        Jane       25
Jennifer    Doe        28
----------  ---------  ---
```

Since the first list in the list of lists contains the names of columns as its elements, we can set it as the column or header names by passing *'firstrow'* as the argument for the ***headers*** parameter:

print(tabulate(**table**, headers='firstrow'))

```
print(tabulate(table, headers='firstrow'))
```

```
First Name    Last Name      Age
------------  -----------  -----
John          Smith           39
Mary          Jane            25
Jennifer      Doe             28
```

The ***tabulate*** function also contains a ***tablefmt*** parameter, which allows us to improve the appearance of our table using pseudo-graphics:

print(tabulate(table, headers='firstrow', tablefmt='grid'))

```
print(tabulate(table, headers='firstrow', tablefmt='grid'))
```

```
+--------------+-------------+-------+
| First Name   | Last Name   |   Age |
+==============+=============+=======+
| John         | Smith       |    39 |
+--------------+-------------+-------+
| Mary         | Jane        |    25 |
+--------------+-------------+-------+
| Jennifer     | Doe         |    28 |
+--------------+-------------+-------+
```

Use the *'fancy_grid'* argument for ***tablefmt***:

```
print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
```

```
print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
```

First Name	Last Name	Age
John	Smith	39
Mary	Jane	25
Jennifer	Doe	28

II) dictionary of iterables

We can create the same table above using a dictionary:

```
info = {'First Name': ['John', 'Mary', 'Jennifer'], 'Last Name': ['Smith', 'Jane', 'Doe'], 'Age': [39, 25, 28]}
```

In the case of a dictionary, the ***keys*** will be the ***column headers***, and the **values** will be the **elements of those columns**. We specify that the keys will be the headers by passing *'keys'* as the argument for the ***headers*** parameter:

```
print(tabulate(info, headers='keys'))
```

```
print(tabulate(info, headers='keys'))
```

```
First Name    Last Name      Age
------------  -----------  -----
John          Smith           39
Mary          Jane            25
Jennifer      Doe             28
```

And of course we can use the ***tablefmt*** parameter to improve the table's appearance:

print(tabulate(**info**, headers='keys', tablefmt='fancy_grid'))

```
print(tabulate(info, headers='keys', tablefmt='fancy_grid'))
```

First Name	Last Name	Age
John	Smith	39
Mary	Jane	25
Jennifer	Doe	28

III) adding an index

We can also add an **index** to our table with the ***showindex*** parameter:

```
print(tabulate(info, headers='keys', tablefmt='fancy_grid', showindex=True))
```

	First Name	Last Name	Age
0	John	Smith	39
1	Mary	Jane	25
2	Jennifer	Doe	28

We can add a custom index by passing in an *iterable* to the ***showindex*** parameter. For example, if we want the index to start at 1, we can pass in a *range object* as the argument:

```
print(tabulate(info, headers='keys', tablefmt='fancy_grid', showindex=range(1,4)))
```

	First Name	Last Name	Age
1	John	Smith	39
2	Mary	Jane	25
3	Jennifer	Doe	28

IV) missing values

If we remove *'Jennifer'* from the above ***info*** dictionary, our table will contain an empty field:

```
print(tabulate({'First Name': ['John', 'Mary'], 'Last Name': ['Smith', 'Jane', 'Doe'], 'Age': [39, 25, 28]}, headers="keys", tablefmt='fancy_grid'))
```

First Name	Last Name	Age
John	Smith	39
Mary	Jane	25
	Doe	28

If there any **missing values** in our table, we can choose what to fill them in with using the ***missingval*** parameter. The default value for ***missingval*** is an empty string. If we change it to *'N/A'*, this is what what our table will look like:

```
print(tabulate({'First Name': ['John', 'Mary'], 'Last Name': ['Smith', 'Jane', 'Doe'], 'Age': [39, 25, 28]}, headers="keys", tablefmt='fancy_grid', missingval='N/A'))
```

First Name	Last Name	Age
John	Smith	39
Mary	Jane	25
N/A	Doe	28

For more detailed info about tabulate, visit: https://pypi.org/project/tabulate/

WEEK 4: (i) Consider 100 observations, find out Correlation "cor()" and Covariance "cov()" and programs on Frequencies and Crosstabs.

Let Σ(X) and Σ(Y) be the expected values of the variables, the covariance formula can be represented as:

$$\text{Covariance}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Where,

- xi = data value of x
- yi = data value of y
- $\bar{x}$ = mean of x
- $\bar{y}$ = mean of y

N = number of data values

What Is Correlation?

In statistics, correlation is a measure that determines the degree to which two or more random variables move in sequence. When an equivalent movement of another variable reciprocates the movement of one variable in some way or another during the study of two variables, the variables are said to be correlated. The formula for correlation is:

$$\rho_{xy} = Correlation\ (x, y) = \frac{cov(x, y)}{\sqrt{var(x)}\sqrt{var(y).}}$$

where,

var(X) = standard deviation of X

var(Y) = standard deviation of Y

Positive correlation occurs when two variables move in the same direction. When variables move in the opposite direction, they are said to be negatively correlated

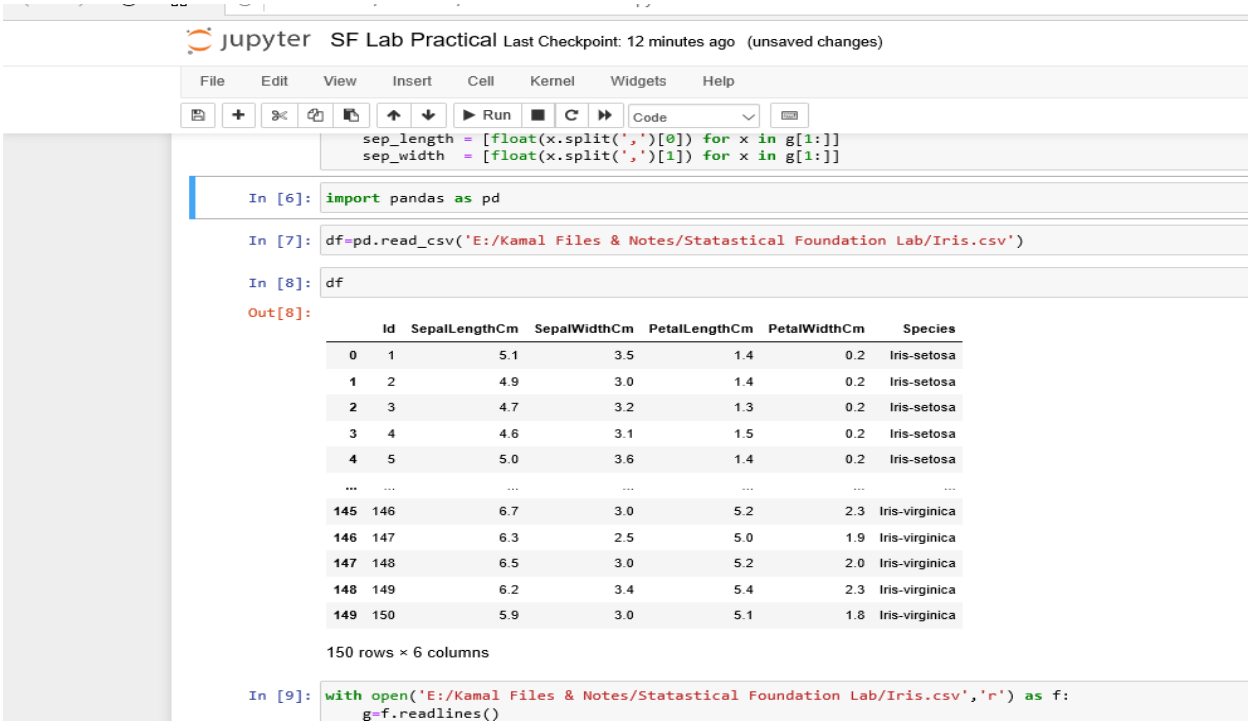
Let's we will be working on the well-known Iris dataset.

(Download IRIS Dataset from https://www.kaggle.com/datasets/uciml/iris)

Consider: **setosa** species field or attribute

```
>>>import pandas as pd
>>>df=pd.read_csv('E:/Kamal Files & Notes/Statastical Foundation Lab/Iris.csv')
```

Output Screen:



To Find Covariance:

```
def covariance(x, y):
    # Finding the mean of the series x and y
    mean_x = sum(x)/float(len(x))
    mean_y = sum(y)/float(len(y))
    # Subtracting mean from the individual elements
    sub_x = [i - mean_x for i in x]
    sub_y = [i - mean_y for i in y]
    numerator = sum([sub_x[i]*sub_y[i] for i in range(len(sub_x))])
    denominator = len(x)-1
    cov = numerator/denominator
    return cov

with open('<--file path--->', 'r') as f:
    cov_func = covariance(sep_length, sep_width)
```

print("Covariance from the custom function:", cov_func)

In [13]:
```
def covariance(x, y):
    # Finding the mean of the series x and y
    mean_x = sum(x)/float(len(x))
    mean_y = sum(y)/float(len(y))
    # Subtracting mean from the individual elements
    sub_x = [i - mean_x for i in x]
    sub_y = [i - mean_y for i in y]
    numerator = sum([sub_x[i]*sub_y[i] for i in range(len(sub_x))])
    denominator = len(x)-1
    cov = numerator/denominator
    return cov

with open('E:/Kamal Files & Notes/Statastical Foundation Lab/Iris.csv', 'r') as f:
    cov_func = covariance(sep_length, sep_width)
    print("Covariance from the custom function:", cov_func)
```

```
Covariance from the custom function: 25.782885906040274
```

To Find Correlation:

```
def correlation(x, y):
    # Finding the mean of the series x and y
    mean_x = sum(x)/float(len(x))
    mean_y = sum(y)/float(len(y))
    # Subtracting mean from the individual elements
    sub_x = [i-mean_x for i in x]
    sub_y = [i-mean_y for i in y]
    # covariance for x and y
    numerator = sum([sub_x[i]*sub_y[i] for i in range(len(sub_x))])
    # Standard Deviation of x and y
    std_deviation_x = sum([sub_x[i]**2.0 for i in range(len(sub_x))])
    std_deviation_y = sum([sub_y[i]**2.0 for i in range(len(sub_y))])
    # squaring by 0.5 to find the square root
    denominator = (std_deviation_x*std_deviation_y)**0.5 # short but equivalent to
(std_deviation_x**0.5) * (std_deviation_y**0.5)
    cor = numerator/denominator
    return cor
with open(<<Dataset Path>>, 'r') as f:
    cor_func = correlation(sep_length, sep_width)
    print("Correlation from the custom function:", cor_func)
```

```
In [12]: def correlation(x, y):
             # Finding the mean of the series x and y
             mean_x = sum(x)/float(len(x))
             mean_y = sum(y)/float(len(y))
             # Subtracting mean from the individual elements
             sub_x = [i-mean_x for i in x]
             sub_y = [i-mean_y for i in y]
             # covariance for x and y
             numerator = sum([sub_x[i]*sub_y[i] for i in range(len(sub_x))])
             # Standard Deviation of x and y
             std_deviation_x = sum([sub_x[i]**2.0 for i in range(len(sub_x))])
             std_deviation_y = sum([sub_y[i]**2.0 for i in range(len(sub_y))])
             # squaring by 0.5 to find the square root
             denominator = (std_deviation_x*std_deviation_y)**0.5 # short but equivalent to (std_deviation_x**0.5) * (std_deviation_y**0.5
             cor = numerator/denominator
             return cor

         with open('E:/Kamal Files & Notes/Statastical Foundation Lab/Iris.csv', 'r') as f:
             cor_func = correlation(sep_length, sep_width)
             print("Correlation from the custom function:", cor_func)

Correlation from the custom function: 0.7166762728539001
```

II Analyse the Missing Data (Using Python)

Working with Missing Data in Pandas

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- isnull()
- notnull()

Is null() and notnull()

```
# importing pandas package
import pandas as pd
# making data frame from csv file
data = pd.read_csv(<<<dataset_path>>>)
# creating boolean series True for NaN values
boolean_series = pd.isnull(data["Col_Name"])
#Col_Name is the one where the blank values are there
# filtering data
# displaying data only with Gender = NaN
data[boolean_series]
```

In [20]:
```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv('E:/Kamal Files & Notes/Statistical Foundation Lab/Dept_Table.csv')

# creating boolean series True for NaN values
boolean_series = pd.isnull(data["MANAGER_ID"])

# filtering data
# displaying data only with Gender = NaN
data[boolean_series]
```

Out[20]:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
11	120	Treasury	NaN	1700
12	130	Corporate Tax	NaN	1700
13	140	Control And Credit	NaN	1700
14	150	Shareholder Services	NaN	1700
15	160	Benefits	NaN	1700
16	170	Manufacturing	NaN	1700
17	180	Construction	NaN	1700
18	190	Contracting	NaN	1700
19	200	Operations	NaN	1700
20	210	IT Support	NaN	1700
21	220	NOC	NaN	1700
22	230	IT Helpdesk	NaN	1700

NOT NULL()

```
# importing pandas package
import pandas as pd
# making data frame from csv file
data = pd.read_csv(<<<dataset_path>>>)
# creating boolean series True for NaN values
boolean_series = pd.notnull(data["Col_Name"])
#Col_Name is the one where the blank values are there
# filtering data
# displaying data only with Gender = NaN
data[boolean_series]
```

In [22]:
```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv('E:/Kamal Files & Notes/Statistical Foundation Lab/Dept_Table.csv')

# creating boolean series True for NaN values
boolean_series = pd.notnull(data["MANAGER_ID"])

# filtering data
# displaying data only with Gender = NaN
data[boolean_series]
```

Out[22]:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
0	10	Administration	200.0	1700
1	20	Marketing	201.0	1800
2	30	Purchasing	114.0	1700
3	40	Human Resources	203.0	2400
4	50	Shipping	121.0	1500
5	60	IT	103.0	1400
6	70	Public Relations	204.0	2700
7	80	Sales	145.0	2500
8	90	Executive	100.0	1700
9	100	Finance	108.0	1700
10	110	Accounting	205.0	1700

To Count the Blank Cells or Missing values in a data_frame

To count how many blank cell are there in CSV file
data['Col_Name'].isna().sum()

In [24]:
```
# To count how many blank cell are there in CSV file
data['MANAGER_ID'].isna().sum()
```

Out[24]: 16

In [26]:
```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv('E:/Kamal Files & Notes/Statistical Foundation Lab/Dept_Table.csv')

# creating boolean series True for NaN values
boolean_series = pd.isnull(data["MANAGER_ID"])

# filtering data
# displaying data only with Gender = NaN
data[boolean_series]

count_nan = data['MANAGER_ID'].isna().sum()

print ('Count of NaN: ' + str(count_nan))
```

```
Count of NaN: 16
```

WEEK 5: Sorting, transposing and merging data. Reshaping a data frame. Basics of text processing. Reading unusual data files. Basics of variable coercion**.**

The various forms of reshaping data in a data frame are:

Transpose of a Matrix,Joining Rows and Columns,Merging of Data Frames

```
# R program to find the transpose of a matrix

first <- matrix(c(1:12), nrow=4, byrow=TRUE)
print("Original Matrix")
first

first <- t(first)
print("Transpose of the Matrix")
first
```

Output:

```
[1] "Original Matrix"
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
[1] "Transpose of the Matrix"
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Joining Rows and Columns in Data Frame

In R, we can join two vectors or merge two data frames using functions. There are basically two functions that perform these tasks:

cbind():

We can combine vectors, matrix or data frames by columns using **cbind()** function.
Syntax: *cbind(x1, x2, x3)*
where x1, x2 and x3 can be vectors or matrices or data frames.

rbind():

We can combine vectors, matrix or data frames by rows using **rbind()** function.
Syntax: *rbind(x1, x2, x3)*
where x1, x2 and x3 can be vectors or matrices or data frames.

```
# Cbind and Rbind function in R
name <- c("Shaoni", "esha", "soumitra", "soumi")
age <- c(24, 53, 62, 29)
address <- c("puducherry", "kolkata", "delhi", "bangalore")

# Cbind function
info <- cbind(name, age, address)
print("Combining vectors into data frame using cbind ")
print(info)

# creating new data frame
newd <- data.frame(name=c("sounak", "bhabani"),
                   age=c("28", "87"),
                   address=c("bangalore", "kolkata"))

# Rbind function
new.info <- rbind(info, newd)
print("Combining data frames using rbind ")
print(new.info)
```

Output:

```
[1] "Combining vectors into data frame using cbind "
     name       age address
[1,] "Shaoni"   "24" "puducherry"
[2,] "esha"     "53" "kolkata"
[3,] "soumitra" "62" "delhi"
[4,] "soumi"    "29" "bangalore"
[1] "Combining data frames using rbind "
      name age    address
1   Shaoni  24 puducherry
2     esha  53    kolkata
3 soumitra  62      delhi
4    soumi  29  bangalore
5   sounak  28  bangalore
6  bhabani  87    kolkata
```

Merging two Data Frames

In R, we can merge two data frames using the **merge()** function provided both the data frames should have the same column names. We may merge the two data frames based on a key value.

```
# Merging two data frames in R

d1 <- data.frame(name=c("shaoni", "soumi", "arjun"),

                 ID=c("111", "112", "113"))

 d2 <- data.frame(name=c("sounak", "esha"),

                 ID=c("114", "115"))

 total <- merge(d1, d2, all=TRUE)

print(total)
```

Output:

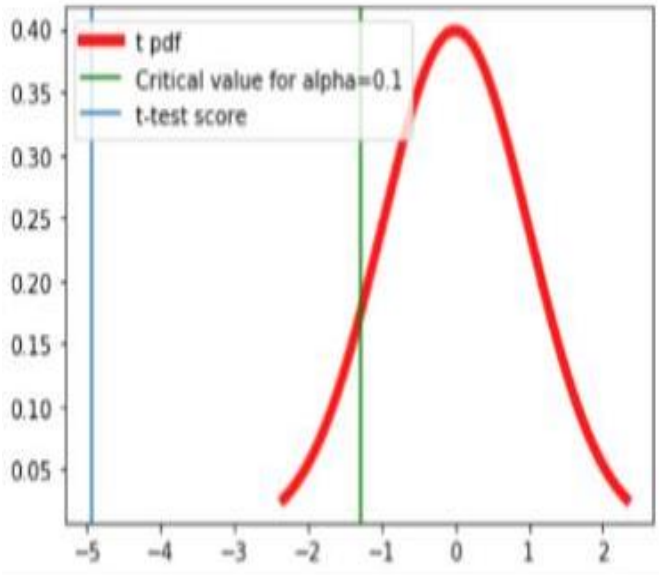
```
  name  ID
1  arjun 113
2 shaoni 111
3  soumi 112
4   esha  115
5 sounak 114
```

WEEK 6: Hypothesis testing and t-test for any given dataset. Find out null hypothesis,alternate hypothesis, draw the picture (graph) to visualize problem. Test the value of population mean.

```
# Manual method
x = df['Overall Performance'].mean()
s = df['Overall Performance'].std()
S_E = s/(1000**0.5)
t_statistic = (x-70)/(S_E)
print("The t_statistic of this test is ", t_statistic)
```

```
The t_statistic of this test is  -4.944863135709332
```

```

from scipy.stats import t
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1)
dof = 999
x = np.linspace(t.ppf(0.01, dof), t.ppf(0.99, dof), 1000)
ax.plot(x, t.pdf(x, dof),
       'r-', lw=5, alpha=0.9, label = 't pdf')
plt.axvline(x=t.ppf(0.1,dof), label='Critical value for alpha=0.1', color='g')
plt.axvline(t_statistic, label='t-test score')
plt.legend()
plt.plot();
```



Step 1: Collect Data

```
import pandas as pd

data = pd.read_csv('diameter.csv')
```

Step 2: Define Null and Alternative Hypotheses

```
H0 = 'Data is normal'

Ha = 'Data is not normal'
```

Step 2: Set the level of significance (α) = 5%

```
alpha = 0.05
```

Step 3: Run a test to check the normality

I am using the Shapiro test to check the normality.

```
from scipy.stats import

shapirop =

round(shapiro(data)[1], 2)
```

Step 4: Conclude using the p-value from step 3

```
if p > alpha:

   print(f"{p} > {alpha}. We fail to reject Null Hypothesis.

{H0}")else:

       print(f"{p} <= {alpha}. We reject Null Hypothesis. {Ha}")
```

The above code outputs "**0.52 > 0.05. We fail to reject Null Hypothesis.Data is Normal.**"

WEEK 7: State alpha level and rejection re-gion, estimate the maximum likely hood and inference.

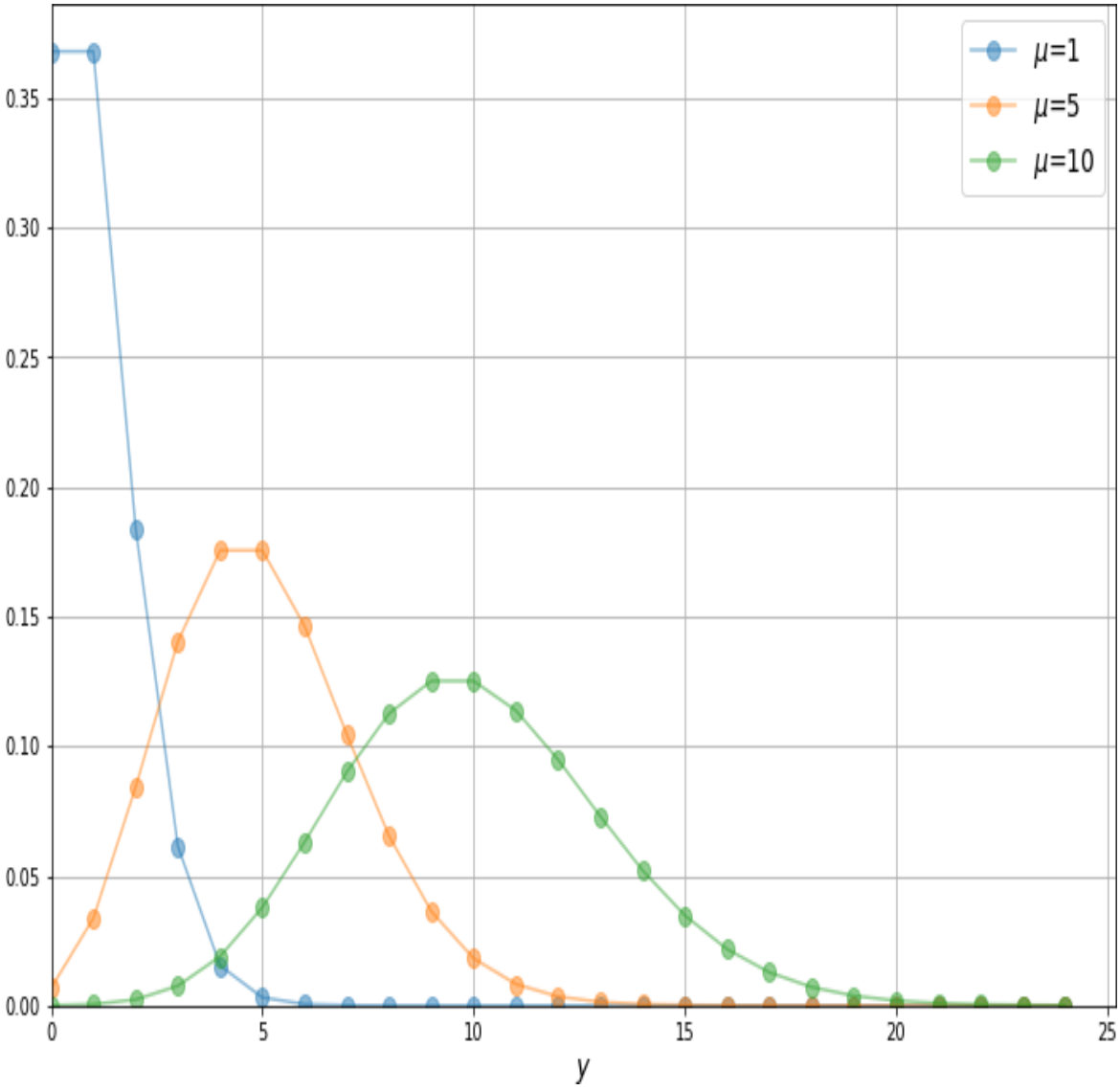
import numpy as np import pandas as pd
from matplotlib import pyplot as pltimport seaborn as sns
from statsmodels import apifrom scipy import stats
from scipy.optimize import minimize

Generate some synthetic data based on the assumptionof Normal Distribution.

# generate an independent variablex = np.linspace(-10, 30, 100)
# generate a normally distributed residuale = np.random.normal(10, 5, 100)
# generate ground truthy = 10 + 4*x + e
df = pd.DataFrame({'x':x, 'y':y})df.head()

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
from numpy import exp
from scipy.special import factorial
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import statsmodels.api as sm
from statsmodels.api import Poisson
from scipy import stats
from scipy.stats import norm
from statsmodels.iolib.summary2 import summary_col
```

```
poisson_pmf = lambda y, μ: μ**y / factorial(y)
* exp(-μ)y_values = range(0, 25)

fig, ax = plt.subplots(figsize=(12, 8))

for μ in [1, 5, 10]:
    distribution =
    [] for y_i in
    y_values:
        distribution.append(poisson_pmf(y
    _i, μ))ax.plot(y_values,
            distribution,
            label=f'$\mu$={μ}'
            , alpha=0.5,
            marker='o',
            markersize=8)

ax.grid()
ax.set_xlabel('$y$', fontsize=14)
ax.set_ylabel('$f(y \mid \mu)$',
fontsize=14)ax.axis(xmin=0, ymin=0)
ax.legend(fontsize=14)

plt.show()
```


μ=1
μ=5
μ=10
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
y

WEEK 8: Binomial simulation: Making the computer flip coins for you. Make use of rbinom function of R to generate samples, and other functions: counts, avgs, mean, sd, sqrt,hist (histogram).

```
# r binomial - binomial simulation in r
rbinom(7, 150,.05)
[1] 10 12 10 2 5 5 14
```

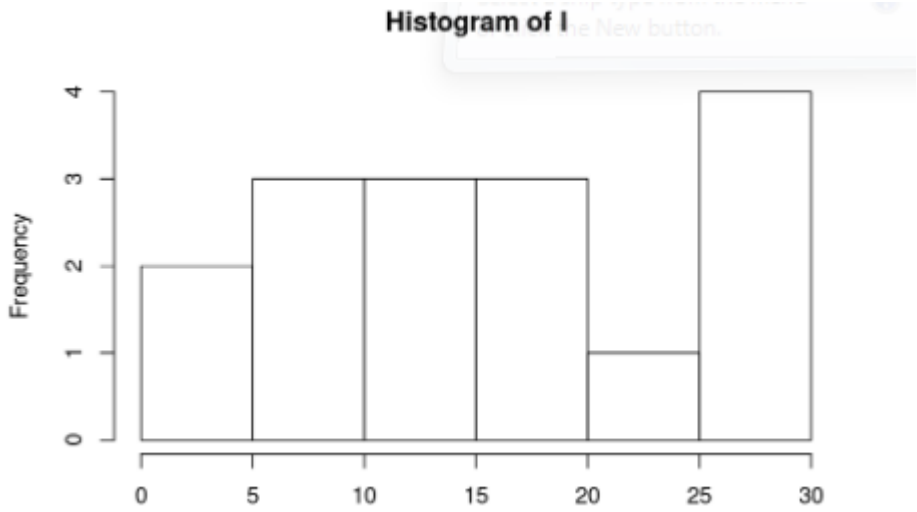
We can model individual Bernoulli trials as well. We do this be setting the trials attribute to one. Here is the outcome of 10 coin flips:

```
# bernoulli distribution in r
rbinom(10, 1,.5)
[1] 1 0 1 1 1 0 0 0 0 1
```

Or stepping it up a bit, here's the outcome of 10 flips of 100 coins:

```
# binomial simulation in r
rbinom(10, 100,.5)
[1] 52 55 51 50 46 42 50 49 46 56
```

```
l <- c(2, 3, 7, 8, 10, 13, 14, 15,
       18, 18, 20, 26, 25, 26, 27, 28)
hist(l)
```

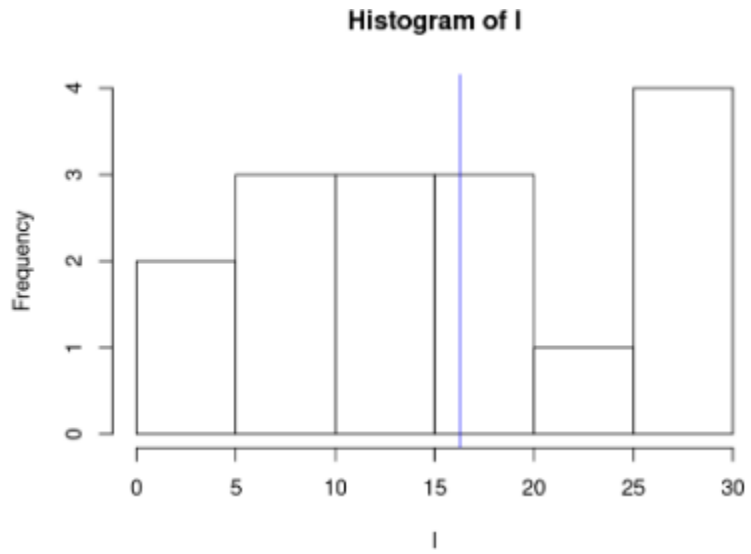


Adding Mean and Median

```
mean <- mean(l)     # Mean: 16.25

med <- median(l)    # Meadian: 16.5
```

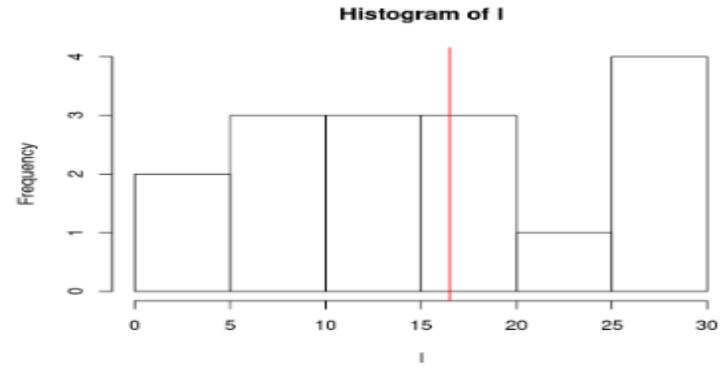
Example 1: **Add Mean to Histogram in R**

```
l <- c(2, 3, 7, 8, 10, 13, 14, 15,
       18, 18, 20, 26, 25, 26, 27, 28)

# Mean of l
mean <- mean(l)

# Plotting histogram and Adding
# Mean line to Histogram
hist(l)
abline(v = mean, col = 'blue')
```



Example 2: Add Median to Histogram in R

```
l <- c(2, 3, 7, 8, 10, 13, 14, 15,
       18, 18, 20, 26, 25, 26, 27, 28)
  # Median of l
med <- median(l)

# Plotting histogram and Adding
# Median line to Histogram
hist(l)
abline(v = med, col = 'red')
```



Computing Average in R Programming

```
# R program to get average of a list

# Taking a list of elements
list = c(2, 4, 4, 4, 5, 5, 7, 9)

# Calculating average using mean()
print(mean(list))
```

Output:

```
[1] 5
```

```
# R program to get average of a list

# Taking a list of elements
list = c(2, 40, 2, 502, 177, 7, 9)

# Calculating average using mean()
print(mean(list))
```

```
[1] 105.5714
```

Computing Variance in R Programming

```
#R program to get variance of a list


 # Taking a list of elements


list = c(2, 4, 4, 4, 5, 5, 7, 9)


 # Calculating variance using var()


print(var(list))
```

Output:

```
[1] 4.571429
```

```
# R program to get variance of a list

# Taking a list of elements
list = c(212, 231, 234, 564, 235)

# Calculating variance using var()
print(var(list))
```

```
[1] 22666.7
```

Standard Deviation in R

```
# R program to get
# standard deviation of a list

# Taking a list of elements
list = c(2, 4, 4, 4, 5, 5, 7, 9)

# Calculating standard
# deviation using sd()
print(sd(list))
```

OUTPUT

```
[1] 2.13809
```

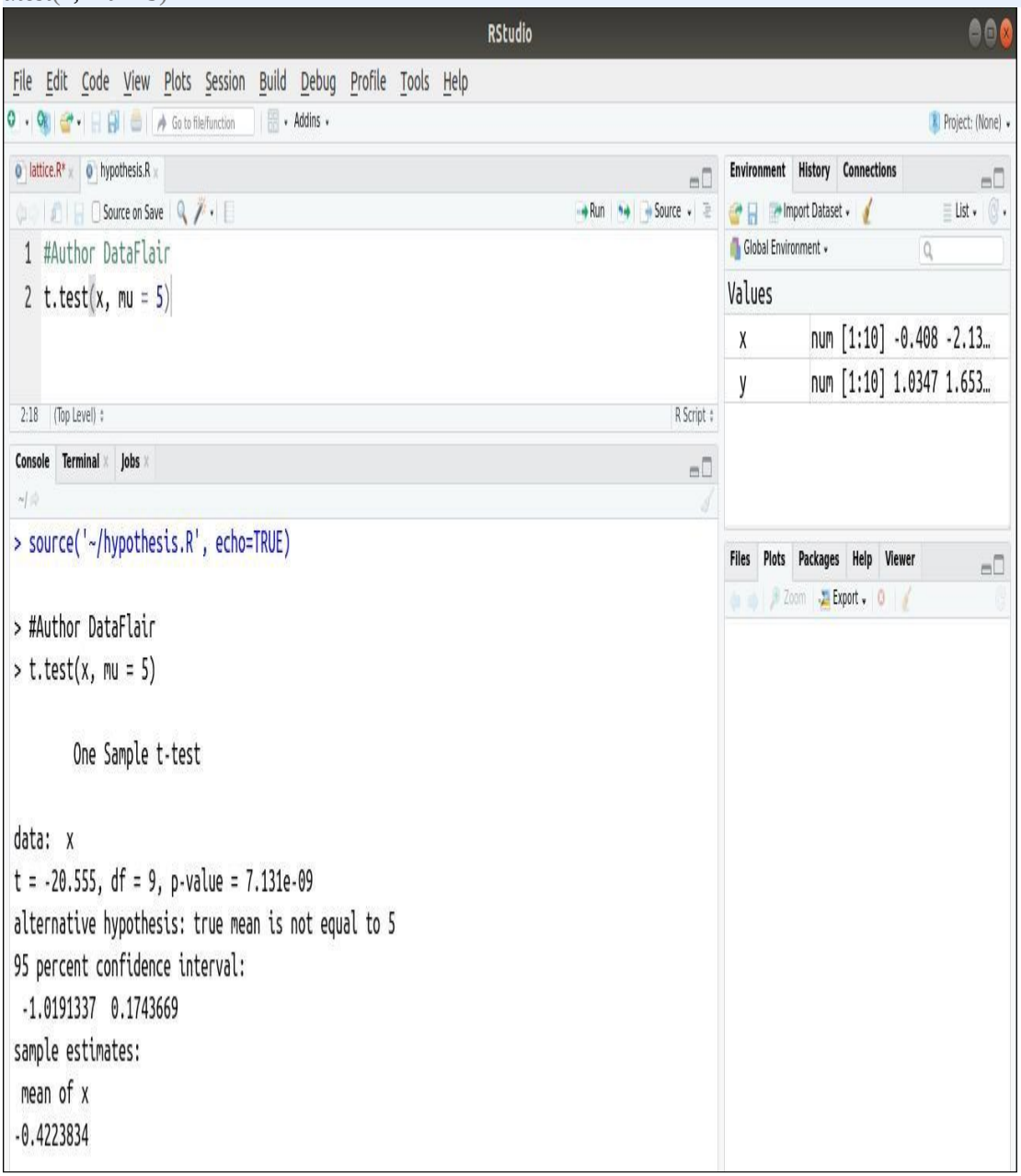
```
# R program to get
# standard deviation of a list

# Taking a list of elements
list = c(290, 124, 127, 899)

# Calculating standard
# deviation using sd()
print(sd(list))
```

OUTPUT

```
[1] 367.6076
```

WEEK 9: Bayesian Hypothesis testing on any given dataset or dataframe.

```
#Author DataFlair
t.test(x, mu = 5)
```




```
#Author DataFlair
t.test(y, mu = 5, alternative = 'greater')
```

```
# Defining sample vector
```

```
x <- rnorm(100)

# One Sample T-Test
t.test(x, mu = 5)
```

Output:

```
    One Sample t-test


data:  x
t = -49.504, df = 99, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 5
95 percent confidence interval:
 -0.1910645  0.2090349
sample estimates:
  mean of x
0.008985172
```

Two Sample T-Testing

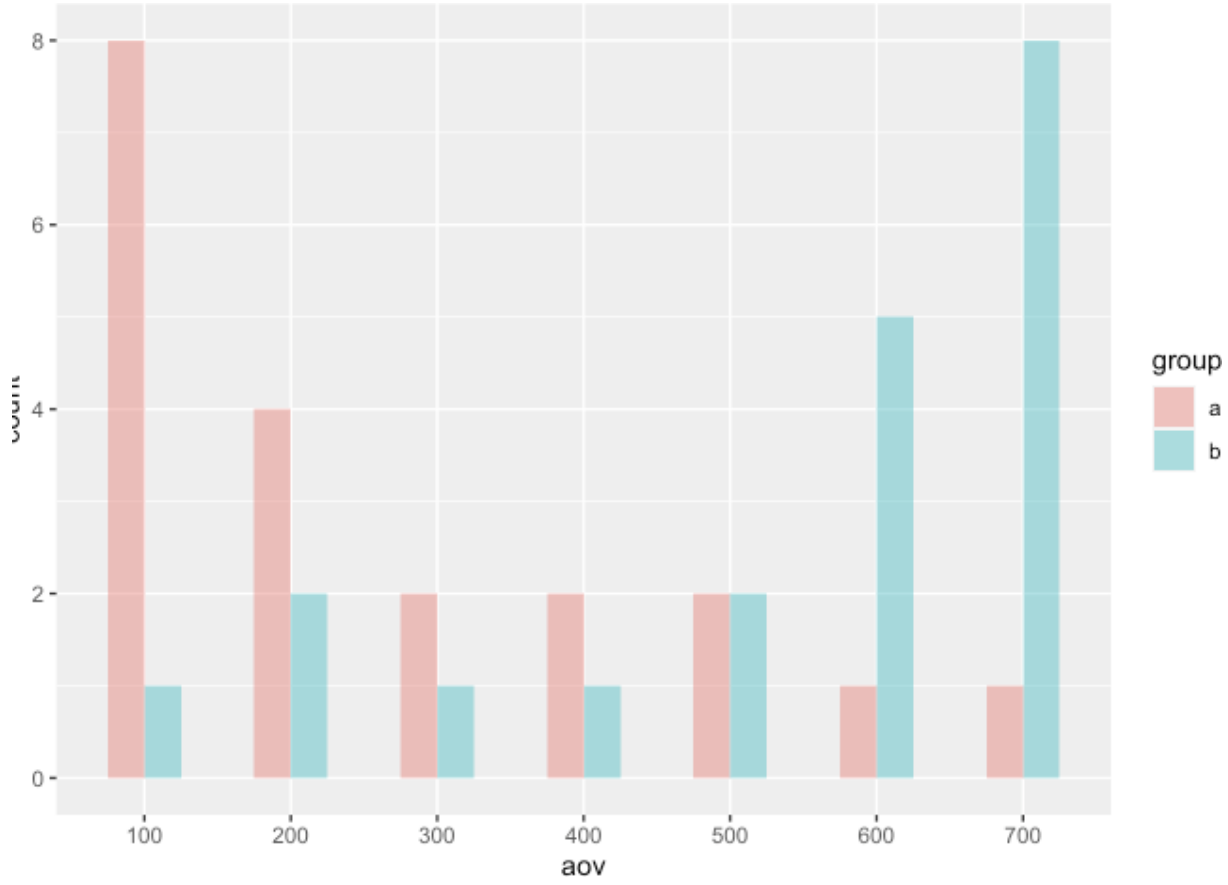
```
# Defining sample vector
x <- rnorm(100)
y <- rnorm(100)

# Two Sample T-Test
t.test(x, y)
Welch Two Sample t-test


data:  x and y
t = -1.0601, df = 197.86, p-value = 0.2904
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.4362140  0.1311918
sample estimates:
  mean of x   mean of y
-0.05075633  0.10175478
```

1. Example Data Frame.

```
#### Dummy example with 2 groups A/B composed of 20 observa-
tions each of Average order value per user.


group <-
c("a","a","a","a","a","a","a","a","a","a","a","a","a","a","a",
"a","a","a","a","a",

"b","b","b","b","b","b","b","b","b","b","b","b","b","b", "b",
"b", "b", "b", "b", "b")

aov <-
c(100,100,100,100,100,100,100,100,200,200,200,200,300,300,400,
400,500,500,600,700,

700,700,700,700,700,700,700,700,600,600,600,600,600,500,500,40
0,300,200,200,100)

df <- cbind(group, aov)
df <- as.data.frame(df)
```

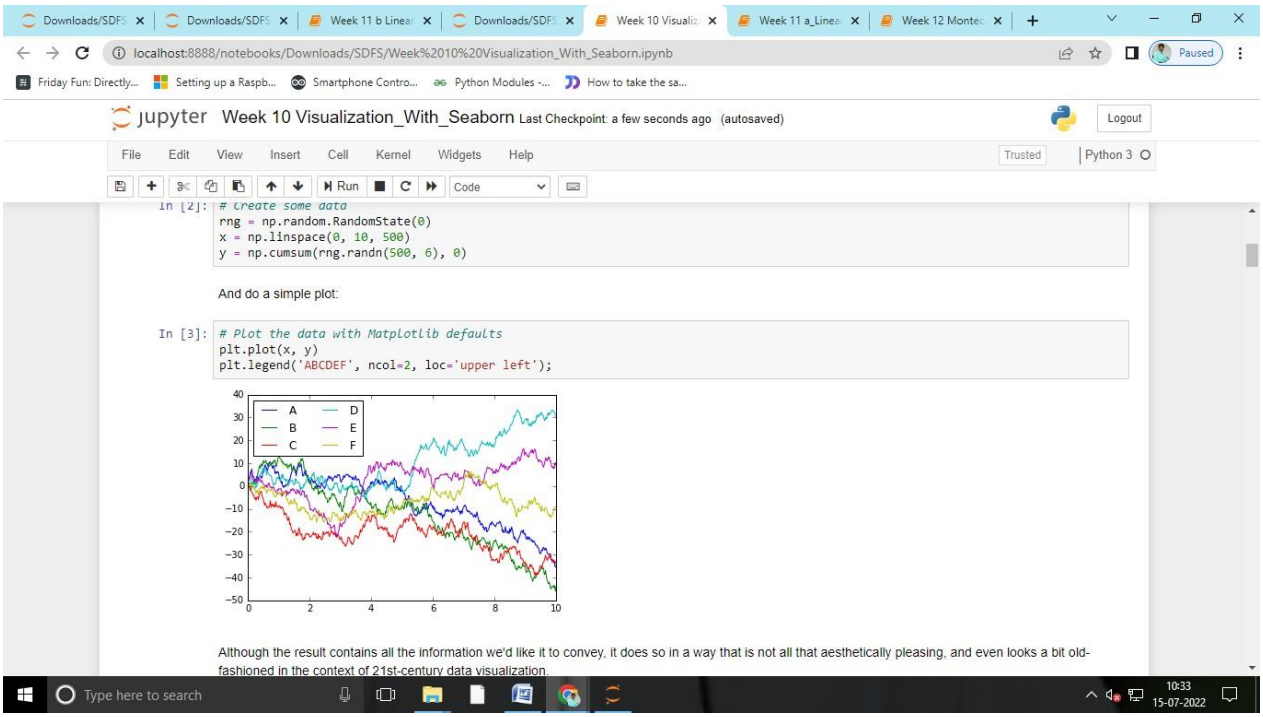
2. Distribution and overlap.

```
ggplot(df, aes(x=aov, fill=group)) +
  stat_count(width = 0.5, alpha=.5, position="dodge")
```



Group B targeting is likely to secure higher AOV.

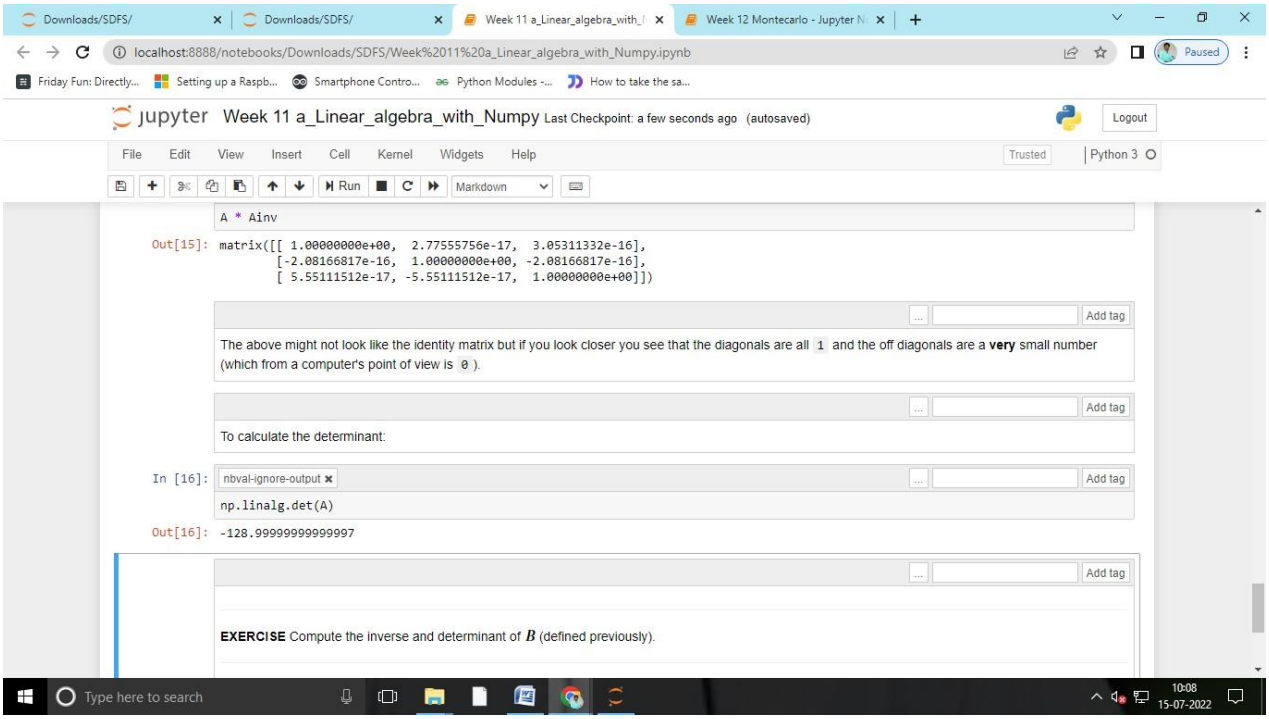
WEEK 10: Use seaborn and combines simple statistical fits with plotting on pandas dataframes.



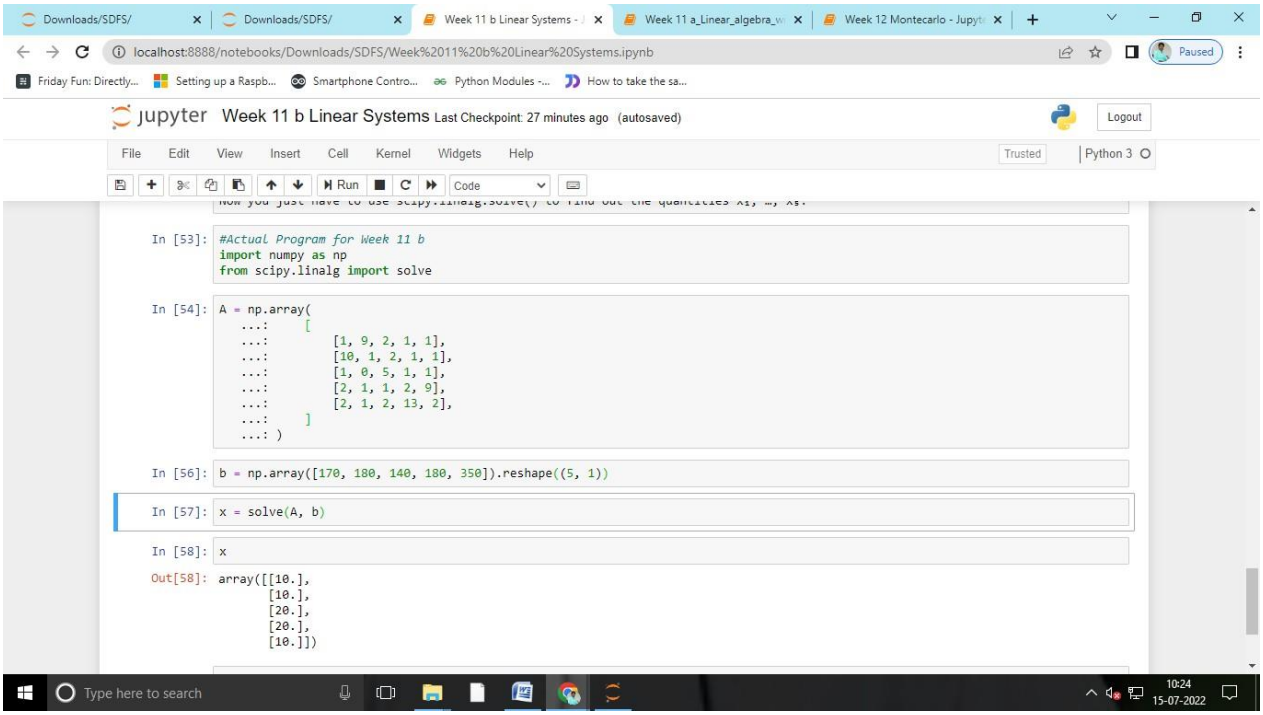
Solution: Juypter Notebook

WEEK 11: Working on Linear Algebra and Linear Systems

11 a) Linear Algebra- Juypter Notebook



11b) Working on Linear Systems – Solution Jupyter Notebook

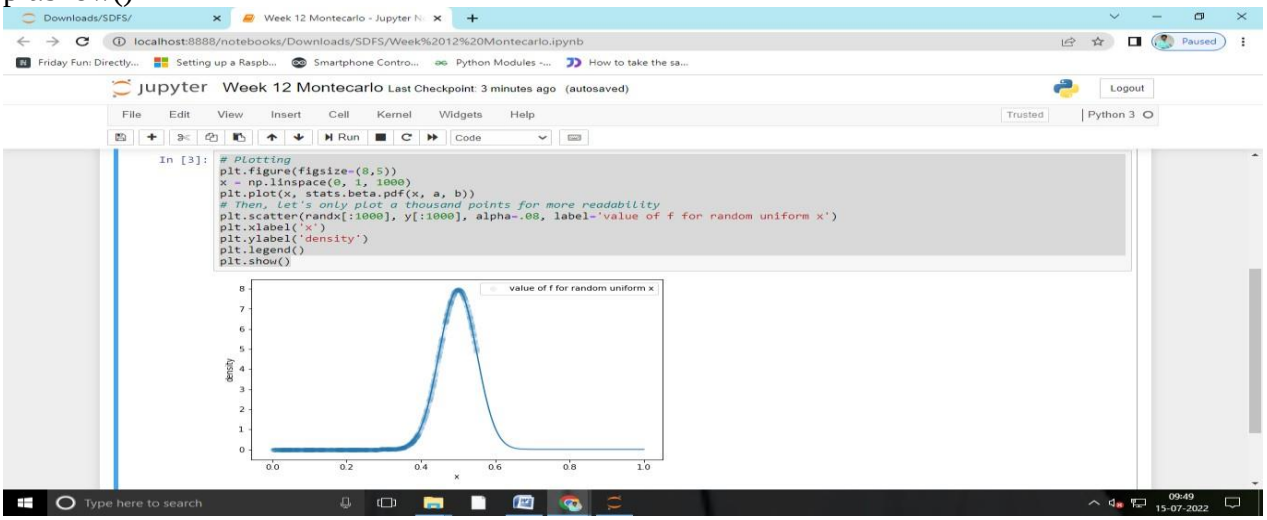


WEEK 12: Working on Monte Carlo Integration (Quasi-random numbers and find out the variance on any data frame)

```
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt

N = 10000
a, b = (50,50)
x_min, x_max = (0, .55)
randx = np.random.uniform(x_min, x_max, N)
y = stats.beta.pdf(randx, a, b)
print(f'Real value to find: {stats.beta(a,b).cdf(.55)}')
print(f'Integral value:  {(x_max-x_min)*y.sum()/N}')
print(f'Calculation error: {np.sqrt((x_max-x_min)*(y*y).sum()/N - (x_max-
x_min)*y.mean()**2)/np.sqrt(N)}')
```

```
Real value to find: 0.8413478010629016
Integral value:  0.8486374584821799
Calculation error: 0.019813380205560064
```

```
# Plotting
plt.figure(figsize=(8,5))
x = np.linspace(0, 1, 1000)
plt.plot(x, stats.beta.pdf(x, a, b))
# Then, let's only plot a thousand points for more readability
plt.scatter(randx[:1000], y[:1000], alpha=.08, label='value of f for random uniform x')
plt.xlabel('x')
plt.ylabel('density')
plt.legend()
plt.show()
```



The blue points correspond to the 10 000 values of f(x_i) computed from the uniform draws we made over X.