VIGNAN'S
Foundation for Science, Technology & Research
(Deemed to be University)
-Estd. u/s 3 of UGC Act 1956

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

STATISTICAL METHODS AND DATA VISUALIZATION (21HS134)

R21 Regulation Lab Manual

I B.TECH II SEM

Academic Year 2021-2022

21HS134

STATISTICAL METHODS AND DATA VISUALIZATION

Lab Manual

VFSTR Deemed to be University

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

LIST OF EXPERIMENTS

1. Creation of Python modules and packages:

Objective of this exercise is to create a python package and hence keep the coding simple, focus on organization, packaging, and testing and do the following:

a. Create a new folder with the proper directory structure for a python package called unit convert. Initialize this directory as a new git repository.

b. In the package directory, create two modules:

i.A module called temperature which implements two functions: fahrenheit_to_celsius and celsius_to_fahrenheit.

ii. A module called distance which implements two functions: miles_to_kilometers and kilometers_to_miles.

iii. Include __init__.py

c. Make sure your modules all have complete and properly-formatted doc-strings.

d. Write a test suite which checks each function in your package and run it using pytest

e. Write a setup.py script which makes your package installable.

f. Push your project to a public github repository

2. This exercise is to gain some hands-on experience with the Pandas library for analyzing tabular data.

a. Load data from CSV files and understand your data

b. Query and index operations on the above data frame

c. Insert, delete and update your data

d. Apply various filters on the data

e. Group, merge and aggregate data in the data frames

f. Identify and Fix missing values in the data

3. Visualize data with the help of the following graphical representations:

a. Line plots b. Bar plots

c. Error Plots d. Scatter plots

e. KDE Plots f. Heat Maps

g. Box Plots h. Pie graph

i. Histogram j. multiple graphs in single figure

k. saving figures

4. Sampling and Resampling:

• Generate a population of random numbers.

• Generate multiple samples using Random sampling with and without random sampling.

• Load a balanced dataset and visualize the class distribution.

• Load an imbalanced dataset and visualize the class distribution.

5. Interpreting Data Using Descriptive Statistics:

Compute Mean, Median, Mode, Standard Deviation, Variance, Co-variance, Interquartile Range and Skewness for two different datasets and write your interpretations about these statistical measures. Which measure is best suitable? Justify.

6. Generating Samples from Probability Distributions:

• Generate a set of random numbers (which corresponds to a uniform distribution)

using the function rand and plot its histogram. What is the shape of this histogram and why?

• Investigate how the shape of the histogram is affected by the number of random numbers you have generated.

• Similarly generate numbers using Bernoulli, Binomial distributions and plot a histogram and check the shape.

• Generate numbers using exponential and poisson distributions and plot a histogram and check the shape.

7. Implement the Central Limit Theorem in Python.

8. Hypothesis tests:

Implement the following three popular statistical techniques for hypothesis testing: Chisquare test, T-test and ANOVA test (Calculate the Test Statistic and P-value by running a Hypothesis test that well suits your data and Make Conclusions).

9. Linear Regression Analysis:

Download house prediction dataset and explore the data, Prepare the dataset for training,

Train a linear regression model, and Make predictions and evaluate the model.

TEXT BOOKS:

1. Thomas Haslwanter, "An Introduction to Statistics with Python With Applications in the Life Sciences, - Springer- ISSN 1431-8784 - ISBN 978-3-319-28315-9 Springer International Publishing Switzerland 2016.

2 Zed A. Shaw, "Learn Python 3 the Hard Way", 1st edition, Pearson Education Inc 2018,

REFERENCE BOOKS:

1. Peter Bruce, Andrew Bruce, Peter Gedeck, "Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python" 2nd edition, Oreilly Publishers, 2020.

2 Bharti Motwani, "Data Analytics using Python" , 1st edition, Wiley Publisheres, 2021

Manual

Experiment 1: Creating Python Modules and Packages

1. Create a new folder named Student_Roll

2. Inside Student_Roll, create a subfolder with the name ' unit_convert '

3. Create an empty __init__.py file in the unit_convert folder

4. Using a Python-aware editor like IDLE, create modules Celsius_to_from_Fahrenheit.py with the following code:

```
def convert_c(fahrenheit):
c = float(fahrenheit)
c = (c-32)*5/9
return(c)
def convert_f(celsius):
f = float(celsius)
f = (f * 9/5) + 32.
return(f)
```

5. Return to Student_Roll

6. Write python and write the following code

Importing Packages

1. from unit_convert import Celsius_to_from_Fahrenheit

2. Run Celsius_to_from_Fahrenheit.convert_c (100): 37.778

Importing modules from __init__.py

Modify __init__.py as below:

1. from .Celsius_to_from_Fahrenheit import convert_c, convert_f

Testing

1. Inside unit_convert create test_cf.py in the Student_Roll folder to test. Write the following code:

```
def test_convert_c(fahrenheit):
c = float(fahrenheit)
c = (c-32)*5/9
return(c)
def test_convert_f(celsius):
f = float(celsius)
f = (f * 9/5) + 32.
return(f)
```

2. Run pytest and check for errors

Creating setup.py

1. Inside Student_Roll create setup.py

2. Write the following code

```
from setuptools import setup, find_packages
setup(
name='unitconverter',
version='0.1.0',
packages=find_packages(include=['unit_convert', 'unit_convert.*'])
)
```

3. Run pip install -e . in the command prompt/terminal

Uploading project on GitHub

1. Sign in to GitHub and create a new empty repo

2. From your terminal, run the following commands after navigating to folder you would like to add:

a. git init

b. git add -A

c. git commit -m 'Added my project'

d. git remote add origin git@github.com: maiiDeep/unit-converter-project.git

e. git push -u -f origin master

Experiment 2: Creating hands-on experience with Pandas library

1. **Load data from CSV files and understand the data**
2. **Query and index operations on the above data frame**
3. **Insert, delete and update your data**
4. **Apply various filters on the data**
5. **Group, merge, and aggregate data in data frames**
6. **Identify and fix missing values in data**

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns. CSV stands for "Comma Separated Values." It is the simplest form of storing data in tabular form as plain text. It is important to know to work with CSV because we mostly rely on CSV data in our day-to-day lives as data scientists.

1. **Load data from CSV files and understand the data**

```
import pandas as pd
data= pd.read_csv("Salary_Data.csv")
print (data)
print (data.columns)
print (data.Salary)
print (data.index)
print (data.shape)
print (data.size)
```

Output:

```
    YearsExperience    Salary
0               1.1   39343.0
1               1.3   46205.0
2               1.5   37731.0
3               2.0   43525.0
4               2.2   39891.0
5               2.9   56642.0
6               3.0   60150.0
7               3.2   54445.0
8               3.2   64445.0
9               3.7   57189.0
10              3.9   63218.0
11              4.0   55794.0
12              4.0   56957.0
13              4.1   57081.0
14              4.5   61111.0
15              4.9   67938.0
16              5.1   66029.0
17              5.3   83088.0
18              5.9   81363.0
19              6.0   93940.0
20              6.8   91738.0
21              7.1   98273.0
22              7.9  101302.0
23              8.2  113812.0
24              8.7  109431.0
25              9.0  105582.0
26              9.5  116969.0
27              9.6  112635.0
28             10.3  122391.0
```

```
Index(['YearsExperience', 'Salary'], dtype='object')
```

```
0      39343.0
1      46205.0
2      37731.0
3      43525.0
4      39891.0
5      56642.0
6      60150.0
7      54445.0
8      64445.0
9      57189.0
10     63218.0
11     55794.0
12     56957.0
13     57081.0
14     61111.0
15     67938.0
16     66029.0
17     83088.0
18     81363.0
19     93940.0
20     91738.0
21     98273.0
22    101302.0
23    113812.0
24    109431.0
25    105582.0
26    116969.0
27    112635.0
28    122391.0
29    121872.0
Name: Salary, dtype: float64
```

```
RangeIndex(start=0, stop=30, step=1)
```

```
(30, 2)
```

```
60
```

2. Query and index operations on the above data frame

There are a lot of ways to pull the elements, rows, and columns from a DataFrame. There is some indexing method in Pandas which help in getting an element from a DataFrame. These indexing methods appear very similar but behave very differently. Pandas support four types of Multi-axes indexing they are:

I. Dataframe.[ ] ; This function also known as indexing operator
II. Dataframe.loc[ ] : This function is used for labels.
III. Dataframe.iloc[ ] : This function is used for positions or integer based

```
import pandas as pd
data= pd.read_csv("nba-2.csv", index_col ="Name")
# retrieving columns by indexing operator
first = data["Age"]
print(first)
#Selecting multiple columns
second  = data[["Age", "College", "Salary"]]
print(second)
# Selecting a single row
# retrieving row by .loc method
third = data.loc["Avery Bradley"]
fourth = data.loc["R.J. Hunter"]
print(third, "\n\n\n",fourth)
```

```
#Selecting multiple rows
fifth = data.loc[["Avery Bradley", "R.J. Hunter"]]
print(fifth)
# retrieving two rows and three columns by loc method
sixth = data.loc[["Avery Bradley", "R.J. Hunter"],
                 ["Team", "Number", "Position"]]
print(sixth)
# retrieving all rows and some columns by loc method
seventh = data.loc[:, ["Team", "Number", "Position"]]
print(seventh)
#Indexing a DataFrame using .iloc[ ]
# retrieving rows by iloc method
eighth = data.iloc[3]
print(eighth)
#Selecting multiple rows
ninth = data.iloc[[3, 5, 7]]
print(ninth)
# retrieving two rows and two columns by iloc method
tenth = data.iloc [[3, 4], [1, 2]]
print(tenth)
# retrieving all rows and some columns by iloc method
eleventh = data.iloc [:, [1, 2]]

print(eleventh)
```

Output:

One)

```
Name
Avery Bradley    25.0
Jae Crowder      25.0
John Holland     27.0
R.J. Hunter      22.0
Jonas Jerebko    29.0
                 ...
Shelvin Mack     26.0
Raul Neto        24.0
Tibor Pleiss     26.0
Jeff Withey      26.0
NaN               NaN
Name: Age, Length: 458, dtype: float64
```

Two

```
                Age            College     Salary
Name
Avery Bradley  25.0              Texas  7730337.0
Jae Crowder    25.0          Marquette  6796117.0
John Holland   27.0  Boston University        NaN
R.J. Hunter    22.0      Georgia State  1148640.0
Jonas Jerebko  29.0                NaN  5000000.0
...             ...                ...        ...
Shelvin Mack   26.0             Butler  2433333.0
Raul Neto      24.0                NaN   900000.0
Tibor Pleiss   26.0                NaN  2900000.0
Jeff Withey    26.0             Kansas   947276.0
NaN             NaN                NaN        NaN
```

Three & Four

```
Team        Boston Celtics
Number                 0.0
Position                PG
Age                   25.0
Height                 6-2
Weight               180.0
College              Texas
Salary           7730337.0
Name: Avery Bradley, dtype: object


 Team        Boston Celtics
Number                28.0
Position                SG
Age                   22.0
Height                 6-5
Weight               185.0
College      Georgia State
Salary           1148640.0
Name: R.J. Hunter, dtype: object
```

Fifth

```
                         Team  Number Position   Age Height  Weight        College     Salary
Name
Avery Bradley  Boston Celtics     0.0       PG  25.0    6-2   180.0          Texas  7730337.0
R.J. Hunter    Boston Celtics    28.0       SG  22.0    6-5   185.0  Georgia State  1148640.0
```

Sixth

```
                         Team  Number Position
Name
Avery Bradley  Boston Celtics     0.0       PG
R.J. Hunter    Boston Celtics    28.0       SG
```

Seventh

```
Name
Avery Bradley  Boston Celtics     0.0        PG
Jae Crowder    Boston Celtics    99.0        SF
John Holland   Boston Celtics    30.0        SG
R.J. Hunter    Boston Celtics    28.0        SG
Jonas Jerebko  Boston Celtics     8.0        PF
...                       ...     ...       ...
Shelvin Mack        Utah Jazz     8.0        PG
Raul Neto           Utah Jazz    25.0        PG
Tibor Pleiss        Utah Jazz    21.0         C
Jeff Withey         Utah Jazz    24.0         C
NaN                       NaN     NaN       NaN

[458 rows x 3 columns]
```

Eighth

```
[458 rows x 3 columns]
Team        Boston Celtics
Number                28.0
Position                SG
Age                   22.0
Height                 6-5
Weight               185.0
College      Georgia State
Salary           1148640.0
Name: R.J. Hunter, dtype: object
```

Ninth

```
                         Team  Number Position   Age Height  Weight        College      Salary
Name
R.J. Hunter    Boston Celtics    28.0       SG  22.0    6-5   185.0  Georgia State   1148640.0
Amir Johnson   Boston Celtics    90.0       PF  29.0    6-9   240.0            NaN  12000000.0
Kelly Olynyk   Boston Celtics    41.0        C  25.0    7-0   238.0        Gonzaga   2165160.0
```

Tenth

```
               Number Position
Name
R.J. Hunter      28.0       SG
Jonas Jerebko     8.0       PF
```

Eleventh

```
                 Number Position
Name
Avery Bradley       0.0       PG
Jae Crowder        99.0       SF
John Holland       30.0       SG
R.J. Hunter        28.0       SG
Jonas Jerebko       8.0       PF
...                 ...      ...
Shelvin Mack        8.0       PG
Raul Neto          25.0       PG
Tibor Pleiss       21.0        C
Jeff Withey        24.0        C
NaN                 NaN      NaN

[458 rows x 2 columns]
```

3. Insert, delete and update your data

(a) Insert Syntax:

DataFrameName.insert(loc, column, value, allow_duplicates = False)

```
import pandas as pd
# reading csv file
data = pd.read_csv("pokemon-3.csv",index_col ="#")
 # displaying  dataframe - Output 1
print (data)
 # inserting column with static value in data frame
data.insert(2, "Team", "Any")
# displaying data frame again - Output 2
# list output
print (data)
# inserting row using append
df2  =  pd.DataFrame  ({  'Name':  ['new'],    'Type':['Water'],'Team':
['Any'] })
data = data.append(df2, ignore_index = True)
print (data)
```

Output:

```
                    Name   Type
#
1              Bulbasaur  Grass
2                Ivysaur  Grass
3               Venusaur  Grass
3  VenusaurMega Venusaur  Grass
4             Charmander   Fire
                    Name   Type Team
#
1              Bulbasaur  Grass  Any
2                Ivysaur  Grass  Any
3               Venusaur  Grass  Any
3  VenusaurMega Venusaur  Grass  Any
4             Charmander   Fire  Any
D:\Vignan\SUBJECTS\Python Lab\Python lab
will be removed from pandas in a future 
  data = data.append(df2)
                    Name   Type Team
1              Bulbasaur  Grass  Any
2                Ivysaur  Grass  Any
3               Venusaur  Grass  Any
3  VenusaurMega Venusaur  Grass  Any
4             Charmander   Fire  Any
0                    new  Water  Any
```

(b) Delete a row in dataframe

Syntax:

DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')

Parameters:

labels: String or list of strings referring row or column name.

axis: int or string value, 0 ‘index’ for Rows and 1 ‘columns’ for Columns.

index or columns: Single label or list. index or columns are an alternative to axis and cannot be used together.

level: Used to specify level in case data frame is having multiple level index.

inplace: Makes changes in original Data Frame if True.

errors: Ignores error if any value from the list doesn’t exists and drops rest of the values when errors = ‘ignore’

Return type: Dataframe with dropped values

Delete rows and columns

```
import pandas as pd
# making data frame from csv file
data = pd.read_csv("nba-2.csv", index_col ="Name" )
print (data)
# dropping passed values
data.drop(["Avery Bradley", "John Holland", "R.J. Hunter",
                              "R.J. Hunter"], inplace = True)
 # display
print (data)
# dropping passed columns
data.drop(["Team", "Weight"], axis = 1, inplace = True)
# display
print (data)
```

```
                          Team  Number Position   Age Height  Weight            College     Salary
Name
Avery Bradley  Boston Celtics     0.0       PG  25.0    6-2   180.0              Texas  7730337.0
Jae Crowder    Boston Celtics    99.0       SF  25.0    6-6   235.0          Marquette  6796117.0
John Holland   Boston Celtics    30.0       SG  27.0    6-5   205.0  Boston University        NaN
R.J. Hunter    Boston Celtics    28.0       SG  22.0    6-5   185.0      Georgia State  1148640.0
Jonas Jerebko  Boston Celtics     8.0       PF  29.0   6-10   231.0                NaN  5000000.0
...                        ...     ...      ...   ...    ...     ...                ...        ...
Shelvin Mack         Utah Jazz     8.0       PG  26.0    6-3   203.0             Butler  2433333.0
Raul Neto            Utah Jazz    25.0       PG  24.0    6-1   179.0                NaN   900000.0
Tibor Pleiss         Utah Jazz    21.0        C  26.0    7-3   256.0                NaN  2900000.0
Jeff Withey          Utah Jazz    24.0        C  26.0    7-0   231.0             Kansas   947276.0
NaN                        NaN     NaN      NaN   NaN    NaN     NaN                NaN        NaN

[458 rows x 8 columns]
                          Team  Number Position   Age Height  Weight    College      Salary
Name
Jae Crowder    Boston Celtics    99.0       SF  25.0    6-6   235.0  Marquette   6796117.0
Jonas Jerebko  Boston Celtics     8.0       PF  29.0   6-10   231.0        NaN   5000000.0
Amir Johnson   Boston Celtics    90.0       PF  29.0    6-9   240.0        NaN  12000000.0
Jordan Mickey  Boston Celtics    55.0       PF  21.0    6-8   235.0        LSU   1170960.0
Kelly Olynyk   Boston Celtics    41.0        C  25.0    7-0   238.0    Gonzaga   2165160.0
...                        ...     ...      ...   ...    ...     ...        ...         ...
Shelvin Mack         Utah Jazz     8.0       PG  26.0    6-3   203.0     Butler   2433333.0
Raul Neto            Utah Jazz    25.0       PG  24.0    6-1   179.0        NaN    900000.0
Tibor Pleiss         Utah Jazz    21.0        C  26.0    7-3   256.0        NaN   2900000.0
Jeff Withey          Utah Jazz    24.0        C  26.0    7-0   231.0     Kansas    947276.0
```

```
               Number Position   Age Height    College      Salary
Name
Jae Crowder      99.0       SF  25.0    6-6  Marquette   6796117.0
Jonas Jerebko     8.0       PF  29.0   6-10        NaN   5000000.0
Amir Johnson     90.0       PF  29.0    6-9        NaN  12000000.0
Jordan Mickey    55.0       PF  21.0    6-8        LSU   1170960.0
Kelly Olynyk     41.0        C  25.0    7-0    Gonzaga   2165160.0
...               ...      ...   ...    ...        ...         ...
Shelvin Mack      8.0       PG  26.0    6-3     Butler   2433333.0
Raul Neto        25.0       PG  24.0    6-1        NaN    900000.0
Tibor Pleiss     21.0        C  26.0    7-3        NaN   2900000.0
Jeff Withey      24.0        C  26.0    7-0     Kansas    947276.0
NaN               NaN      NaN   NaN    NaN        NaN         NaN

[455 rows x 6 columns]
```

(c) Update a dataframe

```
import pandas as pd
 # making data frame from csv file
data = pd.read_csv("pokemon-3.csv", index_col ="Name" )
print (data)
#update
data['Type'] = data['Type'].replace({'Grass': 'Water'})
 # display
print (data)
```

Output:

```
                         #   Type
Name
Bulbasaur                1  Grass
Ivysaur                  2  Grass
Venusaur                 3  Grass
VenusaurMega Venusaur    3  Grass
Charmander               4   Fire
                         #   Type
Name
Bulbasaur                1  Water
Ivysaur                  2  Water
Venusaur                 3  Water
VenusaurMega Venusaur    3  Water
Charmander               4   Fire
```

4. Apply various filters on the data

Pandas dataframe.filter() function is used to Subset rows or columns of dataframe according to labels in the specified index

(a) Use filter() function to filter out any three columns of the dataframe.

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba-2.csv")

# Print the dataframe
print (df)

# applying filter function
print (df.filter(["Name", "College", "Salary"]))

## Using regular expression to extract all
# columns which has letter 'a' or 'A' in its name.
print (df.filter(regex ='[aA]'))
```

Output:

```
1      Jae Crowder  Boston Celtics    99.0       SF  25.0     6-6   235.0          Marquette  6796117.0
2     John Holland  Boston Celtics    30.0       SG  27.0     6-5   205.0  Boston University        NaN
3      R.J. Hunter  Boston Celtics    28.0       SG  22.0     6-5   185.0      Georgia State  1148640.0
4    Jonas Jerebko  Boston Celtics     8.0       PF  29.0    6-10   231.0                NaN  5000000.0
..             ...             ...     ...      ...   ...     ...     ...                ...        ...
453   Shelvin Mack       Utah Jazz     8.0       PG  26.0     6-3   203.0             Butler  2433333.0
454      Raul Neto       Utah Jazz    25.0       PG  24.0     6-1   179.0                NaN   900000.0
455   Tibor Pleiss       Utah Jazz    21.0        C  26.0     7-3   256.0                NaN  2900000.0
456    Jeff Withey       Utah Jazz    24.0        C  26.0     7-0   231.0             Kansas   947276.0
457            NaN             NaN     NaN      NaN   NaN     NaN     NaN                NaN        NaN

[458 rows x 9 columns]
              Name            College     Salary
0    Avery Bradley              Texas  7730337.0
1      Jae Crowder          Marquette  6796117.0
2     John Holland  Boston University        NaN
3      R.J. Hunter      Georgia State  1148640.0
4    Jonas Jerebko                NaN  5000000.0
..             ...                ...        ...
453   Shelvin Mack             Butler  2433333.0
454      Raul Neto                NaN   900000.0
455   Tibor Pleiss                NaN  2900000.0
456    Jeff Withey             Kansas   947276.0
457            NaN                NaN        NaN
```

```
              Name            Team   Age     Salary
0    Avery Bradley  Boston Celtics  25.0  7730337.0
1      Jae Crowder  Boston Celtics  25.0  6796117.0
2     John Holland  Boston Celtics  27.0        NaN
3      R.J. Hunter  Boston Celtics  22.0  1148640.0
4    Jonas Jerebko  Boston Celtics  29.0  5000000.0
..             ...             ...   ...        ...
453   Shelvin Mack       Utah Jazz  26.0  2433333.0
454      Raul Neto       Utah Jazz  24.0   900000.0
455   Tibor Pleiss       Utah Jazz  26.0  2900000.0
456    Jeff Withey       Utah Jazz  26.0   947276.0
457            NaN             NaN   NaN        NaN

[458 rows x 4 columns]
```

5. Group, merge, and aggregate data in data frames

```
# import module
import pandas as pd
```

```
# Creating our dataset
df = pd.DataFrame([[9, 4, 8, 9],
                   [8, 10, 7, 6],
                   [7, 6, 8, 5]],
                  columns=['Maths',  'English',
                           'Science', 'History'])

# display dataset
print(df)
#The sum() function is used to calculate the sum of every value
print (df.sum())
#The describe() function is used to get a summary of our dataset
print (df.describe())
#We used agg() function to calculate the sum, min, and max of each column in our dataset.

print (df.agg(['sum', 'min', 'max']))
#Grouping is used to group data using some criteria from our dataset

a = df.groupby('Maths')
print(a.first())
```

Output:

```
   Maths  English  Science  History
0      9        4        8        9
1      8       10        7        6
2      7        6        8        5
Maths      24
English    20
Science    23
History    20
dtype: int64
       Maths    English   Science   History
count    3.0   3.000000  3.000000  3.000000
mean     8.0   6.666667  7.666667  6.666667
std      1.0   3.055050  0.577350  2.081666
min      7.0   4.000000  7.000000  5.000000
25%      7.5   5.000000  7.500000  5.500000
50%      8.0   6.000000  8.000000  6.000000
75%      8.5   8.000000  8.000000  7.500000
max      9.0  10.000000  8.000000  9.000000
     Maths  English  Science  History
sum     24       20       23       20
min      7        4        7        5
```

```
       English  Science  History
Maths
7            6        8        5
8           10        7        6
9            4        8        9
```

5. **Identify and fix missing values in data**

```
import pandas as pd
import numpy as np

# create two columns of randomly generated values, replace a few examples with NaNs
data = {"X1": [np.nan, 0.7636183 , 0.61735332, 0.73848657, np.nan,
    0.71623709, 0.73075927, np.nan, 0.71073827, 0.54693503],
    "X2": [0.87505771, 0.77210971, 0.64369448, 0.54238232, 0.0710951 ,
    0.6854597 , np.nan, 0.20935994, 0.54764129, np.nan ]}
```

```
df = pd.DataFrame(data)
print(df)
#Imputation Method 1: Mean or Median
df_mean_imputed = df.fillna(df.mean())
print (df_mean_imputed)
df_median_imputed = df.fillna(df.median())
print (df_median_imputed)
#Imputation Method 2: Zero
df_zero_imputed = df.fillna(0)
print (df_zero_imputed)
```

Output:

```
         X1        X2
0       NaN  0.875058
1  0.763618  0.772110
2  0.617353  0.643694
3  0.738487  0.542382
4       NaN  0.071095
5  0.716237  0.685460
6  0.730759       NaN
7       NaN  0.209360
8  0.710738  0.547641
9  0.546935       NaN
         X1        X2
0  0.689161  0.875058
1  0.763618  0.772110
2  0.617353  0.643694
3  0.738487  0.542382
4  0.689161  0.071095
5  0.716237  0.685460
6  0.730759  0.543350
7  0.689161  0.209360
8  0.710738  0.547641
9  0.546935  0.543350
```

	X1	X2
0	0.716237	0.875058
1	0.763618	0.772110
2	0.617353	0.643694
3	0.738487	0.542382
4	0.716237	0.071095
5	0.716237	0.685460
6	0.730759	0.595668
7	0.716237	0.209360
8	0.710738	0.547641
9	0.546935	0.595668

	X1	X2
0	0.000000	0.875058
1	0.763618	0.772110
2	0.617353	0.643694
3	0.738487	0.542382
4	0.000000	0.071095
5	0.716237	0.685460
6	0.730759	0.000000
7	0.000000	0.209360
8	0.710738	0.547641
9	0.546935	0.000000

Experiment 3: Visualize data with the help of the following graphical representations:

a. Line plots

b. Bar plots

c. Error Plots

d. Scatter plots

e. KDE Plots

f. Heat Maps

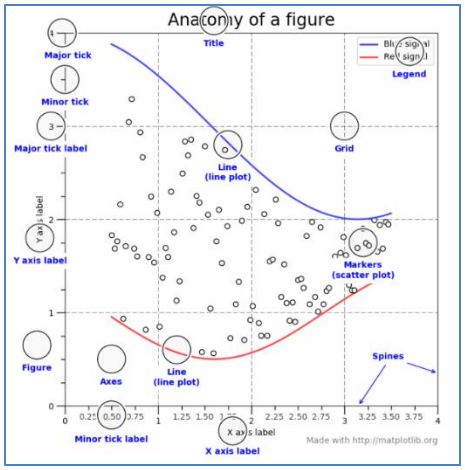
g. Box Plots

h. Pie graph

i. Histogram

j. multiple graphs in single figure

Data Visualization is the process of presenting data in the form of graphs or charts. It helps to understand large and complex amounts of data very easily. **Matploptib** is a low-level library of Python which is used for data visualization. It is easy to use and emulates MATLAB like graphs and visualization. This library is built on the top of NumPy arrays and consists of several plots like line chart, bar chart, histogram, etc. **Pyplot** is a Matplotlib module that provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python and the advantage of being free and open-source. The anatomy of a plot drawn using Matplotlib is given below:



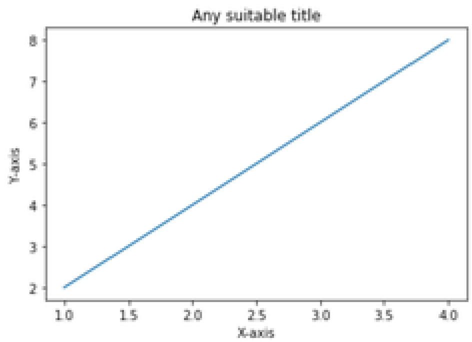
Type pip install Matplotlib to install

a. Line plots

Code 1:

```
import matplotlib.pyplot as plt
import numpy as np
# define data values
x = np.array([1, 2, 3, 4])  # X-axis points
y = x*2  # Y-axis points
plt.plot(x, y)  # Plot the chart
plt.show()  # display
plt.savefig('plt1.jpg')
```

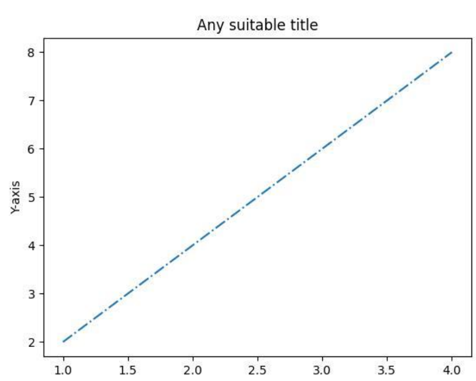
Output 1:



Code 2:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1, 2, 3, 4])
y = x*2
plt.plot(x, y,  '-.')
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Any suitable title")
plt.show() # show first chart
plt.savefig('plt2.jpg')
```

Output 2:

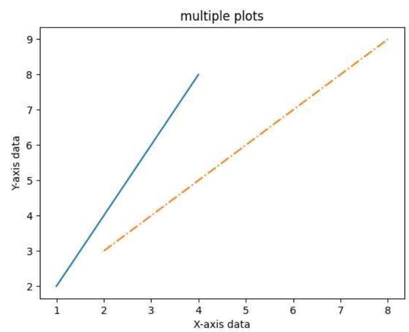


Code 3: Add 2 plots within the same axis.

```
import matplotlib.pyplot as plt
```

```
import numpy as np
x = np.array([1, 2, 3, 4])
y = x*2
# first plot with X and Y data
plt.plot(x, y)
x1 = [2, 4, 6, 8]
y1 = [3, 5, 7, 9]
# second plot with x1 and y1 data
plt.plot(x1, y1, '-.')
plt.xlabel("X-axis data")
plt.ylabel("Y-axis data")
plt.title('multiple plots')
plt.savefig('plt3.jpg')
plt.show()
```

Output 3:

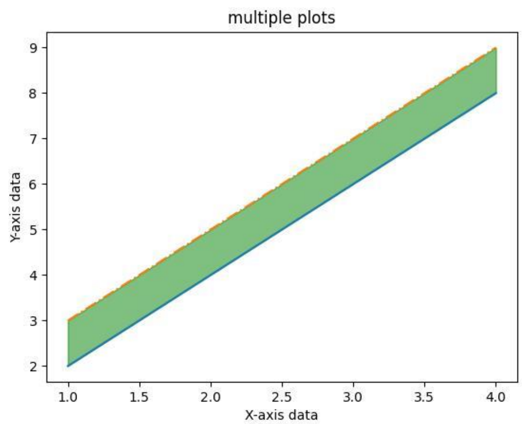


Code 4: Using the pyplot.fill_between() function we can fill in the region between two line plots in the same graph

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1, 2, 3, 4])
y = x*2
```

```
plt.plot(x, y)
x1 = [2, 4, 6, 8]
y1 = [3, 5, 7, 9]
plt.plot(x, y1, '-.')
plt.xlabel("X-axis data")
plt.ylabel("Y-axis data")
plt.title('multiple plots')
plt.fill_between(x, y, y1, color='green', alpha=0.5)
plt.savefig('plt4.jpg')
plt.show()
```

Output 4:

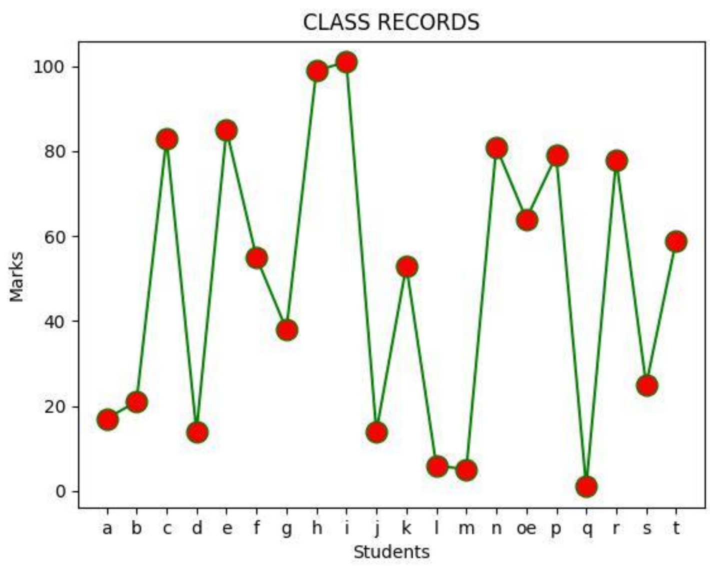


Code 5: Using line Styles

```
import matplotlib.pyplot as plt
import random as random

students = ["a","b","c","d","e",
                "f","g","h","i","j",
                "k","l","m","n","oe",
                "p","q","r","s","t"]

marks=[]
```

```
for i in range(0,len(students)):
        marks.append(random.randint(0, 101))


plt.xlabel("Students")
plt.ylabel("Marks")
plt.title("CLASS RECORDS")
plt.plot(students, marks, color = 'green',
                linestyle = 'solid', marker = 'o',
                markerfacecolor = 'red', markersize = 12)

plt.savefig('plt5.jpg')
plt.show()
```

Output 5



:

b. Bar plots

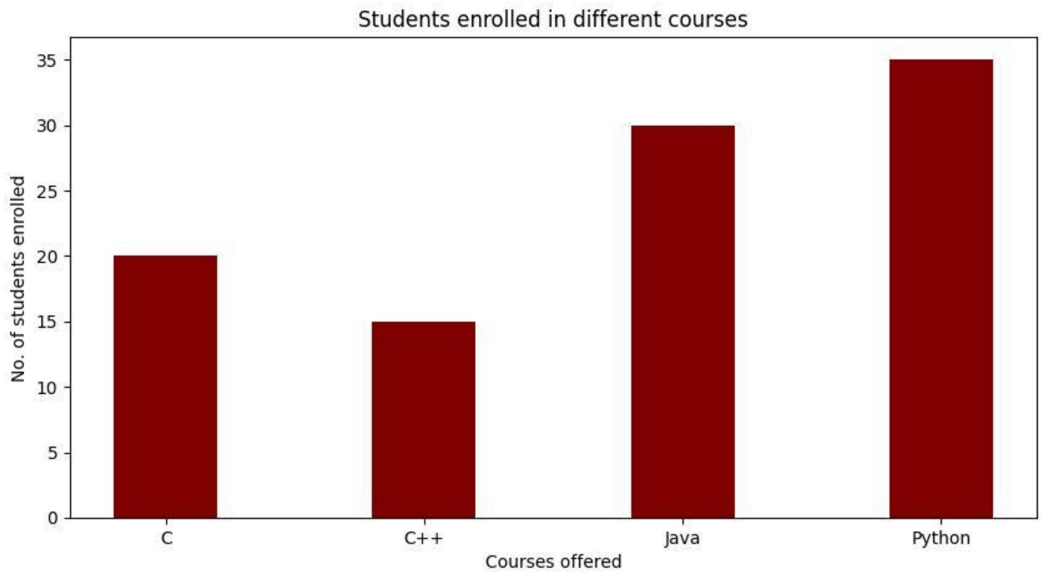
The syntax of the bar() function to be used with the axes is as follows:-

```
plt.bar(x, height, width, bottom, align)
```

Code 1:

```
import numpy as np
import matplotlib.pyplot as plt
 # creating the dataset
data = {'C':20, 'C++':15, 'Java':30,
        'Python':35}
courses = list(data.keys())
values = list(data.values())
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
        width = 0.4)

plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.savefig('plt6.jpg')
plt.show()
```

Output 1

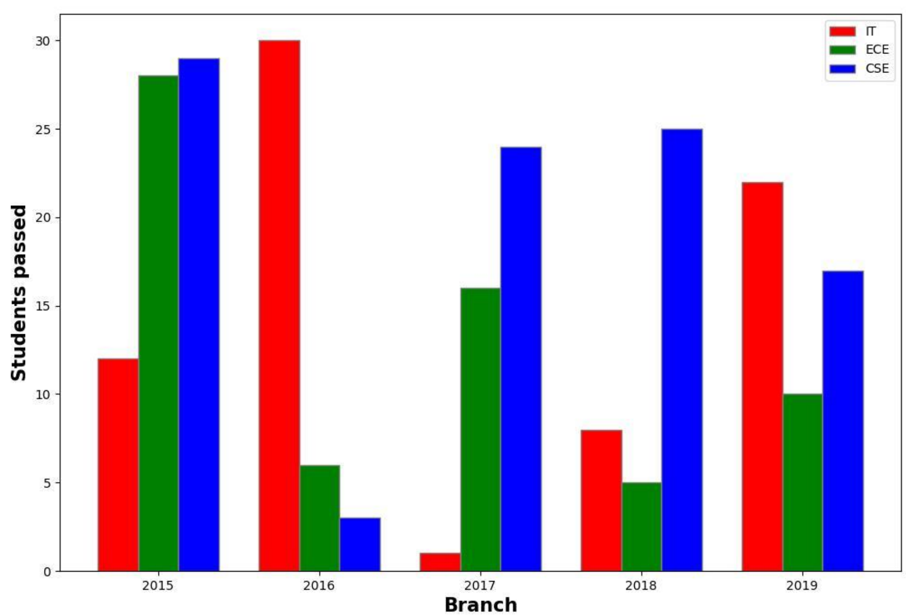


Code 2: Multiple Bar Plots

```
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.25
fig = plt.subplots(figsize =(12, 8))

# set height of bar
IT = [12, 30, 1, 8, 22]
ECE = [28, 6, 16, 5, 10]
CSE = [29, 3, 24, 25, 17]

# Set position of bar on X axis
br1 = np.arange(len(IT))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
```

```
# Make the plot
plt.bar(br1, IT, color ='r', width = barWidth,
        edgecolor ='grey', label ='IT')
plt.bar(br2, ECE, color ='g', width = barWidth,
        edgecolor ='grey', label ='ECE')
plt.bar(br3, CSE, color ='b', width = barWidth,
        edgecolor ='grey', label ='CSE')

# Adding Xticks
plt.xlabel('Branch', fontweight ='bold', fontsize = 15)
plt.ylabel('Students passed', fontweight ='bold', fontsize = 15)
plt.xticks([r + barWidth for r in range(len(IT))],
        ['2015', '2016', '2017', '2018', '2019'])
plt.legend()
plt.savefig('plt7.jpg')
plt.show()
```

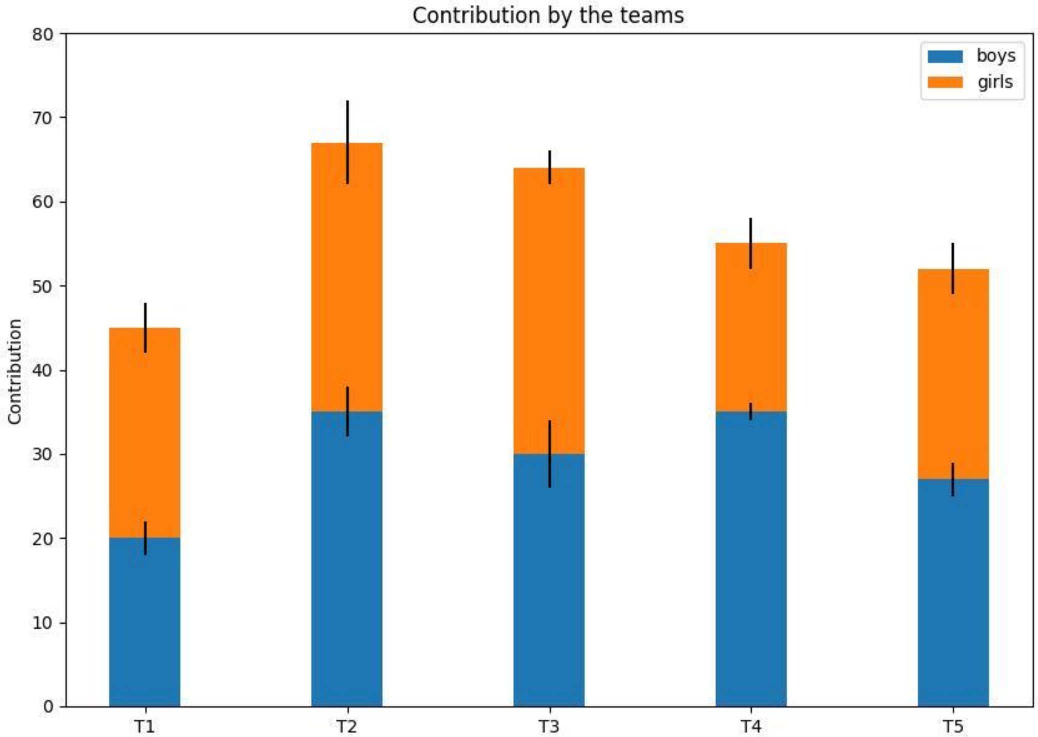
Output 2:



Code 3: Stacked bar plot represent different groups on top of one another

```
import numpy as np
import matplotlib.pyplot as plt
N = 5
boys = (20, 35, 30, 35, 27)
girls = (25, 32, 34, 20, 25)
boyStd = (2, 3, 4, 1, 2)
girlStd = (3, 5, 2, 3, 3)
ind = np.arange(N)
width = 0.35
fig = plt.subplots(figsize =(10, 7))
p1 = plt.bar(ind, boys, width, yerr = boyStd)
p2 = plt.bar(ind, girls, width,
                                    bottom = boys, yerr = girlStd)
plt.ylabel('Contribution')
plt.title('Contribution by the teams')
plt.xticks(ind, ('T1', 'T2', 'T3', 'T4', 'T5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('boys', 'girls'))
plt.savefig('plt8.jpg')
plt.show()
```

Output 3:



c. Error Plots : An error bar is a line through a point on a graph, parallel to one of the axes, which represents the uncertainty or variation of the corresponding coordinate of the point. The errorbar() function in pyplot module of matplotlib library is used to plot y versus x as lines and/or markers with attached errorbars.

Syntax:

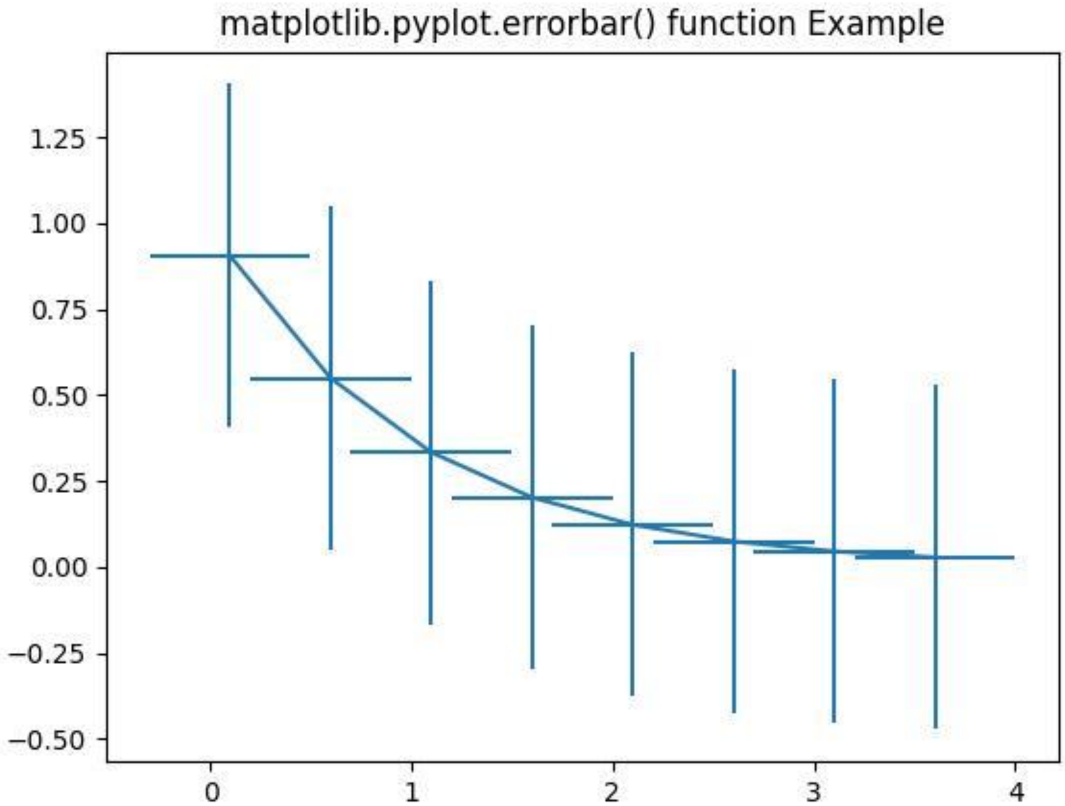
matplotlib.pyplot.errorbar(x, y, yerr=None, xerr=None, fmt=”, ecolor=None, elinewidth=None, capsize=None, barsabove=False, lolims=False, uplims=False, xlolims=False, xuplims=False, errorevery=1, capthick=None, *, data=None, **kwargs)

Code 1:

```
import numpy as np
import matplotlib.pyplot as plt
# example data
```

```
xval = np.arange(0.1, 4, 0.5)
yval = np.exp(-xval)
plt.errorbar(xval, yval, xerr = 0.4, yerr = 0.5)
plt.title('matplotlib.pyplot.errorbar() function Example')
plt.savefig('plt9.jpg')
plt.show()
```

Output 1

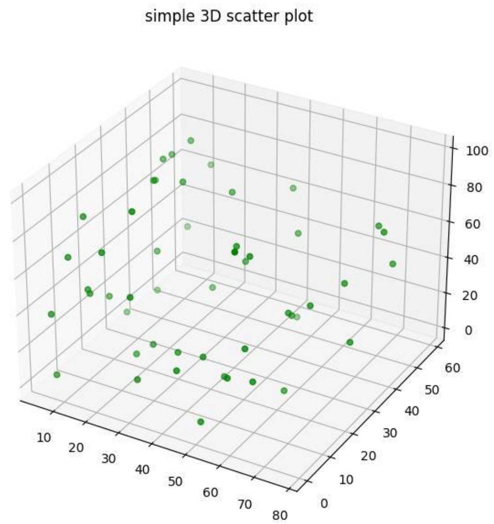


d. Scatter plots: A 3D Scatter Plot is a mathematical diagram, the most basic version of three-dimensional plotting used to display the properties of data as three variables of a dataset using the cartesian coordinates

Code 1:

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
 # Creating dataset
z = np.random.randint(100, size =(50))
x = np.random.randint(80, size =(50))
y = np.random.randint(60, size =(50))
 # Creating figure
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
# Creating plot
ax.scatter3D(x, y, z, color = "green")
plt.title("simple 3D scatter plot")
# show plot
plt.savefig('plt10.jpg')
plt.show()
```

Output 1:

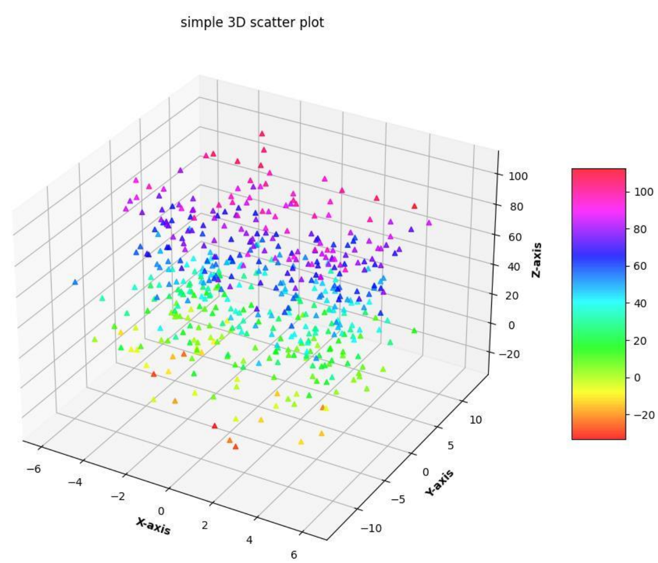


Code 2:

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
# Creating dataset
z = 4 * np.tan(np.random.randint(10, size =(500))) + np.random.randint(100, size
=(500))
x = 4 * np.cos(z) + np.random.normal(size = 500)
y = 4 * np.sin(z) + 4 * np.random.normal(size = 500)
# Creating figure
fig = plt.figure(figsize = (16, 9))
ax = plt.axes(projection ="3d")

# Add x, y gridlines
ax.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.3,
        alpha = 0.2)
```

```
# Creating color map
my_cmap = plt.get_cmap('hsv')
# Creating plot
sctt = ax.scatter3D(x, y, z,
                    alpha = 0.8,
                    c = (x + y + z),
                    cmap = my_cmap,
                    marker ='^')
plt.title("simple 3D scatter plot")
ax.set_xlabel('X-axis', fontweight ='bold')
ax.set_ylabel('Y-axis', fontweight ='bold')
ax.set_zlabel('Z-axis', fontweight ='bold')
fig.colorbar(sctt, ax = ax, shrink = 0.5, aspect = 5)
# show plot
plt.savefig('plt11.jpg')
plt.show()
```

Output 2:

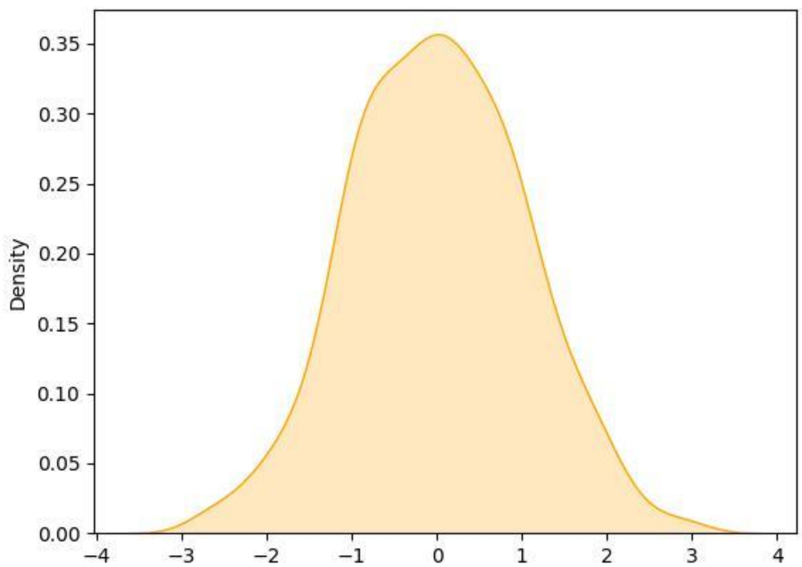


e. KDE Plots: KDE Plot described as Kernel Density Estimate is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization.

Code 1:

```
import seaborn as sn
import matplotlib.pyplot as plt
import numpy as np
data = np.random.randn(500)
res = sn.kdeplot(data, color='orange', shade='True')
plt.show()
```

Output 1:



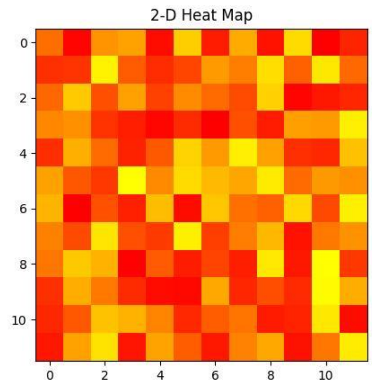
f. Heat Maps: A 2-D Heatmap is a data visualization tool that helps to represent the magnitude of the phenomenon in form of colors

Code 1:

```
import seaborn as sn
```

```
import matplotlib.pyplot as plt
import numpy as np
data = np.random.randn(500)
res = sn.kdeplot(data, color='orange', shade='True')
plt.show()
```

Output 1:

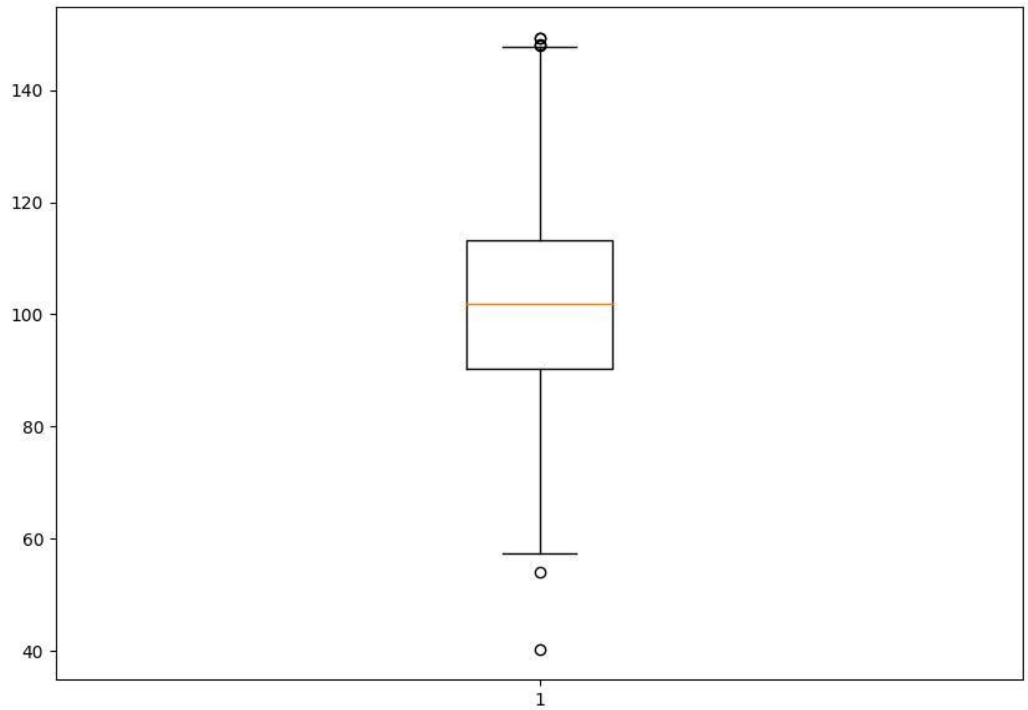


g. Box Plots: A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum

Code 1:

```
import matplotlib.pyplot as plt
import numpy as np
# Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)
fig = plt.figure(figsize =(10, 7))
# Creating plot
plt.boxplot(data)
# show plot
plt.savefig('plt14.jpg')
plt.show()
```

Output 1:



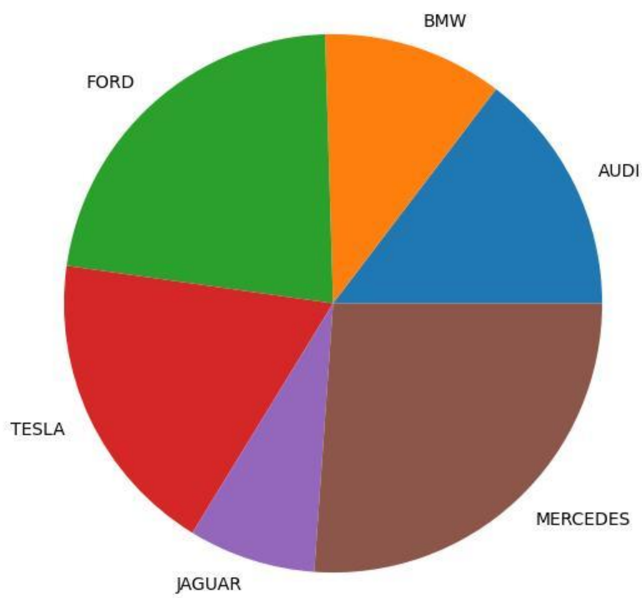
h. Pie graph: A Pie Chart is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data.

Code 1:

```
from matplotlib import pyplot as plt
import numpy as np
# Creating dataset
cars = ['AUDI', 'BMW', 'FORD',
        'TESLA', 'JAGUAR', 'MERCEDES']
data = [23, 17, 35, 29, 12, 41]
# Creating plot
fig = plt.figure(figsize =(10, 7))
```

```
plt.pie(data, labels = cars)
# show plot
plt.savefig('plt15.jpg')
plt.show()
```

Output 1:



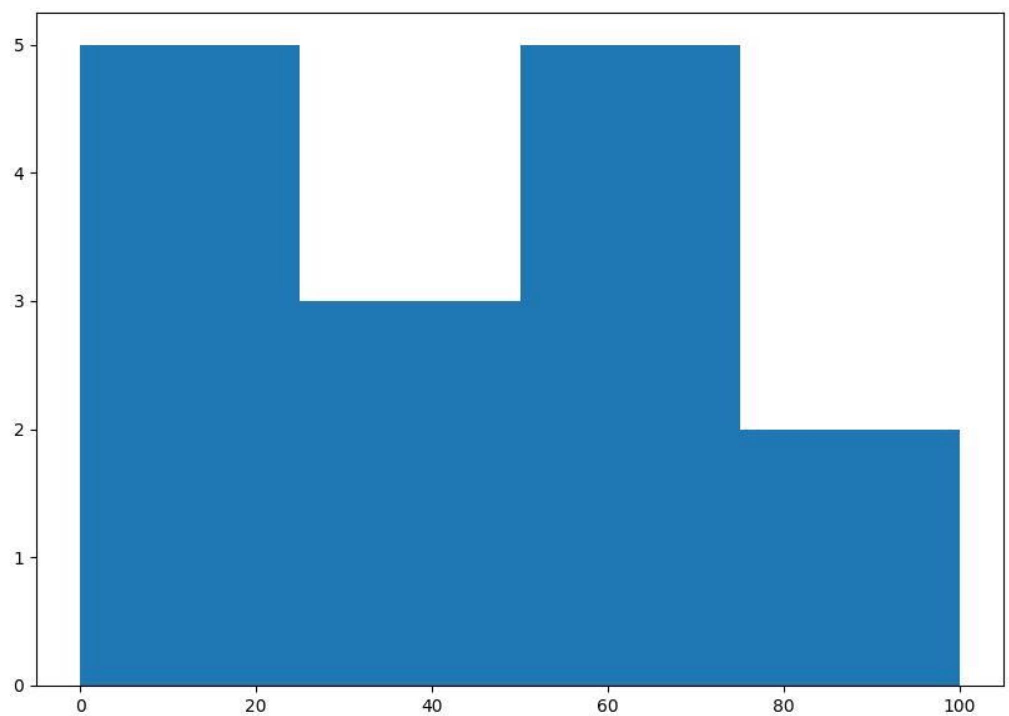
i. Histogram: A histogram is basically used to represent data provided in a form of some groups. It is accurate method for the graphical representation of numerical data distribution.

Code 1:

```
from matplotlib import pyplot as plt
import numpy as np
# Creating dataset
a = np.array([22, 87, 5, 43, 56,
```

```
            73, 55, 54, 11,
            20, 51, 5, 79, 31,
            27])
# Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(a, bins = [0, 25, 50, 75, 100])
# show plot
plt.savefig('plt16.jpg')
plt.show()
```

Output 1:



Experiment 4

Sampling and Resampling:

- ***Generate a population of random numbers.***
- ***Generate multiple samples using Random sampling with and without random sampling.***
- ***Load a balanced dataset and visualize the class distribution.***
- ***Load an imbalanced dataset and visualize the class distribution.***

Generate a population of random numbers.

```
import random
print("Random Population")
random_list = [random.random() for i in range(4)]
print(random_list)
```

output:

Random Population

[0.17508347620070508, 0.0034908367588027955, 0.019492005956522118, 0.6749462483066933]

Generate multiple samples using Random sampling with and without random sampling.

```
import numpy as np
l1=np.random.random_sample((5,))
print("Random sample 1")
print(l1)
l2=np.random.random_sample((5,))
print("Random sample 2")
print(l2)
```

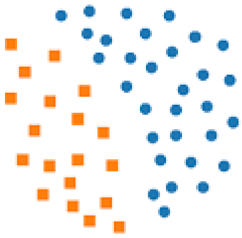
```
l3=np.random.random_sample((5,))
print("Random sample 3")
print(l1)
```

Output:

```
Random sample 1
[0.11668843 0.65219934 0.30282386 0.71349194 0.06380978]
Random sample 2
[0.8605331  0.07084149 0.28931083 0.68654742 0.23567268]
Random sample 3
[0.11668843 0.65219934 0.30282386 0.71349194 0.06380978]
```

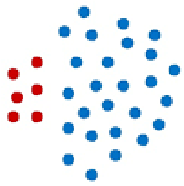
What are Balanced and Imbalanced Datasets?

Balanced Dataset: — Let's take a simple example if in our data set we have positive values which are approximately same as negative values. Then we can say our dataset in balance



Consider Orange color as a positive values and Blue color as a Negative value. We can say that the number of positive values and negative values in approximately same.

Imbalanced Dataset: — If there is the very high different between the positive values and negative values. Then we can say our dataset in Imbalance Dataset.

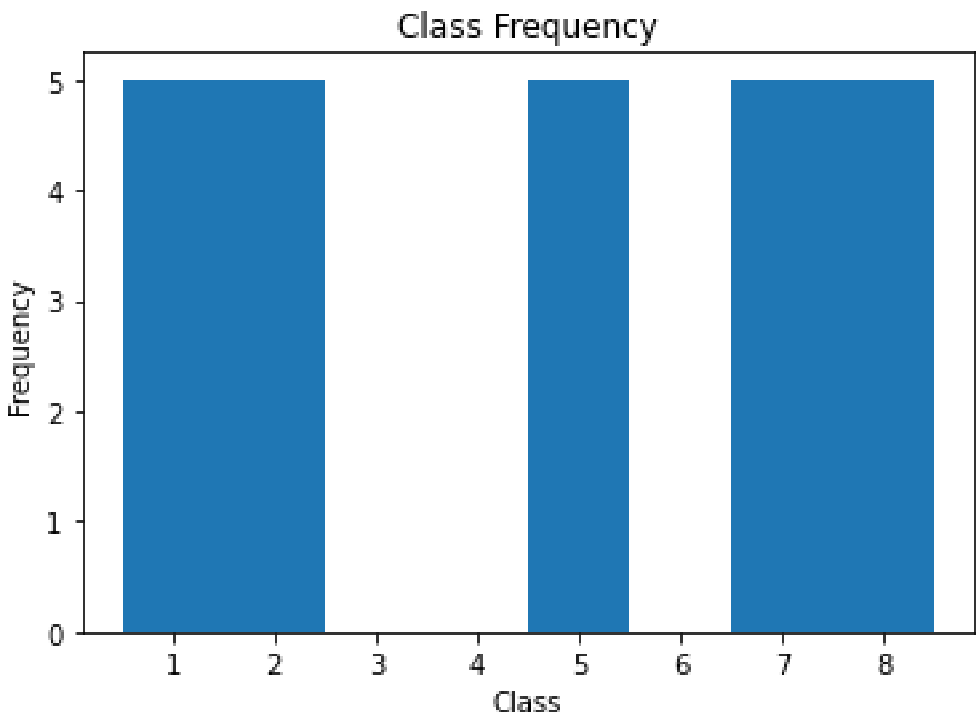


Load a balanced dataset and visualize the class distribution.

```
import numpy as np

import matplotlib.pyplot as plt

uniform_data=[1,1,1,1,1,2,2,2,2,2,5,5,5,5,5,7,7,7,7,7,8,8,8,8,8]

unique, counts = np.unique(uniform_data, return_counts=True)

print(uniform_data)

plt.bar(unique, counts, 1)

plt.title('Class Frequency')

plt.xlabel('Class')

plt.ylabel('Frequency')

plt.show()
```

Output:

[1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8]

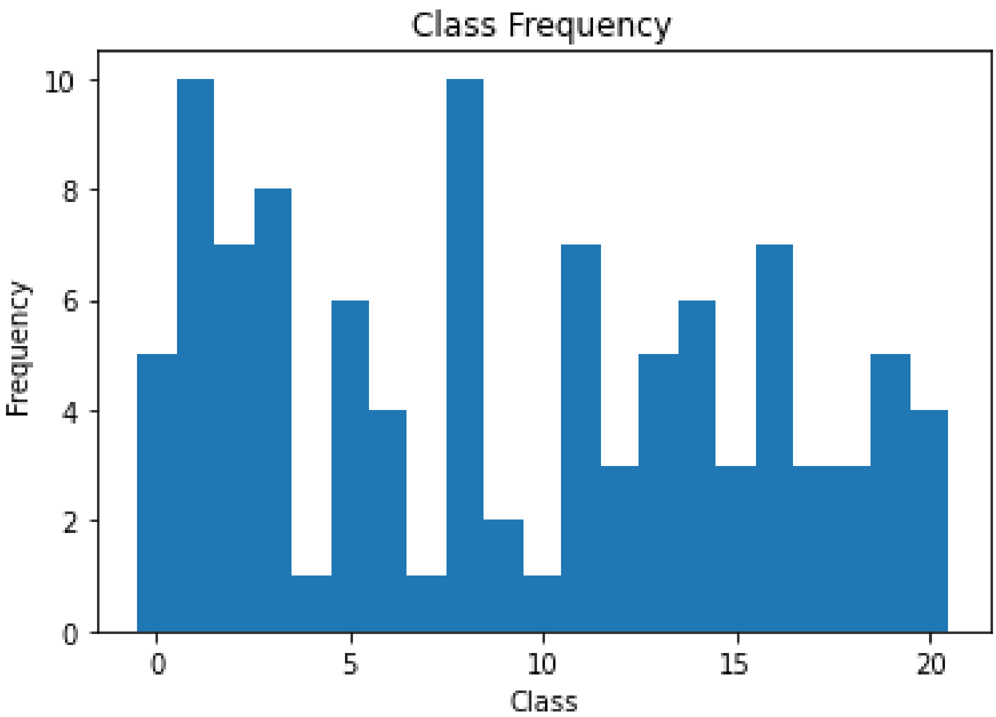


Load an imbalanced dataset and visualize the class distribution.

```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
y_values = [random.randint(0,20) for _ in range(101)]
print(y_values)
unique, counts = np.unique(y_values, return_counts=True)
plt.bar(unique, counts, 1)
plt.title('Class Frequency')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```

Output:

[12, 15, 6, 16, 14, 19, 4, 11, 8, 16, 1, 14, 15, 6, 2, 13, 9, 10, 17, 13, 13, 20, 20, 20, 14, 0, 10, 1, 11, 10, 0, 16, 1, 12, 16, 16, 3, 15, 10, 0, 9, 9, 18, 7, 8, 1, 9, 12, 0, 19, 20, 16, 7, 12, 12, 1, 13, 4, 11, 11, 4, 5, 19, 10, 20, 19, 2, 7, 3, 1, 2, 3, 2, 7, 14, 18, 1, 12, 1, 17, 1, 16, 19, 10, 19, 8, 3, 7, 10, 16, 1, 6, 17, 20, 17, 20, 15, 10, 18, 12, 0]



Experiment 5

Interpreting Data Using Descriptive Statistics:

Compute Mean, Median, Mode, Standard Deviation, Variance, Co-variance, Interquartile Range and Skewness for two different datasets and write your interpretations about these statistical measures. Which measure is best suitable? Justify.

```
import statistics

import numpy

from scipy.stats import skew

sample=[10,20,30,40,50,60,60]

sample2=[73,43,65,23,67,879,34,89,34]

print("Mean of the sample is",statistics.mean(sample))

print("Mean of the sample2 is",statistics.mean(sample2))


print("Median of the sample is",statistics.median(sample))

print("Median of the sample2 is",statistics.median(sample2))


print("Mode of the sample is",statistics.mode(sample))

print("Mode of the sample2 is",statistics.mode(sample2))


print("Standard deviation of the sample is",statistics.stdev(sample))

print("Standard deviation of the sample2 is",statistics.stdev(sample2))


print("variance of the sample is",statistics.variance(sample))
```

```
print("variance of the sample2 is",statistics.variance(sample2))


print("Covariance of the sample is",numpy.cov(sample))

print("Covariance of the sample2 is",numpy.cov(sample2))


q3,q1=numpy.percentile(sample, [75 ,25])

iqr=q3-q1

print("The interquartile range of the sample is",iqr)

q3,q1=numpy.percentile(sample2, [75 ,25])

iqr=q3-q1

print("The interquartile range of the sample2 is",q3-q1)


print("The skewness of the sample2 is")

print(skew(sample2, axis=0, bias=True))
```

output:

Median of the sample is 40

Median of the sample2 is 65

Mode of the sample is 60

Mode of the sample2 is 34

Standard deviation of the sample is 19.518001458970662

Standard deviation of the sample2 is 276.0184675786105

variance of the sample is 380.95238095238096

variance of the sample2 is 76186.19444444445

Covariance of the sample is 380.95238095238096

Covariance of the sample2 is 76186.19444444447

The interquartile range of the sample is 30.0

The interquartile range of the sample2 is 39.0

The skewness of the sample is

-0.22234764798058884

The skewness of the sample2 is

2.44558983254843

Experiment 6

Generating Samples from Probability Distributions:

• Generate a set of random numbers (which corresponds to a uniform distribution) using the function rand and plot its histogram. What is the shape of this histogram and why?

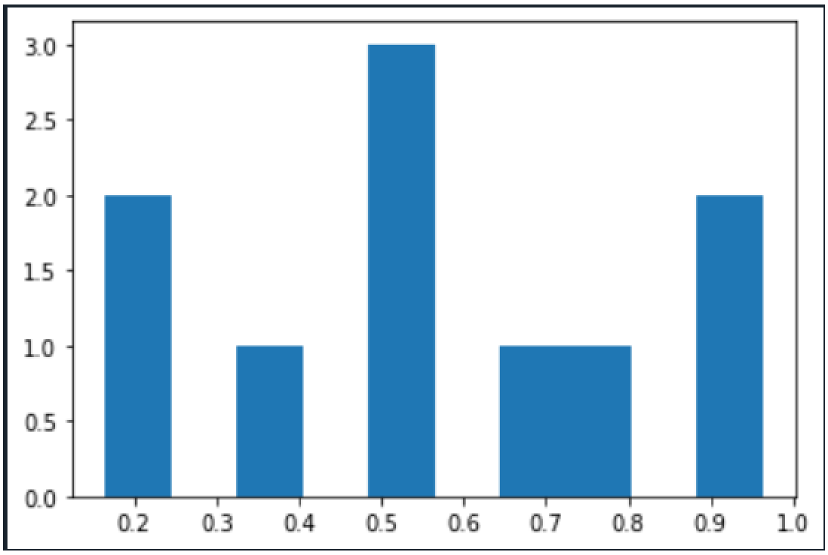
• Investigate how the shape of the histogram is affected by the number of random numbers you have generated.

• Similarly generate numbers using Bernoulli, Binomial distributions and plot a histogram and check the shape.

• Generate numbers using exponential and poisson distributions and plot a histogram and check the shape.

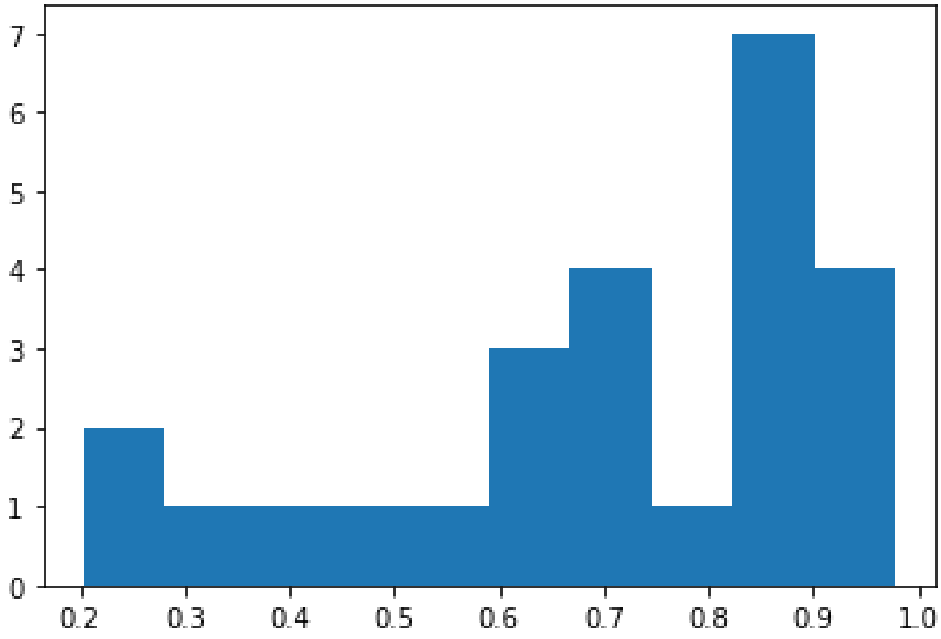
Generate a set of random numbers (which corresponds to a uniform distribution) using the function rand and plot its histogram. What is the shape of this histogram and why?

```
import numpy as np

import matplotlib.pyplot as plt

l1 = np.random.rand(10)

print(l1)

plt.hist(l1)
```

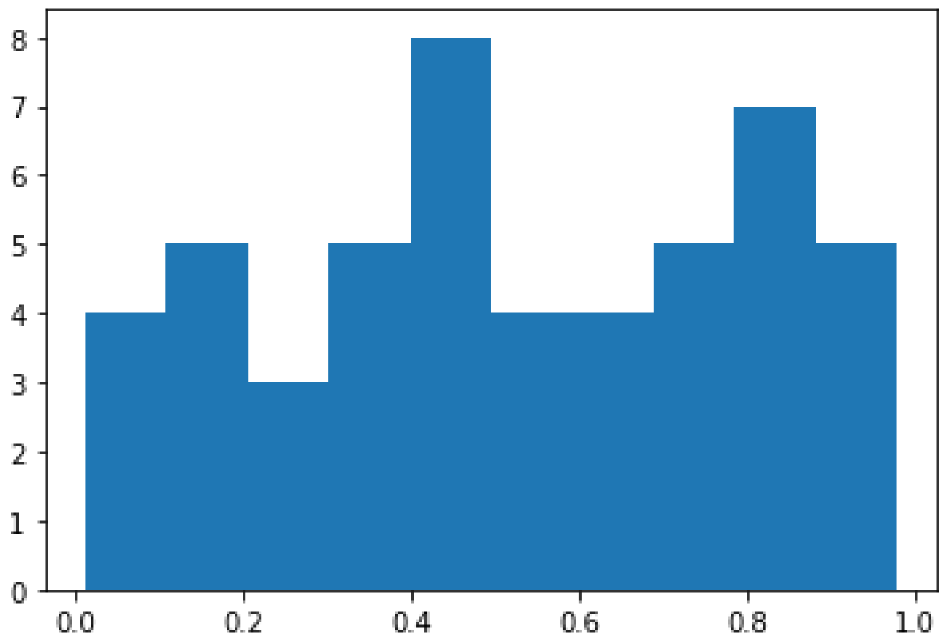


Investigate how the shape of the histogram is affected by the number of random numbers you have generated.

```
import numpy as np

import matplotlib.pyplot as plt

l1 = np.random.rand(25)

print(l1)

plt.hist(l1)
```

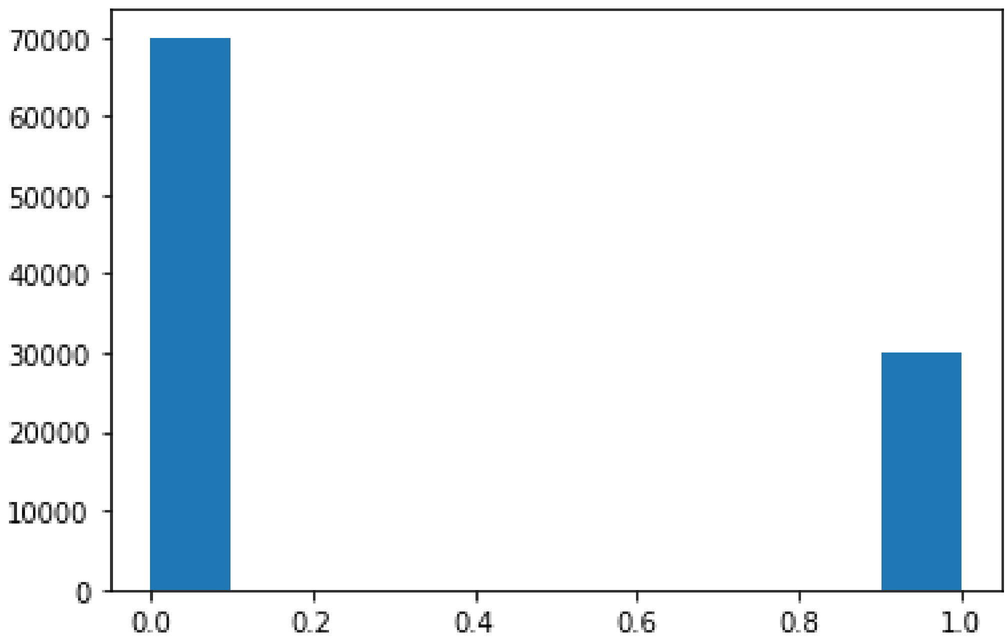



```
l1 = np.random.rand(50)
```



Similarly generate numbers using Bernoulli, Binomial distributions and plot a histogram and check the shape.

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import bernoulli, binom

p = 0.3

X = bernoulli(p)

X_samples = X.rvs(100000)

plt.hist(X_samples, discrete=True, shrink=0.2);
```



Binomial distribution is a probability distribution that summarises the likelihood that a variable will take one of two independent values under a given set of parameters. The distribution is obtained by performing a number of Bernoulli trials.

A Bernoulli trial is assumed to meet each of these criteria :

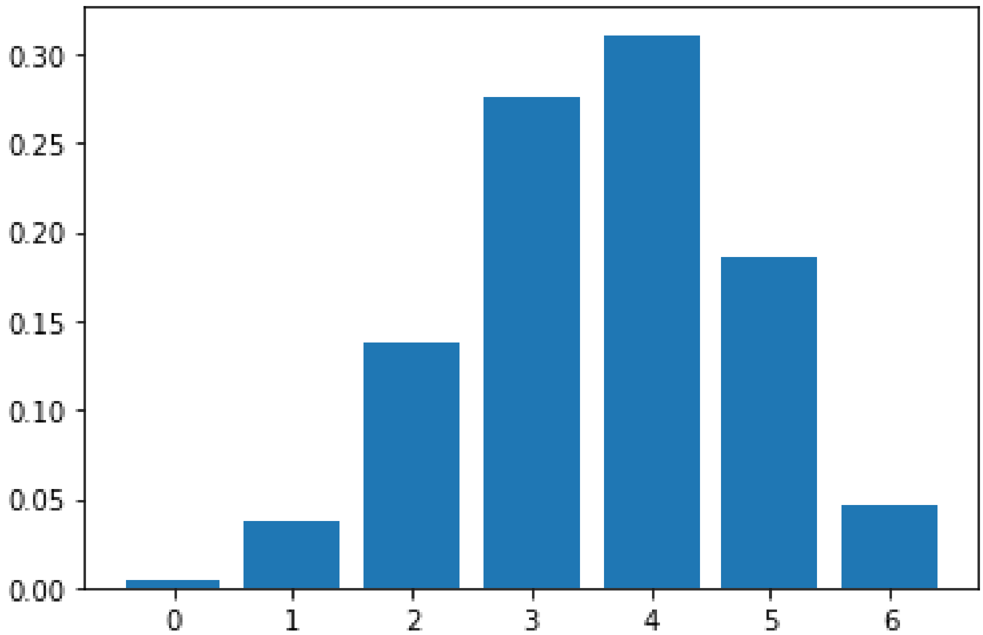
- There must be only 2 possible outcomes.
- Each outcome has a fixed probability of occurring. A success has the probability of p, and a failure has the probability of 1 – p.
- Each trial is completely independent of all others.

scipy.stats.binom.pmf() function is used to obtain the probability mass function for a certain value of r, n and p. We can obtain the distribution by passing all possible values of r(0 to n).

Syntax : scipy.stats.binom.pmf(r, n, p)

```
from scipy.stats import binom

import matplotlib.pyplot as plt

# setting the values

# of n and p

n = 6
```

```
p = 0.6
# defining list of r values
r_values = list(range(n + 1))
# list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values ]
# plotting the graph
plt.bar(r_values, dist)
plt.show()
```

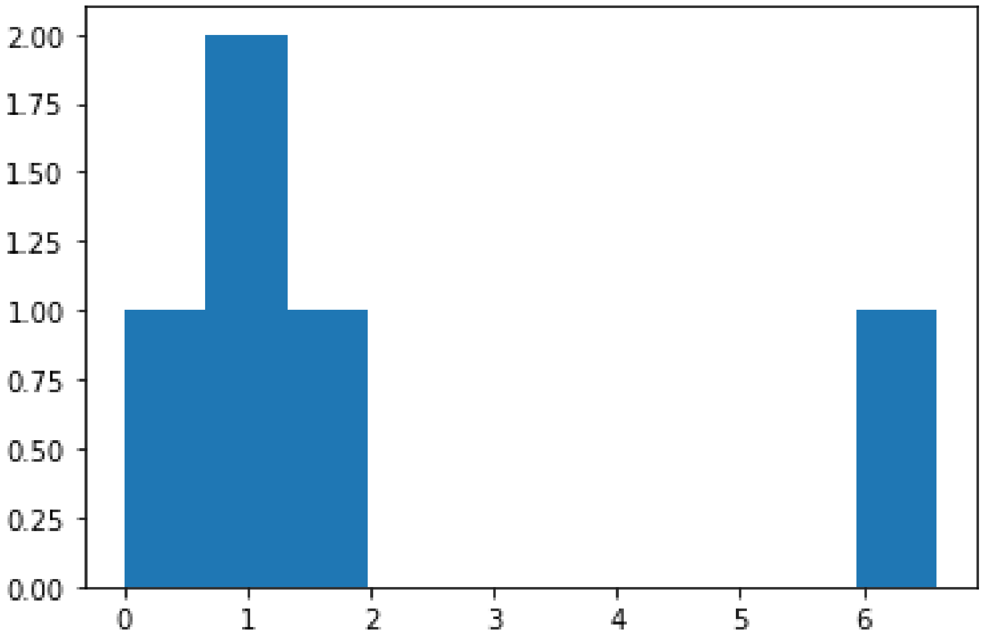


Generate numbers using exponential and poisson distributions and plot a histogram and check the shape.

Exponential distribution in python is implemented using an inbuilt function exponential() which is included in the random module of NumPy library. The exponential() function takes in two parameters. First parameter "size" is the mandatory parameter and it is size of the output array which could be 1D, 2D, 3D or n-dimensional (depending on the programmer's requirements). The second parameter is inverse of rate defined by "scale", it is an optional parameter and if it is not explicitly defined, its value is taken as 1.0.

```
from numpy import random
```

```
import matplotlib.pyplot as plt

res_arr= random. exponential(scale = 2.5, size=5)

print('1D array of size(1,5) having exponential distribution with scale 2.5:\n')

print(res_arr)

plt.hist(res_arr)
```

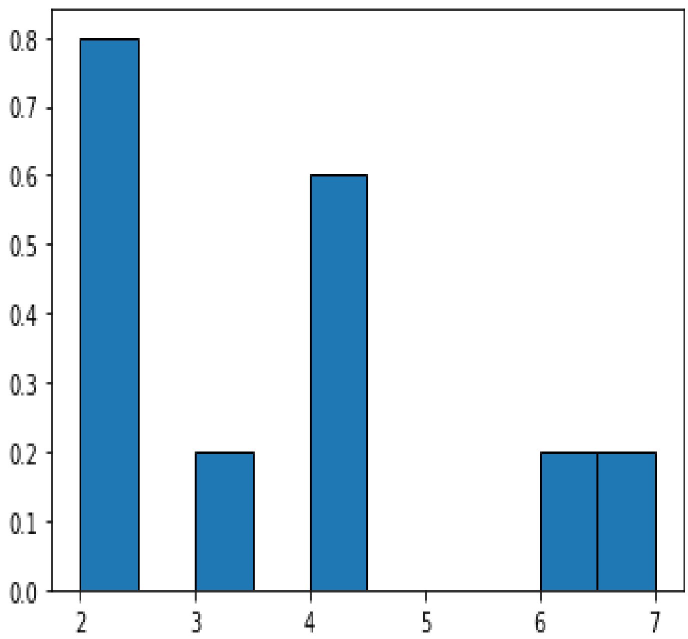


Poisson distribution

The Poisson distribution describes the probability of obtaining k successes during a given time interval.

You can use the poisson.rvs(mu, size) function to generate random values from a Poisson distribution with a specific mean value and sample size:

```
from scipy.stats import poisson

import matplotlib.pyplot as plt

#generate random values from Poisson distribution with mean=3 and sample size=10

x=poisson.rvs(mu=3, size=10)

plt.hist(x, density=True, edgecolor='black')
```


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2
3
4
5
6
7

Experiment 7

Implement the Central Limit Theorem in Python.

Central Limit Theorem

- The CLT describes the shape of the distribution of sample means as a Gaussian commonly known as normal distribution, which is very popular in statistics.
- An example of simulated dice rolls in Python to demonstrate the central limit theorem.
- And finally with CLT knowledge of the Gaussian distribution is used to make inferences about model performance in applied machine learning.

Before going to statistical definition lets consider an example to understand better.

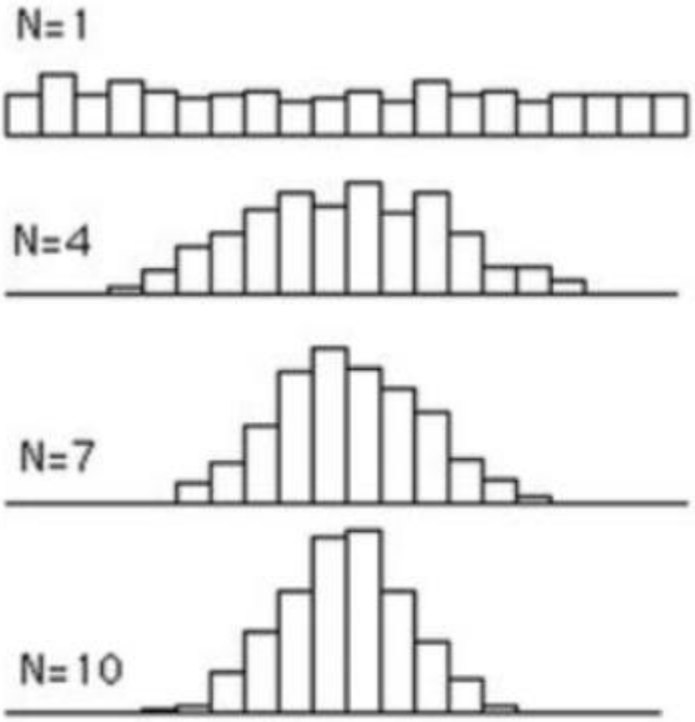
Consider there are 200 students in a class of Computer Science, Our task is to calculate the average marks scored by the class in the Data structure subject. Sounds simple right?

One way will be consider every student's marks sum them all and then divide by total number of students to find out the average.

But what if the size of data is massive, let’s say you are asked to do a study on all the computer science student population of India. Now considering each students marks will be tiresome and long process. So what can be an alternate approach?

First draw groups of student in random from the class(population), which are called as sample. We will draw a multiple sample each consisting of 30 students.

Note: Well why 30, CLT theorem will be true regardless of whether the source population is normal or skewed, provided the sample size is sufficiently large (usually $n > 30$). If population is normal distribution , you can consider sample size less then 30 and CLT theorem will hold good which also will be a true representation of population.



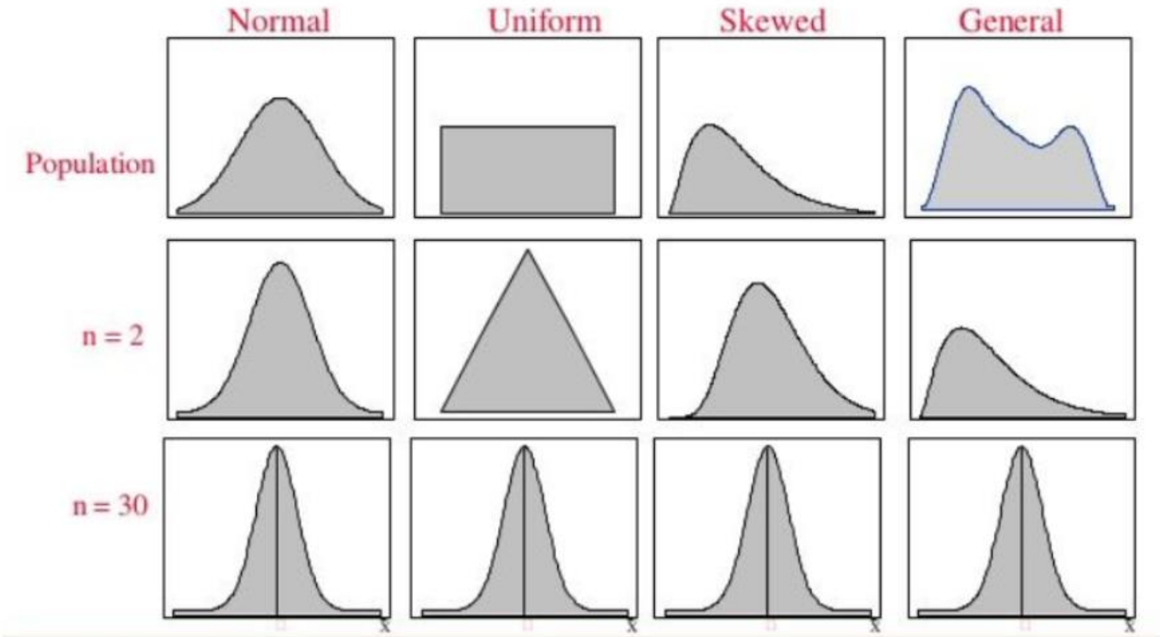
Depiction of distribution of marks changes with sample size

- Calculate the individual mean of these samples
- Calculate the mean of these sample means
- This value will give us the approximate mean marks of the students in the computer science department
- Additionally, the histogram of the sample mean marks of students will resemble a bell curve (or normal distribution)

Formal definition :

The central limit theorem in statistics states that, given a sufficiently large sample size, the sampling distribution of the mean for a variable will approximate a normal distribution regardless of that variable's distribution in the population.

Unpacking the definition, Irrespective of the population distribution if samples size is sufficient. The distribution of sample means, calculated from repeated sampling, will tend to normality as the size of your samples gets larger.



Formula:

Central limit theorem is applicable for a sufficiently large sample sizes (n≥30). The formula for central limit theorem can be stated as follows:

$$\mu_{\overline{x}} = \mu$$

and

$$\sigma_{\overline{x}} = \frac{\sigma}{\sqrt{n}}$$

Where,
μ = Population mean
σ = Population standard deviation
$\mu_{\overline{x}}$ = Sample mean
$\sigma_{\overline{x}}$ = Sample standard deviation
n = Sample size

It is important that each trial that results in an observation be independent and performed in the same way. This is to ensure that the sample is drawing from the same underlying population distribution. More formally, this expectation is referred to as independent and identically distributed, or iid.

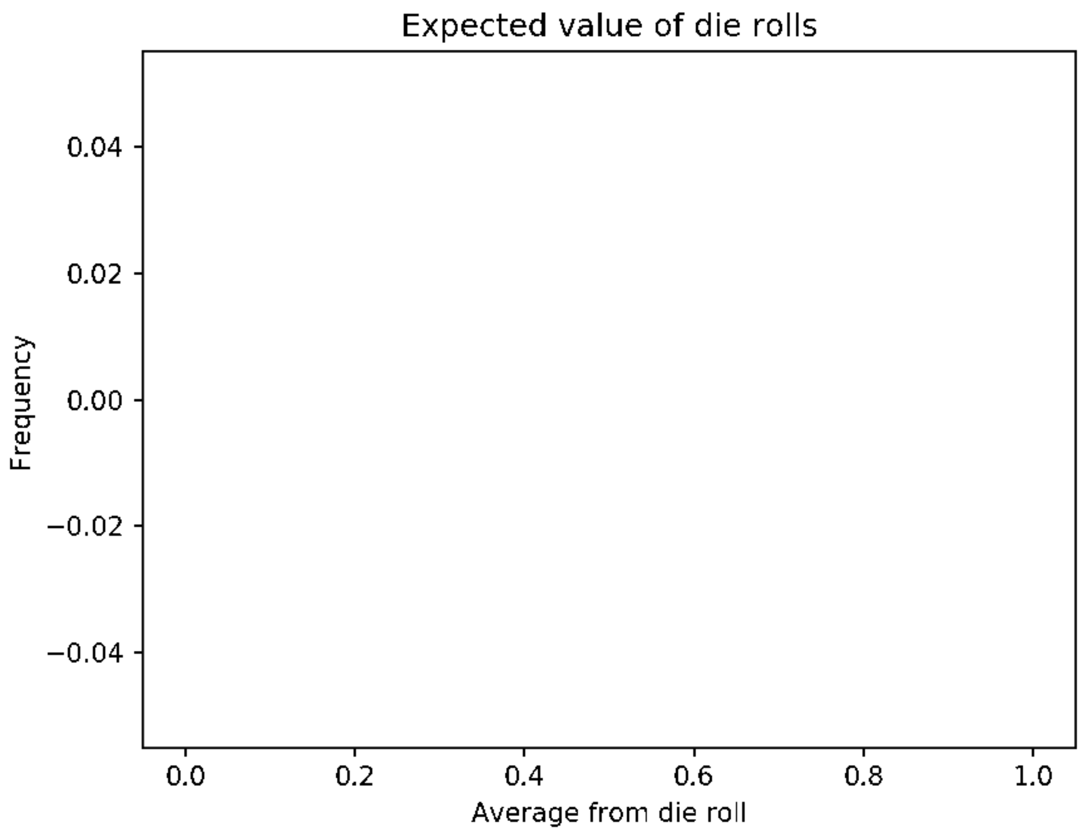
Firstly, the central limit theorem is impressive, especially as this will occur no matter the shape of the population distribution from which we are drawing samples. It demonstrates that the distribution of errors from estimating the population mean fit a distribution that the field of statistics knows a lot about.

Secondly, this estimate of the Gaussian distribution will be more accurate as the size of the samples drawn from the population is increased. This means that if we use our knowledge of the Gaussian distribution in general to start making inferences about the means of samples drawn from a population, that these inferences will become more useful as we increase our sample size.

Implementing the Central Limit Theorem in Python

The below code help us understand the CLT with help of die roll done n times, I used 1000 simulation, but you can go ahead and try with different simulation by changing n lets say 10, 200, 500. and see the normal distribution changes with each n value by yourself.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from wand.image import Image
from wand.display import display


# 1000 simulations of die roll
n = 1000


# In each simulation, there is one trial more than the previous
simulation
avg = []
for i in range(2,n):
    a = np.random.randint(1,7,i)
    avg.append(np.average(a))


print(avg[0:5])


# Function that will plot the histogram, where current is the latest
figure
def clt(current):
    # if animation is at the last frame, stop it
    plt.cla()
    if current == 500:
        a.event_source.stop()


plt.hist(avg[0:current])


plt.gca().set_title('Expected value of die rolls')
    plt.gca().set_xlabel('Average from die roll')
    plt.gca().set_ylabel('Frequency')


plt.annotate('Die roll = {}'.format(current), [3,27])


fig = plt.figure()
a = animation.FuncAnimation(f                rval=5)
plt.show()
```


Expected value of die rolls
Frequency
0.04
0.02
0.00
−0.02
−0.04
0.0
0.2
0.4
0.6
0.8
1.0
Average from die roll

Experiment-8

Hypothesis tests:

Implement the following three popular statistical techniques for hypothesis testing: Chisquare test, T-test and ANOVA test

(Calculate the Test Statistic and P-value by running a Hypothesis test that well suits your data and Make Conclusions).

What is hypothesis testing ?

Hypothesis testing is a statistical method that is used in making statistical decisions using experimental data. Hypothesis Testing is basically an assumption that we make about the population parameter.

Ex : you say avg student in class is 40 or a boy is taller than girls.

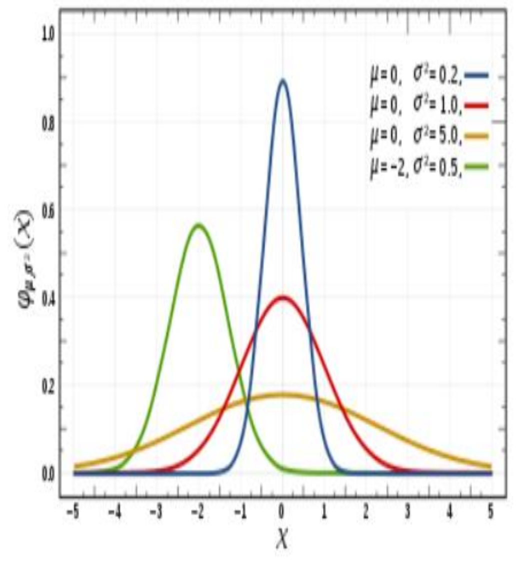
all those example we assume need some statistic way to prove those. we need some mathematical conclusion what ever we are assuming is true.

Why do we use it ?

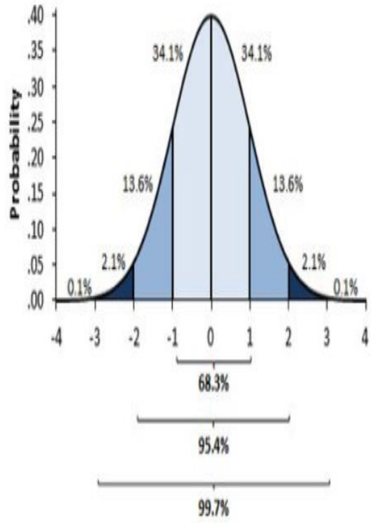
Hypothesis testing is an essential procedure in statistics. A hypothesis test evaluates two mutually exclusive statements about a population to determine which statement is best supported by the sample data. When we say that a finding is statistically significant, it’s thanks to a hypothesis test.

What are basic of hypothesis ?

The basic of hypothesis is normalisation and standard normalisation. all our hypothesis is revolve around basic of these 2 terms. let’s see these.



Normal Curve images with different mean and variance



Standardised Normal curve image and separation on data in percentage in each section.

You must be wondering what's difference between these two image, in 1st first you can see there are different normal curve all those normal curve can have different mean's and variances where as in 2nd image if you notice the graph is properly distributed and mean =0 and variance =1 always. concept of z-score comes in picture when we use standardised normal data.

Normal Distribution -

A variable is said to be normally distributed or have a normal distribution if its distribution has the shape of a normal curve — a special bell-shaped curve. … The graph of a normal distribution is called the normal curve, which has all of the following properties: 1. The mean, median, and mode are equal.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Standardised Normal Distribution —

A standard normal distribution is a normal distribution with mean 0 and standard deviation 1

$$x_{new} = \frac{x - \mu}{\sigma}$$

Which are important parameter of hypothesis testing ?

Null hypothesis :- In inferential statistics, the null hypothesis is a general statement or default position that there is no relationship between two measured phenomena, or no association among groups

In other words it is a basic assumption or made based on domain or problem knowledge.

Example : a company production is = 50 unit/per day etc.

Alternative hypothesis :-

The alternative hypothesis is the hypothesis used in hypothesis testing that is contrary to the null hypothesis. It is usually taken to be that the observations are the result of a real effect (with some amount of chance variation superposed)

Example : a company production is !=50 unit/per day etc.

Level of significance: Refers to the degree of significance in which we accept or reject the null-hypothesis. 100% accuracy is not possible for accepting or rejecting a hypothesis, so we therefore select a level of significance that is usually 5%.

This is normally denoted with alpha(maths symbol) and generally it is 0.05 or 5% , which means your output should be 95% confident to give similar kind of result in each sample.

Type I error: When we reject the null hypothesis, although that hypothesis was true. Type I error is denoted by alpha. In hypothesis testing, the normal curve that shows the critical region is called the alpha region

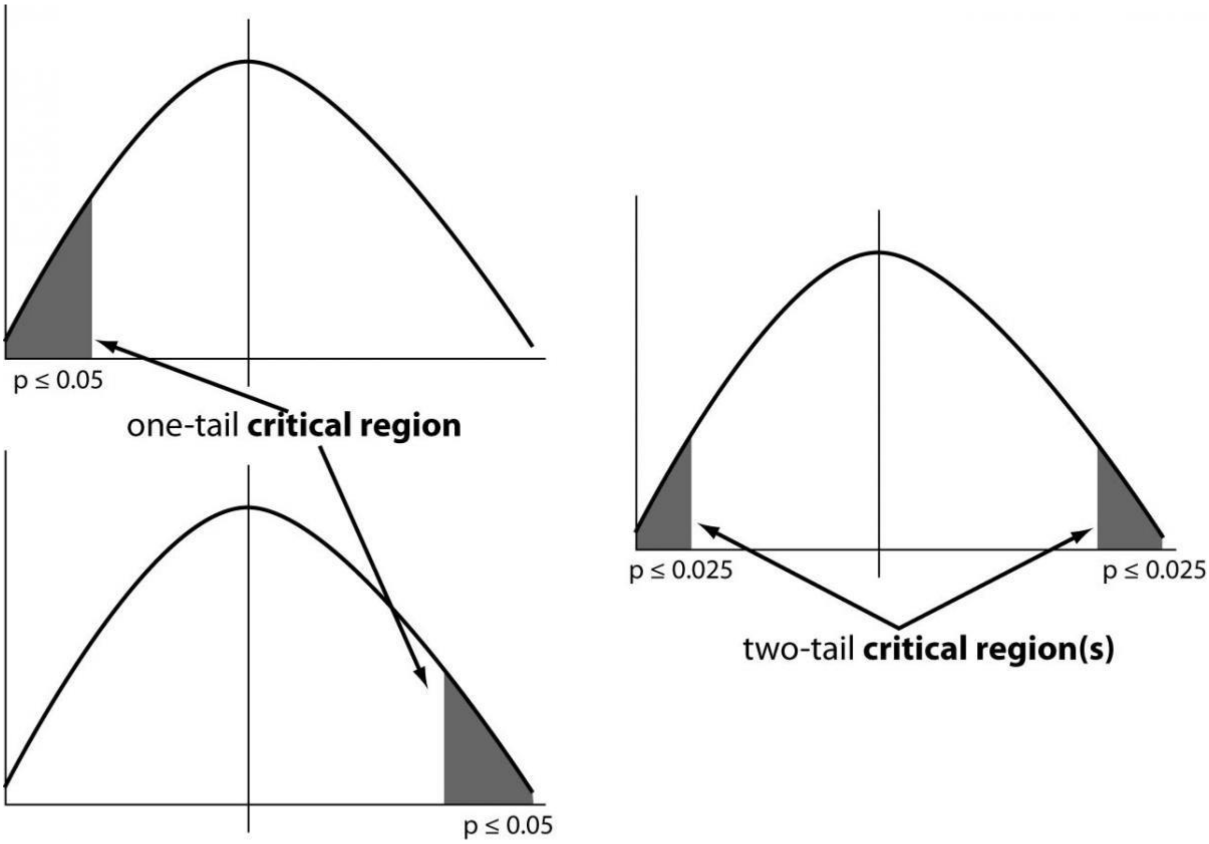
Type II errors: When we accept the null hypothesis but it is false. Type II errors are denoted by beta. In Hypothesis testing, the normal curve that shows the acceptance region is called the beta region.

One tailed test :- A test of a statistical hypothesis , where the region of rejection is on only one side of the sampling distribution , is called a one-tailed test.

Example :- a college has ≥ 4000 student or data science ≤ 80% org adopted.

Two-tailed test :- A two-tailed test is a statistical test in which the critical area of a distribution is two-sided and tests whether a sample is greater than or less than a certain range of values. If the sample being tested falls into either of the critical areas, the alternative hypothesis is accepted instead of the null hypothesis.

Example : a college != 4000 student or data science != 80% org adopted



P-value :- The P value, or calculated probability, is the probability of finding the observed, or more extreme, results when the null hypothesis (H 0) of a study question is true — the definition of 'extreme' depends on how the hypothesis is being tested.

If your P value is less than the chosen significance level then you reject the null hypothesis i.e. accept that your sample gives reasonable evidence to support the alternative hypothesis. It does NOT imply a "meaningful" or "important" difference; that is for you to decide when considering the real-world relevance of your result.

Example : you have a coin and you don't know whether that is fair or tricky so let's decide null and alternate hypothesis

H0 : a coin is a fair coin.

H1 : a coin is a tricky coin. and alpha = 5% or 0.05

Now let’s toss the coin and calculate p- value (probability value).

Toss a coin 1st time and result is tail- P-value = 50% (as head and tail have equal probability)

Toss a coin 2nd time and result is tail, now p-value = 50/2 = 25%

and similarly we Toss 6 consecutive time and got result as P-value = 1.5% but we set our significance level as 95% means 5% error rate we allow and here we see we are beyond that level i.e. our null- hypothesis does not hold good so we need to reject and propose that this coin is a tricky coin which is actually.

Degree of freedom :- Now imagine you’re not into hats. You’re into data analysis. You have a data set with 10 values. If you’re not estimating anything, each value can take on any number, right? Each value is completely free to vary. But suppose you want to test the population mean with a sample of 10 values, using a 1-sample t test. You now have a constraint — the estimation of the mean. What is that constraint, exactly? By definition of the mean, the following relationship must hold: The sum of all values in the data must equal n x mean, where n is the number of values in the data set.

So if a data set has 10 values, the sum of the 10 values must equal the mean x 10. If the mean of the 10 values is 3.5 (you could pick any number), this constraint requires that the sum of the 10 values must equal 10 x 3.5 = 35.

With that constraint, the first value in the data set is free to vary. Whatever value it is, it’s still possible for the sum of all 10 numbers to have a value of 35. The second value is also free to vary, because whatever value you choose, it still allows for the possibility that the sum of all the values is 35.\

Now Let’s see some of widely used hypothesis testing type :-

- T Test
- ANOVA Test
- Chi-Square Test

T Test

A t-test is a type of inferential statistic which is used to determine if there is a significant difference between the means of two groups which may be related in certain features. It is mostly used when the data sets, like the set of data recorded as outcome from flipping a coin a 100 times, would follow a normal distribution and may have unknown variances. T test is used as a hypothesis testing tool, which allows testing of an assumption applicable to a population.

T-test has 2 types : 1. one sampled t-test 2. two-sampled t-test.

One sample t-test : The One Sample t Test determines whether the sample mean is statistically different from a known or hypothesised population mean. The One Sample t Test is a parametric test.

Example :- you have 10 ages and you are checking whether avg age is 30 or not.

```
from scipy.stats import ttest_1samp
import numpy as np

ages = np.genfromtxt("ages.csv")

print(ages)

ages_mean = np.mean(ages)
print(ages_mean)
tset, pval = ttest_1samp(ages, 30)

print("p-values",pval)

if pval < 0.05:    # alpha value is 0.05 or 5%
   print(" we are rejecting null hypothesis")
else:
  print("we are accepting null hypothesis")
```

Output

```
[ 32.  34.  29.  29.  22.  39.  38.  37.  38.  36.  30.  26.
22.  22.]
('mean ages', 31.0)
('p-values', 0.56051558881713792)
we are accepting null hypothesis
```

Two sampled T-test :-The Independent Samples t Test or 2-sample t-test compares the means of two independent groups in order to determine whether there is statistical evidence that the associated population means are significantly different. The Independent Samples t Test is a parametric test. This test is also known as: Independent t Test.

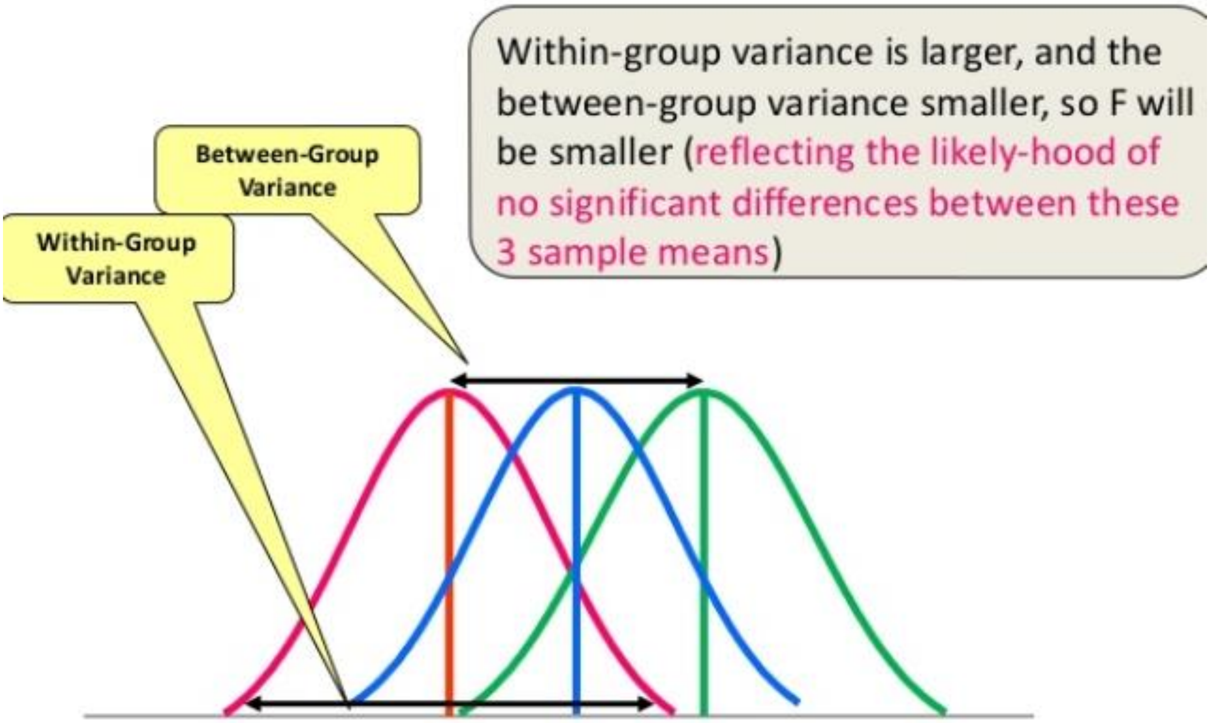
```python
from scipy.stats import ttest_ind
import numpy as np

week1 = np.genfromtxt("week1.csv",  delimiter=",")
week2 = np.genfromtxt("week2.csv",  delimiter=",")

print(week1)
print("week2 data :-\n")
print(week2)
week1_mean = np.mean(week1)
week2_mean = np.mean(week2)

print("week1 mean value:",week1_mean)
print("week2 mean value:",week2_mean)

week1_std = np.std(week1)
week2_std = np.std(week2)

print("week1 std value:",week1_std)
print("week2 std value:",week2_std)

ttest,pval = ttest_ind(week1,week2)
print("p-value",pval)

if pval <0.05:
  print("we reject null hypothesis")
else:
  print("we accept null hypothesis")
```

Output:

```
 20.83415999  23.79367158  19.7556718   29.54421084
20.1433138 ]
week2 data :-

[ 18.63431907  31.28788036  34.96797943  21.81678117
28.21619974
  39.39313736  35.52223207  27.54222109  33.64395433
25.31673581
  28.81392191  30.7358016   26.37241881  26.0945555
26.34073477
  19.42196017  32.58797652  24.84001926  28.93348335
20.43667584
  22.72495967  32.31728012  35.384306    29.66709637
24.53512973
  30.91406007  19.56117513  24.90816833  30.13163726
31.47466199
  27.77683598  16.51307462  35.0770162   31.74818107
36.36053496
  27.70500593  29.49869936  27.65575346  37.18504075
25.16055104
  29.26553553  38.22163057  28.92102091  24.8215439
38.30155495
  34.76020645  22.26869162  28.82593733  32.00975127
36.46437665]
('week1 mean value:', 25.448059395144654)
('week2 mean value:', 29.021568107746155)
('week1 std value:', 4.5316933870843146)
('week2 std value:', 5.4979667086536512)
('p-value', 0.00067676769000677567)
we reject null hypothesis
```

ANOVA (F-TEST) :- The t-test works well when dealing with two groups, but sometimes we want to compare more than two groups at the same time. For example, if we wanted to test whether voter age differs based on some categorical variable like race, we have to compare the means of each level or group the variable. We could carry out a separate t-test for each pair of groups, but when you conduct many tests you increase the chances of false positives. The analysis of variance or ANOVA is a statistical inference test that lets you compare multiple groups at the same time.

F = Between group variability / Within group variability



Unlike the t-distributions, the F-distribution does not have any negative values because between and within-group variability are always positive due to squaring each deviation.

One Way F-test(Anova) :- It tell whether two or more groups are similar or not based on their mean similarity and f-score.

```
df_anova = pd.read_csv('PlantGrowth.csv')
df_anova = df_anova[['weight','group']]

grps = pd.unique(df_anova.group.values)
d_data = {grp:df_anova['weight'][df_anova.group == grp] for grp in
grps}

F, p = stats.f_oneway(d_data['ctrl'], d_data['trt1'], d_data['trt2'])

print("p-value for significance is: ", p)

if p<0.05:
    print("reject null hypothesis")
else:
    print("accept null hypothesis")
```

Chi-Square Test- The test is applied when you have two categorical variables from a single population. It is used to determine whether there is a significant association between the two variables.

For example, in an election survey, voters might be classified by gender (male or female) and voting preference (Democrat, Republican, or Independent). We could use a chi-square test for independence to determine whether gender is related to voting preference

```
df_chi = pd.read_csv('chi-test.csv')
contingency_table=pd.crosstab(df_chi["Gender"],df_chi["Shopping?"])
print('contingency_table :-\n',contingency_table)


#Observed Values
Observed_Values = contingency_table.values
print("Observed Values :-\n",Observed_Values)


b=stats.chi2_contingency(contingency_table)
Expected_Values = b[3]
print("Expected Values :-\n",Expected_Values)


no_of_rows=len(contingency_table.iloc[0:2,0])
no_of_columns=len(contingency_table.iloc[0,0:2])
ddof=(no_of_rows-1)*(no_of_columns-1)
print("Degree of Freedom:-",ddof)
alpha = 0.05


from scipy.stats import chi2
chi_square=sum([(o-e)**2./e for o,e in
zip(Observed_Values,Expected_Values)])
chi_square_statistic=chi_square[0]+chi_square[1]
print("chi-square statistic:-",chi_square_statistic)


critical_value=chi2.ppf(q=1-alpha,df=ddof)
print('critical_value:',critical_value)
```

```
#p-value
p_value=1-chi2.cdf(x=chi_square_statistic,df=ddof)
print('p-value:',p_value)


print('Significance level: ',alpha)
print('Degree of Freedom: ',ddof)
print('chi-square statistic:',chi_square_statistic)
print('critical_value:',critical_value)
print('p-value:',p_value)


if chi_square_statistic>=critical_value:
    print("Reject H0,There is a relationship between 2 categorical 
variables")
else:
    print("Retain H0,There is no relationship between 2 categorical 
variables")

if p_value<=alpha:
    print("Reject H0,There is a relationship between 2 categorical 
variables")
else:
    print("Retain H0,There is no relationship between 2 categorical 
variables")
```

Exercise 9

Linear Regression Analysis:

Download house prediction dataset and explore the data, Prepare the dataset for training, Train a linear regression model, and Make predictions and evaluate the model.

What is Linear Regression Model in Machine Learning

Linear Regression is a Supervised Machine Learning Model for finding the relationship between independent variables and dependent variable. Linear regression performs the task to predict the response (dependent) variable value (y) based on a given (independent) explanatory variable (x). So, this regression technique finds out a linear relationship between x (input) and y (output).

About House Prediction Data Set

Problem Statement – A real state agents want help to predict the house price for regions in the USA. He gave you the dataset to work on and you decided to use the Linear Regression Model. Create a model that will help him to estimate of what the house would sell for.

The dataset contains 7 columns and 5000 rows with CSV extension. The data contains the following columns :

'Avg. Area Income' – Avg. The income of the householder of the city house is located.

'Avg. Area House Age' – Avg. Age of Houses in the same city.

'Avg. Area Number of Rooms' – Avg. Number of Rooms for Houses in the same city.

'Avg. Area Number of Bedrooms' – Avg. Number of Bedrooms for Houses in the same city.

'Area Population' – Population of the city.

'Price' – Price that the house sold at.

'Address' – Address of the houses.

Predicting house prices with linear regression using SciKit-Learn, Pandas, Seaborn and NumPy

Import Libraries

We will be importing SciKit-Learn, Pandas, Seaborn, Matplotlib and Numpy.

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt
```

Importing Data and Checking out

As data is in the CSV file, we will read the CSV using pandas read_csv function and check the first 5 rows of the data frame using head().

```
HouseDF = pd.read_csv('USA_Housing.csv')

HouseDF.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

```
HouseDF.info()
```

```
OUTPUT

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
Avg. Area Income                5000 non-null float64
Avg. Area House Age             5000 non-null float64
Avg. Area Number of Rooms       5000 non-null float64
Avg. Area Number of Bedrooms    5000 non-null float64
Area Population                 5000 non-null float64
Price                           5000 non-null float64
Address                         5000 non-null object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

HouseDF.describe()

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

HouseDF.columns

```
OUTPUT

Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of
Rooms','Avg. Area Number of Bedrooms', 'Area Population', 'Price',
'Address'],       dtype='object')
```

Get Data Ready For Training a Linear Regression Model

Let’s now begin to train out the regression model. We will need to first split up our data into an X list that contains the features to train on, and a y list with the target variable, in this case, the Price column. We will ignore the Address column because it only has text which is not useful for linear regression modeling.

X and y List

X = HouseDF[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',

'Avg. Area Number of Bedrooms', 'Area Population']]

y = HouseDF['Price']

Split Data into Train, Test

Now we will split our dataset into a training set and testing set using sklearn train_test_split(). the training set will be going to use for training the model and testing set for testing the model. We are creating a split of 40% training data and 60% of the training set.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)

X_train and y_train contain data for the training model. X_test and y_test contain data for the testing model. X and y are features and target variable names.

Creating and Training the LinearRegression Model

We will import and create sklearn linearmodel LinearRegression object and fit the training dataset in it.

from sklearn.linear_model import LinearRegression lm = LinearRegression()

lm.fit(X_train,y_train)

```
OUTPUT

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

LinearRegression Model Evaluation

Now let's evaluate the model by checking out its coefficients and how we can interpret them.

print(lm.intercept_)

OUTPUT

-2640159.796851911

coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient']) coeff_df

	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

What does coefficient of data says:

Holding all other features fixed, a 1 unit increase in Avg. Area Income is associated with an increase of $21.52 .

Holding all other features fixed, a 1 unit increase in Avg. Area House Age is associated with an increase of $164883.28 .

Holding all other features fixed, a 1 unit increase in Avg. Area Number of Rooms is associated with an increase of $122368.67 .

Holding all other features fixed, a 1 unit increase in Avg. Area Number of Bedrooms is associated with an increase of $2233.80 .

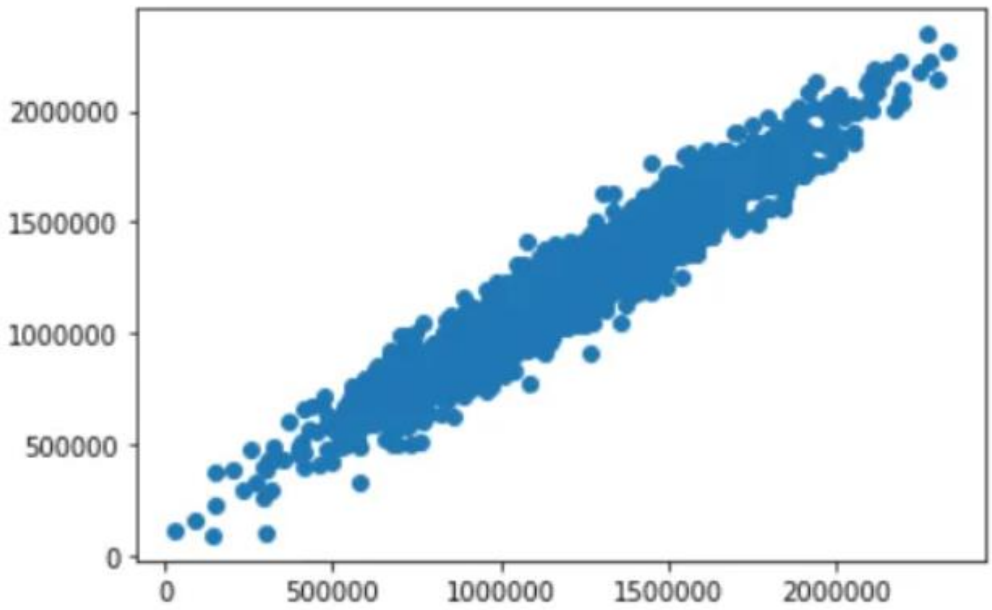
Holding all other features fixed, a 1 unit increase in Area Population is associated with an increase of $15.15 .

Predictions from our Linear Regression Model

Let’s find out the predictions of our test set and see how well it perform.

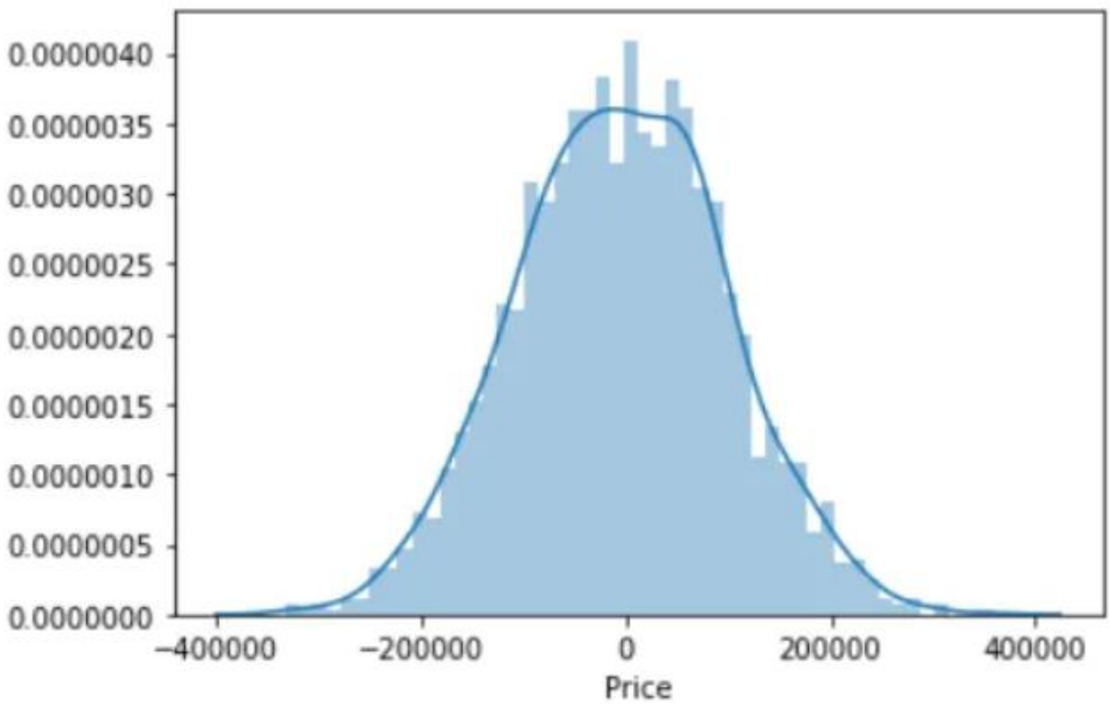
```
predictions = lm.predict(X_test)

plt.scatter(y_test,predictions)
```

<matplotlib.collections.PathCollection at 0x67b87ccc88>



In the above scatter plot, we see data is in a line form, which means our model has done good predictions.

sns.distplot((y_test-predictions),bins=50);



In the above histogram plot, we see data is in bell shape (Normally Distributed), which means our model has done good predictions.

Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac 1n\sum_{i=1}^n|y_i-\hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac 1n\sum_{i=1}^n(y_i-\hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac 1n\sum_{i=1}^n(y_i-\hat{y}_i)^2}$$

Comparing these metrics:

MAE is the easiest to understand because it’s the average error.

MSE is more popular than MAE because MSE “punishes” larger errors, which tends to be useful in the real world.

RMSE is even more popular than MSE because RMSE is interpretable in the “y” units.

All of these are loss functions because we want to minimize them.

```
from sklearn import metrics
```

print('MAE:', metrics.mean_absolute_error(y_test, predictions)) print('MSE:', metrics.mean_squared_error(y_test, predictions)) print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))

OUTPUT

```
MAE: 82288.22251914957
MSE: 10460958907.209501
RMSE: 102278.82922291153
```