**Name: Harsh Chheda**

**Roll Number: 22-15405**

**Class: Msc. Computer Science (Part 2)**

**Subject: Machine Learning**

**Year: 2022-23**

|  |  |  |  |
| --- | --- | --- | --- |
| **INDEX** | | | |
| **NO** | **TITLE** | **PAGE NO** | **SIGN** |
| 1 | Basic Data Pre-Processing | 3-8 |  |
| 2 | Data Handling and Data Modelling | 9-27 |  |
| 3 | Feature Engineering | 28-45 |  |
| 4 | Probability | 46-51 |  |
| 5 | Bayes Theorem | 52 |  |
| 6 | Hypothesis Testing | 53-61 |  |
| 7 | 1. Simple Linear Regression 2. Multiple Linear Regression | 62-70 |  |
| 8 | Logistic Regression | 71-78 |  |
| 9 | K-Means Clustering | 79-80 |  |
| 10 | Random Forest Algorithm | 81-88 |  |
| 11 | Support Vector Machine | 89-97 |  |
| 12 | ANN | 98-100 |  |

**Practical 1**

**Q1) Performing the basic data pre-processing steps.**

🡪

# Importing the libraries

*import* numpy as np

*import* matplotlib.pyplot as plt

*import* pandas as pd

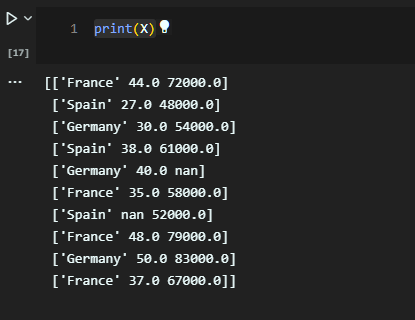
# Importing the dataset

dataset = pd.read\_csv('Data.csv')

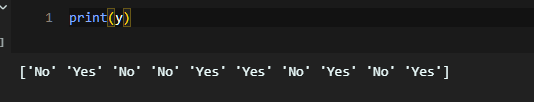
X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

print(X)



print(y)



# Taking care of missing data

*from* sklearn.impute *import* SimpleImputer

imputer = SimpleImputer(*missing\_values*=np.nan, *strategy*='mean')

imputer.fit(X[:, 1:3])

X[:, 1:3] = imputer.transform(X[:, 1:3])

# Encoding categorical data

## Encoding the Independent Variable:

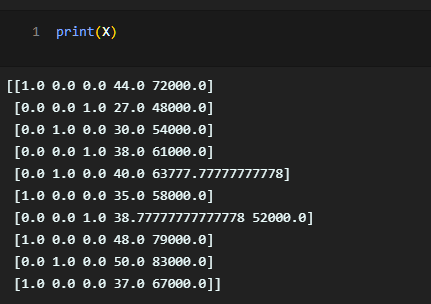
*from* sklearn.compose *import* ColumnTransformer

*from* sklearn.preprocessing *import* OneHotEncoder

ct = ColumnTransformer(*transformers*=[('encoder', OneHotEncoder(), [0])], *remainder*='passthrough')

X = np.array(ct.fit\_transform(X))

print(X)



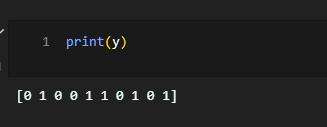
## Encoding the Dependent Variable

*from* sklearn.preprocessing *import* LabelEncoder

le = LabelEncoder()

y = le.fit\_transform(y)

print(y)

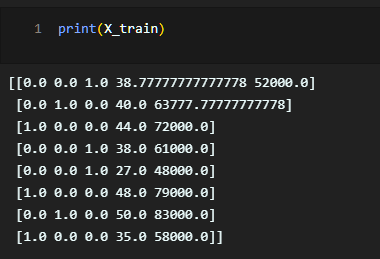


# Splitting the dataset into the Training set and Test set

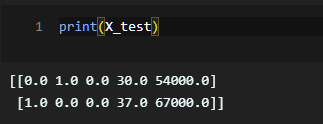
*from* sklearn.model\_selection *import* train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size* = 0.2, *random\_state* = 1)

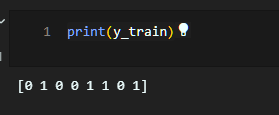
print(X\_train)



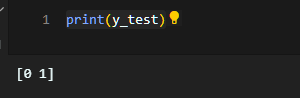
print(X\_test)



print(y\_train)



print(y\_test)



# Feature Scaling

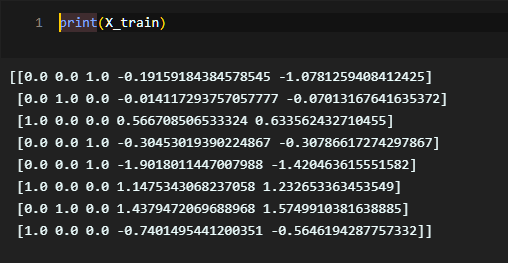
*from* sklearn.preprocessing *import* StandardScaler

sc = StandardScaler()

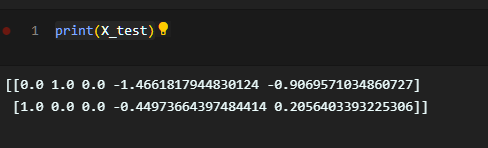
X\_train[:, 3:] = sc.fit\_transform(X\_train[:, 3:])

X\_test[:, 3:] = sc.transform(X\_test[:, 3:])

print(X\_train)



print(X\_test)



**Practical 2**

**Q1) Performing the basic data collection and data modelling steps.**

🡪

**Data Collection**:

* Data collection is defined as the procedure of collecting, measuring and analysing accurate insights for research using standard validated techniques.
* A researcher can evaluate their hypothesis on the basis of collected data. In most cases, data collection is the primary and most important step for research, irrespective of the field of research.
* The approach of data collection is different for different fields of study, depending on the required information.
* The most critical objective of data collection is ensuring that information-rich and reliable data is collected for statistical analysis so that data-driven decisions can be made for research.

my\_dict={'Name':["a","b","c","d","e","f","g"],

        'age':[20,27,35,45,55,43,35],

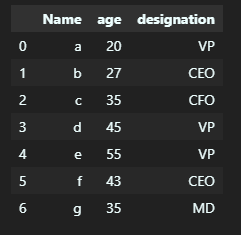
        'designation':["VP","CEO","CFO","VP","VP","CEO","MD"]}

*import* pandas as pd

*import* numpy as np

df=pd.DataFrame(my\_dict)

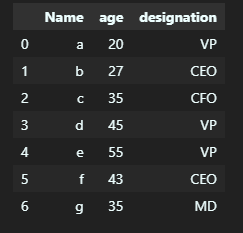
df



## Let’s first create our own CSV file using the data that is currently present in the DataFrame, we can store the data of this DataFrame in CSV format using the API called to\_csv(...) of Pandas DataFrame as:

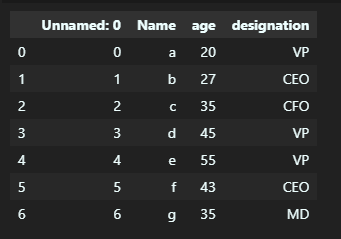
df.to\_csv('Csv example')

df



df\_csv=pd.read\_csv('Csv example')

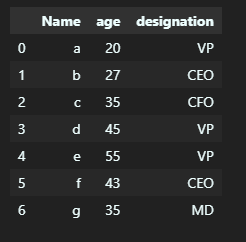
df\_csv



df.to\_csv('CSV Ex',*index*=False)

df\_csv=pd.read\_csv('CSV Ex')

df\_csv



## Load data from csv file and display data without headers

*import* pandas as pd

Location = "student-mat.csv"

df = pd.read\_csv(Location, *header*=None)

df.head()

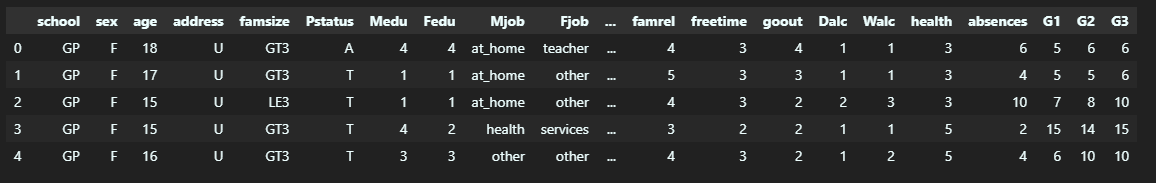


*import* pandas as pd

Location = "student-mat.csv"

df = pd.read\_csv(Location)

df.head()



## Loading Data from CSV File and Adding Headers

*import* pandas as pd

Location = "student-mat.csv"

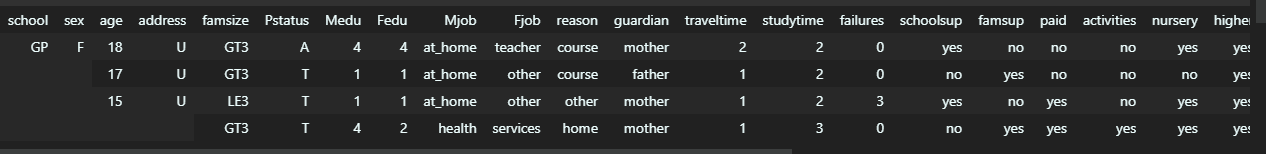
*# To add headers as we load the data...*

df = pd.read\_csv(Location, *names*=['RollNo','Names','Grades'])

*# To add headers to a dataframe*

df.columns = ['RollNo','Names','Grades']

df.head()



*import* pandas as pd

names = ['Bob','Jessica','Mary','John','Mel']

grades = [76,95,77,78,99]

bsdegrees = [1,1,0,0,1]

msdegrees = [2,1,0,0,0]

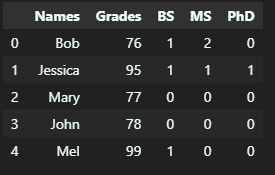
phddegrees = [0,1,0,0,0]

Degrees = zip(names,grades,bsdegrees,msdegrees,phddegrees)

columns = ['Names','Grades','BS','MS','PhD']

df = pd.DataFrame(*data* = Degrees, *columns*=columns)

df



%pip install openpyxl xlrd  xlwt xlsxwriter

*import* pandas as pd

Location = "gradedata.xlsx"

df = pd.read\_excel(Location)

*#Changing column Names*

df.columns = ['first','last','sex','age','exer','hrs','grd','addr']

df.head()



*import* pandas as pd

names = ['Bob','Jessica','Mary','John','Mel']

grades = [76,95,77,78,99]

GradeList = zip(names,grades)

df = pd.DataFrame(*data* = GradeList,*columns*=['Names','Grades'])

writer = pd.ExcelWriter('dataframe.xlsx', *engine*='xlsxwriter')

df.to\_excel(writer, *sheet\_name*='Sheet1')

writer.save()

## Load Data from sqlite

*import* sqlite3

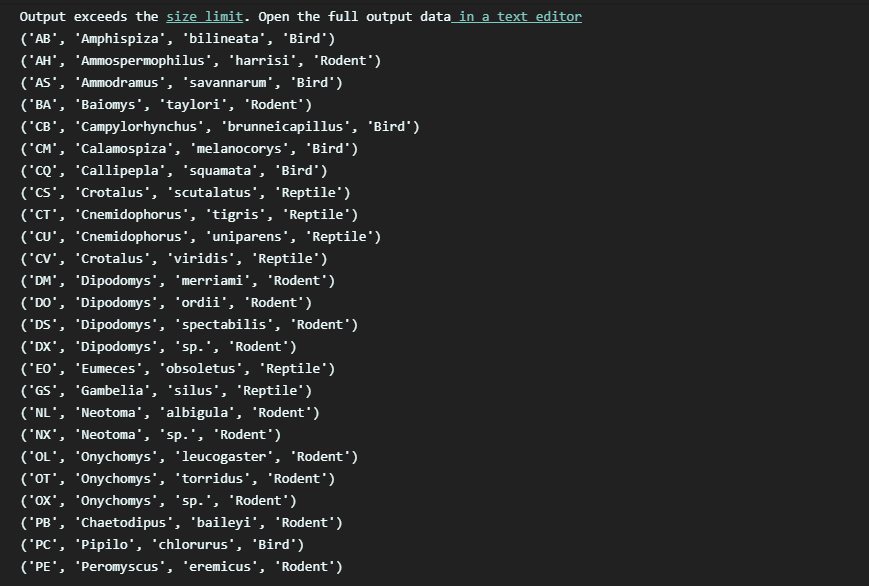
con = sqlite3.connect("portal\_mammals.sqlite")

cur = con.cursor()

*for* row in cur.execute('SELECT \* FROM species;'):

    print(row)

con.close()



*import* sqlite3

*# Create a SQL connection to our SQLite database*

con = sqlite3.connect("portal\_mammals.sqlite")

cur = con.cursor()

*# Return all results of query*

cur.execute('SELECT plot\_id FROM plots WHERE plot\_type="Control"')

print(cur.fetchall())

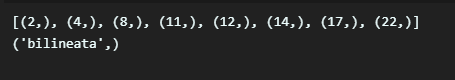
*# Return first result of query*

cur.execute('SELECT species FROM species WHERE taxa="Bird"')

print(cur.fetchone())

*# Be sure to close the connection*

con.close()



*import* pandas as pd

*import* sqlite3

*# Read sqlite query results into a pandas DataFrame*

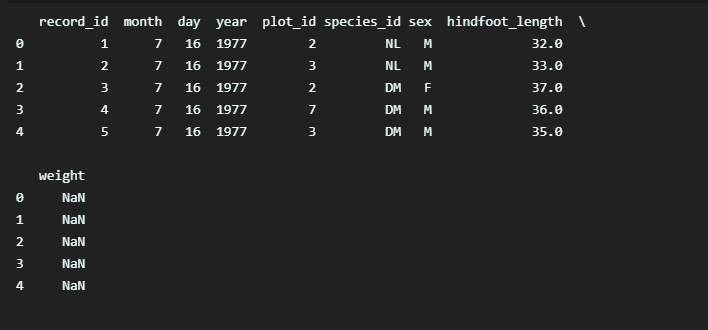
con = sqlite3.connect("portal\_mammals.sqlite")

df = pd.read\_sql\_query("SELECT \* from surveys", con)

*# Verify that result of SQL query is stored in the dataframe*

print(df.head())

con.close()



## Saving data to SQL

*from* pandas *import* DataFrame

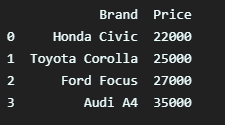
Cars={'Brand':['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],

      'Price':[22000,25000,27000,35000]

      }

df=DataFrame(Cars,*columns*=['Brand','Price'])

print(df)



*import* sqlite3

conn=sqlite3.connect('TestDB1.db')

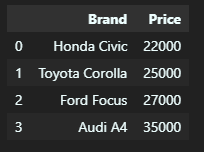
c=conn.cursor()

c.execute('CREATE TABLE CARS2(Brand text, Price number)')

conn.commit()

df.to\_sql('CARS2',conn,*if\_exists*='replace',*index*=False)

df



c.execute('''

SELECT Brand,max(Price) from CARS2

''')



%pip install sqlalchemy

*import* pandas as pd

*import* os

*import* sqlite3 as lite

*from* sqlalchemy *import* create\_engine

studentId=["rj101","rj150","rj134","rj70"]

SName=["Saurabh","Giftson","Vikas","Radha"]

LName=["Chavan","Paul","Bisoi","Rai"]

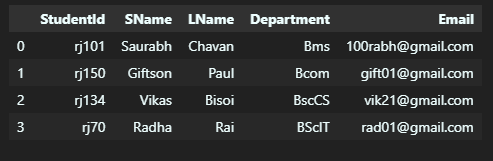
Department=["Bms","Bcom","BscCS","BScIT"]

Email=["100rabh@gmail.com","gift01@gmail.com","vik21@gmail.com","rad01@gmail.com"]

studata = zip(studentId,SName,LName,Department,Email)

df = pd.DataFrame(*data* =studata, *columns*=['StudentId','SName','LName','Department','Email'])

df



df1=df.to\_csv('studentdata.csv',*index*=False,*header*=True)

df1

df2=df.to\_excel('studentdata2.xlsx',*index*=False,*header*=True)

df2

db\_filename = r'studentdata.db'

con = lite.connect(db\_filename)

df.to\_sql('student',

con,

*schema*=None,

*if\_exists*='replace',

*index*=True,

*index\_label*=None,

*chunksize*=None,

*dtype*=None)

con.close()

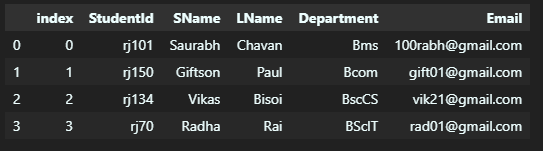
db\_file = r'studentdata.db'

engine = create\_engine(r"sqlite:///{}" .format(db\_file))

sql = 'SELECT \* from student '

studf = pd.read\_sql(sql, engine)

studf



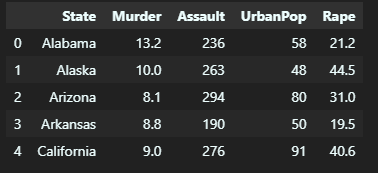
# Data Preprocessing

*import* numpy as np

*import* pandas as pd

state=pd.read\_csv("US\_violent\_crime.csv")

state.head()

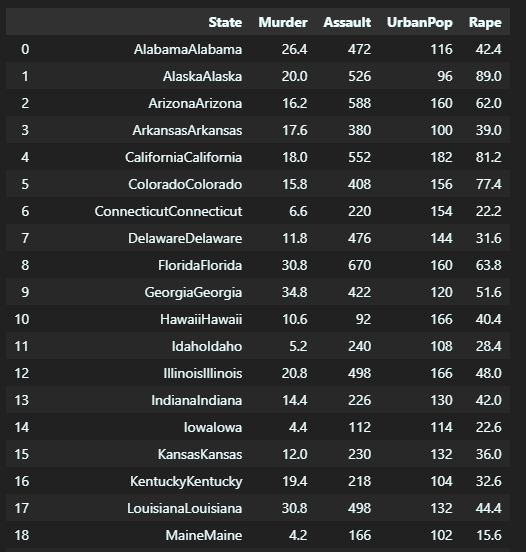


def some\_func(*x*):

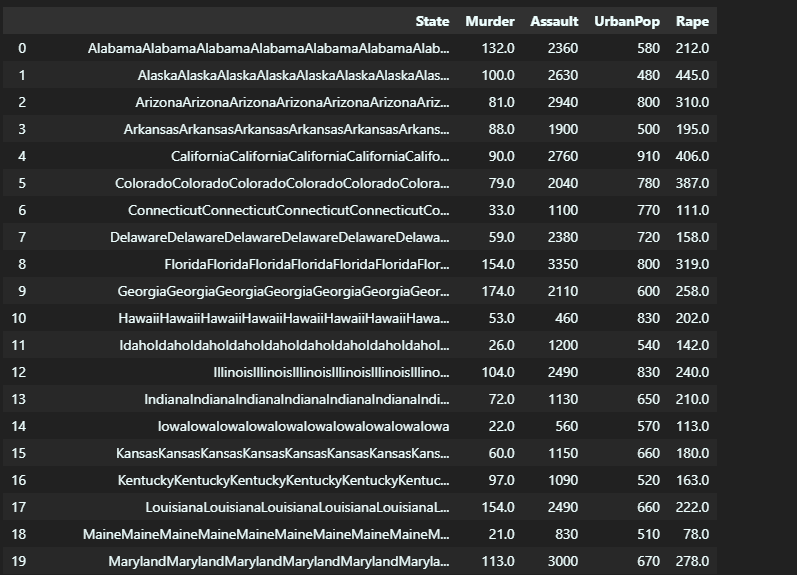
*return* *x*\*2

state.apply(some\_func) *#update each entry of dataframe without any loop*

state.apply(lambda *n*: *n*\*2) *#lambda also works the same*



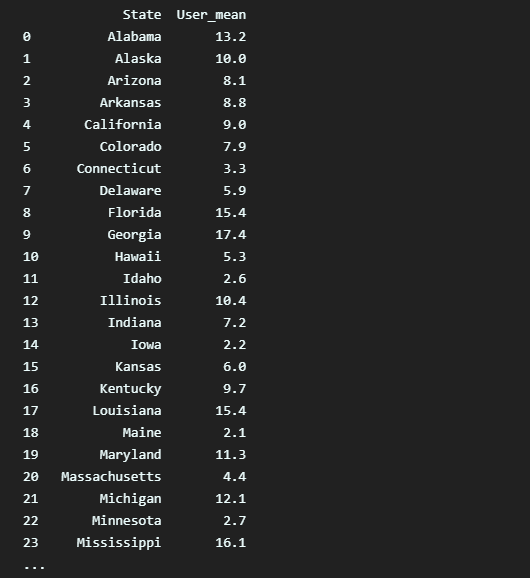
state.transform(*func* = lambda *x* : *x* \* 10)



#usinggroupby

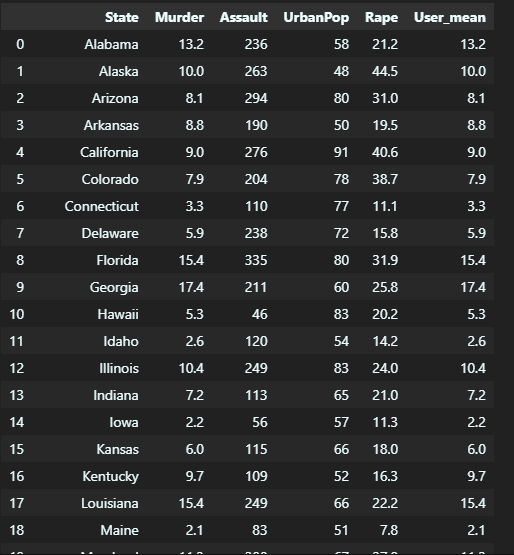
mean\_purchase =state.groupby('State')["Murder"].mean().rename("User\_mean").reset\_index()

print(mean\_purchase)



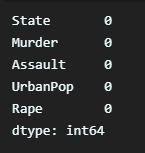
mer=state.merge(mean\_purchase)

mer



*#checking for missing values*

print(state.isnull().sum())



*import* pandas as pd

*import* numpy as np

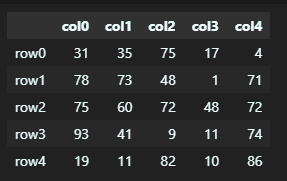
cols=['col0', 'col1', 'col2', 'col3', 'col4']

rows=['row0', 'row1', 'row2', 'row3', 'row4']

data=np.random.randint(0, 100, *size*=(5,5))

df=pd.DataFrame(data, *columns*=cols, *index*=rows)

df.head()



df.iloc[4,2]



## Dealing with 0 and NAN values NaN stands for Not A Number and is one of the common ways to represent the missing value in the data.

df.iloc[3, 3]=0

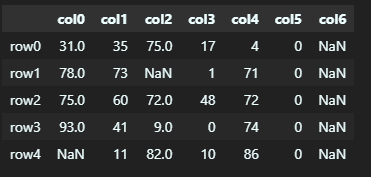
df.iloc[1, 2]=np.nan

df.iloc[4, 0]=np.nan

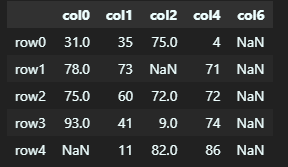
df['col5']=0

df['col6']=np.nan

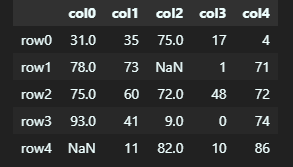
df.head()



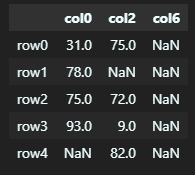
df.loc[:,df.all()]



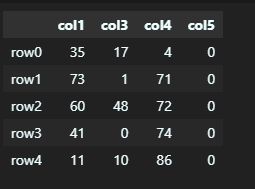
df.loc[:,df.any()]



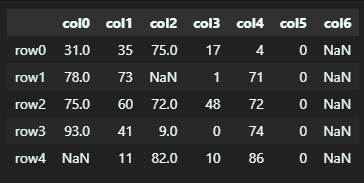
df.loc[:,df.isnull().any()]



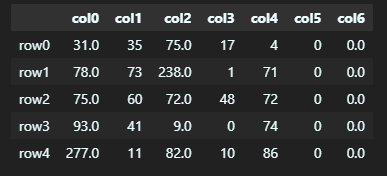
df.loc[:,df.notnull().all()]



df.dropna(*how*="all",*axis*=0)



df.fillna(df.sum())



*#Demonstrate transfomr function using pandas in python*

*import* pandas as pd

*import* numpy as np

*import* random

data = pd.DataFrame({

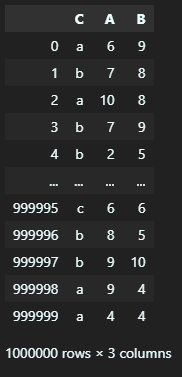
    'C' : [random.choice(('a','b','c')) *for* i in range(1000000)],

    'A' : [random.randint(1,10) *for* i in range(1000000)],

    'B' : [random.randint(1,10) *for* i in range(1000000)]

})

data

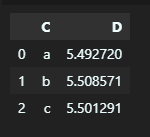


v=data.groupby('C')["A"].mean

v

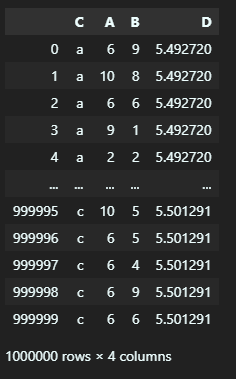
mean=data.groupby('C')["A"].mean().rename("D").reset\_index()

mean



df\_1=data.merge(mean)

df\_1



**Practical 3**

**Q1) Performing the basic feature engineerign steps.**

🡪

# What is feature engineering?

* All machine learning algorithms use some input data to generate outputs. Input data contains many features which may not be in proper form to be given to the model directly. It needs some kind of processing and here feature engineering helps. Feature engineering fulfils mainly two goals:
* It prepares the input dataset in the form which is required for a specific model or machine learning algorithm.
* Feature engineering helps in improving the performance of machine learning models magically.
* According to some surveys, data scientists spend their time on data preparation

*import* pandas as pd

*import* numpy as np

The main feature engineering techniques that will be discussed are:

1. Missing data imputation

2. Categorical encoding

3. Variable transformation

4. Outlier engineering

5. Date and time engineering

## Missing Data Imputation for Feature Engineering

* In your input data, there may be some features or columns which will have missing data, missing values. It occurs if there is no data stored for a certain observation in a variable. Missing data is very common and it is an unavoidable problem especially in real-world data sets. If this data containing a missing value is used then you can see the significance in the results. So, imputation is the act of replacing missing data with statistical estimates of the missing values. It helps you to complete your training data which can then be provided to any model or an algorithm for prediction.
* There are multiple techniques for missing data imputation. These are as follows:-
* Complete case analysis
* Mean / Median / Mode imputation
* Missing Value Indicator

## Complete Case Analysis for Missing Data Imputation

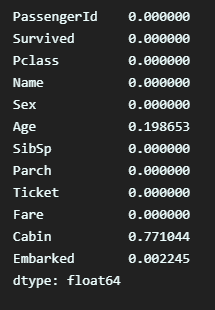
* Complete case analysis is basically analyzing those observations in the dataset that contains values in all the variables. Or you can say, remove all the observations that contain missing values. But this method can only be used when there are only a few observations which has a missing dataset otherwise it will reduce the dataset size and then it will be of not much use.
* So, it can be used when missing data is small but in real-life datasets, the amount of missing data is always big. So, practically, complete case analysis is never an option to use, although you can use it if the missing data size is small.
* Let’s see the use of this on the titanic dataset.

titanic = pd.read\_csv('train.csv')

*# make a copy of titanic dataset*

data1 = titanic.copy()

data1.isnull().mean()



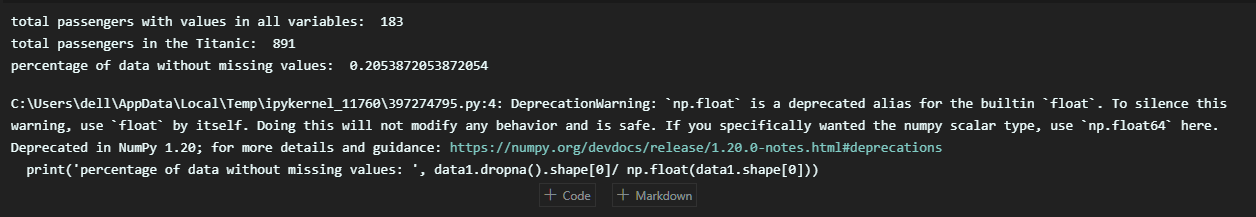
If we remove all the missing observations, we would end up with a very small dataset, given that the Cabin is missing for 77% of the observations

*# check how many observations we would drop*

print('total passengers with values in all variables: ', data1.dropna().shape[0])

print('total passengers in the Titanic: ', data1.shape[0])

print('percentage of data without missing values: ', data1.dropna().shape[0]/ np.float(data1.shape[0]))



So, we have complete information for only 20% of our observations in the Titanic dataset. Thus, Complete Case Analysis method would not be an option for this dataset.

## Mean/ Median/ Mode for Missing Data Imputation

Missing values can also be replaced with the mean, median, or mode of the

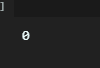
variable(feature). It is widely used in data competitions and in almost every situation. It is suitable to use this technique where data is missing at random places and in small proportions.

*# impute missing values in age in train and test set*

median =data1.Age.median()

data1['Age'].fillna(median, *inplace*=True)

data1['Age'].isnull().sum()



0 represents that now the Age feature has no null values.

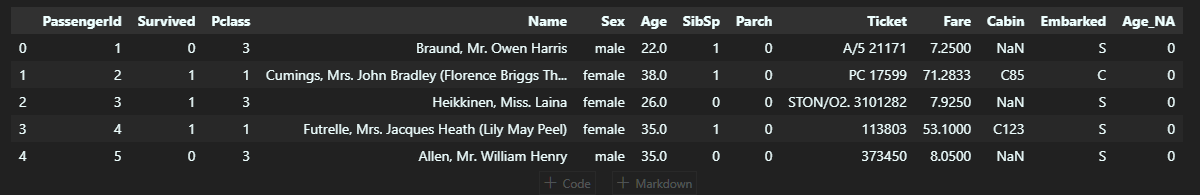
One important point to consider while doing imputation is that it should be done over the training set first and then to the test set. All missing values in the train set and test set should be filled with the value which is extracted from the train set only. This helps in avoiding overfitting.

## Missing Value Indicator for Missing Value Indication

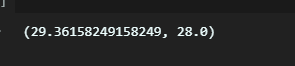
This technique involves adding a binary variable to indicate whether the value is missing for a certain observation. This variable takes the value 1 if the observation is missing, or 0 otherwise. But we still need to replace the missing values in the original variable, which we tend to do with mean or median imputation. By using these 2 techniques together, if the missing value has predictive power, it will be captured by the missing indicator, and if it doesn’t it will be masked by the mean / median imputation.

data1['Age\_NA'] = np.where(data1['Age'].isnull(), 1, 0)

data1.head()



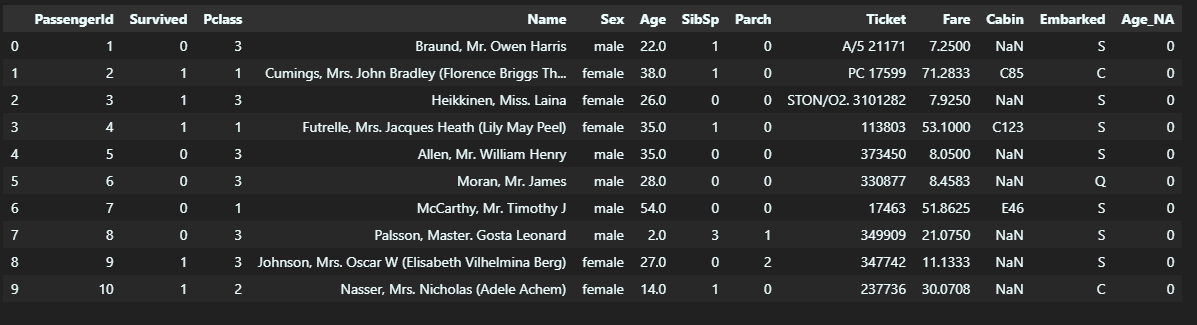
data1.Age.mean(), data1.Age.median()



Now, since mean and median are the same, let’s replace them with the median.

data1['Age'].fillna(data1.Age.median(), *inplace*=True)

data1.head(10)



So, the Age\_NA variable was created to capture the missingness.

# Categorical encoding in Feature Engineering

* Categorical data is defined as that data that takes only a number of values. Let’s understand this with an example. Parameter Gender in a dataset will have categorical values like Male, Female. If a survey is done to know which car people own then the result will be categorical (because the answers would be in categories like Honda, Toyota, Hyundai, Maruti, None, etc.). So, the point to notice here is that data falls in a fixed set of categories.
* If you directly give this dataset with categorical variables to a model, you will get an error. Hence, they are required to be encoded. There are multiple techniques to do so:

1. One-Hot encoding (OHE)
2. Ordinal encoding
3. Count and Frequency encoding
4. Target encoding / Mean encoding

## One-Hot Encoding

* It is a commonly used technique for encoding categorical variables. It basically creates binary variables for each category present in the categorical variable. These binary variables will have 0 if it is absent in the category or 1 if it is present. Each new variable is called a dummy variable or binary variable.
* Example: If the categorical variable is Gender with labels female and male, two boolean variables can be generated called male and female. Male will take 1 if the person is male or 0 otherwise. Similarly for a female variable. See this code below for the titanic dataset.

pd.get\_dummies(data1['Sex']).head()

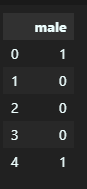


pd.concat([data1['Sex'], pd.get\_dummies(data1['Sex'])], *axis*=1).head()



But you can see that we only need 1 dummy variable to represent Sex categorical variable. So, you can take it as a general formula where if there are n categories, you only need an n-1 dummy variable. So you can easily drop anyone dummy variable. To get n-1 dummy variables simply use this:

pd.get\_dummies(data1['Sex'], *drop\_first*=True).head()



## Count and Frequency Encoding

* In this encoding technique, categories are replaced by the count of the observations that show that category in the dataset. Replacement can also be done with the frequency of the percentage of observations in the dataset. Suppose, if 30 of 100 genders are male we can replace male with 30 or by 0.3.
* This approach is popularly used in data science competitions, so basically it represents how many times each label appears in the dataset.

## Target / Mean Encoding

* In target encoding, also called mean encoding, we replace each category of a variable with the mean value of the target for the observations that show a certain category.
* For example, there is a categorical variable “city”, and we want to predict if the customer will buy a TV provided we send a letter. If 30 percent of the people in the city “London” buy the TV, we would replace London with 0.3.
* So it helps in capturing some information regarding the target at the time of encoding the category and it also does not expands the feature space.
* Hence, it also can be considered as an option for encoding. But it may cause over-fitting to the model, so be careful. Look at this code for implementation:

*import* pandas as pd

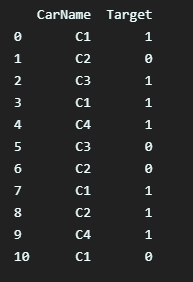
*# creating dataset*

data={'CarName':['C1','C2','C3','C1','C4','C3','C2','C1','C2','C4','C1'],

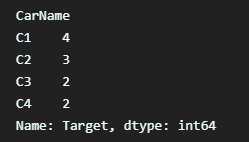
      'Target':[1,0,1,1,1,0,0,1,1,1,0]}

df = pd.DataFrame(data)

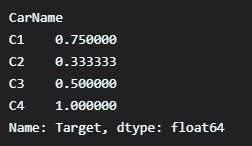
print(df)



df.groupby(['CarName'])['Target'].count()



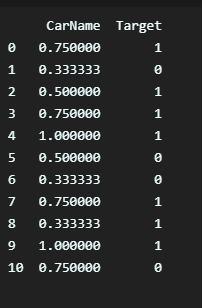
df.groupby(['CarName'])['Target'].mean()



Mean\_encoded = df.groupby(['CarName'])['Target'].mean().to\_dict()

df['CarName'] = df['CarName'].map(Mean\_encoded)

print(df)



## Variable Transformation

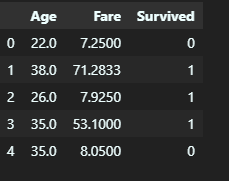
* Machine learning algorithms like linear and logistic regression assume that the variables are normally distributed. If a variable is not normally distributed, sometimes it is possible to find a mathematical transformation so that the transformed variable is Gaussian. Gaussian distributed variables many times boost the machine learning algorithm performance.
* Commonly used mathematical transformations are:

1. Logarithm transformation – log(x)
2. Square root transformation – sqrt(x)
3. Reciprocal transformation – 1 / x
4. Exponential transformation – exp(x)

* Let’s check these out on the titanic dataset.

cols\_reqiuired = ['Age', 'Fare', 'Survived']

data1[cols\_reqiuired].head()



First, we need to fill in missing data. We will start with filling missing data with a random sample.

def impute(*data1*, *variable*):

    df = *data1*.copy()

    df[*variable*+'\_random'] = df[*variable*]

*# extract the random sample to fill the na*

    random\_sample = df[*variable*].dropna().sample(df[*variable*].isnull().sum(), *random\_state*=0)

    random\_sample.index = df[df[*variable*].isnull()].index

    df.loc[df[*variable*].isnull(), *variable*+'\_random'] = random\_sample

*return* df[*variable*+'\_random']

*# fill na*

data1['Age'] = impute(data1, 'Age')

## Now, to visualize the distribution of the age variable we will plot histogram and Q-Q-plot.

## Date and Time Feature Engineering

* Date variables are considered a special type of categorical variable and if they are processed well they can enrich the dataset to a great extent. From the date we can extract various important information like: Month, Semester, Quarter, Day, Day of the week, Is it a weekend or not, hours, minutes, and many more. Let’s use some dataset and do some coding around it.
* For this, we will use the Lending club dataset.
* We will use only two columns from the dataset: issue\_d and last\_pymnt\_d.

*import* pandas as pd

*import* numpy as np

use\_cols = ['issue\_d', 'last\_pymnt\_d']

data = pd.read\_csv('loan.csv', *usecols*=use\_cols, *nrows*=10000)

data.head()

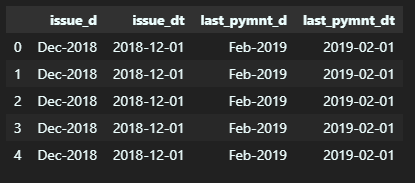


Now, parse dates into DateTime format as they are coded in strings currently.

data['issue\_dt'] = pd.to\_datetime(data.issue\_d)

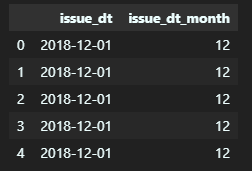
data['last\_pymnt\_dt'] = pd.to\_datetime(data.last\_pymnt\_d)

data[['issue\_d','issue\_dt','last\_pymnt\_d', 'last\_pymnt\_dt']].head()



data['issue\_dt\_month'] = data['issue\_dt'].dt.month

data[['issue\_dt', 'issue\_dt\_month']].head()



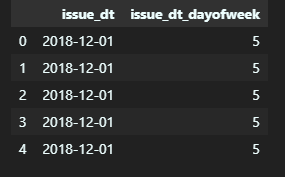
data['issue\_dt\_quarter'] = data['issue\_dt'].dt.quarter

data[['issue\_dt', 'issue\_dt\_quarter']].head()



data['issue\_dt\_dayofweek'] = data['issue\_dt'].dt.dayofweek

data[['issue\_dt', 'issue\_dt\_dayofweek']].head()



# Outlier engineering

Outliers are defined as those values that are unusually high or low with respect to the rest of the observations of the variable. Some of the techniques to handle outliers are:

1. Outlier removal

2. Treating outliers as missing values

3. Outlier capping

How to identify outliers?

For that, the basic form of detection is an extreme value analysis of data. If the distribution of the variable is Gaussian then outliers will lie outside the mean plus or minus three times the standard deviation of the variable. But if the variable is not normally distributed, then quantiles can be used. Calculate the quantiles and then inter quartile range:

Inter quantile is 75th quantile-25quantile.

upper boundary: 75th quantile + (IQR \* 1.5)

lower boundary: 25th quantile – (IQR \* 1.5)

So, the outlier will sit outside these boundaries. Outlier removal

In this technique, simply remove outlier observations from the dataset. In datasets if outliers are not abundant, then dropping the outliers will not affect the data much. But if multiple variables have outliers then we may end up removing a big chunk of data from our dataset. So, this point has to be kept in mind whenever dropping the outliers. Treating outliers as missing values

You can also treat outliers as missing values. But then these missing values also have to be filled. So to fill missing values you can use any of the methods as discussed above in this article. Outlier capping

This procedure involves capping the maximum and minimum values at a predefined value. This value can be derived from the variable distribution. If a variable is normally distributed we can cap the maximum and minimum values at the mean plus or minus three times the standard deviation. But if the variable is skewed, we can use the inter-quantile range proximity rule or cap at the bottom percentiles.

*import* pandas as pd

df = pd.read\_csv('height.csv')

df.head()



max\_thresold = df['height'].quantile(0.95)

max\_thresold



df[df['height']>max\_thresold]

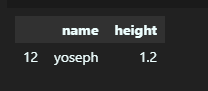


min\_thresold = df['height'].quantile(0.05)

min\_thresold



df[df['height']<min\_thresold]



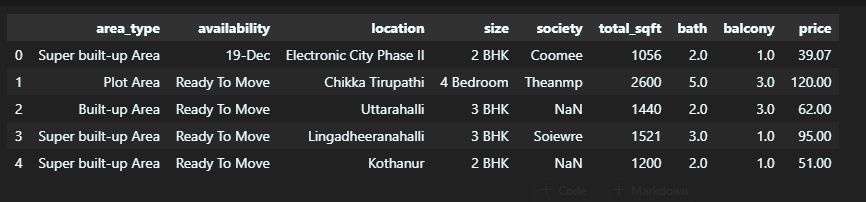
## Remove outliers

df[(df['height']<max\_thresold) & (df['height']>min\_thresold)]



df = pd.read\_csv("BHP.csv")

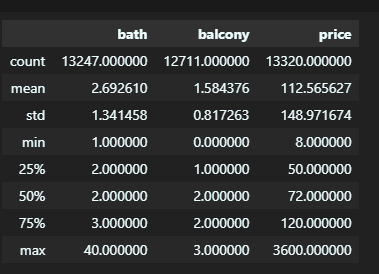
df.head()



df.shape



df.describe()



min\_thresold, max\_thresold = df.price.quantile([0.001, 0.999])

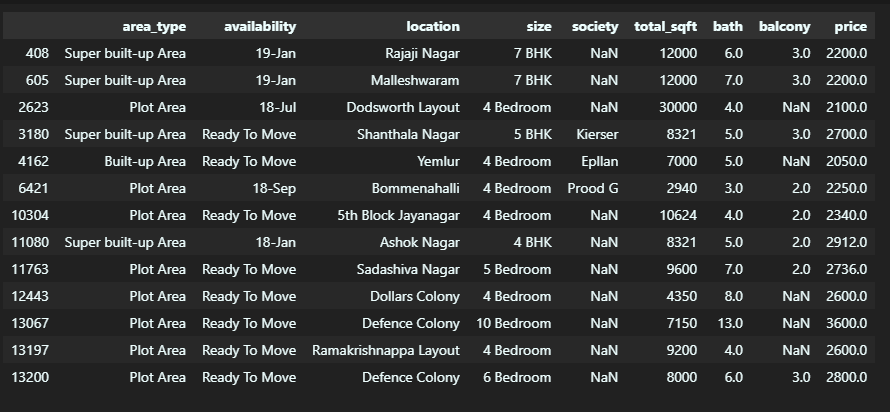
min\_thresold, max\_thresold



df[df.price < min\_thresold]



df[df.price > max\_thresold]

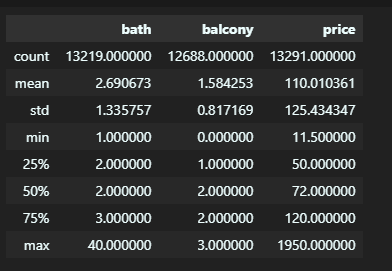


df2 = df[(df.price<max\_thresold) & (df.price>min\_thresold)]

df2.shape



df2.describe()



**Practical 4**

**Q1) Performing the Probability Operations.**

🡪

*# probability of getting 3 when a die is rolled*

ns = 6 *#n(S) = {1,2,3,4,5,6}*

na = 1 *#n(A) = {3}*

pa = na/ns *# P(A)*

print("probability of getting 3 is:",pa)



*# probability of atleast getting one head when a coin is tossed thrice*

ns = 8 *#n(S) = {HHH, HHT, HTH, THH, TTH, THT, HTT, TTT}*

na = 7 *#n(A) = {HHH, HHT, HTH, THH, TTH, THT, HTT}*

pa = na/ns *# P(A)*

print("probability of getting atleast one head is:",pa)



*# A glass jar contains 5 red, 3 blue and 2 green jelly beans. If a jelly bean is chosen at random from the jar,*

*#  mwhat is the probability that it is not blue?*

ns = 10 *#n(S) = {5red,3blue,2green}*

na = 7 *#n(A) = {5red, 2green}*

pa = na/ns *# P(A)*

print("probability of getting not blue jellybean is:",pa)



# Independent and Dependent Events

*# If the probability that person A will be alive in 20 years*

*#is 0.7 and the probability that person B will be alive in*

*# 20 years is 0.5, what is the probability that they will*

*#both be alive in 20 years?*

*#These are independent events, so*

P = 0.7\*0.5

print("probability that they will be alive after 20 years is:",P)



def event\_probability(*n*,*s*):

*return* *n*/*s*

*#A fair die is tossed twice. Find the probability of getting a 4 or 5 on the first toss and a 1,2, or 3 in the second toss.*

pa = event\_probability(2,6) *# probability of getting a 4 or 5 on the first toss*

pb = event\_probability(3,6) *# probability of getting 1,2,3 in second toss*

P = pa\*pb

print("probability of getting a 4 or 5 on the first toss and a 1,2, or 3 in the second toss is:",P)



*# A bag contains 5 white marbles, 3 black marbles and 2 green marbles. In each draw, a marble is drawn from the bag*

*# and not replaced. In three draws, find the probability of obtaining white, black and green in that order.*

pw = event\_probability(5,10)

pb = event\_probability(3,9)

pg = event\_probability(2,8)

print("the probability of obtaining white, black and green in that order is ",(pw\*pb\*pg))



*# Sample Space*

cards = 52

*# Calculate the probability of drawing a heart or a club*

hearts = 13

clubs = 13

heart\_or\_club = event\_probability(hearts, cards) + event\_probability(clubs, cards)

print(heart\_or\_club )



*# Calculate the probability of drawing an ace, king, or a queen*

aces = 4

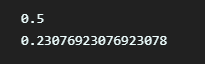
kings = 4

queens = 4

ace\_king\_or\_queen = event\_probability(aces, cards) + event\_probability(kings, cards) + event\_probability(queens, cards)

print(heart\_or\_club)

print(ace\_king\_or\_queen)



*# Calculate the probability of drawing a heart or an ace*

hearts = 13

aces = 4

ace\_of\_hearts = 1

heart\_or\_ace = event\_probability(hearts, cards) + event\_probability(aces, cards) - event\_probability(ace\_of\_hearts, cards)

print(round(heart\_or\_ace, 1))



red\_cards = 26

face\_cards = 12

red\_face\_cards = 6

red\_or\_face\_cards = event\_probability(red\_cards, cards) + event\_probability(face\_cards, cards) - event\_probability(red\_face\_cards, cards)

print(round(heart\_or\_ace, 1))

print(round(red\_or\_face\_cards, 1))



# Complementary Events

*#probabiltiy of not getting 5 when a fair die is rolled*

ns = 6 *#n(S) = {1,2,3,4,5,6}*

na = 1 *#n(A) = {5}*

pa = na/ns *# P(A)*

print("probabilty of not getting 5 is:",1-pa)



*import* pandas as pd

*import* numpy as np

df = pd.read\_csv('student-mat.csv')

df.head(3)



len(df)



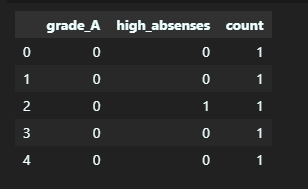
df['grade\_A'] = np.where(df['G3']\*5 >= 80, 1, 0)

df['high\_absenses'] = np.where(df['absences'] >= 10, 1, 0)

df['count'] = 1

df = df[['grade\_A','high\_absenses','count']]

df.head()



final= pd.pivot\_table(

    df,

*values*='count',

*index*=['grade\_A'],

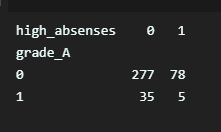
*columns*=['high\_absenses'],

*aggfunc*=np.size,

*fill\_value*=0

)

print(final)



**Practical 5**

**Q1) Bayes Theorem.**

🡪

def bayes\_theorem(*p\_a*, *p\_b\_given\_a*, *p\_b\_given\_not\_a*):

*# calculate P(not A)*

    not\_a = 1 - *p\_a*

*# calculate P(B)*

    p\_b = *p\_b\_given\_a* \* *p\_a* + *p\_b\_given\_not\_a* \* not\_a

*# calculate P(A|B)*

    p\_a\_given\_b = (*p\_b\_given\_a* \* *p\_a*) / p\_b

*return* p\_a\_given\_b

*# P(A)*

p\_a = 0.0002

*# P(B|A)*

p\_b\_given\_a = 0.85

*# P(B|not A)*

p\_b\_given\_not\_a = 0.05

*# calculate P(A|B)*

result = bayes\_theorem(p\_a, p\_b\_given\_a, p\_b\_given\_not\_a)

*# summarize*

print('P(A|B) = %.3f%%' % (result \* 100))



**Practical 6**

**Q1) Hypothesis Testing.**

🡪

%pip install statsmodels

*import* pandas as pd

*import* numpy as np

*import* matplotlib.pyplot as plt

*import* seaborn as sns

*import* scipy.stats as stats

*from* scipy.stats *import* ttest\_1samp

*from* statsmodels.stats.power *import* tt\_ind\_solve\_power

# T test

ages=[10,20,35,50,28,40,55,18,16,55,30,25,43,18,30,28,14,24,16,17,32,35,26,27,65,18,43,23,21,20,19,70]

ages\_mean=np.mean(ages)

print(ages\_mean)



*#Lets take sample*

sample\_size=10

age\_sample=np.random.choice(ages,sample\_size)

age\_sample



*from* scipy.stats *import* ttest\_1samp

ttest,p\_value=ttest\_1samp(age\_sample,30)

print(p\_value)



*if* p\_value < 0.05:

    print("We are rejecting null hypothesis")

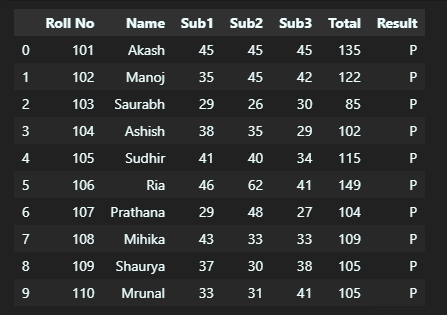
*else*:

    print("We are accepting null hypothesis")

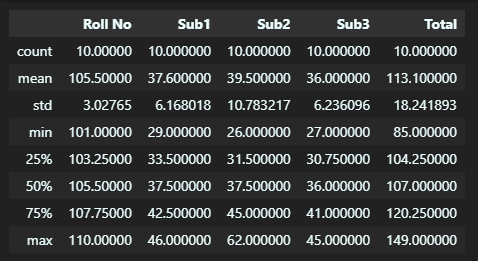


df=pd.read\_excel('result.xlsx')

df



df.describe()



# One way hypothesis

Ho = "mu <= 113"

*# alt hyp*

Ha = "mu > 113"

*# alpha*

al = 0.05

*# mu -> mean*

mu = 113

*# tail type*

tt = 1

*# data*

marks = df['Total'].values

print("Ho:", Ho)

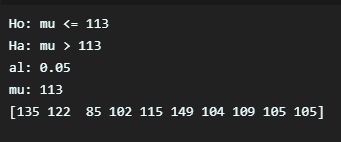
print("Ha:", Ha)

print("al:", al)

print("mu:", mu)

print(marks)

print("")



ts, pv = ttest\_1samp(marks, mu)

print("t-stat",ts)

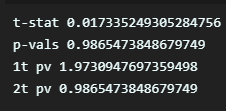
print("p-vals",pv)

t2pv = pv

t1pv = pv\*2

print("1t pv",t1pv)

print("2t pv",t2pv)



*if* tt == 1:

*if* t1pv < al:

        print("Null Hypothesis: Rejected")

        print("Conclusion:",Ha)

*else*:

        print("Null Hypothesis: Not Rejected")

        print("Conclusion:",Ho)

*else*:

*if* t2pv < al/2:

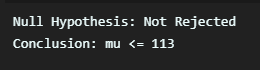
        print("Null Hypothesis: Rejected")

        print("Conclusion:",Ha)

*else*:

        print("Null Hypothesis: Not Rejected")

        print("Conclusion:",Ho)



# Two way hypothesis

*# Problem: Check if the total mean marks is  equal to 113*

*#Ho:    m = 113*

*#Ha:    m != 113*

*#Tail: Two*

*#Test: One Sample Mean without std*

*# null hyp*

Ho = "mu = 113"

*# alt hyp*

Ha = "mu != 113"

*# alpha*

al = 0.05

*# mu - mean*

mu = 113

*# tail type*

tt = 2

*# data*

marks = df['Total'].values

*# print*

print("Ho:", Ho)

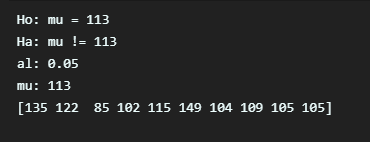
print("Ha:", Ha)

print("al:", al)

print("mu:", mu)

print(marks)

print("")



ts, pv = ttest\_1samp(marks, mu)

print("t-stat",ts)

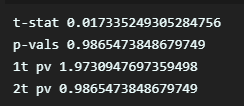
print("p-vals",pv)

t2pv = pv

t1pv = pv\*2

print("1t pv",t1pv)

print("2t pv",t2pv)



*if* tt == 1:

*if* t1pv < al:

        print("Null Hypothesis: Rejected")

        print("Conclusion:",Ha)

*else*:

        print("Null Hypothesis: Not Rejected")

        print("Conclusion:",Ho)

*else*:

*if* t2pv < al/2:

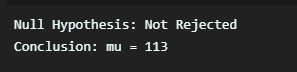
        print("Null Hypothesis: Rejected")

        print("Conclusion:",Ha)

*else*:

        print("Null Hypothesis: Not Rejected")

        print("Conclusion:",Ho)

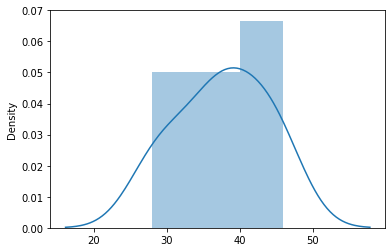


# AB Testing

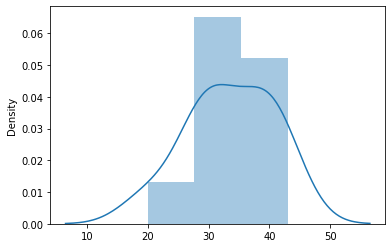
subj1 = np.array([45,36,29,40,46,37,43,39,28,33])

subj2 = np.array([40,20,30,35,29,43,40,39,28,31])

sns.distplot(subj1)



sns.distplot(subj2)



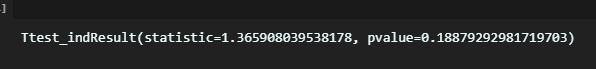
t\_stat, p\_val= stats.ttest\_ind(subj1,subj2)

t\_stat , p\_val



*#perform two sample t-test with equal variances*

stats.ttest\_ind(subj1, subj2, *equal\_var*=True)



effect\_size=abs((subj1.mean()-subj2.mean())/(subj1.std()-subj2.std()))

sample\_size=10

alpha=0.05

ratio=1.0

statistical\_power = tt\_ind\_solve\_power(*effect\_size*=effect\_size, *nobs1*=sample\_size, *alpha*=alpha, *ratio*=1.0, *alternative*='two-sided')

print(statistical\_power)



type\_2\_error=1-statistical\_power

type\_2\_error



**Practical 7**

**Q1) Simple Linear Regression**

🡪

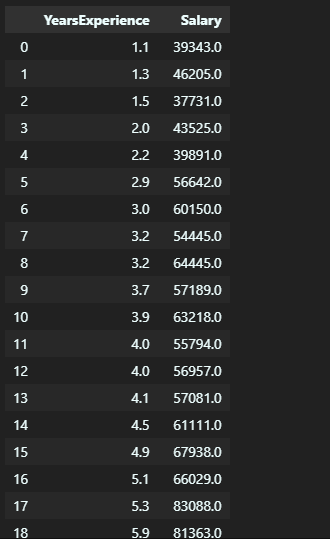
*import* numpy as np

*import* matplotlib.pyplot as plt

*import* pandas as pd

dataset = pd.read\_csv('Salary\_Data.csv')

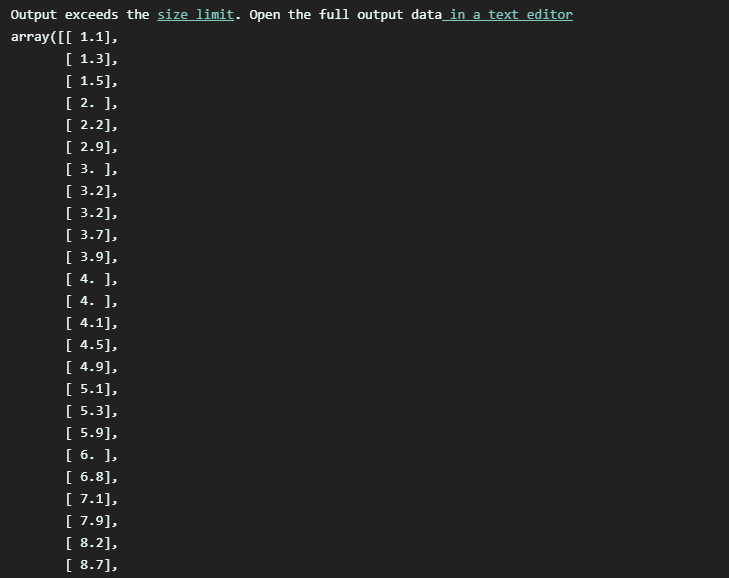
dataset



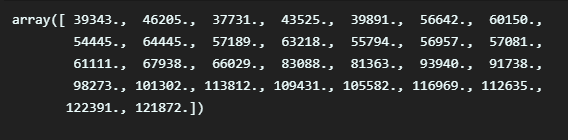
x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

x



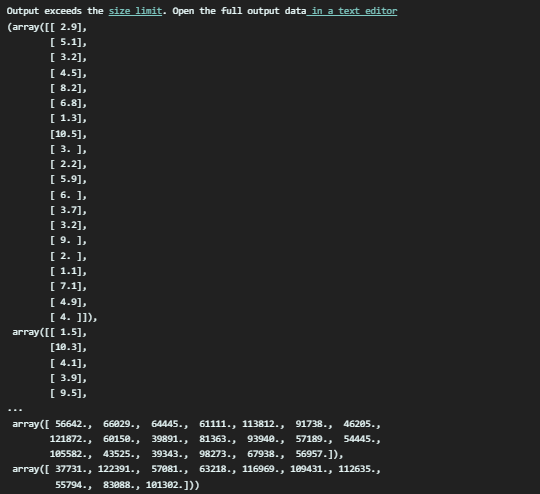
Y



*from* sklearn.model\_selection *import* train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, *test\_size* = 1/3, *random\_state* = 0)

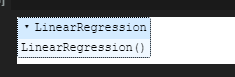
x\_train, x\_test, y\_train, y\_test



*from* sklearn.linear\_model *import* LinearRegression

regressor = LinearRegression()

regressor.fit(x\_train, y\_train)



y\_pred= regressor.predict(x\_test)

x\_pred= regressor.predict(x\_train)

plt.scatter(x\_train, y\_train, *color* = 'red')

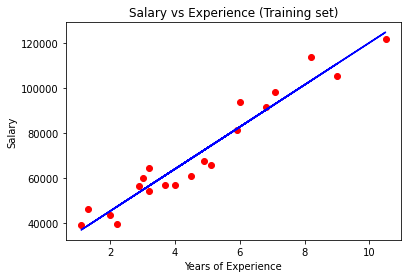
plt.plot(x\_train, regressor.predict(x\_train), *color* = 'blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()



plt.scatter(x\_test, y\_test, *color* = 'red')

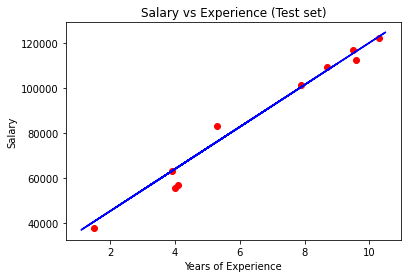
plt.plot(x\_train, regressor.predict(x\_train), *color* = 'blue')

plt.title('Salary vs Experience (Test set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()



**Q2) Multiple Linear Regression**

**🡪**

*import* numpy as np

*import* matplotlib.pyplot as plt

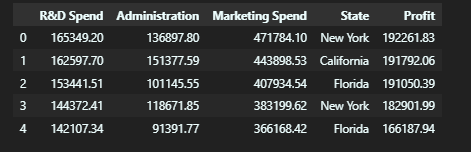
*import* pandas as pd

dataset = pd.read\_csv('50\_Startups.csv')

dataset.head()

dataset = pd.read\_csv('50\_Startups.csv')

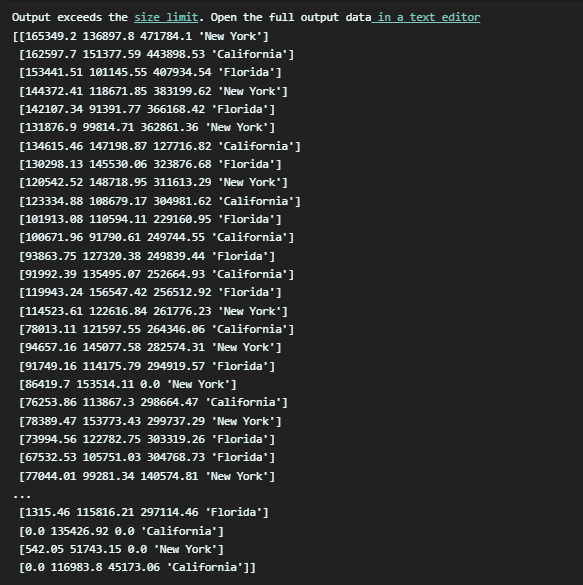
dataset.head()

****

x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

print(x)

****

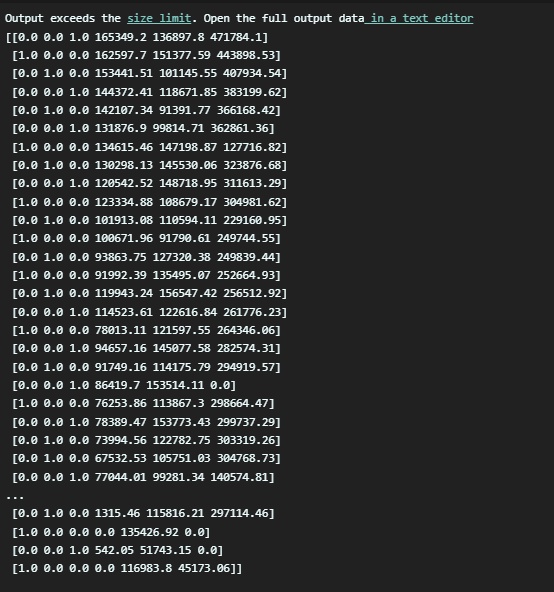
from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(*transformers*=[('encoder', OneHotEncoder(), [3])], *remainder*='passthrough')

x = np.array(ct.fit\_transform(x))

print(x)

****

*from* sklearn.model\_selection *import* train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, *test\_size* = 0.2, *random\_state* = 0)

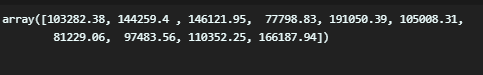
*from* sklearn.linear\_model *import* LinearRegression

regressor = LinearRegression()

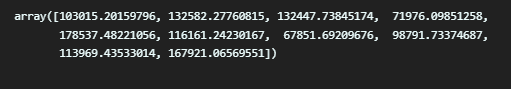
regressor.fit(X\_train, y\_train)

y\_pred = regressor.predict(X\_test)

y\_test

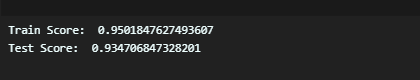
****

y\_pred

****

print('Train Score: ', regressor.score(X\_train, y\_train))

print('Test Score: ', regressor.score(X\_test, y\_test))

****

**Practical 8**

**Q1) K-Nearest Neightbors**

🡪

*import* numpy as np

*import* matplotlib.pyplot as plt

*import* pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

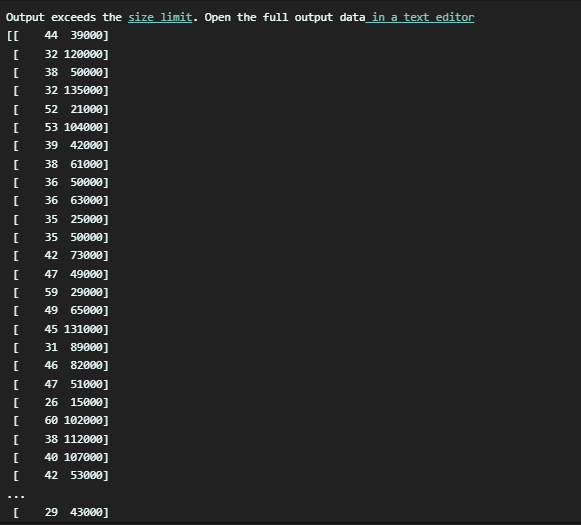
X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

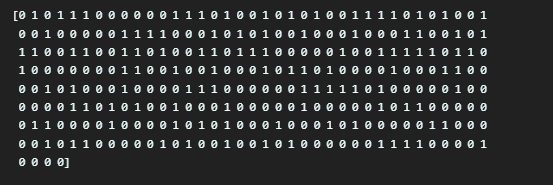
*from* sklearn.model\_selection *import* train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size* = 0.25, *random\_state* = 0)

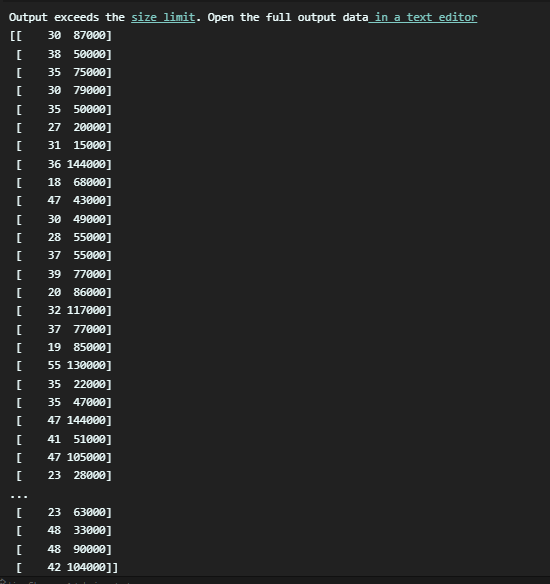
print(X\_train)



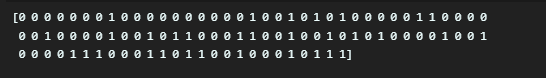
print(y\_train)



print(X\_test)



print(y\_test)



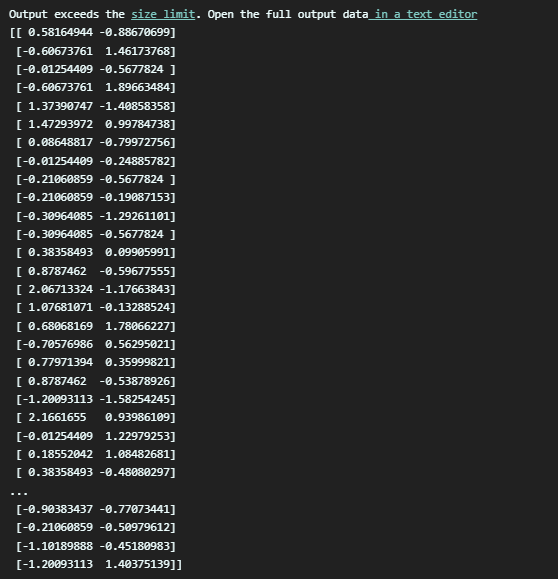
*from* sklearn.preprocessing *import* StandardScaler

sc = StandardScaler()

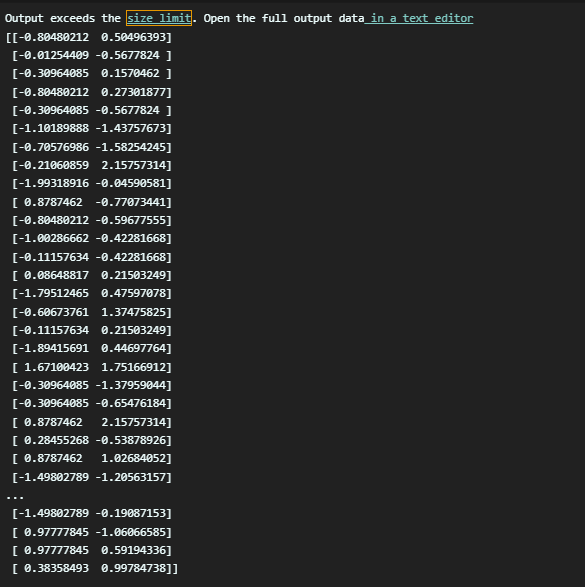
X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

print(X\_train)



print(X\_test)



from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(*n\_neighbors* = 5, *metric* = 'minkowski', *p* = 2)

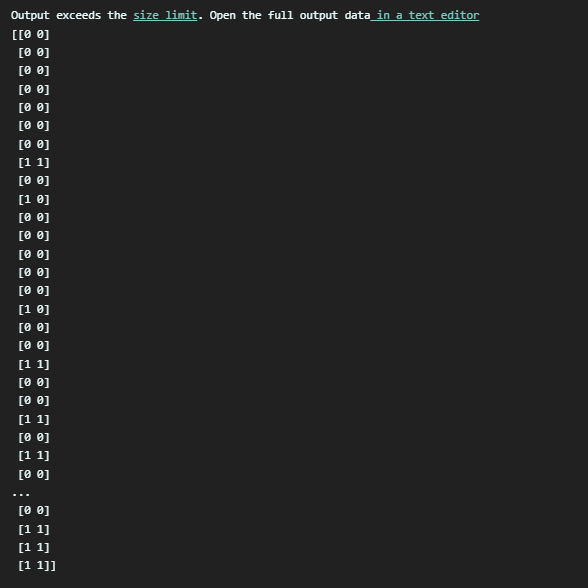
classifier.fit(X\_train, y\_train)

print(classifier.predict(sc.transform([[30,87000]])))



y\_pred = classifier.predict(X\_test)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))



*from* sklearn.metrics *import* confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)



*from* matplotlib.colors *import* ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(*start* = X\_set[:, 0].min() - 10, *stop* = X\_set[:, 0].max() + 10, *step* = 1),

                     np.arange(*start* = X\_set[:, 1].min() - 1000, *stop* = X\_set[:, 1].max() + 1000, *step* = 1))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

*alpha* = 0.75, *cmap* = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

*for* i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], *c* = ListedColormap(('red', 'green'))(i), *label* = j)

plt.title('K-NN (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

*from* matplotlib.colors *import* ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(*start* = X\_set[:, 0].min() - 10, *stop* = X\_set[:, 0].max() + 10, *step* = 1),

                     np.arange(*start* = X\_set[:, 1].min() - 1000, *stop* = X\_set[:, 1].max() + 1000, *step* = 1))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

*alpha* = 0.75, *cmap* = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

*for* i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], *c* = ListedColormap(('red', 'green'))(i), *label* = j)

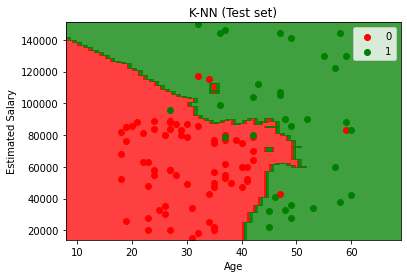
plt.title('K-NN (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



**Practical 9**

**Q1) K-Means Clustering**

🡪

*import* numpy as np

*import* matplotlib.pyplot as plt

*import* pandas as pd

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

*from* sklearn.cluster *import* KMeans

wcss = []

*for* i in range(1, 11):

    kmeans = KMeans(*n\_clusters* = i, *init* = 'k-means++', *random\_state* = 42)

    kmeans.fit(X)

    wcss.append(kmeans.inertia\_)

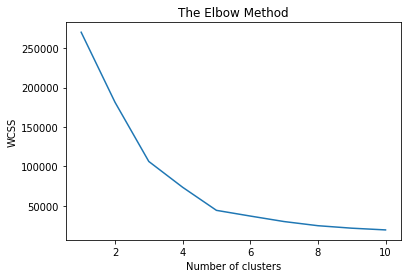
plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()



kmeans = KMeans(*n\_clusters* = 5, *init* = 'k-means++', *random\_state* = 42)

y\_kmeans = kmeans.fit\_predict(X)

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], *s* = 100, *c* = 'red', *label* = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], *s* = 100, *c* = 'blue', *label* = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], *s* = 100, *c* = 'green', *label* = 'Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], *s* = 100, *c* = 'cyan', *label* = 'Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], *s* = 100, *c* = 'magenta', *label* = 'Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], *s* = 300, *c* = 'yellow', *label* = 'Centroids')

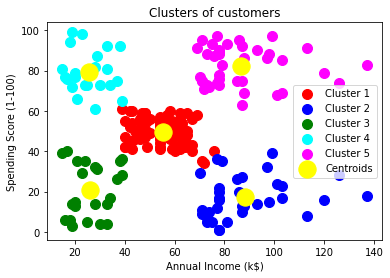
plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()



**Practical 10**

**Q1) Random Forest Classification**

🡪

*import* numpy as np

*import* matplotlib.pyplot as plt

*import* pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

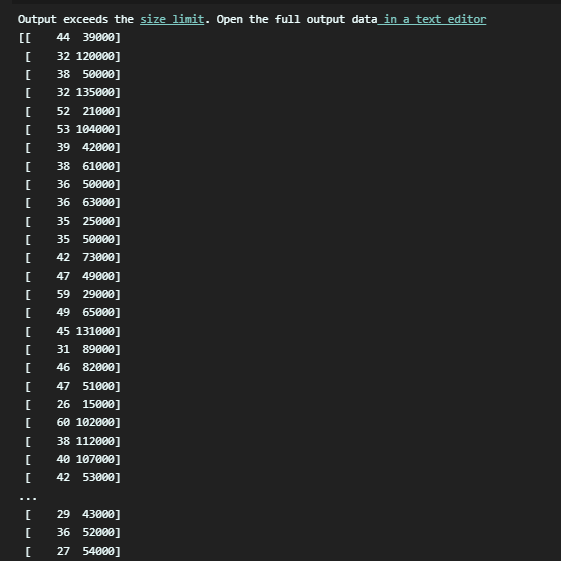
X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

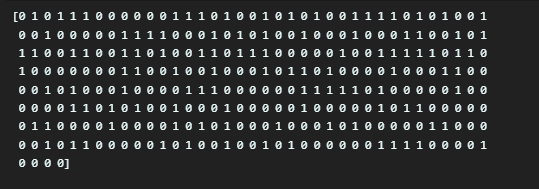
*from* sklearn.model\_selection *import* train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size* = 0.25, *random\_state* = 0)

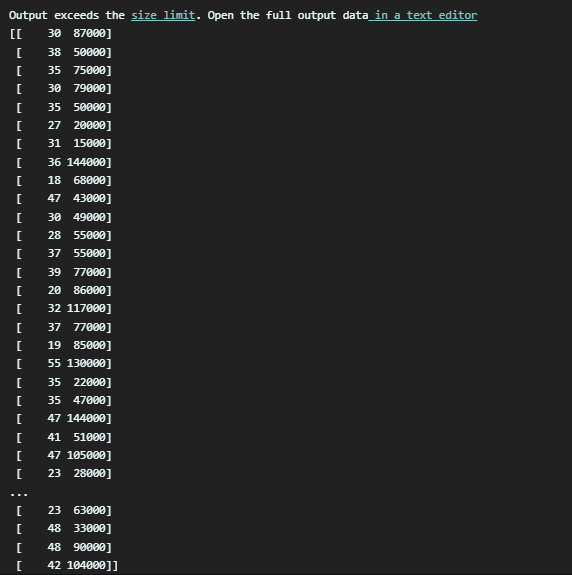
print(X\_train)



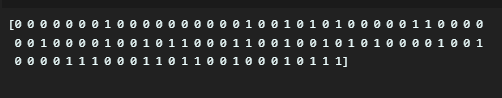
print(y\_train)



print(X\_test)



print(y\_test)



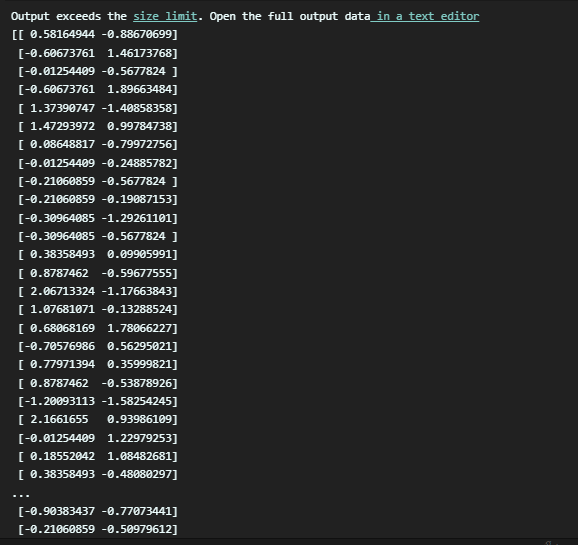
*from* sklearn.preprocessing *import* StandardScaler

sc = StandardScaler()

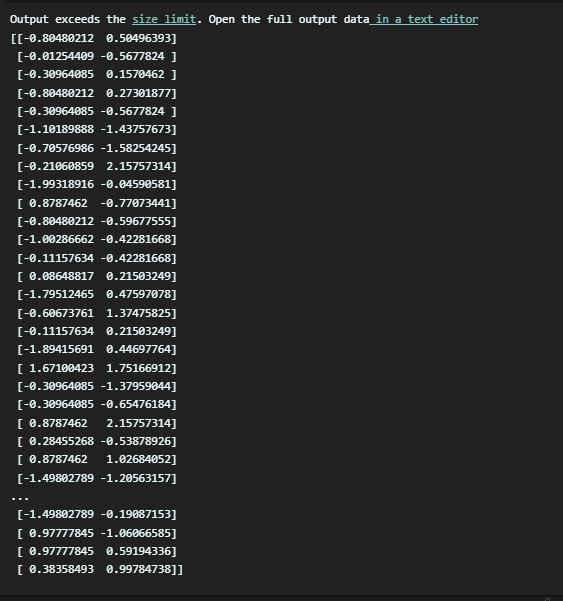
X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

print(X\_train)



print(X\_test)



from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(*n\_estimators* = 10, *criterion* = 'entropy', *random\_state* = 0)

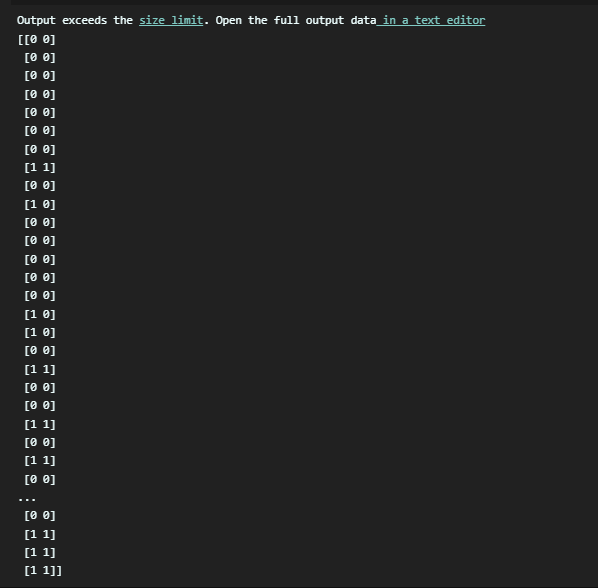
classifier.fit(X\_train, y\_train)

print(classifier.predict(sc.transform([[30,87000]])))



y\_pred = classifier.predict(X\_test)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))



from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)



*from* matplotlib.colors *import* ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(*start* = X\_set[:, 0].min() - 10, *stop* = X\_set[:, 0].max() + 10, *step* = 0.25),

                     np.arange(*start* = X\_set[:, 1].min() - 1000, *stop* = X\_set[:, 1].max() + 1000, *step* = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

*alpha* = 0.75, *cmap* = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

*for* i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], *c* = ListedColormap(('red', 'green'))(i), *label* = j)

plt.title('Random Forest Classification (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

*from* matplotlib.colors *import* ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(*start* = X\_set[:, 0].min() - 10, *stop* = X\_set[:, 0].max() + 10, *step* = 0.25),

                     np.arange(*start* = X\_set[:, 1].min() - 1000, *stop* = X\_set[:, 1].max() + 1000, *step* = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

*alpha* = 0.75, *cmap* = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

*for* i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], *c* = ListedColormap(('red', 'green'))(i), *label* = j)

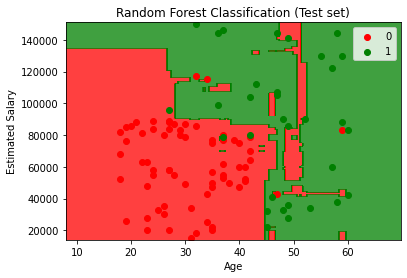
plt.title('Random Forest Classification (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



**Practical 11**

**Q1) Support Vector Machine**

🡪

*import* numpy as np

*import* matplotlib.pyplot as plt

*import* pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

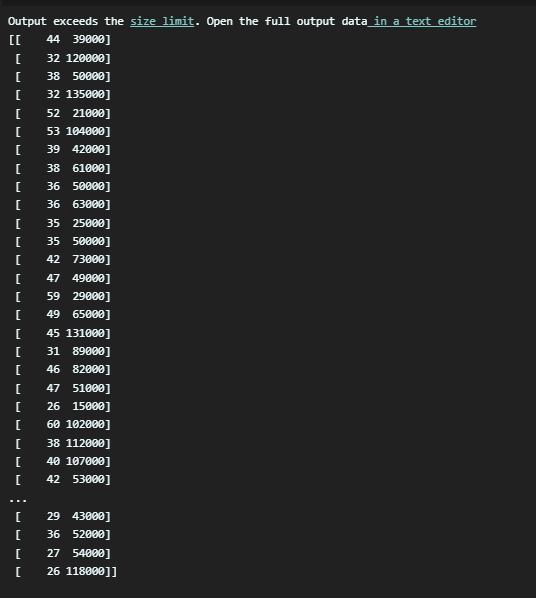
X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

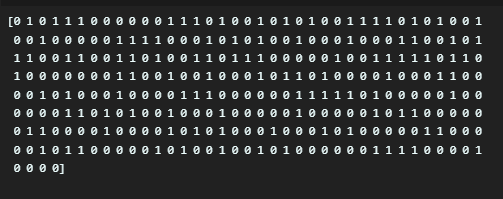
*from* sklearn.model\_selection *import* train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size* = 0.25, *random\_state* = 0)

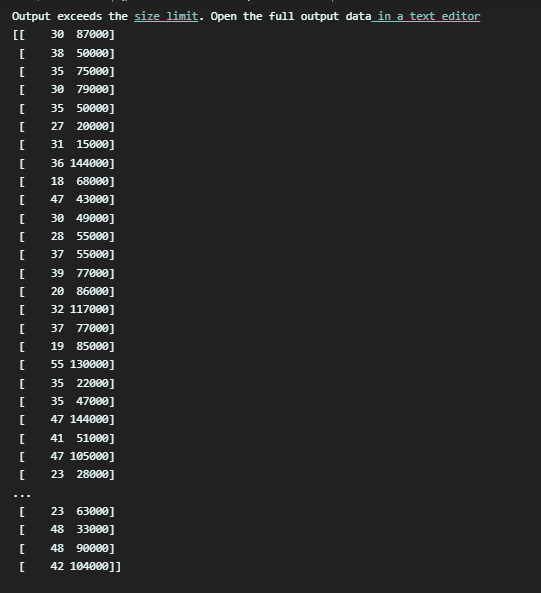
print(X\_train)



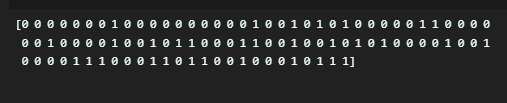
print(y\_train)



print(X\_test)



print(y\_test)



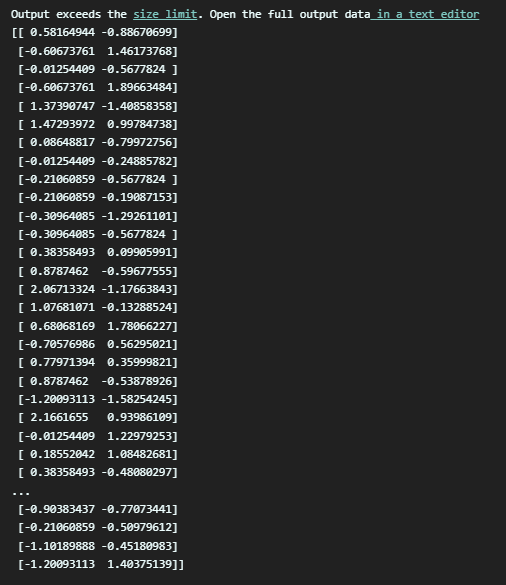
*from* sklearn.preprocessing *import* StandardScaler

sc = StandardScaler()

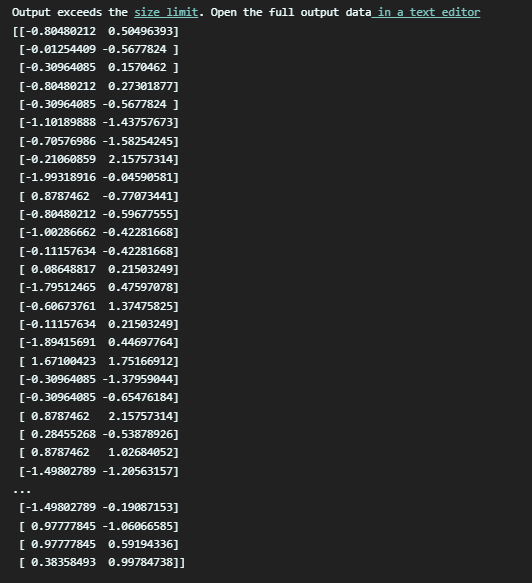
X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

print(X\_train)



print(X\_test)



*from* sklearn.svm *import* SVC

classifier = SVC(*kernel* = 'linear', *random\_state* = 0)

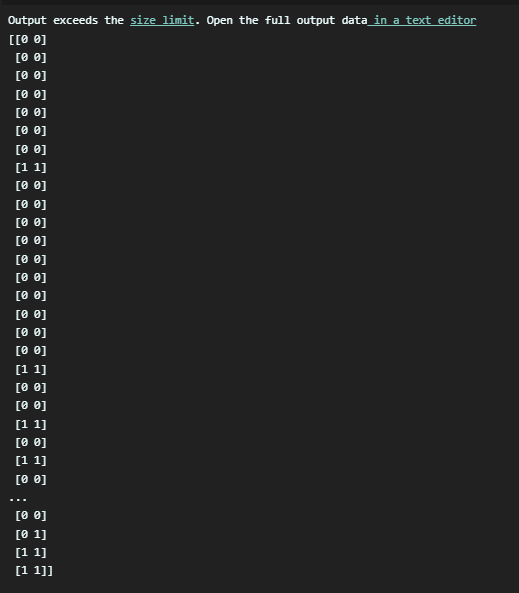
classifier.fit(X\_train, y\_train)

print(classifier.predict(sc.transform([[30,87000]])))



y\_pred = classifier.predict(X\_test)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

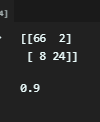


from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)



*from* matplotlib.colors *import* ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(*start* = X\_set[:, 0].min() - 10, *stop* = X\_set[:, 0].max() + 10, *step* = 0.25),

                     np.arange(*start* = X\_set[:, 1].min() - 1000, *stop* = X\_set[:, 1].max() + 1000, *step* = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

*alpha* = 0.75, *cmap* = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

*for* i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], *c* = ListedColormap(('red', 'green'))(i), *label* = j)

plt.title('SVM (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

*from* matplotlib.colors *import* ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(*start* = X\_set[:, 0].min() - 10, *stop* = X\_set[:, 0].max() + 10, *step* = 0.25),

                     np.arange(*start* = X\_set[:, 1].min() - 1000, *stop* = X\_set[:, 1].max() + 1000, *step* = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

*alpha* = 0.75, *cmap* = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

*for* i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], *c* = ListedColormap(('red', 'green'))(i), *label* = j)

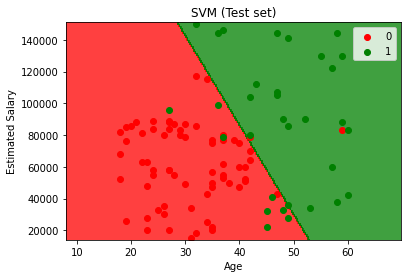
plt.title('SVM (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



**Practical 12**

**Q1) ANN**

🡪

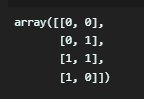
*import* numpy as np

*#assign input values*

input\_value = np.array([[0,0],[0,1],[1,1],[1,0]])

input\_value.shape

input\_value



*#assign output values*

output=np.array([0,1,1,0])

output=output.reshape(4,1)

output.shape



*#assign weights*

weigths=np.array([[0.1],[0.2]])

weigths



*#assign bias*

bias=0.3

*#Activation function*

def sigmoid\_func(*x*):

*return* 1/(1 + np.exp(-*x*))

*#Derivative of Sigmoid function*

def der(*x*):

*return* sigmoid\_func(*x*) \* (1- sigmoid\_func(*x*))

*#Updating weigths*

*for* epochs in range(10000):

    input\_arr = input\_value

    weighted\_sum=np.dot(input\_arr, weigths) + bias

    first\_output=sigmoid\_func(weighted\_sum)

    error =first\_output - output

    total\_error=np.square(np.subtract(first\_output,output)).mean()

    first\_der = error

    second\_der = der(first\_output)

    derivative = first\_der \* second\_der

    t\_input = input\_value.T

    final\_derivative = np.dot(t\_input, derivative)

*#update weights*

    weigths = weigths - 0.05 \* final\_derivative

*#update bias*

*for* i in derivative:

        bias = bias - 0.05 \* i

print(bias)



pred = np.array([0,1])

result = np.dot(pred, weigths) + bias

res = sigmoid\_func(result)

print(res)

