Group B
Assignment No. 1.

* Title:- Predict the price of the uber ride from a given pickup point to the agreed drop-off location. Perform following tasks.

1. Pre-process of the dataset.
2. Implement linear regression and random forest regression models.
3. Evaluate the models and compare their respective scores like R2, RMSE, etc.

* Dataset Description :-

The project is about on world's largest taxi company Uber inc. In this project, we're looking to predict the fare for their future transactional cases. Uber delivers service to lakhs of customers daily.

Link : https://www.kaggle.com/datasets/yasserh/uber-fares-dataset.

JSPM
Page No. :

Objective :

Students should be able to preprocess dataset and identify outliers to check correlation and implement linear regression and random forest regression models. Evaluate them with respective sources like R2, AMSE etc.

Theory :-

Data Preprocessing :-

Data Preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the crucial step while creating a machine learning model.

When creating a machine learning project, it is raw data and making it suitable for amachine learning.

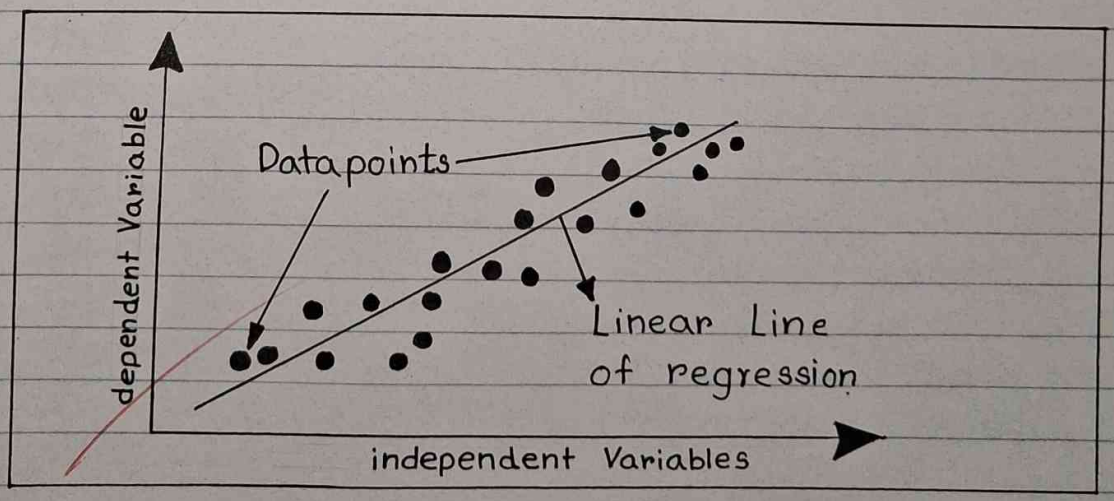
Linear Regression :

Linear regression is one of the easiest and most popular Machine Learning algorithm. It is a statistical method that

is used for predictive analysis. Linear regression make predictions for continuous /real or numeric variables such as salary, age, product price, etc.

Linear regression algorithm show a linear relationship between (y) and one or more independent (y) variable, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped line representing the relationship between the variables. Consider the below diagram:-



JSPM
Page No. :

Random Forest Regression Models:

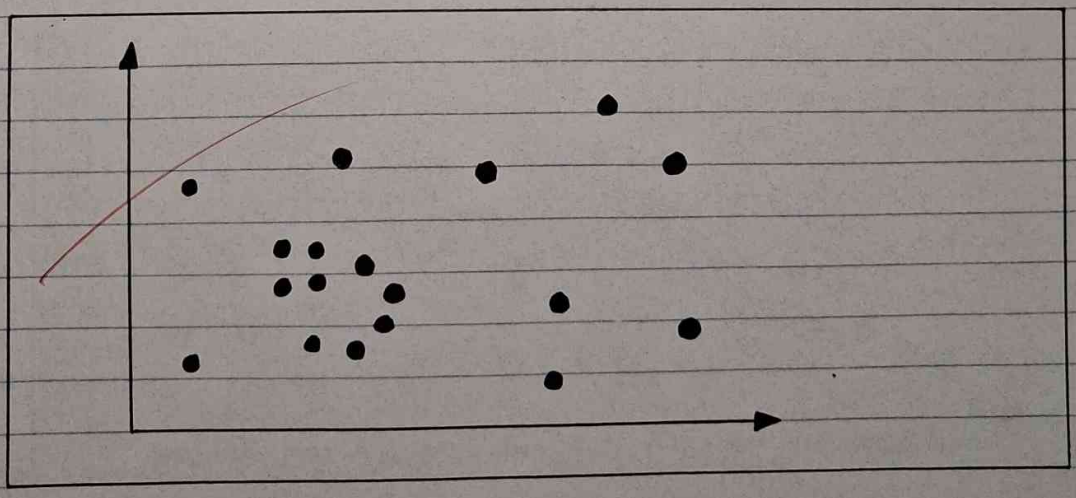
Random Forest is a popular machine learning algorithm that belongs to the supervisedlearning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classiers to solve a complex problem and to improve the performance of model.

Outlier :

The major thing above the outliers is what you do with them. If you are going to analyze any task to analyze data sets, you will always have some assumptions based on how this data is generated. If you some data points that are likely to contain some form of error, then these are definitely outliers. and some data points that are likely to contain some form of error, then these are data mining process involves the analysis and prediction of data that the holds. In 1969, Grubbs introduced.

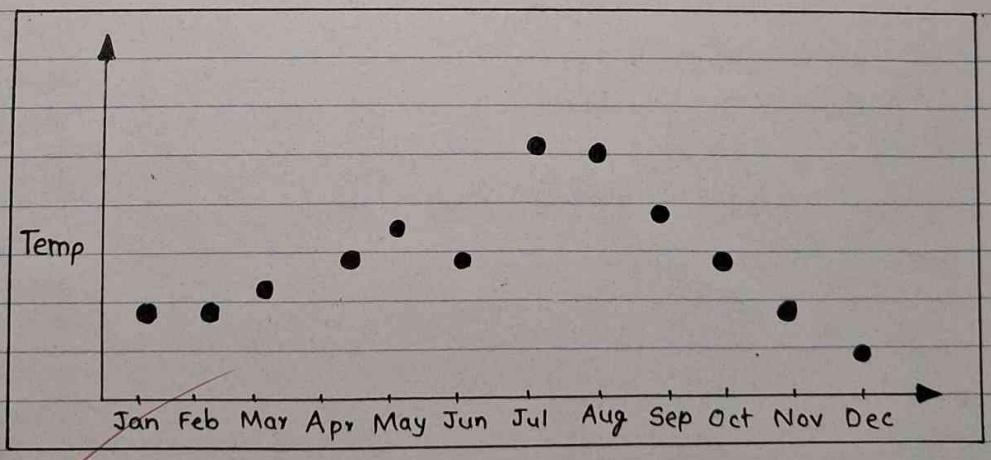
Collective Outliers:

In a given set of data, when a group of data points deviates from the rest of the data set is called collective outliers. Here, the particular set of data objects may not be outliers, but when you consider the data objects as a whole, they may behave as outliers. To identify thetypes of different outliers. you need to go through background information about the relationship between the behavior of outliers shown by different data objects. For example, in an Intrusion Detection System, the DOS package from one system to another is taken as normal behavior.



Contextual Outliers

As the name suggests, "Contextual" means this outlier introduced within a context. For example, in the speech recognition technique, the single backgro--und noise. Contextual outliers are also known as Conditional outliers. These types of outliers happen if a data objective deviates from the outlier data points because of anay condition and behavioral attributes. Contextual outlier analysis enables the users to examine outlier in different contexts and conditions, which can be useful in various applications. For example, A temperature reading of 45 degrees Celsius may behave as an outlier in a rainy season.



Haversine:

The Haversine Formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface. It is important for use in navigation.

Matplotlib:

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multiplatform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Mean Squared Error:

The Mean Squared Error (MSE) or Mean

Squared Deviation (MSD) of an estimator measures the average of error squares i.e. the average squared difference between the estimated values and true value. It is a risk function, corresponding to the expected value of the squared error loss. It is always non-negative and value close to zero are better. The MSE is the second moment of the error (about the origin) and thus incorporates both the variance of the estimator and its bias.

✳ Conclusion :

In this way we have explored concept correlation and implement linear regression and random forest regression models.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import chi2_contingency
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from random import randrange, uniform

Train_Data = pd.read_csv(r'trainn.csv')
Train_Data.head(1)
```

Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pass

```
Train_Data.drop(labels='Unnamed: 0',axis=1,inplace=True)

Train_Data.drop(labels='key',axis=1,inplace=True)

Train_Data.shape
```

```
(200000, 7)
```

```
test = pd.read_csv(r'testt.csv')
test.head(1)
```

Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latit

```
test.shape,Train_Data.shape
```

```
((50000, 10), (200000, 7))
```

```
Train_Data.head()
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

```
test.head()
```

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latit
0	0	37338	31401407	2011-02-10 19:06:00.000000169	2011-02-10 19:06:00 UTC	-73.951662	40.790710	-73.947570	40.756
1	1	160901	33158465	2011-06-23 09:24:00.000000157	2011-06-23 09:24:00 UTC	-73.951007	40.771508	-73.974075	40.763
2	2	40428	10638355	2012-07-14 10:37:00.000000149	2012-07-14 10:37:00 UTC	-73.996473	40.747930	-73.990298	40.756
3	3	63353	3836845	2014-10-19 22:27:05.0000002	2014-10-19 22:27:05 UTC	-73.997934	40.716890	-73.952617	40.727

As this is Taxi fare data and we know there are many factors which affect the price of taxi like

1. Travelled distance
2. Time of Travel
3. Demand and Availability of Taxi
4. Some special places are more costlier like Airport or other places where there might be toll

```
print(Train_Data.info())
print(test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 7 columns):
fare_amount          200000 non-null float64
pickup_datetime      200000 non-null object
pickup_longitude     200000 non-null float64
pickup_latitude      200000 non-null float64
dropoff_longitude    199999 non-null float64
dropoff_latitude     199999 non-null float64
passenger_count      200000 non-null int64
dtypes: float64(5), int64(1), object(1)
memory usage: 10.7+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 10 columns):
Unnamed: 0           50000 non-null int64
Unnamed: 0.1         50000 non-null int64
Unnamed: 0.1.1       50000 non-null int64
key                  50000 non-null object
pickup_datetime      50000 non-null object
pickup_longitude     50000 non-null float64
pickup_latitude      50000 non-null float64
dropoff_longitude    50000 non-null float64
dropoff_latitude     50000 non-null float64
passenger_count      50000 non-null int64
dtypes: float64(4), int64(4), object(2)
memory usage: 3.8+ MB
None
```

here we can see there are 8columns in which 6 numerics and 2 are object.

Lets change the type of pickup_datetime from object to DateTime

```
Train_Data["pickup_datetime"] = pd.to_datetime(Train_Data["pickup_datetime"])

print(Train_Data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 7 columns):
fare_amount          200000 non-null float64
pickup_datetime      200000 non-null datetime64[ns, UTC]
pickup_longitude     200000 non-null float64
pickup_latitude      200000 non-null float64
dropoff_longitude    199999 non-null float64
dropoff_latitude     199999 non-null float64
passenger_count      200000 non-null int64
dtypes: datetime64[ns, UTC](1), float64(5), int64(1)
memory usage: 10.7 MB
None
```

```
Train_Data.describe()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	12.500000	-73.967154	40.767158	-73.963658	40.768001	2.000000
max	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

1.Here first thing which we can see is minimum value of fare is negative which is -52 which is not the valid value, so we need to remove the fare which are negative values.

2.Secondly, passenger_count minimum value is 0 and maximum value is 208 which impossible, so we need to remove them as well, for safer side we can think that a taxi can have maximum 7 people.

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
Train_Data.isnull().sum()
```

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

```
Train_Data.dropna(axis = 0, inplace= True)
```

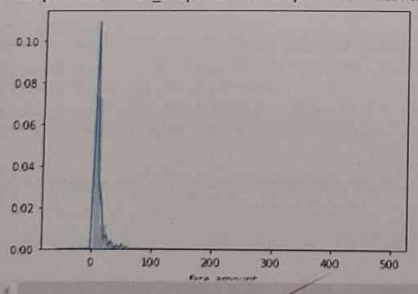
```
print(Train_Data.isnull().sum())
```

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

Lets see the statistics of our data

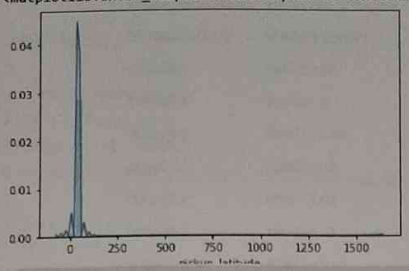
```
sns.distplot(Train_Data['fare_amount'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xd297c2b608>
```



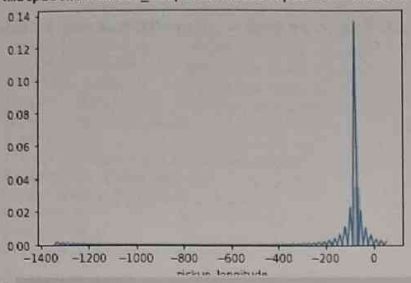

```
sns.distplot(Train_Data['pickup_latitude'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xd297d1b648>
```



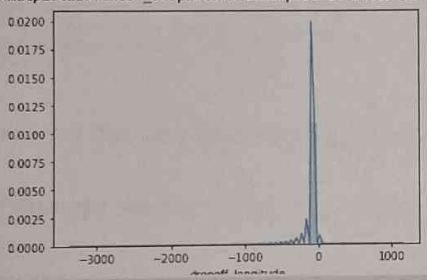

```
sns.distplot(Train_Data['pickup_longitude'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xd297e87648>
```



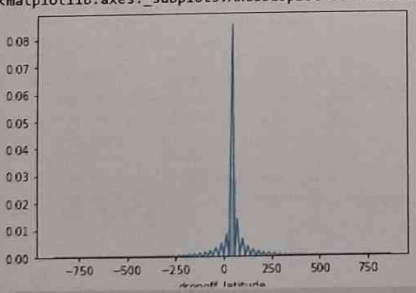

```
sns.distplot(Train_Data['dropoff_longitude'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xd297e84c48>
```




```
sns.distplot(Train_Data['dropoff_latitude'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xd298b91588>
```




```
print("drop_off latitude min value",Train_Data["dropoff_latitude"].min())
print("drop_off latitude max value",Train_Data["dropoff_latitude"].max())
print("drop_off longitude min value", Train_Data["dropoff_longitude"].min())
print("drop_off longitude max value",Train_Data["dropoff_longitude"].max())
print("pickup latitude min value",Train_Data["pickup_latitude"].min())
```

```
print("pickup latitude max value",Train_Data["pickup_latitude"].max())
print("pickup longitude min value",Train_Data["pickup_longitude"].min())
print("pickup longitude max value",Train_Data["pickup_longitude"].max())
```

```
drop_off latitude min value -881.9855130000002
drop_off latitude max value 872.6976279999999
drop_off longitude min value -3356.6663
drop_off longitude max value 1153.5726029999996
pickup latitude min value -74.01551500000002
pickup latitude max value 1644.421482
pickup longitude min value -1340.64841
pickup longitude max value 57.418457
```

```
print("drop_off latitude min value",test["dropoff_latitude"].min())
print("drop_off latitude max value",test["dropoff_latitude"].max())
print("drop_off longitude min value", test["dropoff_longitude"].min())
print("drop_off longitude max value",test["dropoff_longitude"].max())
print("pickup latitude min value",test["pickup_latitude"].min())
print("pickup latitude max value",test["pickup_latitude"].max())
print("pickup longitude min value",test["pickup_longitude"].min())
print("pickup longitude max value",test["pickup_longitude"].max())
```

```
drop_off latitude min value -74.00110699999998
drop_off latitude max value 47.433332
drop_off longitude min value -1491.194073
drop_off longitude max value 40.796262
pickup latitude min value -74.001047
pickup latitude max value 42.46842
pickup longitude min value -80.734728
pickup longitude max value 40.812005
```

```
min_longitude=-1491.194073,
min_latitude=-74.001047,
max_longitude=40.812005,
max_latitude=41.709555
```

```
min_longitude=-1491.194073,
min_latitude=-74.001047,
max_longitude=40.812005,
max_latitude=41.709555
```

```
tempdf=Train_Data[(Train_Data["dropoff_latitude"]<min_latitude) |
                  (Train_Data["pickup_latitude"]<min_latitude) |
                  (Train_Data["dropoff_longitude"]<min_longitude) |
                  (Train_Data["pickup_longitude"]<min_longitude) |
                  (Train_Data["dropoff_latitude"]>max_latitude) |
                  (Train_Data["pickup_latitude"]>max_latitude) |
                  (Train_Data["dropoff_longitude"]>max_longitude) |
                  (Train_Data["pickup_longitude"]>max_longitude) ]
print("before droping",Train_Data.shape)
Train_Data.drop(tempdf.index,inplace=True)
print("after droping",Train_Data.shape)
```

```
before droping (199999, 7)
after droping (199961, 7)
```

```
import calendar
Train_Data['day']=Train_Data['pickup_datetime'].apply(lambda x:x.day)
Train_Data['hour']=Train_Data['pickup_datetime'].apply(lambda x:x.hour)
Train_Data['month']=Train_Data['pickup_datetime'].apply(lambda x:x.month)
Train_Data['year']=Train_Data['pickup_datetime'].apply(lambda x:x.year)
Train_Data['weekday']=Train_Data['pickup_datetime'].apply(lambda x: calendar.day_name[x.weekday()])
```

```
Train_Data.weekday = Train_Data.weekday.map({'Sunday':0,'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,'Saturday':6})
```

```
Train_Data.drop(labels = 'pickup_datetime',axis=1,inplace=True)
```

```
Train_Data.head(1)
Train_Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199961 entries, 0 to 199999
Data columns (total 11 columns):
fare_amount          199961 non-null float64
```

```
pickup_longitude     199961 non-null float64
pickup_latitude      199961 non-null float64
dropoff_longitude    199961 non-null float64
dropoff_latitude     199961 non-null float64
passenger_count      199961 non-null int64
day                  199961 non-null int64
hour                 199961 non-null int64
month                199961 non-null int64
year                 199961 non-null int64
weekday              199961 non-null int64
dtypes: float64(5), int64(6)
memory usage: 18.3 MB
```

Model Building

```
from sklearn.model_selection import train_test_split

x=Train_Data.drop("fare_amount", axis=1)
x
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	month	year	weekday
0	-73.999817	40.738354	-73.999512	40.723217	1	7	19	5	2015	4
1	-73.994355	40.728225	-73.994710	40.750325	1	17	20	7	2009	5
2	-74.005043	40.740770	-73.962565	40.772647	1	24	21	8	2009	1
3	-73.976124	40.790844	-73.965316	40.803349	3	26	8	6	2009	5
4	-73.925023	40.744085	-73.973082	40.761247	5	28	17	8	2014	4
...	...	...	...	...	...	...	...	...	...	...
199995	-73.987042	40.739367	-73.986525	40.740297	1	28	10	10	2012	0
199996	-73.984722	40.736837	-74.006672	40.739620	1	14	1	3	2014	5
199997	-73.986017	40.756487	-73.858957	40.692588	2	29	0	6	2009	1
199998	-73.997124	40.725452	-73.983215	40.695415	1	20	14	5	2015	3
199999	-73.984395	40.720077	-73.985508	40.768793	1	15	4	5	2010	6

199961 rows x 10 columns

```
y=Train_Data["fare_amount"]

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=101)

x_train.head()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	month	year	weekday
36449	-73.963597	40.761797	-73.970199	40.762533	1	4	22	9	2014	4
177679	-74.013143	40.705700	-73.867012	40.768862	4	2	17	1	2013	3
36877	-73.993683	40.702455	-73.917713	40.684747	2	31	3	10	2010	0
20428	-73.954686	40.780613	-73.971005	40.758253	1	29	10	8	2012	3
18927	-73.978887	40.777162	-73.993860	40.746392	1	14	16	7	2013	0

```
x_test.head()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	month	year	weekday
35707	-73.979422	40.743943	-73.969162	40.758608	1	26	16	6	2012	2
37333	-74.000873	40.747298	-73.991410	40.764548	2	14	17	4	2011	4
131999	-74.007640	40.732222	-73.988398	40.748832	5	29	8	11	2010	1
106818	-73.960133	40.719825	-73.942702	40.717567	4	6	21	9	2013	5
52881	-73.995711	40.764551	-73.991177	40.750312	2	1	8	5	2009	5

```
pickup_longitude     199961 non-null float64
pickup_latitude      199961 non-null float64
dropoff_longitude    199961 non-null float64
dropoff_latitude     199961 non-null float64
passenger_count      199961 non-null int64
day                  199961 non-null int64
hour                 199961 non-null int64
month                199961 non-null int64
year                 199961 non-null int64
weekday              199961 non-null int64
dtypes: float64(5), int64(6)
memory usage: 18.3 MB
```

Model Building

```
from sklearn.model_selection import train_test_split

x=Train_Data.drop("fare_amount", axis=1)
x
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	month	year	weekday
0	-73.999817	40.738354	-73.999512	40.723217	1	7	19	5	2015	4
1	-73.994355	40.728225	-73.994710	40.750325	1	17	20	7	2009	5
2	-74.005043	40.740770	-73.962565	40.772647	1	24	21	8	2009	1
3	-73.976124	40.790844	-73.965316	40.803349	3	26	8	6	2009	5
4	-73.925023	40.744085	-73.973082	40.761247	5	28	17	8	2014	4
...	...	...	...	...	...	...	...	...	...	...
199995	-73.987042	40.739367	-73.986525	40.740297	1	28	10	10	2012	0
199996	-73.984722	40.736837	-74.006672	40.739620	1	14	1	3	2014	5
199997	-73.986017	40.756487	-73.858957	40.692588	2	29	0	6	2009	1
199998	-73.997124	40.725452	-73.983215	40.695415	1	20	14	5	2015	3
199999	-73.984395	40.720077	-73.985508	40.768793	1	15	4	5	2010	6

199961 rows x 10 columns

```
y=Train_Data["fare_amount"]

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=101)

x_train.head()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	month	year	weekday
36449	-73.963597	40.761797	-73.970199	40.762533	1	4	22	9	2014	4
177679	-74.013143	40.705700	-73.867012	40.768862	4	2	17	1	2013	3
36877	-73.993683	40.702455	-73.917713	40.684747	2	31	3	10	2010	0
20428	-73.954686	40.780613	-73.971005	40.758253	1	29	10	8	2012	3
18927	-73.978887	40.777162	-73.993860	40.746392	1	14	16	7	2013	0

```
x_test.head()
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	month	year	weekday
35707	-73.979422	40.743943	-73.969162	40.758608	1	26	16	6	2012	2
37333	-74.000873	40.747298	-73.991410	40.764548	2	14	17	4	2011	4
131999	-74.007640	40.732222	-73.988398	40.748832	5	29	8	11	2010	1
106818	-73.960133	40.719825	-73.942702	40.717567	4	6	21	9	2013	5
52881	-73.995711	40.764551	-73.991177	40.750312	2	1	8	5	2009	5

```
y_train.head()
```

```
36449      4.5
177679    47.3
36877     20.9
20428      8.9
18927     11.0
Name: fare_amount, dtype: float64
```

```
y_test.head()
```

```
35707     7.3
37333     6.5
131999    8.1
106818    8.5
52881     9.7
Name: fare_amount, dtype: float64
```

```
print(x_train.shape)
print(x_test.shape)
print(y_test.shape)
print(y_train.shape)
```

```
(159968, 10)
(39993, 10)
(39993,)
(159968,)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression

lrmodel=LinearRegression()
lrmodel.fit(x_train, y_train)
```

```
LinearRegression()
```

```
predictedvalues = lrmodel.predict(x_test)

from sklearn.metrics import mean_squared_error

lrmodelrmse = np.sqrt(mean_squared_error(predictedvalues, y_test))
print("RMSE value for Linear regression is", lrmodelrmse)
```

```
RMSE value for Linear regression is 10.05963609768883
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
rfrmodel = RandomForestRegressor(n_estimators=100, random_state=101)

rfrmodel.fit(x_train,y_train)
rfrmodel_pred= rfrmodel.predict(x_test)

rfrmodel_rmse=np.sqrt(mean_squared_error(rfrmodel_pred, y_test))
print("RMSE value for Random forest regression is ",rfrmodel_rmse)
```

```
RMSE value for Random forest regression is  4.838358623297524
```

```
rfrmodel_pred.shape
```

```
(39993,)
```

Working on Test Data

```
test = pd.read_csv(r'testt.csv')
```

```
test.drop(test[['Unnamed: 0','Unnamed: 0.1','Unnamed: 0.1.1','key']],axis=1,inplace=True)
```

```
test.isnull().sum()
```

```
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

```
test["pickup_datetime"] = pd.to_datetime(test["pickup_datetime"])
```

```
test['day']=test['pickup_datetime'].apply(lambda x:x.day)
test['hour']=test['pickup_datetime'].apply(lambda x:x.hour)
test['month']=test['pickup_datetime'].apply(lambda x:x.month)
test['year']=test['pickup_datetime'].apply(lambda x:x.year)
test['weekday']=test['pickup_datetime'].apply(lambda x: calendar.day_name[x.weekday()])
```

```
test.head(5)
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	month	year	weekday
0	-73.951662	40.790710	-73.947570	40.756220	1	10	19	2	2011	4
1	-73.951007	40.771508	-73.974075	40.763553	1	23	9	6	2011	4
2	-73.996473	40.747930	-73.990298	40.756152	6	14	10	7	2012	6
3	-73.997934	40.716890	-73.952617	40.727149	1	19	22	10	2014	0
4	-73.952583	40.714039	-73.906128	40.711281	1	25	22	5	2015	1

```
test.drop(['pickup_datetime'], axis = 1, inplace = True)
```

```
test.weekday = test.weekday.map({'Sunday':0,'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,'Saturday':6})
```

```
rfrmodel_pred= rfrmodel.predict(test)
```

```
df = pd.DataFrame(rfrmodel_pred)
df
```

	0
0	11.2380
1	9.1880
2	5.9320
3	11.8570
4	11.9080
...	...
49995	6.7250
49996	27.4532
49997	7.4750
49998	7.7190
49999	10.8940

50000 rows × 1 columns

JSPM
Page No. :

Assignment No. 2

✳ Title : Classify the email using binary classification method.

Email Spam detection has two states:

a) Normal State - Not Spam,
b) Abnormal State - Spam

Use k-Nearest Neighbors and support Vector Machine for classification Analyze their performance.

✳ Dataset Description:

The csv file contains 5172 rows, each row for each email. There are 3002 columns. The first column indicates Email name. The name has been set with numbers and not receipts' name to protect privacy. The last column has been set with numbers. The last column has the labels for prediction: 1 for spam, 0 for not spam. The remaining 3000 columns are the 3000 most common words in all the emails, after excluding the non-alphabetical characters/ words.

For each row, the count of each word (column) in that emails (row) is stored in the respective cells. Thus, information regarding all 5172 emails are stored in a compact dataframe rather than asseparate text files.

Link: https://www.kaggle.com/datasets/balakal18/email-spam-classification-dataset-csv

✳ Objective :

Students should be able to classify email using the binary classification and implement email spam detection technique by using K-Nearest Neighbors and support Vector Machine algorithm.

✳ Prerequisite :

1. Basic knowledge of Python
2. Concept of K-Nearest Neighbors and Support Vector Machine for classification.

✳ Contents of the Theory :

1. Data Preprocessing
2. Binary classification

3. K-Nearest Neighbours
4. Support Vector Machine
5. Train, Test and Split Procedure

Data Preprocessing :

Data preprocessing is a process of preparing the raw data and making it suitable for amachine learning model. It is the 1st and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data processing task.

Why do we need Data Preprocessing?

A real-world data ~~eng~~ generally contains noise, missing values, and may be in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making

it suitable for a machine learning model which also increases the accuracy and eficiency of a machine learning model.

It involves below steps:

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set.
- Feature scalin

→» K-Nearest Neighbors (KNN)

- KNN Algorithm: A non-parametric, instance-based learning algorithm. It classifies a data point based on the majority class of its k-nearest neighbors in the feature space.

→» Advantages:

- Simple to understand and implement.
- No training phase, making it faster for small datasets.

→» Disadvantages:

- Slow for large datasets, as predictions require computation over the entire training set.
- Sensitive to irrelevant features and the choice of k.

✳ Support Vector Machine (SVM)

→» SVM Algorithm:
A supervised learning model that finds the hyperplane which best separates the data points of different classes (spam and not spam) in a higher-dimensional space.

→» Advantages: Effective in high-dimensional spaces and when the number of

features exceeds the number of samples.

- Works well with clear margin separation.

→ Disadvantages:

- Less effective with noisy data or overlapping classes.

- Requires careful parameter tuning (e.g. choosing the right kernel & regularization parameters).

✳ Performance Evaluation:

→ Evaluation Metrics:

Accuracy, precision, recall, and F1-score will be used to evaluate the performance of KNN and SVM.

✳ Confusion Matrix:

Provides insights into True Positives, False Positives, True Negatives and False Negatives.

JSPM
Page No. :

Conclusion :

In this way we classify the email using binary classification method.

```
import pandas as pd

df = pd.read_csv('Data/ham-spam.csv')
df.head()
```

	IsSpam	Text
0	0	key issues going forwarda year end reviews rep...
1	0	congrats contratulations the execution the cen...
2	0	key issues going forwardall under control set...
3	0	epmi files protest entergy transcoattached our...
4	0	california power please contact kristin walsh ...

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   IsSpam  1000 non-null   int64
 1   Text    1000 non-null   object
dtypes: int64(1), object(1)
memory usage: 15.8+ KB
```

Check for duplicate rows in the dataset.

```
df.groupby('IsSpam').describe()
```

	Text			
	count	unique	top	freq
IsSpam				
0	500	499	paso firm capacity award memorandumlouise del...	2
1	500	500	you get your order immediately via gra levitr ...	1

```
df = df.drop_duplicates()
df.groupby('IsSpam').describe()
```

	Text			
	count	unique	top	freq
IsSpam				
0	499	499	brazil commercial update version delete previ...	1
1	500	500	you get your order immediately via gra levitr ...	1

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(ngram_range=(1, 2), stop_words='english')
x = vectorizer.fit_transform(df['Text'])
y = df['IsSpam']
```

Show the vocabulary that `CountVectorizer` built from the training e-mails.

```
vectorizer.vocabulary_
```

```
{'key': 50036,
 'issues': 48310,
 'going': 40119,
 'forwarda': 37692,
 'year': 99702,
 'end': 31413,
 'reviews': 77698,
 'report': 76304,
```

```
 'needs': 60510,
 'generating': 39353,
 'like': 52433,
 'mid': 57723,
 'documenting': 27881,
 'business': 12428,
 'unit': 93760,
 'performance': 66771,
 'review': 77618,
 'completion': 18806,
 'david': 23666,
 'john': 49105,
 'work': 98732,
 'plan': 67886,
 'generation': 39362,
 'nim': 61313,
 'employees': 31202,
 'hpl': 44127,
 'transition': 92241,
 'ongoing': 63430,
 'officially': 63018,
 'transferred': 92194,
 'regardsdelainey': 75386,
 'key issues': 50047,
 'issues going': 48337,
 'going forwarda': 40142,
 'forwarda year': 37693,
 'year end': 99733,
 'end reviews': 31450,
 'reviews report': 77707,
 'report needs': 76342,
 'needs generating': 60519,
 'generating like': 39356,
 'like mid': 52526,
 'mid year': 57738,
 'year documenting': 99731,
 'documenting business': 27882,
 'business unit': 12542,
 'unit performance': 93775,
 'performance review': 66798,
 'review completion': 77630,
 'completion david': 18810,
 'david john': 23680,
 'john work': 49173,
 'work plan': 98799,
 'plan generation': 67922,
 'generation nim': 39379,
 'nim issues': 61314,
 'issues employees': 48325,
```

```
text = vectorizer.transform(['Why pay MORE for * expensive meds when you can ...123... order them online and save $$$?'])
text = vectorizer.inverse_transform(text)
print(text)
```

```
[array(['expensive', 'meds', 'online', 'order', 'order online', 'pay',
       'save'], dtype='<U401')]
```

Split the dataset so that 80% can be used for training and 20% for testing.

```
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)
```

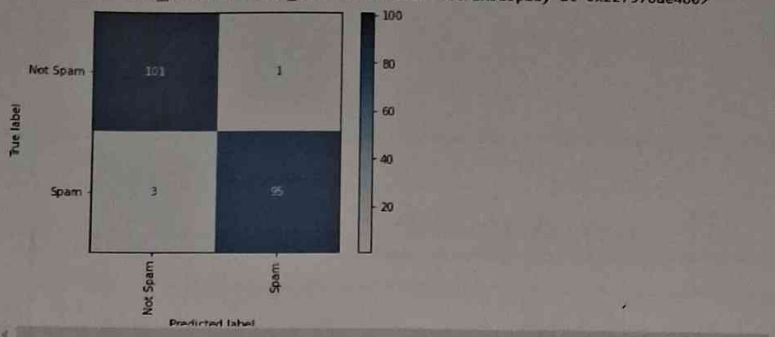
```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(x_train, y_train)
```

```
MultinomialNB()
```

Validate the trained model with the 20% of the dataset aside for testing and show a confusion matrix.

```
%matplotlib inline
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(model, x_test, y_test, display_labels=['Not Spam', 'Spam'], cmap='Blues', xticks_rotation='vertical')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x227576de460>
```




```
model.score(x_test, y_test)
```

```
0.98
```

```
from sklearn.metrics import roc_auc_score

probabilities = model.predict_proba(x_test)
roc_auc_score(y_test, probabilities[:, 1])
```

```
0.9992997198879552
```

Use the model to classify e-mails

```
message = vectorizer.transform(['Can you attend a code review on Tuesday? Need to make sure the logic is rock solid.'])
model.predict(message)[0]
```

```
0
```

```
model.predict_proba(message)[0][0]
```

```
0.9999170457201042
```

```
message = vectorizer.transform(['Why pay more for expensive meds when you can order them online and save $$$?'])
model.predict(message)[0]
```

```
1
```

```
model.predict_proba(message)[0][0]
```

```
0.00021423891260677753
```

```
model.predict_proba(message)[0][1]
```

```
0.9997857610873945
```

Assignment No. 3

✳ Title : Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

✳ Dataset Description : The case study is for an open-source dataset from kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, Credit Score, Geography, Gender, Age, Tenure, Balance, etc.

Link : https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling.

✳ Perform the following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.

5. Print the accuracy score and confusion matrix.

✳ Objective :

Students should be able to distinguish the feature and target set and divide the data set into training and test sets and normalize them and students should build the model on the basis of that.

✳ Prerequisite:

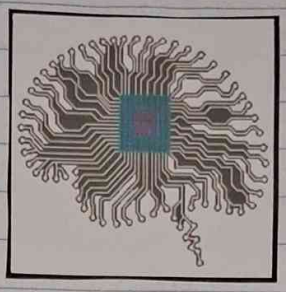
1. Basic knowledge of Python.
2. Concept of confusion Matrix.

✳ Contents of the Theory:

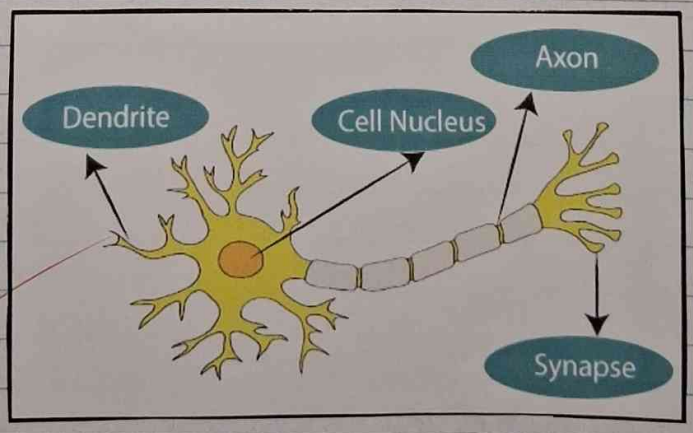
1. Artificial Neural Network
2. Keras
3. tensorflow
4. Normalization
5. Confusion Matrix.

JSPM
Page No. :

Artificial Neural Network:

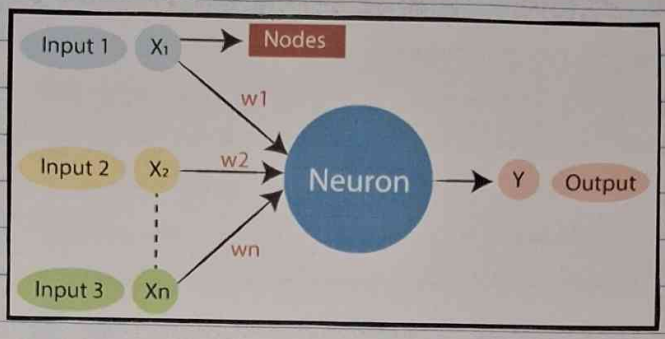


The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of thenet works. These neurons are known as nodes.



The given figure illustrates the typical diagram of Biological Netural Network.

The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

An artificial Neural Network in the field of Artificial intelligence where it attempts to mimic the network of neurons makes up a human brain so that computer will have an option to understand things and make decisions in a human-like manner. The artificial neural networks designed by programmeing computers to behave simply like interconnected brain cells.

* Artificial Neural Network primarily consists of three layers:

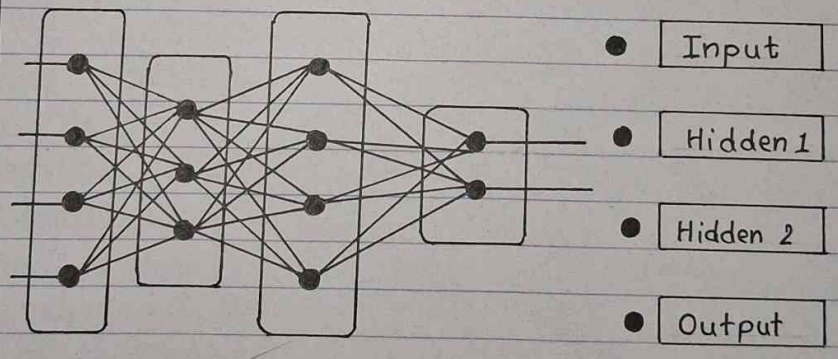


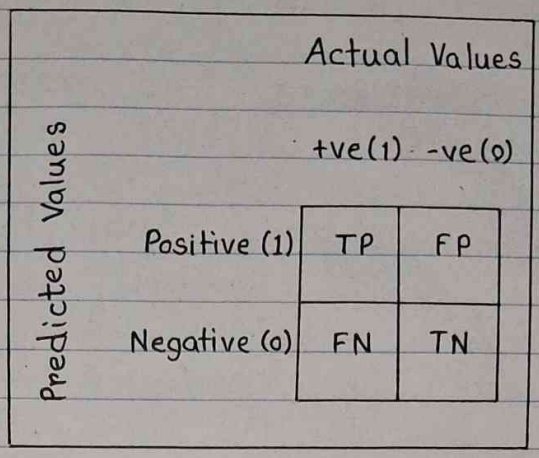
Fig. Artificial Neural Network

The hidden layer presents in-between input & output layer.

Confusion Matrix :-

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can be determined if the true values for test data are known. The matrix itself can be easily understood, but the matrix, hence also known as an error matrix. Some features of confusion matrix are given below:

- For the 2 prediction classes of classification, the matrix of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are predicted values & actual values along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true value for the given observations.
- It looks like the below table:



The above tables has the following cases:

- True Negative : Model has given prediction No, and real or actual values was also No.
- True Positive : The model value has predicted yes, and the actual value was also true.
- False Negative : The model has predicted no, but the actual value was Yes, it is called as Type-II error.
- False Positive : The model has predicted Yes, but the actual value was. No, it is also called a Type-I error.

JSPM
Page No.:

Need for confusion Matrix in ML:

- It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.
- It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.
- With the help of the confusion matrix, we can calculate the different parameters forthe model, such as accuracy, precision, etc.

✳ Conclusion:

In this way we build a neural network based classificater that can determine whether they willeave or not in the next 6 months.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import io


from google.colab import files
uploaded=files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving bank.csv to bank.csv
```

```
df=pd.read_csv(io.StringIO(uploaded['bank.csv'].decode('utf-8')))
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Esti
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	

```
df=df.drop(['RowNumber','CustomerId','Surname'],axis=1)
df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
df.isna().any()
df.isna().sum()
```

```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
print(df.shape)
df.info()
```

```
(10000, 11)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CreditScore      10000 non-null  int64
 1   Geography        10000 non-null  object
 2   Gender           10000 non-null  object
 3   Age              10000 non-null  int64
 4   Tenure           10000 non-null  int64
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64
 7   HasCrCard        10000 non-null  int64
 8   IsActiveMember   10000 non-null  int64
```

```
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

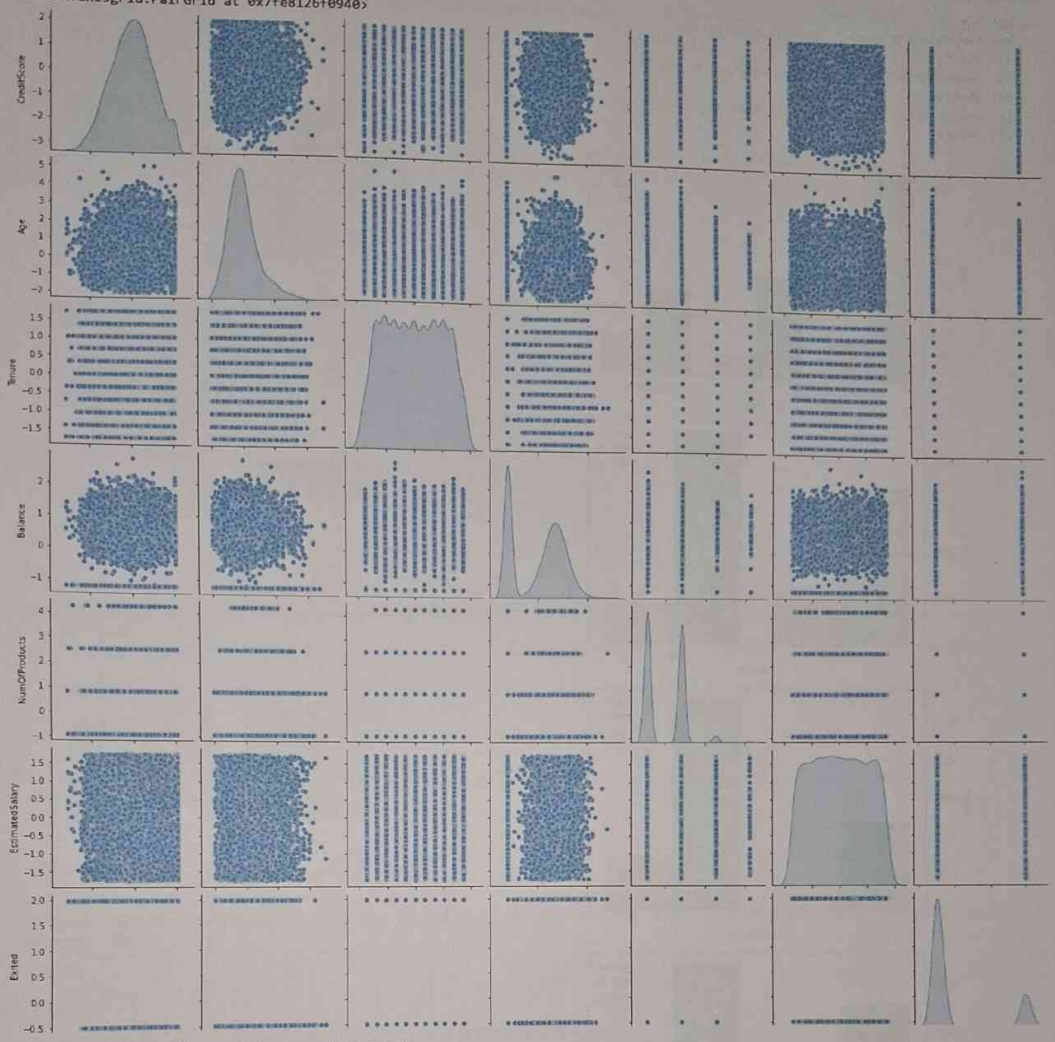
```
df.describe()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

Before performing Bivariate analysis, Lets bring all the features to the same range

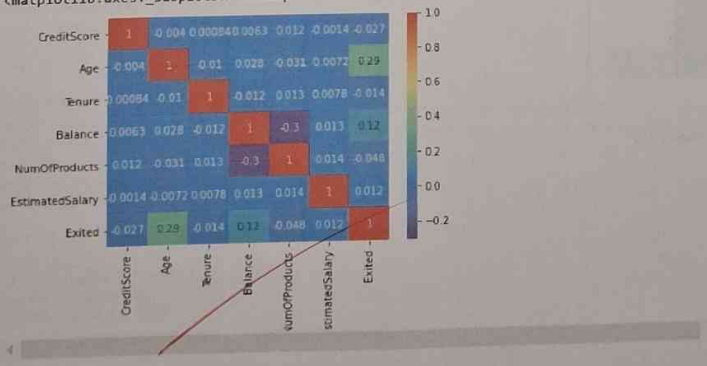
```
scaler=StandardScaler()
## Extract only the Numerical Columns to perform Bivariate Analysis
subset=df.drop(['Geography','Gender','HasCrCard','IsActiveMember'],axis=1)
scaled=scaler.fit_transform(subset)
scaled_df=pd.DataFrame(scaled,columns=subset.columns)
sns.pairplot(scaled_df,diag_kind='kde')
```

```
<seaborn.axisgrid.PairGrid at 0x7fe8126f0940>
```



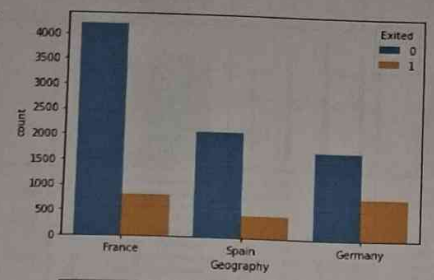

```
sns.heatmap(scaled_df.corr(),annot=True,cmap='rainbow')
```

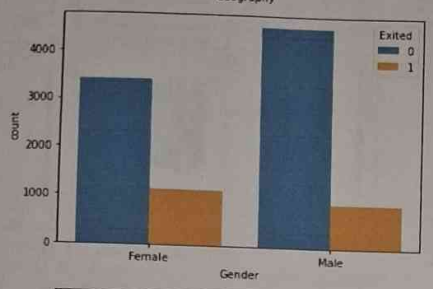
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe7cb4ef9b0>
```

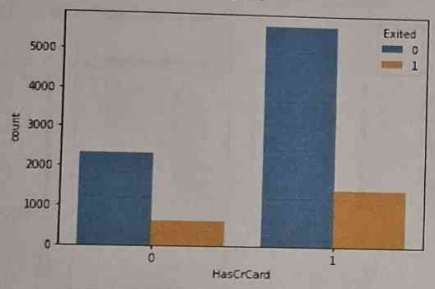


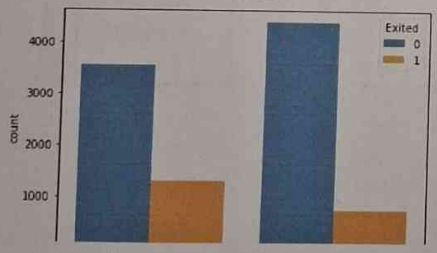

```
## Categorical Features vs Target Variable
sns.countplot(x='Geography',data=df,hue='Exited')
```

```
plt.show()
sns.countplot(x='Gender',data=df,hue='Exited')
plt.show()
sns.countplot(x='HasCrCard',data=df,hue='Exited')
plt.show()
sns.countplot(x='IsActiveMember',data=df,hue='Exited')
plt.show()
```



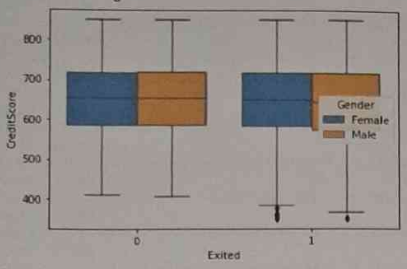




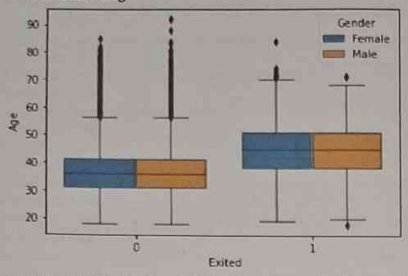



```
subset=subset.drop('Exited',axis=1)
for i in subset.columns:
  sns.boxplot(df['Exited'],df[i],hue=df['Gender'])
  plt.show()
```

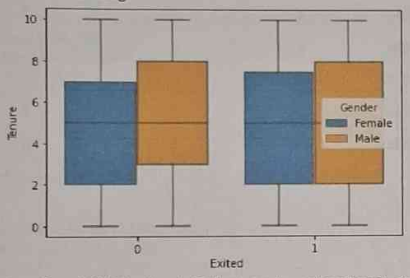
```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y.
  FutureWarning
```



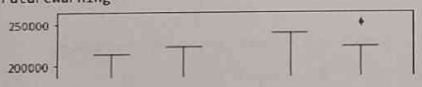

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y.
  FutureWarning
```




```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y.
  FutureWarning
```




```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y.
  FutureWarning
```



Distinguish the Target and Feature Set and divide the dataset into Training and Test sets

```
X=df.drop('Exited',axis=1)
y=df.pop('Exited')
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.10,random_state=5)
X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,test_size=0.10,random_state=5)
print("X_train size is {}".format(X_train.shape[0]))
print("X_val size is {}".format(X_val.shape[0]))
print("X_test size is {}".format(X_test.shape[0]))
```

```
X_train size is 8100
X_val size is 900
X_test size is 1000
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

```
num_cols=['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary']
num_subset=scaler.fit_transform(X_train[num_cols])
X_train_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_train_num_df['Geography']=list(X_train['Geography'])
X_train_num_df['Gender']=list(X_train['Gender'])
X_train_num_df['HasCrCard']=list(X_train['HasCrCard'])
X_train_num_df['IsActiveMember']=list(X_train['IsActiveMember'])
X_train_num_df.head()
## Standardise the Validation data
num_subset=scaler.fit_transform(X_val[num_cols])
X_val_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_val_num_df['Geography']=list(X_val['Geography'])
X_val_num_df['Gender']=list(X_val['Gender'])
X_val_num_df['HasCrCard']=list(X_val['HasCrCard'])
X_val_num_df['IsActiveMember']=list(X_val['IsActiveMember'])
## Standardise the Test data
num_subset=scaler.fit_transform(X_test[num_cols])
X_test_num_df=pd.DataFrame(num_subset,columns=num_cols)
X_test_num_df['Geography']=list(X_test['Geography'])
X_test_num_df['Gender']=list(X_test['Gender'])
X_test_num_df['HasCrCard']=list(X_test['HasCrCard'])
X_test_num_df['IsActiveMember']=list(X_test['IsActiveMember'])
```

```
## Convert the categorical features to numerical
X_train_num_df=pd.get_dummies(X_train_num_df,columns=['Geography','Gender'])
X_test_num_df=pd.get_dummies(X_test_num_df,columns=['Geography','Gender'])
X_val_num_df=pd.get_dummies(X_val_num_df,columns=['Geography','Gender'])
X_train_num_df.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	HasCrCard	IsActiveMember	Geography_France	Geography_Ger
0	-1.178587	-1.041960	-1.732257	0.198686	0.820905	1.560315	1	1	1	
1	-0.380169	-1.326982	1.730718	-0.022020	-0.907991	-0.713592	1	0	0	
2	-0.349062	1.808258	-0.693364	0.681178	0.820905	-1.126515	1	0	0	
3	0.625629	2.378302	-0.347067	-1.229191	0.820905	-1.682740	1	1	1	
4	-0.203895	-1.136967	1.730718	0.924256	-0.907991	1.332535	1	1	0	

Initialise and build the Model

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model=Sequential()
model.add(Dense(7,activation='relu'))
model.add(Dense(10,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

import tensorflow as tf
optimizer=tf.keras.optimizers.Adam(0.01)
model.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'])

model.fit(X_train_num_df,y_train,epochs=100,batch_size=10,verbose=1)
```

```
Epoch 1/100
810/810 [==============================] - 1s 1ms/step - loss: 0.4511 - accuracy: 0.8054
Epoch 2/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3623 - accuracy: 0.8493
Epoch 3/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3543 - accuracy: 0.8541
Epoch 4/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3433 - accuracy: 0.8561
Epoch 5/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3291 - accuracy: 0.8692
Epoch 6/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3488 - accuracy: 0.8560
Epoch 7/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3439 - accuracy: 0.8644
Epoch 8/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3399 - accuracy: 0.8615
```

```
Epoch 9/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3480 - accuracy: 0.8602
Epoch 10/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3321 - accuracy: 0.8683
Epoch 11/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3329 - accuracy: 0.8614
Epoch 12/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3466 - accuracy: 0.8575
Epoch 13/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3386 - accuracy: 0.8640
Epoch 14/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3342 - accuracy: 0.8667
Epoch 15/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3375 - accuracy: 0.8652
Epoch 16/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3412 - accuracy: 0.8574
Epoch 17/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3256 - accuracy: 0.8652
Epoch 18/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3229 - accuracy: 0.8687
Epoch 19/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3276 - accuracy: 0.8670
Epoch 20/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3399 - accuracy: 0.8581
Epoch 21/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3384 - accuracy: 0.8661
Epoch 22/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3309 - accuracy: 0.8638
Epoch 23/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3441 - accuracy: 0.8592
Epoch 24/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3232 - accuracy: 0.8697
Epoch 25/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3409 - accuracy: 0.8651
Epoch 26/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3361 - accuracy: 0.8622
Epoch 27/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3380 - accuracy: 0.8596
Epoch 28/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3363 - accuracy: 0.8611
Epoch 29/100
810/810 [==============================] - 1s 1ms/step - loss: 0.3347 - accuracy: 0.8636
```

```
y_pred_val=model.predict(X_val_num_df)
y_pred_val[y_pred_val>0.5]=1
y_pred_val[y_pred_val <0.5]=0
```

```
y_pred_val=y_pred_val.tolist()
X_compare_val=X_val.copy()
X_compare_val['y_actual']=y_val
X_compare_val['y_pred']=y_pred_val
X_compare_val.head(10)
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	y_actual	y_pred
340	642	Germany	Female	40	6	129502.49	2	0	1	86099.23	1	[0.0]
8622	706	Germany	Male	36	9	58571.18	2	1	0	40774.01	0	[0.0]
8401	535	Spain	Male	58	1	0.00	2	1	1	11779.98	1	[0.0]
4338	714	Spain	Male	25	2	0.00	1	1	1	132979.43	0	[0.0]
8915	606	France	Male	36	1	155655.46	1	1	1	192387.51	1	[0.0]
2624	605	Spain	Female	29	3	116805.82	1	0	0	4092.75	0	[0.0]
2234	720	France	Female	38	10	0.00	2	1	1	56229.72	1	[0.0]
349	582	France	Male	39	5	0.00	2	1	1	129892.93	0	[0.0]
3719	850	France	Female	62	1	124678.35	1	1	0	70916.00	1	[1.0]
2171	526	Germany	Male	58	9	190298.89	2	1	1	191263.76	0	[0.0]

```
from sklearn.metrics import confusion_matrix
cm_val=confusion_matrix(y_val,y_pred_val)
cm_val
```

```
array([[694,  22],
       [ 96,  88]])
```

```
Accuracy=782/900
print("Accuracy of the Model on the Validation Data set is 86.89%")
```

```
Accuracy of the Model on the Validation Data set is 86.89%
```

```
loss1,accuracy1=model.evaluate(X_train_num_df,y_train,verbose=False)
loss2,accuracy2=model.evaluate(X_val_num_df,y_val,verbose=False)
print("Train Loss {}".format(loss1))
print("Train Accuracy {}".format(accuracy1))
print("Val Loss {}".format(loss2))
```

JSPM
Page No. :

Assignment No. 04

* Title: Implement K-Nearest Neighbors algorithm on diabetes.csv dataset.
Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

* Dataset Description:

We will try to build a machine learning model to accurately predict whether or not the patients in the dataset have or not?
The dataset consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregancies the patient has had, their BMI, insulin level, age, and so on.

Link: Diabetes prediction system with KNN algorithm | Kaggle.

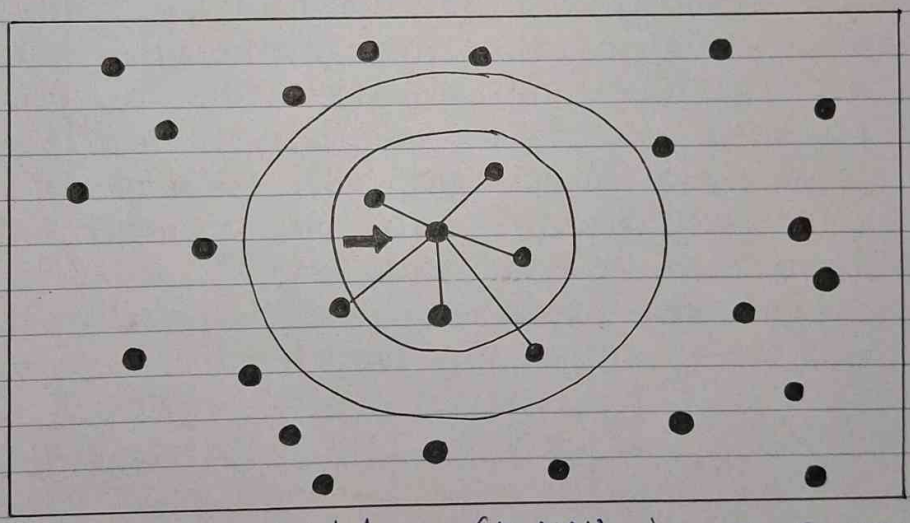
* Objective :

Students should be able to preprocess dataset and identify outliers, to check correlation and implement KNN algorithm

and random forest classification models. Evaluate them with respective scores like confusion_matrix, accuracy_score, mean_squared_error, r2_score, roc_auc_score, roc_curve etc.

✳ Prequisite:

1. Basic knowledge of Python
2. Concept of Confusion Matrix
3. Concept of roc_auc curve.
4. Concept of Random Forest and KNN algorithms.



K-Nearest - Neighbors (K-NN) is a supervised machine learning model. Supervised learning is when a model learns from data that is already labeled.

For our K-NN model, the first step is to read in the data we will use as input. For this example, we are using the diabetes dataset. To start, we will use Pandas to read in data.

I will not go into detail on Pandas, but it is a library you should become familiar with if you're looking to dive further into data science and ML.

K-Fold Cross-Validation:

Cross-validation is when the dataset is randomly split up into 'K' groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group as been used as the test set.

✳ Conclusion:

In this way we build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

```
import pandas as pd
import numpy as np
```

+ Code + Text

```
data = pd.read_csv("diabetes.csv")
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
data.isnull().any()
```

```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

```
data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

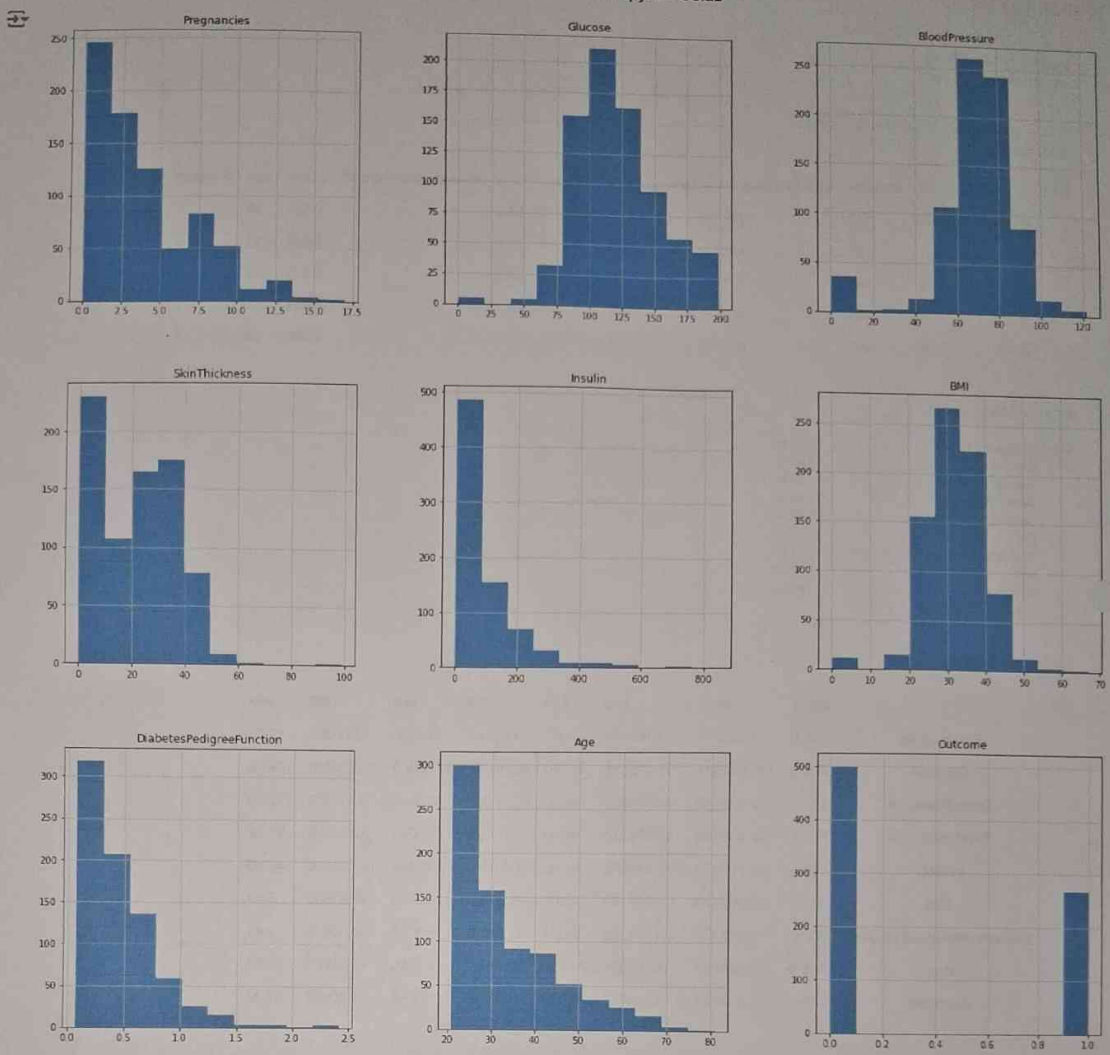
Glucose, BloodPressure, SkinThickness, Insulin, BMI
columns have values 0 which does not ake sense , hence are missing values

```
data_copy = data.copy(deep = True)
data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI
data_copy.isnull().sum()
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

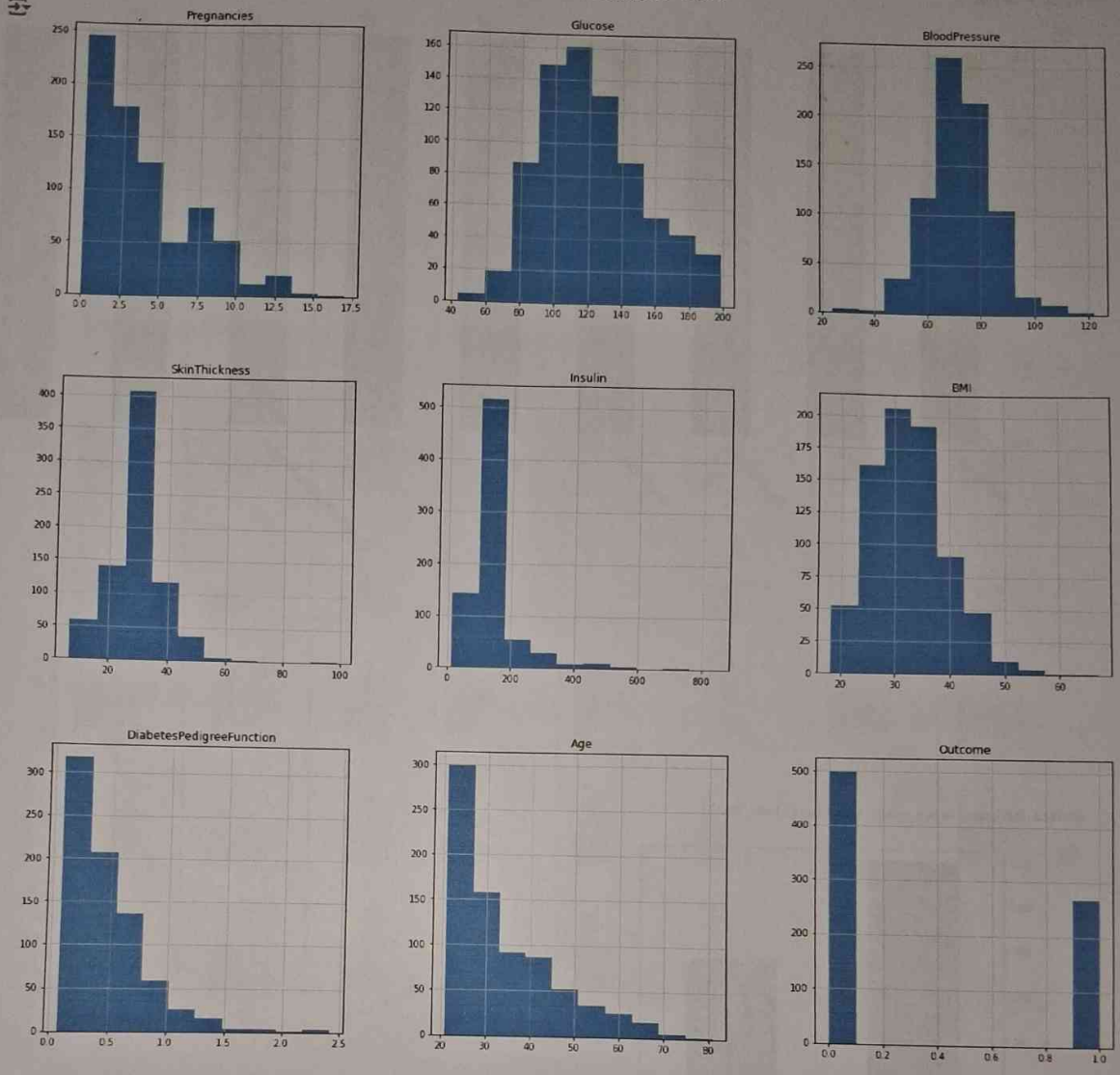
To fill these Nan values the data distribution needs to be understood

```
p = data.hist(figsize = (20,20))
```

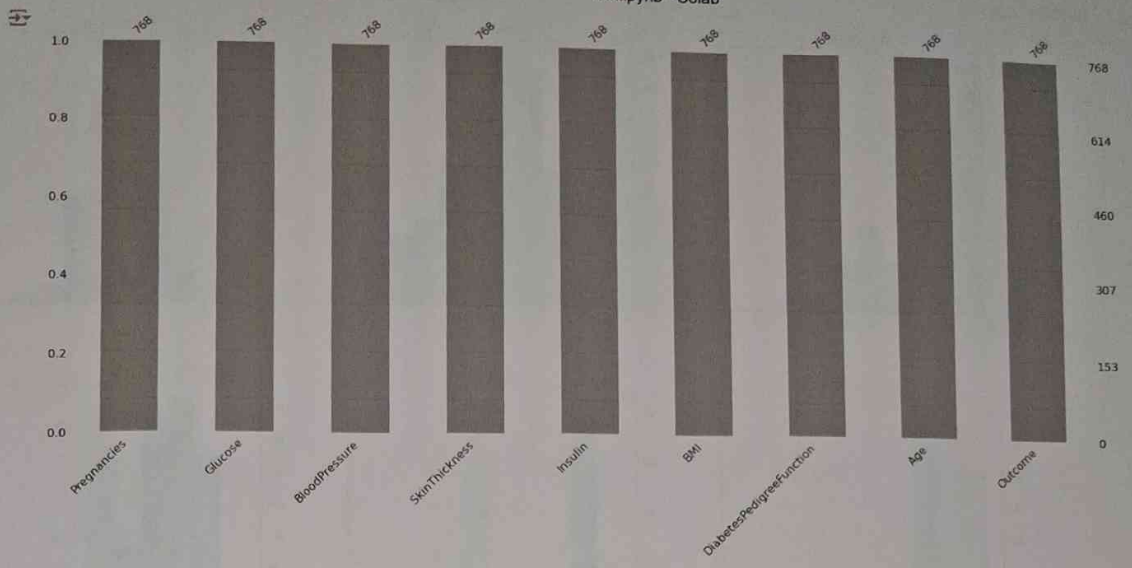



```
data_copy['Glucose'].fillna(data_copy['Glucose'].mean(), inplace = True)
data_copy['BloodPressure'].fillna(data_copy['BloodPressure'].mean(), inplace = True)
data_copy['SkinThickness'].fillna(data_copy['SkinThickness'].median(), inplace = True)
data_copy['Insulin'].fillna(data_copy['Insulin'].median(), inplace = True)
data_copy['BMI'].fillna(data_copy['BMI'].median(), inplace = True)
```

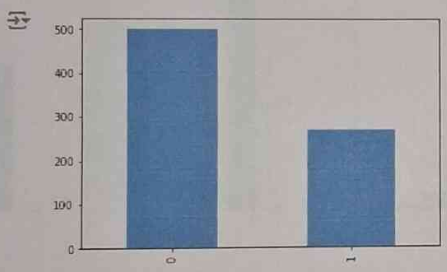
```
p = data_copy.hist(figsize = (20,20))
```




```
import missingno as msno
p = msno.bar(data)
```

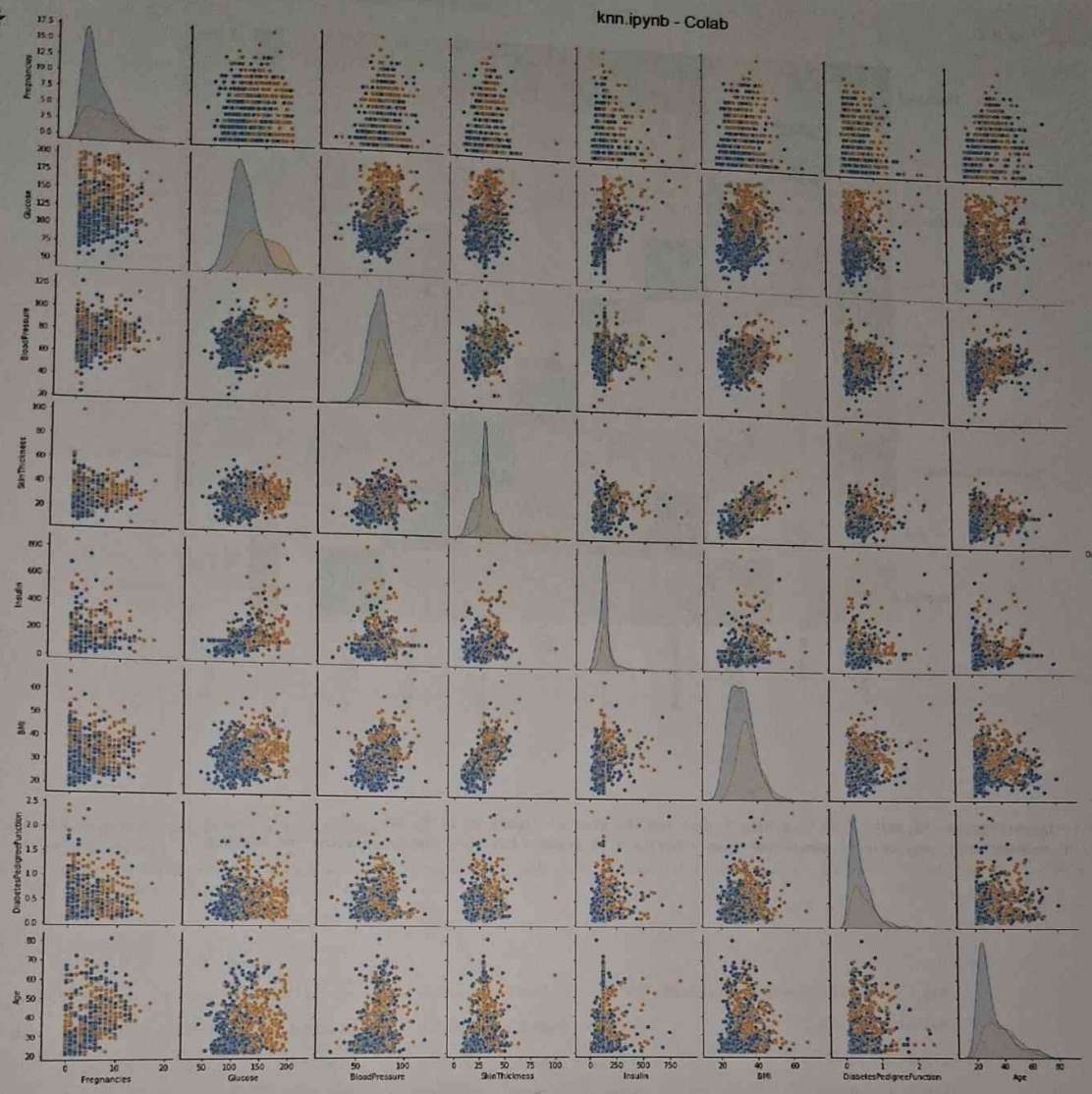



```
p=data.Outcome.value_counts().plot(kind="bar")
```



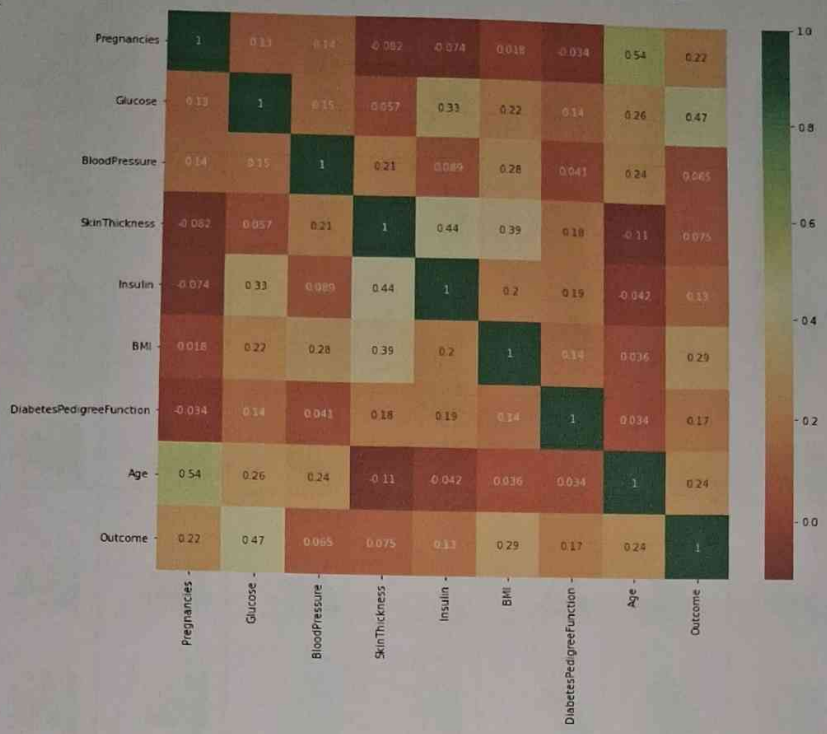
The above graph shows that the data is biased towards datapoints having outcome value as 0 where it means that diabetes was not present actually. The number of non-diabetics is almost twice the number of diabetic patients

```
import seaborn as sns
p=sns.pairplot(data_copy, hue = 'Outcome')
```

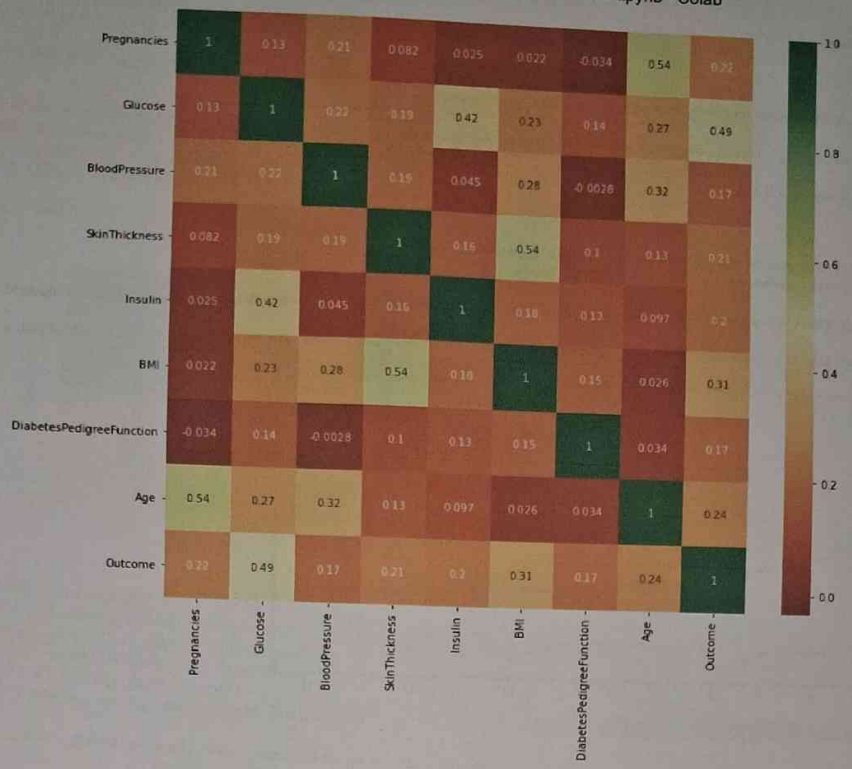


frc

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12 by 10.
p=sns.heatmap(data.corr(), annot=True,cmap ='RdYlGn')  # seaborn has very simple solution for heatmap
```




```
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12 by 10.
p=sns.heatmap(data_copy.corr(), annot=True,cmap ='RdYlGn')  # seaborn has very simple solution for heatmap
```




```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X = pd.DataFrame(sc_X.fit_transform(data_copy.drop(["Outcome"], axis =1),),columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThicknes
       'BMI', 'DiabetesPedigreeFunction', 'Age'])
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468492	1.425995
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365061	-0.190672
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	0.604397	-0.105584
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-0.920763	-1.041549
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484909	-0.020496

```
y =data_copy.Outcome
```

```
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 = 42, stratify=y)
```

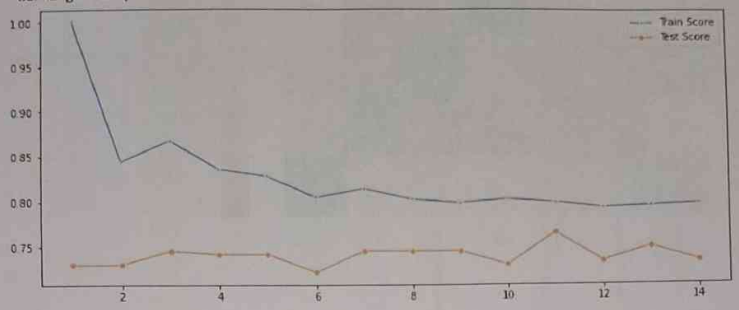
```
from sklearn.neighbors import KNeighborsClassifier

train_scores = []
test_scores = []

for i in range(1,15):
    knn = KNeighborsClassifier(i)
    knn.fit(X_train, y_train)
    train_scores.append(knn.score(X_train, y_train))
    test_scores.append(knn.score(X_test, y_test))
```

```
max_test_score =max(test_scores)


test_score_index = [i for i, v in enumerate(test_scores) if v== max_test_score]

print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda x: x+1, test_score_index))))
```

```
Max test score 76.5625 % and k = [11]
```

```
plt.figure(figsize=(12,5))
p = sns.lineplot(range(1,15),train_scores,marker='*',label='Train Score')
p = sns.lineplot(range(1,15),test_scores,marker='o',label='Test Score')
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y.
  warnings.warn(
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y.
  warnings.warn(
```

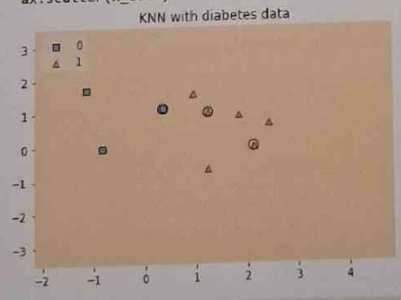



```
# K=11
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(11)

knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

```
0.765625
```

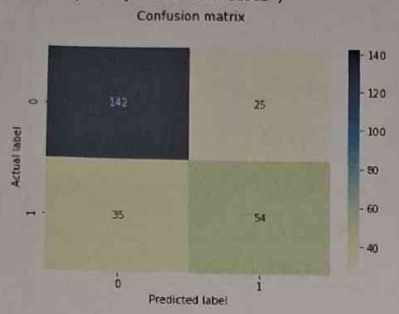
```
from mlxtend.plotting import plot_decision_regions
value = 20000
width  =20000

plot_decision_regions(X.values, y.values, clf = knn, legend =2,filler_feature_values={2: value, 3: value, 4: value, 5: value, 6: value, 7: v
                      filler_feature_ranges={2: width, 3: width, 4: width, 5: width, 6: width, 7: width},
                      X_highlight=X_test.values)
plt.title("KNN with diabetes data")
plt.show()
```

```
C:\Users\Admin\anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:244: UserWarning: No contour levels were found within th
  ax.contour(xx, yy, Z, cset.levels,
C:\Users\Admin\anaconda3\lib\site-packages\mlxtend\plotting\decision_regions.py:316: MatplotlibDeprecationWarning: Using a string of sir
  ax.scatter(x_data,
```



```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, fbeta_score
y_pred = knn.predict(X_test)

cnf_matrix = confusion_matrix(y_test, y_pred)


p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 15.0, 'Predicted label')
```




```
def model_evaluation(y_test, y_pred, model_name):
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    f2 = fbeta_score(y_test, y_pred, beta = 2.0)

    results = pd.DataFrame([[model_name, acc, prec, rec, f1, f2]],
                      columns = ["Model", "Accuracy", "Precision", "Recall",
                                 "F1 SCore", "F2 Score"])
    results = results.sort_values(["Precision", "Recall", "F2 Score"], ascending = False)
    return results


model_evaluation(y_test, y_pred, "KNN")
```

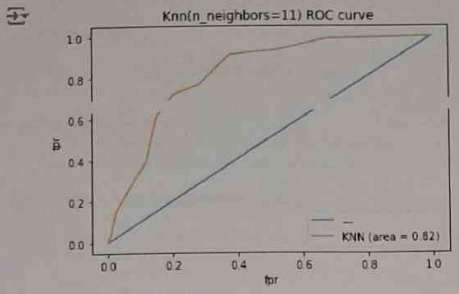
	Model	Accuracy	Precision	Recall	F1 SCore	F2 Score
0	KNN	0.765625	0.683544	0.606742	0.642857	0.62069

```
# Alternate way
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.85      0.83       167
           1       0.68      0.61      0.64        89

    accuracy                           0.77       256
   macro avg       0.74      0.73      0.73       256
weighted avg       0.76      0.77      0.76       256
```

```
from sklearn.metrics import auc, roc_auc_score, roc_curve

y_pred_proba = knn.predict_proba(X_test)[:,-1]
fpr, tpr, threshold = roc_curve(y_test, y_pred_proba)


classifier_roc_auc = roc_auc_score(y_test, y_pred_proba)
plt.plot([0,1],[0,1], label = "---")

plt.plot(fpr, tpr, label ='KNN (area = %0.2f)' % classifier_roc_auc
plt.xlabel("fpr")
```

```
plt.ylabel("tpr")
plt.title('Knn(n_neighbors=11) ROC curve')
plt.legend(loc="lower right", fontsize = "medium")
plt.xticks(rotation=0, horizontalalignment="center")
plt.yticks(rotation=0, horizontalalignment="right")
```




```
#Hyper parameters tuning using GridSearchCV


from sklearn.model_selection import GridSearchCV
parameters_grid = {"n_neighbors": np.arange(0,50)}
knn= KNeighborsClassifier()
knn_GSV = GridSearchCV(knn, param_grid=parameters_grid, cv = 5)
knn_GSV.fit(X, y)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:548: FitFailedWarning: Estimator fit failed. The sc
Traceback (most recent call last):
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_base.py", line 1157, in fit
    return self._fit(X)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\neighbors\_base.py", line 467, in _fit
    raise ValueError(
ValueError: Expected n_neighbors > 0. Got 0
```

JSPM
Page No. :

Assignment No. 5

* Title: Implement K-Means clustering / hierar-chical clustering on sales_data_sample.csv dataset. Determine the number of cluster using the elbow method.

* Dataset Description:

The data included:

1. Customer ID
2. Customer Gender
3. Customer Age
4. Annual Income of the customer (in Thousand Dollars)
5. Spending score of the customer (based on customer behavior and spending nature).

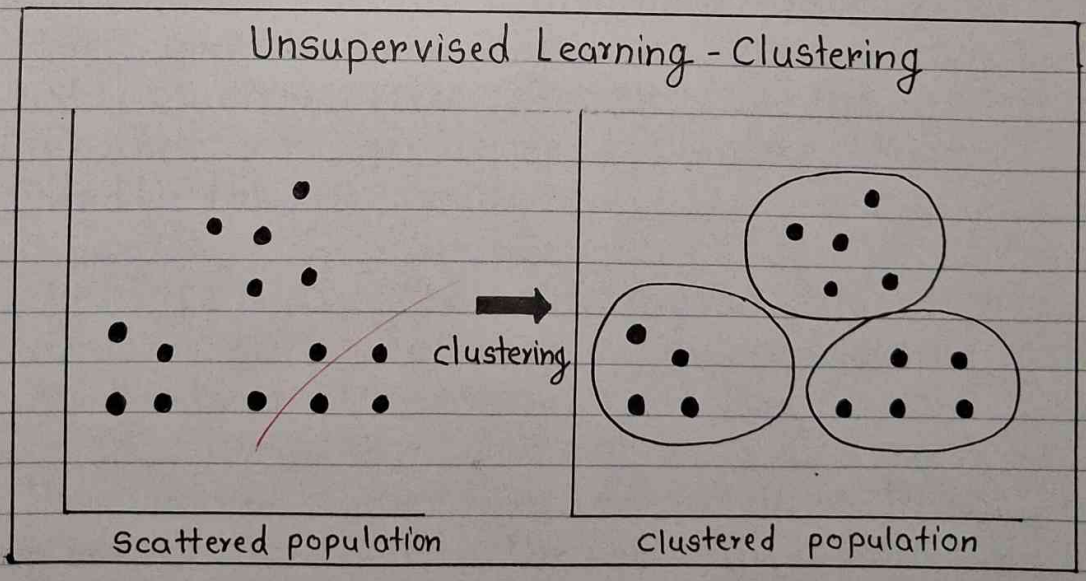
* Objective :

Students should able to understand how to use unsupervised learning to segment different-different clusters or groups and used to them to train your model to predict future things.

Prerequisite:

1. Knowledge of Python
2. Unsupervised learning
3. clustering
4. Elbow method.

- Clustering algorithms try to find natural clustering in data, the various aspects of how the algorithms to cluster data can be turned and modified. Clustering is based on the principle that items within the same cluster must be similar to each other. The data is grouped in such a way that related elements are close to each other.



Diverse and different types of data are subdivided into smaller groups.

* Uses of Clustering:

Marketing:

In the field of marketing, clustering can be used to identify various customer groups with existing customer data. Based on that, customer can be provided with discounts, offers, promo codes etc.

K-Means Clustering

K-Means clustering is an unsupervised machine learning algorithm that divides the given data into the given number of clusters. Here, the "K" is the given number of predefined clusters, that need to be created.

It is a centroid based algorithm in which each cluster is associated with a centroid.

The main idea is to reduce the distance between the data points &

their respective cluster centroid.

The algorithm takes raw unlabelled data as input and divides the dataset into clusters and the process is repeated until the best clusters are found.

K-Means is very easy and simple to implement. It is highly scalable, can be applied to both small and large datasets. There is, however, a problem with choosing the number of clusters or k. Also, with the increase in dimensions, stability decreases. But, overall k Means is a simple and robust algorithm that makes clustering very easy.

```
# Importing the necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
% matplotlib inline
```

The necessary libraries are imported.

Reading the excel file

data = pd.read_excel ("Mall_Customers.xlsx")

The data is read.

The data has 200 entries, that is data from 200 customers.

data.head()

So let us have a look at the data.

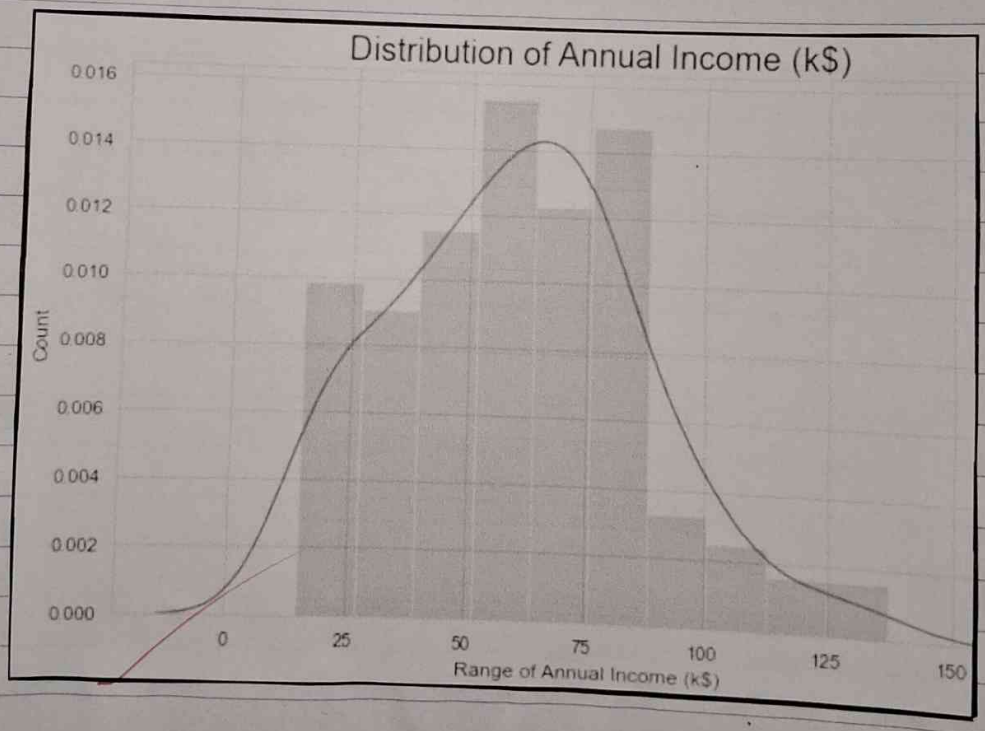
	Customer ID	Gender	Age	Annual Income (k$)	Spending Scored (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	06
3	4	Female	23	16	77
4	5	Female	31	17	40

data.corr()

	Customer ID	Age	Annual Income(k$)	Spending Score
Customer ID	1.000000	-0.02676	0.977548	0.013835
Age	-0.026763	1.000000	-0.012398	-0.327227
Annual Icome(k$)	0.977548	-0.01289	1.000000	0.009903
Spending Score(1-100)	0.013835	-0.327227	0.009903	1.000000

Annual Income Distribution :

```
# Distribution of Annual Income

plt.figure (figsize = (10, 6))
sns.set (style = 'whitegrid')
sns.distplot (data ['Annual Income (k$)'])
plt.title ('Distribution of Annual Income (k$)',
fontsize = 20)
plt.xlabel ('Range of Anual Income (k$)')
plt.ylabel ('count')
```

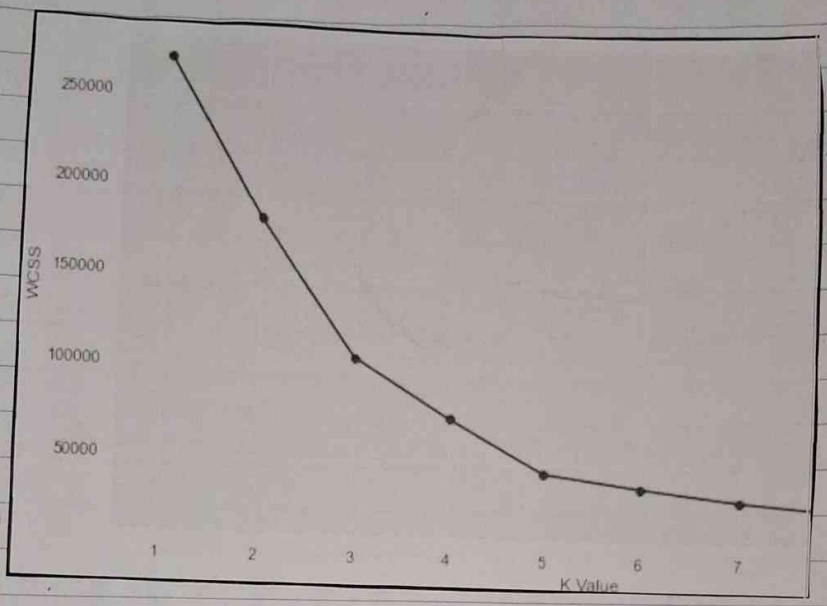


```
# Importing KMeans from sklearn

from sklearn.cluster import KMeans
```

Now we calculate the Within Cluster Sum of Squared Errors (WSS) for different values of k. Next, we choose the k for which WSS first starts to diminish. This value of k gives us the best number of clusters to make from the raw data.

```
wcss = []
for i in range (1,11):
    km = KMeans (n_clusters = i)
    km.fit (X)
    wcss.append (km.inertia_)

# The elbow curve

plt.figure (figsize = (12,6))
plt.plot (range (1,11), wcss)
plt.plot (range (1,11), wcss, linewidth = 2, color = "red",
maker = "8")
plt.xlabel ("K Value")
plt.xtricks (np.arrange (1, 11, 1))
plt.ylabel ("WCSS")
plt.show()
```

The plot:



This is known as the elbow graph, the x-axis being the number of clusters, the number of clusters is taken at the elbow joint point. This point is the point where making clusters is most relevant as here the value of WCSS suddenly stops decreasing. Here in the graph, after 5 the drop is minimal, so we take 5 to be the number of clusters.

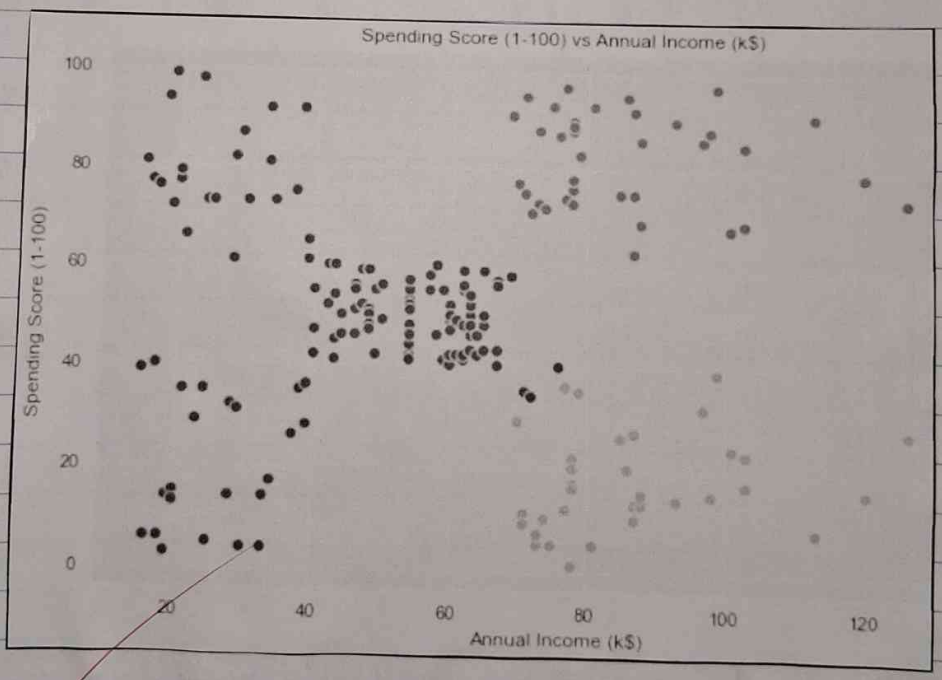
```
# Taking 5 clusters

km1 = kMeans (n_clusters=5)
```

```
# Fitting the input data
km1.fit(x)

# predicting the labels of the input data
y = km1.predict(x)

#adding the labels to a column named label
df1["label"]=y

# The new dataframe with the clustering done
df1.head()
```

The labels added to the data.

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	label
0	1	Male	19	15	39	4
1	2	Male	21	15	81	2
2	3	Female	20	16	06	4
3	4	Female	23	16	77	2
4	5	Female	31	17	40	4

```
# Scatterplot of the clusters

plt.figure(figsize=(10,6))
sns.scatterplot(x='Annual Income (k$)',y
                y='Spending Score (1-100)',
                hue="label", platte=['green',
                'orange','brown','dodgerblue','red'],
legend='full', data=df1, s=60)
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Spending Score (1-100) vs
 Annual Income (k$)')
plt.show()
```



We can clearly see that 5 different clusters have been formed from the data. The red cluster is the customers with least income and least spending score, similarly, the blue cluster is the customer with the most income and most spending score.

✳ K-Means Clustering on the basis of 3D data:

Now, we shall be working on 3 types of data. Apart from the spending score and annual income of clustomer, we shall also take in the age of the customers.

Taking the features:

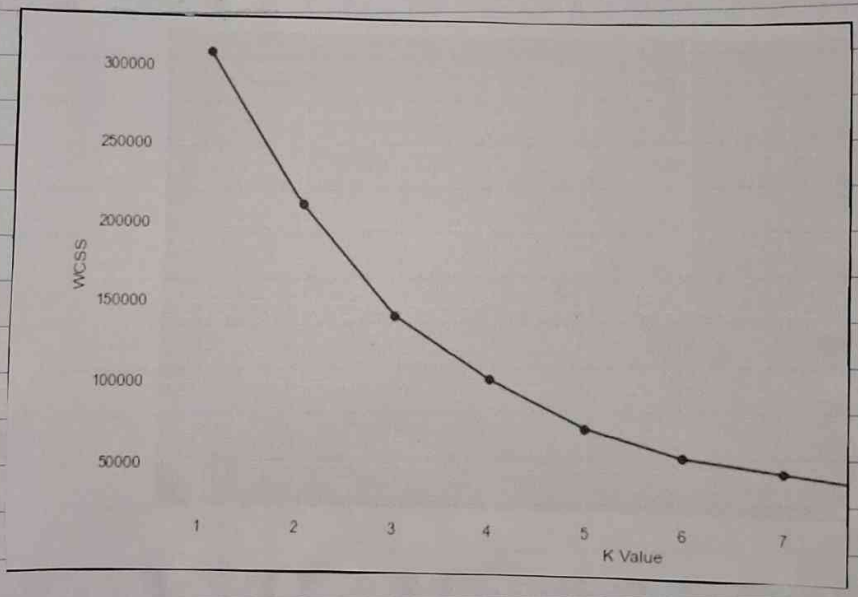
```
X2 = df2 [["Age", "Annual Income (K$)", "spending
     Score (1-100)"]]
```

Now, we calculate the Within Cluster Sum of Squared Errors (WSS) for different values of K.

```
wcss = []
for k in range (1,11):
    kmeans = kMeans (n_clusters=k, init="k-means++")
    kmeans.fit (X2)
    wcss. append (kmeans. intertia_)
```

```
plt. figure (figursize = (12,6))
plt. plot (range (1,11), wcss, linewidth=2,
color = "red", marker = "8")
plt.  xlabel ("K Value")
plt.  xtricks (np. arrange (1,11,1))
plt.  ylabel ("WCSS")
plt.  show ()
```

The WCSS Curve:-



* conclusion:
In this way we implement K-Means clustering /hierarchical clustering on sales_data_sample .csv dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("sales_data_sample.csv", encoding='Latin-1')
data.head()

# While utf-8 supports all languages according to pandas' documentation, utf-8 has a byte structure that must be respected at all times. Some
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLINE1	AD
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long Airport Avenue	
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	59 rue de l'Abbaye	
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	27 rue du Colonel Pierre Avia	
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hillside Dr.	
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Strong St.	

5 rows × 25 columns

```
data.shape
```

```
(2823, 25)
```

```
# Number of NAN values per column in the dataset
data.isnull().sum()
```

```
ORDERNUMBER          0
QUANTITYORDERED      0
PRICEEACH            0
ORDERLINENUMBER      0
SALES                0
ORDERDATE            0
STATUS               0
QTR_ID               0
MONTH_ID             0
YEAR_ID              0
PRODUCTLINE          0
MSRP                 0
PRODUCTCODE          0
CUSTOMERNAME         0
PHONE                0
ADDRESSLINE1         0
ADDRESSLINE2      2521
CITY                 0
STATE             1486
POSTALCODE          76
COUNTRY              0
TERRITORY         1074
CONTACTLASTNAME      0
CONTACTFIRSTNAME     0
DEALSIZE             0
dtype: int64
```

```
data.drop(["ORDERNUMBER", "PRICEEACH", "ORDERDATE", "PHONE", "ADDRESSLINE1", "ADDRESSLINE2", "CITY", "STATE", "TERRITORY", "POSTALCODE", "CC
```

```
data.head()
```

	QUANTITYORDERED	ORDERLINENUMBER	SALES	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MSRP	PRODUCTCODE	CUSTOMERNAME	COUN
0	30	2	2871.00	Shipped	1	2	2003	Motorcycles	95	S10_1678	Land of Toys Inc.	L
1	34	5	2765.90	Shipped	2	5	2003	Motorcycles	95	S10_1678	Reims Collectables	Fra
2	41	2	3884.34	Shipped	3	7	2003	Motorcycles	95	S10_1678	Lyon Souveniers	Fra
3	45	6	3746.70	Shipped	3	8	2003	Motorcycles	95	S10_1678	Toys4GrownUps.com	L
											Corporate Gift Ideas	

```
data.isnull().sum()
```

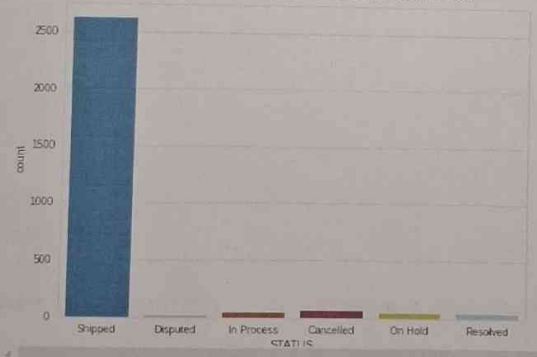
```
QUANTITYORDERED    0
ORDERLINENUMBER    0
SALES              0
STATUS             0
QTR_ID             0
MONTH_ID           0
YEAR_ID            0
PRODUCTLINE        0
MSRP               0
PRODUCTCODE        0
CUSTOMERNAME       0
COUNTRY            0
DEALSIZE           0
dtype: int64
```

```
data.describe()
```

	QUANTITYORDERED	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.00000	2823.000000
mean	35.092809	6.466171	3553.889072	2.717676	7.092455	2003.81509	100.715551
std	9.741443	4.225841	1841.865106	1.203878	3.656633	0.69967	40.187912
min	6.000000	1.000000	482.130000	1.000000	1.000000	2003.00000	33.000000
25%	27.000000	3.000000	2203.430000	2.000000	4.000000	2003.00000	68.000000
50%	35.000000	6.000000	3184.800000	3.000000	8.000000	2004.00000	99.000000
75%	43.000000	9.000000	4508.000000	4.000000	11.000000	2004.00000	124.000000
max	97.000000	18.000000	14082.800000	4.000000	12.000000	2005.00000	214.000000

```
sns.countplot(data = data , x = 'STATUS')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f27c8bbbd50>
```




```
import seaborn as sns

sns.histplot(x = 'SALES' , hue = 'PRODUCTLINE', data = data,
             element="poly")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f27c8b6f210>
```

Here we can see all the catagory lies in the range of price and hence in this we be creating a cluster on targeting the same

```
data['PRODUCTLINE'].unique()
```

```
array(['Motorcycles', 'Classic Cars', 'Trucks and Buses', 'Vintage Cars',
       'Planes', 'Ships', 'Trains'], dtype=object)
```

```
#checking the duplicated values
data.drop_duplicates(inplace=True)
```

```
data.info()
```

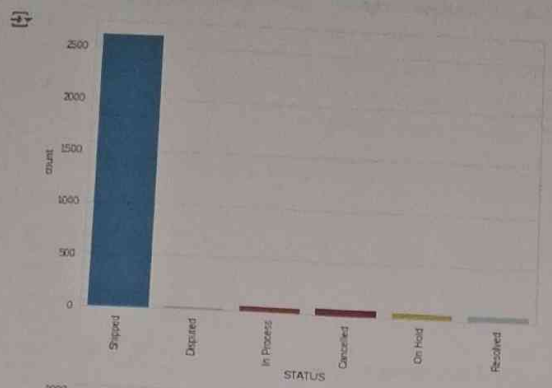
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2823 entries, 0 to 2822
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   QUANTITYORDERED   2823 non-null   int64
 1   ORDERLINENUMBER   2823 non-null   int64
 2   SALES             2823 non-null   float64
 3   STATUS            2823 non-null   object
 4   QTR_ID            2823 non-null   int64
 5   MONTH_ID          2823 non-null   int64
 6   YEAR_ID           2823 non-null   int64
 7   PRODUCTLINE       2823 non-null   object
 8   MSRP              2823 non-null   int64
 9   PRODUCTCODE       2823 non-null   object
 10  CUSTOMERNAME      2823 non-null   object
 11  COUNTRY           2823 non-null   object
 12  DEALSIZE          2823 non-null   object
dtypes: float64(1), int64(6), object(6)
memory usage: 308.8+ KB
```

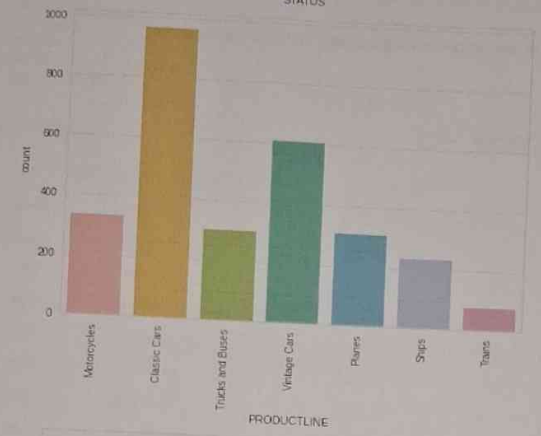
```
list_cat = data.select_dtypes(include=['object']).columns.tolist()
```

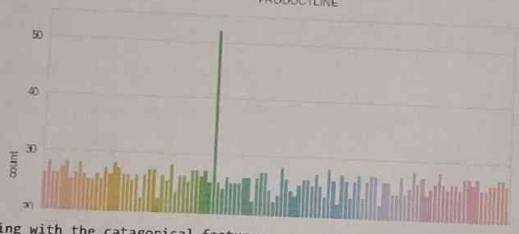
```
list_cat
```

```
['STATUS', 'PRODUCTLINE', 'PRODUCTCODE', 'CUSTOMERNAME', 'COUNTRY', 'DEALSIZE']
```

```
for i in list_cat:
  sns.countplot(data = data ,x = i)
  plt.xticks(rotation = 90)
  plt.show()
```








```
#dealing with the catagorical features
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
for i in list_cat:
  data[i]= le.fit_transform(data[i])
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2823 entries, 0 to 2822
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   QUANTITYORDERED   2823 non-null   int64
 1   ORDERLINENUMBER   2823 non-null   int64
 2   SALES             2823 non-null   float64
 3   STATUS            2823 non-null   int64
 4   QTR_ID            2823 non-null   int64
 5   MONTH_ID          2823 non-null   int64
 6   YEAR_ID           2823 non-null   int64
 7   PRODUCTLINE       2823 non-null   int64
 8   MSRP              2823 non-null   int64
```

```
 9   PRODUCTCODE       2823 non-null   int64
 10  CUSTOMERNAME      2823 non-null   int64
 11  COUNTRY           2823 non-null   int64
 12  DEALSIZE          2823 non-null   int64
dtypes: float64(1), int64(12)
memory usage: 373.3 KB
```

```
data['SALES'] = data['SALES'].astype(int)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2823 entries, 0 to 2822
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   QUANTITYORDERED   2823 non-null   int64
 1   ORDERLINENUMBER   2823 non-null   int64
 2   SALES             2823 non-null   int64
 3   STATUS            2823 non-null   int64
 4   QTR_ID            2823 non-null   int64
 5   MONTH_ID          2823 non-null   int64
 6   YEAR_ID           2823 non-null   int64
 7   PRODUCTLINE       2823 non-null   int64
 8   MSRP              2823 non-null   int64
 9   PRODUCTCODE       2823 non-null   int64
 10  CUSTOMERNAME      2823 non-null   int64
 11  COUNTRY           2823 non-null   int64
 12  DEALSIZE          2823 non-null   int64
dtypes: int64(13)
memory usage: 373.3 KB
```

```
data.describe()
```

	QUANTITYORDERED	ORDERLINENUMBER	SALES	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MSRP	PRODUCT
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.00000	2823.000000	2823.000000	2823.00
mean	35.092809	6.466171	3553.421537	4.782501	2.717676	7.092455	2003.81509	2.515055	100.715551	53.77
std	9.741443	4.225841	1841.865754	0.879416	1.203878	3.656633	0.69967	2.411665	40.187912	31.58
min	6.000000	1.000000	482.000000	0.000000	1.000000	1.000000	2003.00000	0.000000	33.000000	0.00
25%	27.000000	3.000000	2203.000000	5.000000	2.000000	4.000000	2003.00000	0.000000	68.000000	27.00
50%	35.000000	6.000000	3184.000000	5.000000	3.000000	8.000000	2004.00000	2.000000	99.000000	53.00
75%	43.000000	9.000000	4508.000000	5.000000	4.000000	11.000000	2004.00000	5.000000	124.000000	81.00
max	97.000000	18.000000	14082.000000	5.000000	4.000000	12.000000	2005.00000	6.000000	214.000000	108.00

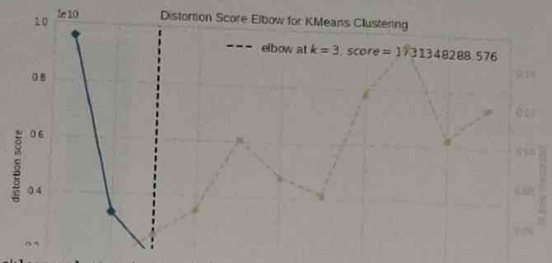
```
## taget feature are Sales and productline
X = data[['SALES','PRODUCTCODE']]
```

```
data.columns
```

```
Index(['QUANTITYORDERED', 'ORDERLINENUMBER', 'SALES', 'STATUS', 'QTR_ID',
       'MONTH_ID', 'YEAR_ID', 'PRODUCTLINE', 'MSRP', 'PRODUCTCODE',
       'CUSTOMERNAME', 'COUNTRY', 'DEALSIZE'],
      dtype='object')
```

K Means implementation

```
from yellowbrick.cluster import KElbowVisualizer
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,12)).fit(X)
visualizer.show()
```




```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=0).fit(X)
```

```
kmeans.labels_
```

```
array([0, 0, 0, ..., 3, 2, 0], dtype=int32)
```

```
kmeans.inertia_
```

```
1042223216.6249831
```

```
kmeans.n_iter_
```

```
24
```

```
kmeans.cluster_centers_
```

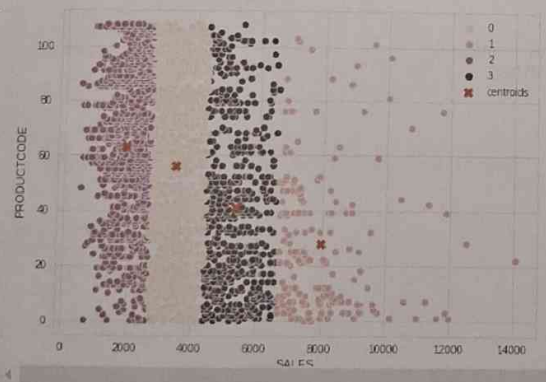
```
array([[3416.59686888,   56.3072407 ],
       [7983.1758794 ,   28.05025126],
       [1879.28363988,   63.25072604],
       [5289.27065026,   41.01230228]])
```

```
#getting the size of the clusters
from collections import Counter
Counter(kmeans.labels_)
```

```
Counter({0: 1024, 3: 565, 2: 1035, 1: 199})
```

Hence the NUmber of Clusters to be choosen Will be 4 according to the elbow method

```
sns.scatterplot(data=X, x="SALES", y="PRODUCTCODE", hue=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            marker="X", c="r", s=80, label="centroids")
plt.legend()
plt.show()
```



JSPM
Page No. :

ML

Assignment No. 6

Title: Analyzing Ups and Downs in the Indian stock Market : A Predictive Approach

Objective: The objective of this analysis is to examine the trends and patterns in the Indian stock market from 2000 to 2020 and develop a predictive model to forecast future stock price returns.

Theory: To analyze the stock market data, we will employ the following technigues:

- Exploratory Data Analysis (EDA):

We will use EDA to understand the distribution of stock prices, identify trends, and detect any anomalies in the data.

- Time Series Analysis:

We will use time series analysis to model the stock prices and identify any seasonal or periodic patterns.

- Linear Regression:

We will use linear regression to develop a predictive model to develop a predictive model that forecasts future stock price returns based on historical data.

- Code:

```
# Import necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import
train_test_split
from sklearn.linear model import
LinearRegression
from sklearn.metrics import mean_squared
_error

# Load the dataset

df = pd.read_csv ('stock_data.csv')

# Perform EDA

Print (df.head ())
```

```
print (df.info())
print (df.describe()).

# Plot the stock prices over time.

plt.figure (figsize = (10,6))
plt.plot (df ['Date'], df['close'])
plt.title ('Stock Prices Over Time']
plt.xlabel ('Data')
plt.ylabel ('close Price')
plt.show()

# Prepare the data for time series
  analysis

df ['Date'] = pd.to_datetime (df['Date'])
df.set_index ('Date', inplace = True)

# Split the data into training &
  testing sets

train_data, test_data = train_test_split
(df, test_size = 0.2, random_state = 42)

# Develop the predictive model

X = train_data.drop ('close', axis = 1)
y = train_data ['close']
model = Linear Regression ()
```

```
model.fit(X, y)

# Make predictions on the test data

predictions = model.predict(test_data.drop('close', axis=1))

# Evaluate the model

mse = mean_squared_error(test_data['close'], predictions)
print(f'Mean Squared Error: {mse:.2f}')

# Use the model to make predictions on future stock price returns

future_date = pd.to_datetime('2021-01-01')
future_date = pd.DataFrame({'Open': [100], 'High': [120], 'Low': [80]})
future_prediction = model.predict(future_data)
print(f'Predicted stock Price Return on {future_date}: {future_prediction[0]:.2f}')
```

Conclusion :

In this analysis, we examined the trends and patterns in the Indian stock market from 2000 to 2020 and developed a predictive model to forecast

future stock price returns. Model uses mea-square error algorithm. Model useful for investors and analysts seeking to make informed decisions about stock market.

miniproject-group-b-1

October 15, 2024

[25]:
```
# Name: Shriharsh J. Deshmukh
# Roll No.71 Div. 'A'
```

[ ]:
```
# Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
from sklearn.preprocessing import StandardScaler
```

[3]:
```
# Load the dataset
df = pd.read_csv("Stock_data.csv")
```

[4]:
```
df.head()
```

[4]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-09-28	234.05	235.95	230.20	233.50	233.75	3069914	7162.35
1	2018-09-27	234.55	236.80	231.10	233.80	233.25	5082859	11859.95
2	2018-09-26	240.00	240.00	232.50	235.00	234.25	2240909	5248.60
3	2018-09-25	233.30	236.75	232.00	236.25	236.10	2349368	5503.90
4	2018-09-24	233.55	239.20	230.75	234.00	233.30	3423509	7999.55

[5]:
```
# Check for missing values
print(df.isnull().sum())
```

```
Date                    0
Open                    0
```

```
High                   0
Low                    0
Last                   0
Close                  0
Total Trade Quantity   0
Turnover (Lacs)        0
dtype: int64
```

[6]:
```
df.dropna(inplace=True)
```

[7]:
```
df.dtypes
```

[7]:
```
Date                    object
Open                   float64
High                   float64
Low                    float64
Last                   float64
Close                  float64
Total Trade Quantity     int64
Turnover (Lacs)        float64
dtype: object
```

[8]:
```
df['Date'] = pd.to_datetime(df['Date'])
```

[9]:
```
# Sort the data by Date
df = df.sort_values('Date')
```

[10]:
```
# Set the Date as the index
df.set_index('Date', inplace=True)
```

[11]:
```
# Plot the stock prices
plt.figure(figsize=(10,6))
plt.plot(df.index, df['Close'], label='Close Price')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title('Stock Closing Prices Over Time')
plt.legend()
plt.show()
```

[12]:
```
# Moving Averages (Short and Long term)
df['SMA_20'] = df['Close'].rolling(window=20).mean()  # Short-term moving␣
 ↪average (20 days)
df['SMA_50'] = df['Close'].rolling(window=50).mean()  # Long-term moving␣
 ↪average (50 days)

# Percentage change in closing price
df['Price_Change'] = df['Close'].pct_change()

# Target variable: if the price will go up (1) or down (0) on the next day
df['Target'] = np.where(df['Price_Change'] > 0, 1, 0)

# Drop rows with NaN values after feature creation
df.dropna(inplace=True)
```

[13]:
```
# Features (excluding the target and date columns)
X = df[['SMA_20', 'SMA_50', 'Price_Change']]
y = df['Target']

# Split into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪shuffle=False)

# Scale the features
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[14]:
```
# Model
model = RandomForestClassifier(n_estimators=100, random_state=42)
```

[15]:
```
# Fit the model
model.fit(X_train_scaled, y_train)
```

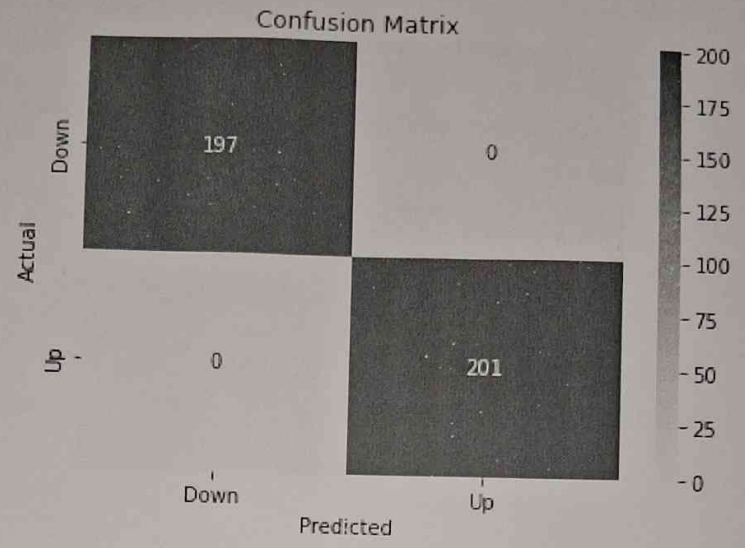
[15]:
```
RandomForestClassifier(random_state=42)
```

[16]:
```
# Make predictions
y_pred = model.predict(X_test_scaled)
```

[17]:
```
# Accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 100.00%
```

[18]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Down', 'Up'],␣
↪yticklabels=['Down', 'Up'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



[19]:
```
# Classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       197
           1       1.00      1.00      1.00       201

    accuracy                           1.00       398
   macro avg       1.00      1.00      1.00       398
weighted avg       1.00      1.00      1.00       398
```

[20]:
```
# Predict for the last N days
future_days = 30
```

[21]:
```
# Prepare the data for future prediction (typically you'd need the latest data␣
↪for prediction)
X_future = df[['SMA_20', 'SMA_50', 'Price_Change']].tail(future_days)
```

[22]:
```
# Scale the future data
X_future_scaled = scaler.transform(X_future)
```

[23]:
```
# Make predictions
future_predictions = model.predict(X_future_scaled)
```

[24]:
```
# Display the predictions
print("Predictions for the next 30 days (1=Up, 0=Down):")
print(future_predictions)
```

```
Predictions for the next 30 days (1=Up, 0=Down):
[1 1 1 1 0 0 0 1 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 1]
```

[ ]: