Assignment No.1

Title: Principal Component Analysis(PCA)

Name: Wavhal Prathmesh Navnath

Roll No: 23107137

Div: TY-B

Batch :B

In [1]:
```
import pandas as pd
import seaborn as sns
import sklearn as sk
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
df = pd.read_csv("C:/Users/prath/Downloads/winequalityN - winequalityN.csv")
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free su dio
count	6487.000000	6489.000000	6494.000000	6495.000000	6495.000000	6497.00
mean	7.216579	0.339691	0.318722	5.444326	0.056042	30.52
std	1.296750	0.164649	0.145265	4.758125	0.035036	17.74
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.00
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.00
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.00
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.00
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.00

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
type                    0
fixed acidity          10
volatile acidity        8
citric acid             3
residual sugar          2
chlorides               2
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      9
sulphates               4
alcohol                 0
quality                 0
dtype: int64
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
type                    0
fixed acidity          10
volatile acidity        8
citric acid             3
residual sugar          2
chlorides               2
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      9
sulphates               4
alcohol                 0
quality                 0
dtype: int64
```

In [8]:
```
df.isna().sum()
```

Out[8]:
```
type                    0
fixed acidity           10
volatile acidity        8
citric acid             3
residual sugar          2
chlorides               2
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      9
sulphates               4
alcohol                 0
quality                 0
dtype: int64
```

In [9]:
```
df1 = df.copy()
```

In [10]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [11]:
```
df1.dropna(inplace =True)
```

In [12]:
```
df1.isnull().sum()
```

Out[12]:
```
type                    0
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

In [13]:
```
df1['type'].value_counts()
```

Out[13]:
```
type
white    4870
red      1593
Name: count, dtype: int64
```

In [14]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df1['type'] = le.fit_transform(df1['type'])
```

In [15]:
```
df1
```

Out[15]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	densit
0	1	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.0010
1	1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.9940
2	1	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.9951
3	1	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.9956
4	1	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.9956
...	...	...	...	...	...	...	...	...	.
6491	0	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.9965
6492	0	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.9949
6494	0	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.9957
6495	0	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.9954
6496	0	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.9954

6463 rows × 13 columns

In [16]:
```
x = df1.drop('type', axis=1)
y = df1['type']
```

In [17]:
```
x
```

Out[17]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00
1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30
2	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26
3	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19
4	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19
...	...	...	...	...	...	...	...	...	..
6491	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42
6492	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45
6494	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42
6495	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57
6496	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39

6463 rows × 12 columns

In [18]:
```
y
```

Out[18]:
```
0       1
1       1
2       1
3       1
4       1
       ..
6491    0
6492    0
6494    0
6495    0
6496    0
Name: type, Length: 6463, dtype: int64
```

In [19]:
```
from sklearn.preprocessing import StandardScaler
sta = StandardScaler()
x_scaled = sta.fit_transform(x)
```

In [20]:
```
from sklearn.decomposition import PCA
model = PCA(n_components=2)
```

In [21]:
```
data_pca = model.fit_transform(x_scaled)
```

In [22]:
```
data_pca
```

Out[22]:
```
array([[ 2.49836923,  3.16466701],
       [-0.08269753, -0.47392403],
       [ 0.18303919,  0.29170561],
       ...,
       [-2.20830864, -0.65283135],
       [-2.53751532, -0.21505932],
       [-1.17991877, -0.62370364]], shape=(6463, 2))
```

In [23]:
```
model.explained_variance_
```

Out[23]:
```
array([3.04385697, 2.64958447])
```

In [24]:
```
model.score
```

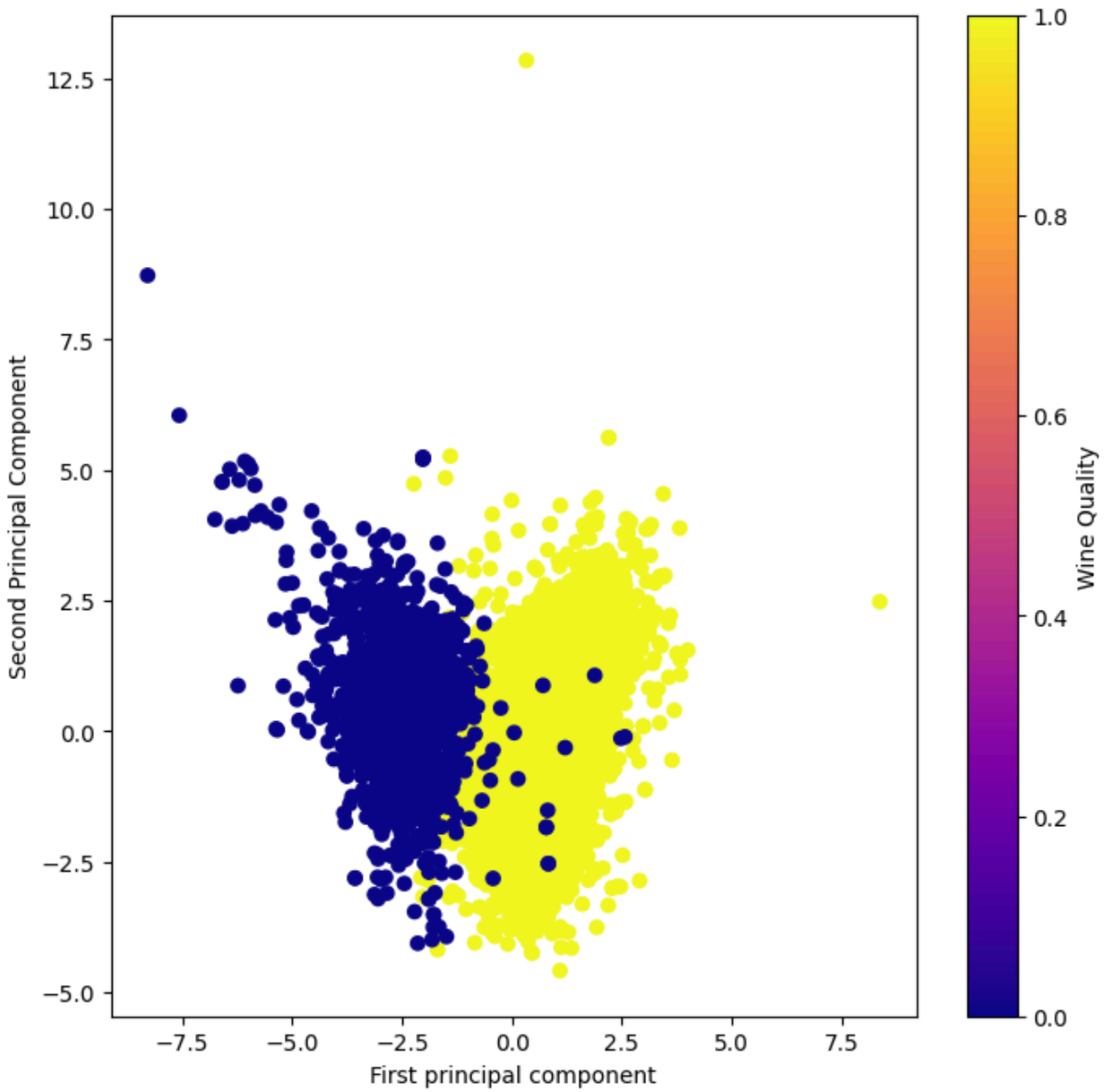
Out[24]:
```
<bound method PCA.score of PCA(n_components=2)>
```

In [25]:
```
model.explained_variance_ratio_
```

Out[25]:
```
array([0.2536155 , 0.22076454])
```

In [26]:
```
plt.figure(figsize=(8,8))
plt.scatter(data_pca[:,0],data_pca[:,1],c=df1['type'],cmap='plasma')
plt.colorbar(label='Wine Quality')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
```

Out[26]:
```
Text(0, 0.5, 'Second Principal Component')
```



In [27]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data_pca, y, test_size=0.2
```

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(x_train, y_train)
```

Out[28]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [29]:
```
y_pred = model.predict(x_test)
```

In [30]:
```
from sklearn.linear_model import LogisticRegression
```

```
model1 = LogisticRegression()
model1.fit(x_train,y_train)
```

Out[30]:

```
▾ LogisticRegression
LogisticRegression()
```

In [31]:

```
y_pred1 = model1.predict(x_test)
```

In [32]:

```
from sklearn.metrics import accuracy_score,classification_report
acc = accuracy_score(y_test, y_pred)
cr = classification_report(y_test,y_pred)
print(cr)
print("Accuracy after PCA on KNn:", acc)
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       322
           1       0.99      0.98      0.99       971

    accuracy                           0.98      1293
   macro avg       0.97      0.98      0.98      1293
weighted avg       0.98      0.98      0.98      1293

Accuracy after PCA on KNn: 0.9814385150812065
```

In [33]:

```
cr1 = classification_report(y_test,y_pred1)
print(cr1)
acc1 = accuracy_score(y_test, y_pred1)
print("Accuracy after PCA on Logistic regression:", acc1)
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.97       322
           1       0.99      0.99      0.99       971

    accuracy                           0.99      1293
   macro avg       0.98      0.99      0.98      1293
weighted avg       0.99      0.99      0.99      1293

Accuracy after PCA on Logistic regression: 0.9868522815158546
```

Assignment No.2

Develop a Ridge and Lasso regression model to predict the number of bike rentals based on

weather conditions and time. Dataset: Bike Sharing Dataset (UCI)

Name: Wavhal Prathmesh Navnath

Roll No:23107137

Class : TY-B

In [247…
```
import pandas as pd
import seaborn as sns
import sklearn as sk
import matplotlib.pyplot as plt
import numpy as np
```

In [266…
```
df= pd.read_csv("C:/Users/prath/Downloads/hour.csv")
```

In [267…
```
df
```

Out[267…

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday
0	1	2011-01-01	1	0	1	0	0	6	0
1	2	2011-01-01	1	0	1	1	0	6	0
2	3	2011-01-01	1	0	1	2	0	6	0
3	4	2011-01-01	1	0	1	3	0	6	0
4	5	2011-01-01	1	0	1	4	0	6	0
...	...	...	...	...	...	...	...	...	...
17374	17375	2012-12-31	1	1	12	19	0	1	1
17375	17376	2012-12-31	1	1	12	20	0	1	1
17376	17377	2012-12-31	1	1	12	21	0	1	1
17377	17378	2012-12-31	1	1	12	22	0	1	1
17378	17379	2012-12-31	1	1	12	23	0	1	1

17379 rows × 17 columns

In [268…
```
df = df.drop(columns=["dteday","instant","season","yr","holiday"])
```

In [269…
```
df
```

Out[269…

	mnth	hr	weekday	workingday	weathersit	temp	atemp	hum	windsp
0	1	0	6	0	1	0.24	0.2879	0.81	0.
1	1	1	6	0	1	0.22	0.2727	0.80	0.
2	1	2	6	0	1	0.22	0.2727	0.80	0.
3	1	3	6	0	1	0.24	0.2879	0.75	0.
4	1	4	6	0	1	0.24	0.2879	0.75	0.
...	...	...	...	...	...	...	...	...	
17374	12	19	1	1	2	0.26	0.2576	0.60	0.
17375	12	20	1	1	2	0.26	0.2576	0.60	0.
17376	12	21	1	1	1	0.26	0.2576	0.60	0.
17377	12	22	1	1	1	0.26	0.2727	0.56	0.
17378	12	23	1	1	1	0.26	0.2727	0.65	0.

17379 rows × 12 columns

In [270…
```
df.isnull().sum()
```

Out[270…
```
mnth          0
hr            0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
cnt           0
dtype: int64
```

In [271…
```
df.isna().sum()
```

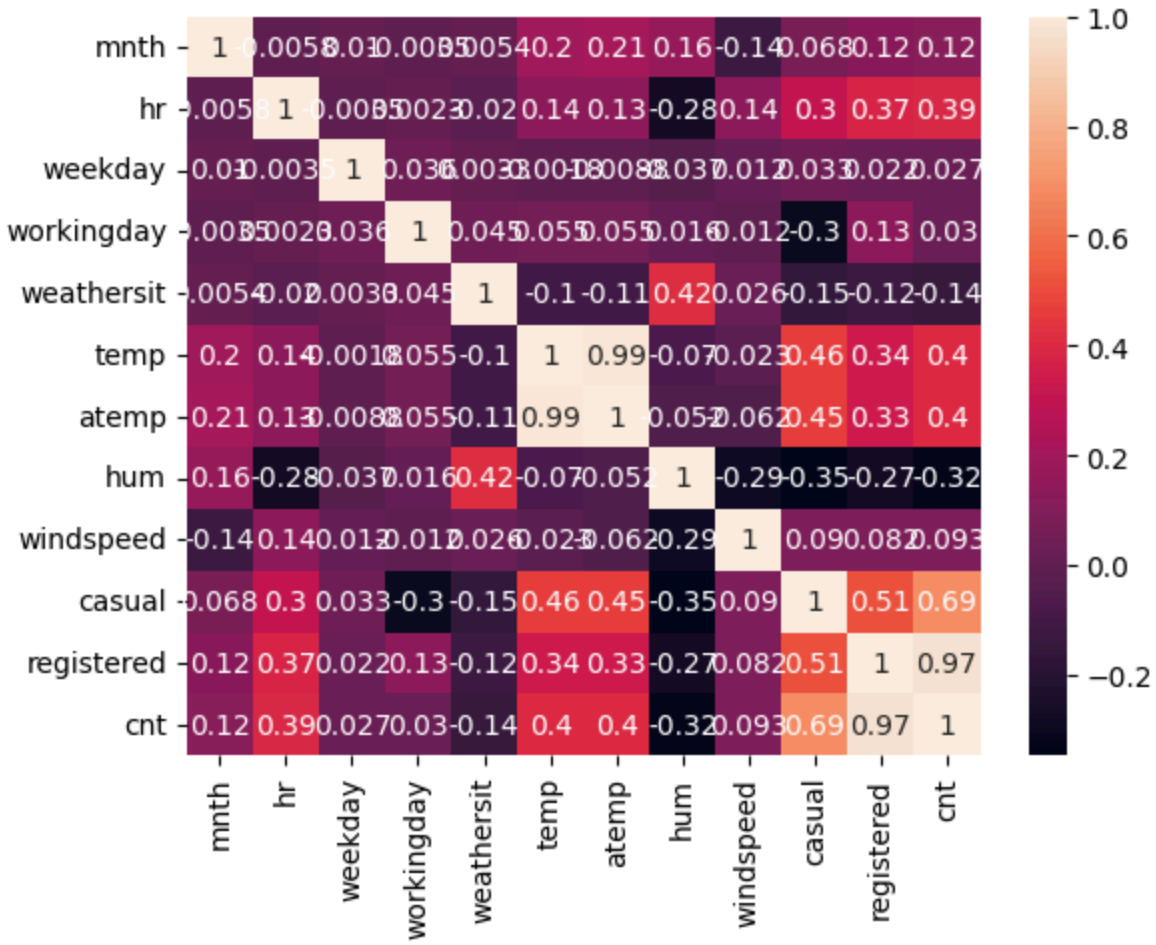
Out[271…
```
mnth          0
hr            0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
cnt           0
dtype: int64
```

In [272…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   mnth        17379 non-null  int64  
 1   hr          17379 non-null  int64  
 2   weekday     17379 non-null  int64  
 3   workingday  17379 non-null  int64  
 4   weathersit  17379 non-null  int64  
 5   temp        17379 non-null  float64
 6   atemp       17379 non-null  float64
 7   hum         17379 non-null  float64
 8   windspeed   17379 non-null  float64
 9   casual      17379 non-null  int64  
 10  registered  17379 non-null  int64  
 11  cnt         17379 non-null  int64  
dtypes: float64(4), int64(8)
memory usage: 1.6 MB
```

In [273…
```
sns.heatmap(df.corr(),annot=True)
```

Out[273…
```
<Axes: >
```



In [274…
```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
```

In [278…
```
month_new1 = ohe.fit_transform(df[['mnth']])
```

In [280…
```
month_new_df1 = pd.DataFrame(month_new1.toarray(),columns=ohe.get_feature_name
```

In [281…
```
month_new_df1
```

Out[281…

	mnth_1	mnth_2	mnth_3	mnth_4	mnth_5	mnth_6	mnth_7	mnth_8	m
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
17374	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
17375	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
17376	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
17377	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
17378	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

17379 rows × 12 columns

In [ ]:

In [285…
```
week_new1 = ohe.fit_transform(df[['weekday']])
```

In [286…
```
week_new_df1 = pd.DataFrame(week_new1.toarray(),columns=ohe.get_feature_names_
```

In [287…
```
df = pd.concat([df.drop(['weekday','mnth'],axis=1),week_new_df1,month_new_df1]
```

In [288…
```
df
```

Out[288…

	hr	workingday	weathersit	temp	atemp	hum	windspeed	casual	reg
0	0	0	1	0.24	0.2879	0.81	0.0000	3	
1	1	0	1	0.22	0.2727	0.80	0.0000	8	
2	2	0	1	0.22	0.2727	0.80	0.0000	5	
3	3	0	1	0.24	0.2879	0.75	0.0000	3	
4	4	0	1	0.24	0.2879	0.75	0.0000	0	
...	...	...	...	...	...	...	...	...	
17374	19	1	2	0.26	0.2576	0.60	0.1642	11	
17375	20	1	2	0.26	0.2576	0.60	0.1642	8	
17376	21	1	1	0.26	0.2576	0.60	0.1642	7	
17377	22	1	1	0.26	0.2727	0.56	0.1343	13	
17378	23	1	1	0.26	0.2727	0.65	0.1343	12	

17379 rows × 29 columns

In [289…

```
x = df.drop("cnt",axis=1)
y  = df['cnt']
```

In [290…

```
x
```

Out[290…

	hr	workingday	weathersit	temp	atemp	hum	windspeed	casual	reg
0	0	0	1	0.24	0.2879	0.81	0.0000	3	
1	1	0	1	0.22	0.2727	0.80	0.0000	8	
2	2	0	1	0.22	0.2727	0.80	0.0000	5	
3	3	0	1	0.24	0.2879	0.75	0.0000	3	
4	4	0	1	0.24	0.2879	0.75	0.0000	0	
...	...	...	...	...	...	...	...	...	
17374	19	1	2	0.26	0.2576	0.60	0.1642	11	
17375	20	1	2	0.26	0.2576	0.60	0.1642	8	
17376	21	1	1	0.26	0.2576	0.60	0.1642	7	
17377	22	1	1	0.26	0.2727	0.56	0.1343	13	
17378	23	1	1	0.26	0.2727	0.65	0.1343	12	

17379 rows × 28 columns

In [291…

```
y
```

Out[291…
```
0         16
1         40
2         32
3         13
4          1
        ...
17374    119
17375     89
17376     90
17377     61
17378     49
Name: cnt, Length: 17379, dtype: int64
```

In [292…
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_sta
```

In [293…
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [294…
```
x_train_scaled = sc.fit_transform(x_train)
```

In [295…
```
x_test_scaled = sc.transform(x_test)
```

In [296…
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

In [297…
```
model.fit(x_train_scaled,y_train)
```

Out[297…

▾ LinearRegression

Parameters

fit_intercept	True
copy_X	True
tol	1e-06
n_jobs	None
positive	False

In [298…
```
y_pred = model.predict(x_test_scaled)
```

In [299…
```
y_pred
```

Out[299…
```
array([425.,  88.,   4., ...,  98., 266., 267.])
```

In [300…
```
from sklearn.metrics import r2_score,mean_absolute_error
mae = mean_absolute_error(y_test,y_pred)
r2= r2_score(y_test,y_pred)
```

In [301…
```
mae
```

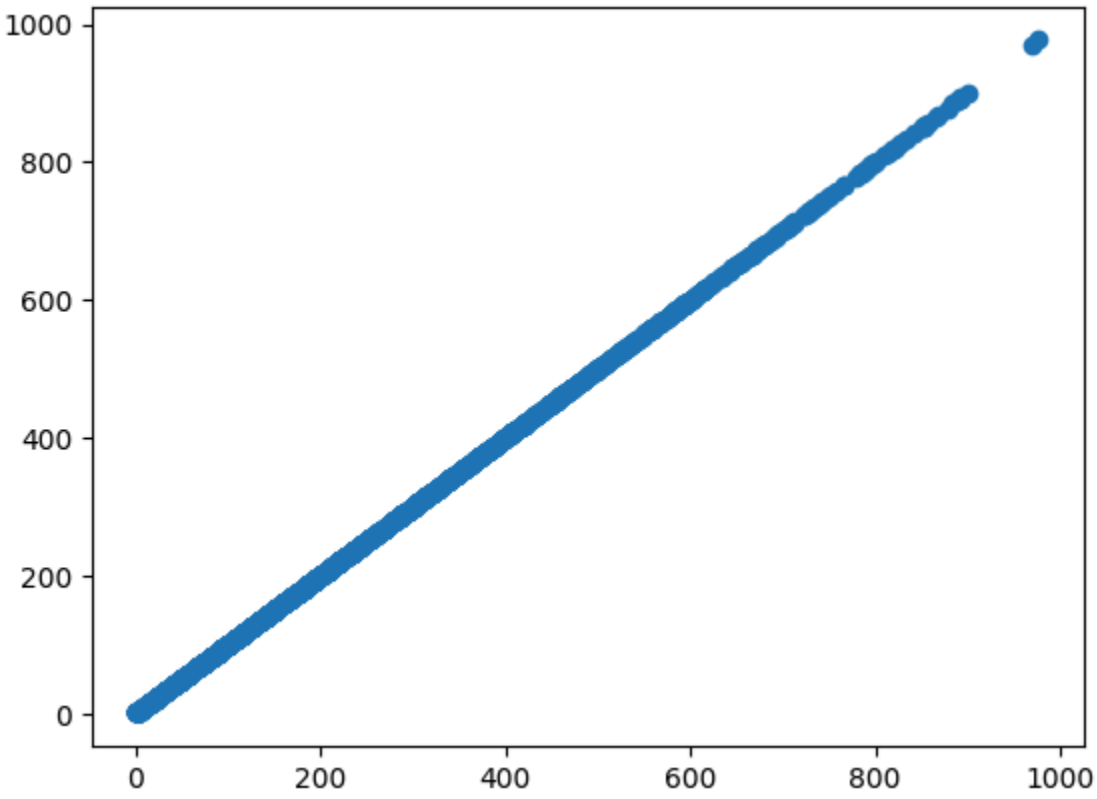
Out[301…
```
1.616546048025899e-13
```

In [302…
```
r2
```

Out[302…
```
1.0
```

In [303…
```
plt.scatter(y_test,y_pred)
```

Out[303…
```
<matplotlib.collections.PathCollection at 0x1f340a014d0>
```



In [304…
```
from sklearn.linear_model import Ridge
model1 = Ridge()
```

In [305…
```
model1.fit(x_train_scaled,y_train)
```

Out[305…

Ridge

Parameters

alpha	1.0
fit_intercept	True
copy_X	True
max_iter	None
tol	0.0001
solver	'auto'
positive	False
random_state	None

In [306…
```
y_pred1 = model1.predict(x_test_scaled)
```

In [307…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
mae =mean_absolute_error(y_test,y_pred1)
mse = mean_squared_error(y_test,y_pred1)
rmse = np.sqrt(mse)
print(mse)
print(mae)
print(rmse)
r2 = r2_score(y_pred1,y_test)
```

```
0.00015910433008547108
0.009077921659339962
0.012613656491496471
```

In [308…
```
r2
```

Out[308… 0.999999994974901

In [ ]:

In [309…
```
from sklearn.linear_model import Lasso
model2 = Lasso(alpha=1)
```

In [310…
```
model2.fit(x_train_scaled,y_train)
```

Out[310…

Lasso

Parameters

alpha	1
fit_intercept	True
precompute	False
copy_X	True
max_iter	1000
tol	0.0001
warm_start	False
positive	False
random_state	None
selection	'cyclic'

In [311…
```
y_pred3= model2.predict(x_test_scaled)
```

In [312…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
mae =mean_absolute_error(y_test,y_pred3)
mse = mean_squared_error(y_test,y_pred3)
rmse = np.sqrt(mse)
print(mse)
print(mae)
print(rmse)
r2 = r2_score(y_pred3,y_test)
```

```
1.258919694212011
0.8788669770931126
1.1220159063988402
```

In [313…
```
r2
```

Out[313… 0.9999597569690253

In [314…
```
from sklearn.linear_model import ElasticNet
es = ElasticNet()
```

In [315…
```
es.fit(x_train_scaled,y_train)
```

Out[315…

ElasticNet

Parameters

alpha	1.0
l1_ratio	0.5
fit_intercept	True
precompute	False
max_iter	1000
copy_X	True
tol	0.0001
warm_start	False
positive	False
random_state	None
selection	'cyclic'

In [316…
```
y_pred2 = es.predict(x_test_scaled)
```

In [317…
```
y_pred2
```

Out[317…
```
array([420.63067753, 114.71354173,  47.61020139, ...,  94.15737458,
       239.54272137, 240.53479038])
```

In [318…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
mae =mean_absolute_error(y_test,y_pred2)
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
print(mse)
print(mae)
print(rmse)
r2 = r2_score(y_pred2,y_test)
```

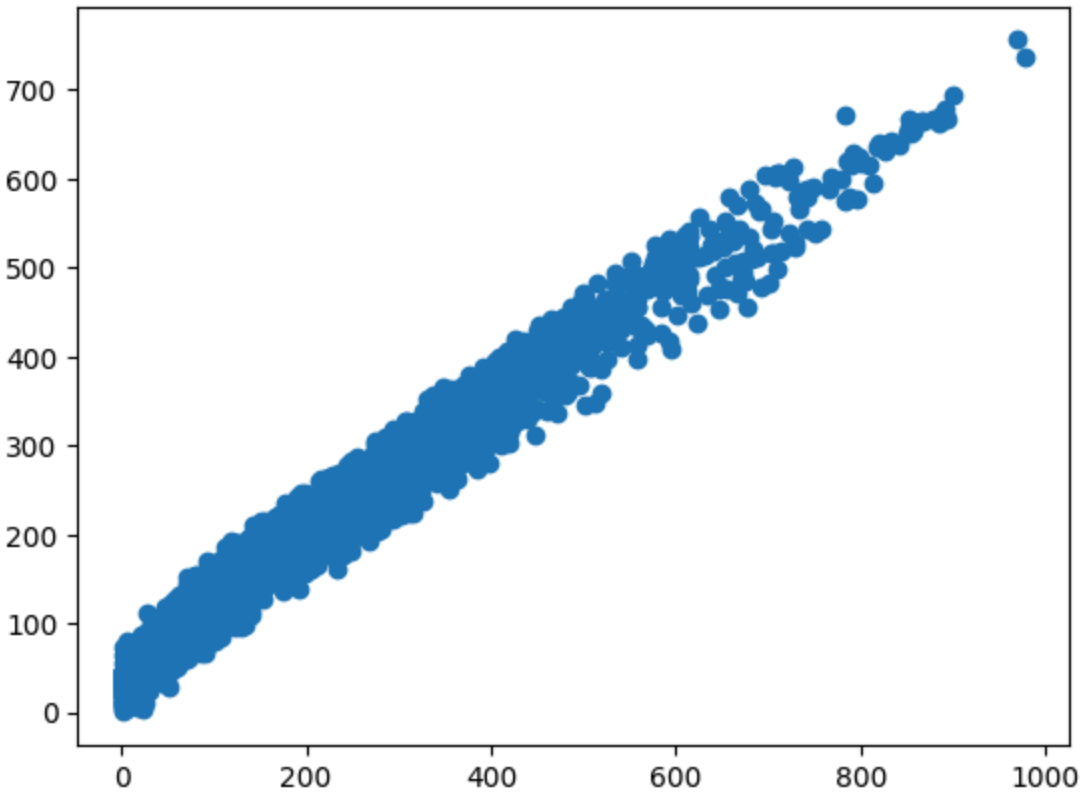
```
4.8268396227577376e-26
36.9727613893678
2.1970069692100974e-13
```

In [319…
```
r2
```

Out[319…
```
0.8623554877942854
```

In [320…
```
plt.scatter(y_test,y_pred2)
```

Out[320…
```
<matplotlib.collections.PathCollection at 0x1f3408fead0>
```



In [ ]:

In [ ]:

Prediction only with the Weather data points

In [180…
```
df1 =  pd.read_csv("C:/Users/prath/Downloads/day.csv")
```

In [181…
```
df1
```

Out[181…

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	wea
0	1	2011-01-01	1	0	1	0	6	0	
1	2	2011-01-02	1	0	1	0	0	0	
2	3	2011-01-03	1	0	1	0	1	1	
3	4	2011-01-04	1	0	1	0	2	1	
4	5	2011-01-05	1	0	1	0	3	1	
...	...	...	...	...	...	...	...	...	
726	727	2012-12-27	1	1	12	0	4	1	
727	728	2012-12-28	1	1	12	0	5	1	
728	729	2012-12-29	1	1	12	0	6	0	
729	730	2012-12-30	1	1	12	0	0	0	
730	731	2012-12-31	1	1	12	0	1	1	

731 rows × 16 columns

In [182…
```
df1 = df1.drop(columns=["dteday","instant","season","yr","holiday"])
```

In [183…
```
df1
```

Out[183…

	mnth	weekday	workingday	weathersit	temp	atemp	hum	win
0	1	6	0	2	0.344167	0.363625	0.805833	0
1	1	0	0	2	0.363478	0.353739	0.696087	0
2	1	1	1	1	0.196364	0.189405	0.437273	0
3	1	2	1	1	0.200000	0.212122	0.590435	0
4	1	3	1	1	0.226957	0.229270	0.436957	0
...	...	...	...	...	...	...	...	
726	12	4	1	2	0.254167	0.226642	0.652917	0
727	12	5	1	2	0.253333	0.255046	0.590000	0
728	12	6	0	2	0.253333	0.242400	0.752917	0
729	12	0	0	1	0.255833	0.231700	0.483333	0
730	12	1	1	2	0.215833	0.223487	0.577500	0

731 rows × 11 columns

In [184…
```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
```

In [185…
```
week_new = ohe.fit_transform(df1[['weekday']])
month_new = ohe.fit_transform(df1[['mnth']])
```

In [158…
```
week_new
```

Out[158…
```
<731x7 sparse matrix of type '<class 'numpy.float64'>'
        with 731 stored elements in Compressed Sparse Row format>
```

In [159…
```
ohe.get_feature_names_out(['weekday'])
```

Out[159…
```
array(['weekday_0', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4',
       'weekday_5', 'weekday_6'], dtype=object)
```

In [186…
```
ohe.get_feature_names_out(['mnth'])
```

Out[186…
```
array(['mnth_1', 'mnth_2', 'mnth_3', 'mnth_4', 'mnth_5', 'mnth_6',
       'mnth_7', 'mnth_8', 'mnth_9', 'mnth_10', 'mnth_11', 'mnth_12'],
      dtype=object)
```

In [160…
```
week_new_df = pd.DataFrame(week_new.toarray(),columns=ohe.get_feature_names_ou
```

In [187…
```
mnth_new_df = pd.DataFrame(month_new.toarray(),columns=ohe.get_feature_names_c
```

In [161…
```
week_new_df
```

Out[161…

	weekday_0	weekday_1	weekday_2	weekday_3	weekday_4	weekday_5	w
0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	1.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	1.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	0.0	0.0	
...	...	...	...	...	...	...	
726	0.0	0.0	0.0	0.0	1.0	0.0	
727	0.0	0.0	0.0	0.0	0.0	1.0	
728	0.0	0.0	0.0	0.0	0.0	0.0	
729	1.0	0.0	0.0	0.0	0.0	0.0	
730	0.0	1.0	0.0	0.0	0.0	0.0	

731 rows × 7 columns

In [190…
```
mnth_new_df
```

Out[190…

	mnth_1	mnth_2	mnth_3	mnth_4	mnth_5	mnth_6	mnth_7	mnth_8	mnth
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
726	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
727	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
728	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
729	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

731 rows × 12 columns

In [191…
```
df1 = pd.concat([df1.drop(['weekday','mnth'],axis=1),week_new_df,mnth_new_df],
```

In [192…
```
df1
```

Out[192…

	workingday	weathersit	temp	atemp	hum	windspeed	casual
0	0	2	0.344167	0.363625	0.805833	0.160446	331
1	0	2	0.363478	0.353739	0.696087	0.248539	131
2	1	1	0.196364	0.189405	0.437273	0.248309	120
3	1	1	0.200000	0.212122	0.590435	0.160296	108
4	1	1	0.226957	0.229270	0.436957	0.186900	82
...	...	...	...	...	...	...	...
726	1	2	0.254167	0.226642	0.652917	0.350133	247
727	1	2	0.253333	0.255046	0.590000	0.155471	644
728	0	2	0.253333	0.242400	0.752917	0.124383	159
729	0	1	0.255833	0.231700	0.483333	0.350754	364
730	1	2	0.215833	0.223487	0.577500	0.154846	439

731 rows × 28 columns

In [196…
```
x = df1.drop(columns=["cnt","weathersit","casual","registered","workingday"],a
y  = df1['cnt']
```

In [197…
```
x
```

Out[197…

	temp	atemp	hum	windspeed	weekday_0	weekday_1	weekday
0	0.344167	0.363625	0.805833	0.160446	0.0	0.0	0
1	0.363478	0.353739	0.696087	0.248539	1.0	0.0	0
2	0.196364	0.189405	0.437273	0.248309	0.0	1.0	0
3	0.200000	0.212122	0.590435	0.160296	0.0	0.0	1
4	0.226957	0.229270	0.436957	0.186900	0.0	0.0	0
...	...	...	...	...	...	...	
726	0.254167	0.226642	0.652917	0.350133	0.0	0.0	0
727	0.253333	0.255046	0.590000	0.155471	0.0	0.0	0
728	0.253333	0.242400	0.752917	0.124383	0.0	0.0	0
729	0.255833	0.231700	0.483333	0.350754	1.0	0.0	0
730	0.215833	0.223487	0.577500	0.154846	0.0	1.0	0

731 rows × 23 columns

In [198…
```
y
```

Out[198…
```
0       985
1       801
2      1349
3      1562
4      1600
       ...
726    2114
727    3095
728    1341
729    1796
730    2729
Name: cnt, Length: 731, dtype: int64
```

In [199…
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_sta
```

In [200…
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [201…
```
x_train_scaled = sc.fit_transform(x_train)
```

In [202…
```
x_test_scaled = sc.transform(x_test)
```

In [203…
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

In [204…
```
model.fit(x_train_scaled,y_train)
```

Out[204…

▾ LinearRegression

Parameters

fit_intercept	True
copy_X	True
tol	1e-06
n_jobs	None
positive	False

In [205…
```
y_pred = model.predict(x_test_scaled)
```

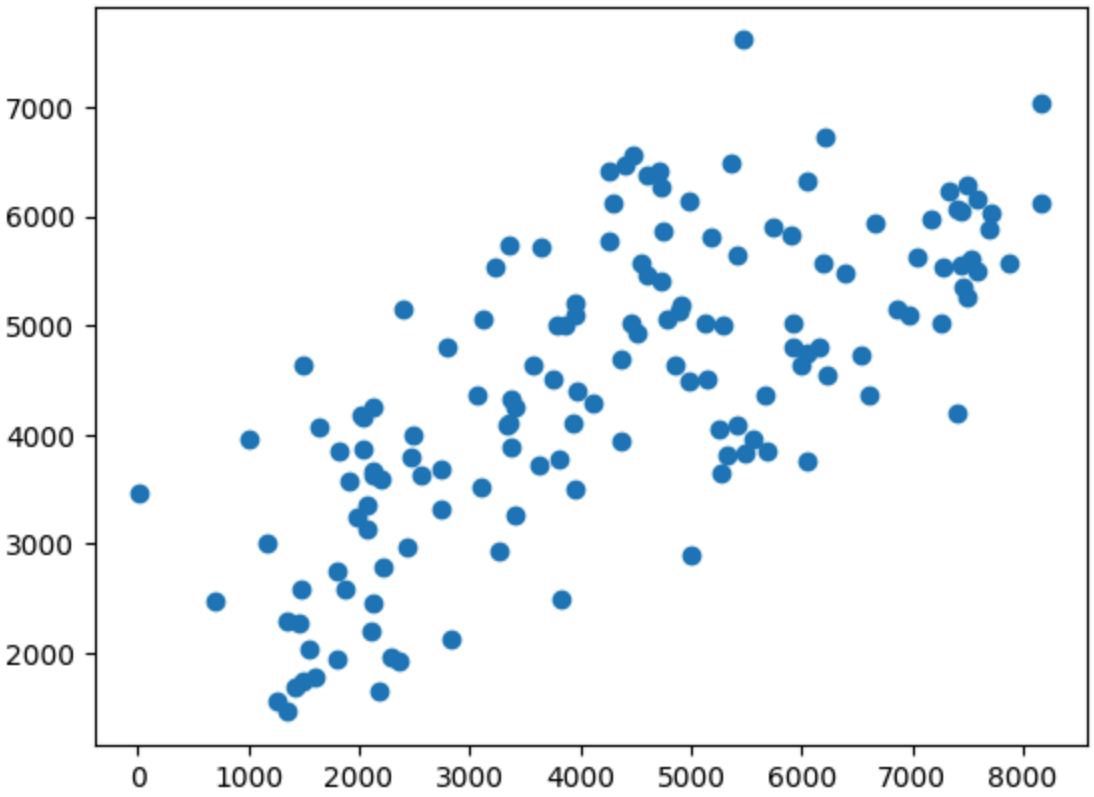
In [206…
```
from sklearn.metrics import r2_score,mean_absolute_error
mae = mean_absolute_error(y_test,y_pred)
r2= r2_score(y_test,y_pred)
```

In [207…
```
r2
```

Out[207… 0.49210234387317797

In [208…
```
plt.scatter(y_test,y_pred)
```

Out[208… <matplotlib.collections.PathCollection at 0x1f33aa59850>



In [209…
```
from sklearn.linear_model import Ridge
model1 = Ridge()
```

In [210…
```
model1.fit(x_train_scaled,y_train)
```

Out[210…

Ridge

Parameters

alpha	1.0
fit_intercept	True
copy_X	True
max_iter	None
tol	0.0001
solver	'auto'
positive	False
random_state	None

In [211…
```
y_pred = model1.predict(x_test_scaled)
```

In [212…
```
from sklearn.metrics import r2_score,mean_absolute_error
```

```
mae = mean_absolute_error(y_test,y_pred)
r2= r2_score(y_test,y_pred)
```

In [213…
```
r2
```

Out[213…
```
0.4930987144804707
```

In [214…
```
from sklearn.linear_model import Lasso
model2 = Lasso(alpha=1)
```

In [215…
```
model2.fit(x_train_scaled,y_train)
```

Out[215…

▾ Lasso

Parameters

alpha	1
fit_intercept	True
precompute	False
copy_X	True
max_iter	1000
tol	0.0001
warm_start	False
positive	False
random_state	None
selection	'cyclic'

In [216…
```
y_pred = model2.predict(x_test_scaled)
```

In [217…
```
from sklearn.metrics import r2_score,mean_absolute_error
mae = mean_absolute_error(y_test,y_pred)
r2= r2_score(y_test,y_pred)
```

In [218…
```
r2
```

Out[218…
```
0.4932945236635691
```

In [219…
```
from sklearn.linear_model import ElasticNet
es = ElasticNet()
```

In [220…
```
es.fit(x_train_scaled,y_train)
```

Out[220…

ElasticNet

Parameters

alpha	1.0
l1_ratio	0.5
fit_intercept	True
precompute	False
max_iter	1000
copy_X	True
tol	0.0001
warm_start	False
positive	False
random_state	None
selection	'cyclic'

In [221…
```
y_pred = es.predict(x_test_scaled)
```

In [222…
```
from sklearn.metrics import r2_score,mean_absolute_error
mae = mean_absolute_error(y_test,y_pred)
r2= r2_score(y_test,y_pred)
```

In [223…
```
r2
```

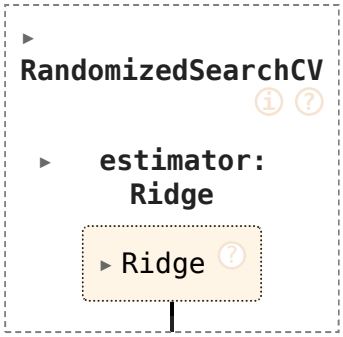
Out[223… 0.473948291371206

In [237…
```
Param_grid = {'alpha' : [1,10,100,1000],
    'max_iter':[1,10,50,100],
    'selection' : ['cyclic', 'random'],
}
```

In [234…
```
from sklearn.model_selection import RandomizedSearchCV
rc = RandomizedSearchCV(model1,param_distributions=Param_grid,cv=5,n_jobs=-1,v
```

In [235…
```
rc
```

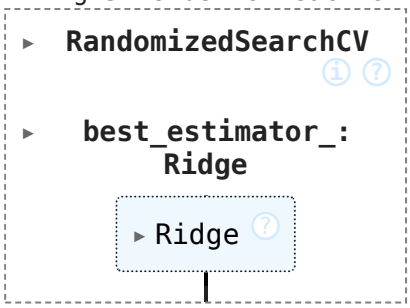
Out[235…



In [236…
```
rc.fit(x_train_scaled,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[236…



In [232…
```
rc.best_params_
```

Out[232…
```
{'solver': 'lsqr', 'max_iter': 50, 'alpha': 10}
```

In [233…
```
rc.best_score_
```

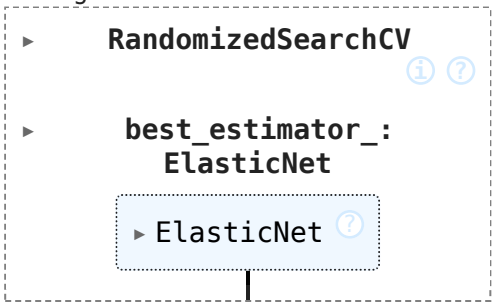
Out[233…
```
0.4944832427303142
```

In [239…
```
from sklearn.model_selection import RandomizedSearchCV
rc1 = RandomizedSearchCV(es,param_distributions=Param_grid,cv=5,n_jobs=-1,verb
```

In [240…
```
rc1.fit(x_train_scaled,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[240…



In [241…
```
rc1.best_estimator_
```

Out[241…

ElasticNet

Parameters

alpha	1
l1_ratio	0.5
fit_intercept	True
precompute	False
max_iter	100
copy_X	True
tol	0.0001
warm_start	False
positive	False
random_state	None
selection	'random'

In [242…
```
rc1.best_params_
```

Out[242… `{'selection': 'random', 'max_iter': 100, 'alpha': 1}`

In [243…
```
rc1.best_score_
```

Out[243… `0.4659627912508756`

In [ ]:

Assignment No.3

Title:Create a multiclass classification model to predict wine quality based on chemical properties.

Dataset: Wine Quality Dataset (UCI)

Name: Wavhal Prathmesh Navnath

Roll No:23107137

Class : TY-B

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn as sk
```

In [3]:
```
df = pd.read_csv("C:/Users/prath/Downloads/WineQT.csv")
```

In [4]:
```
df
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51
...	...	...	...	...	...	...	...	...	..
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57

1143 rows × 13 columns

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
```

In [6]:
```
df.isna().sum()
```

Out[6]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
```

In [7]:
```
df.info()
```

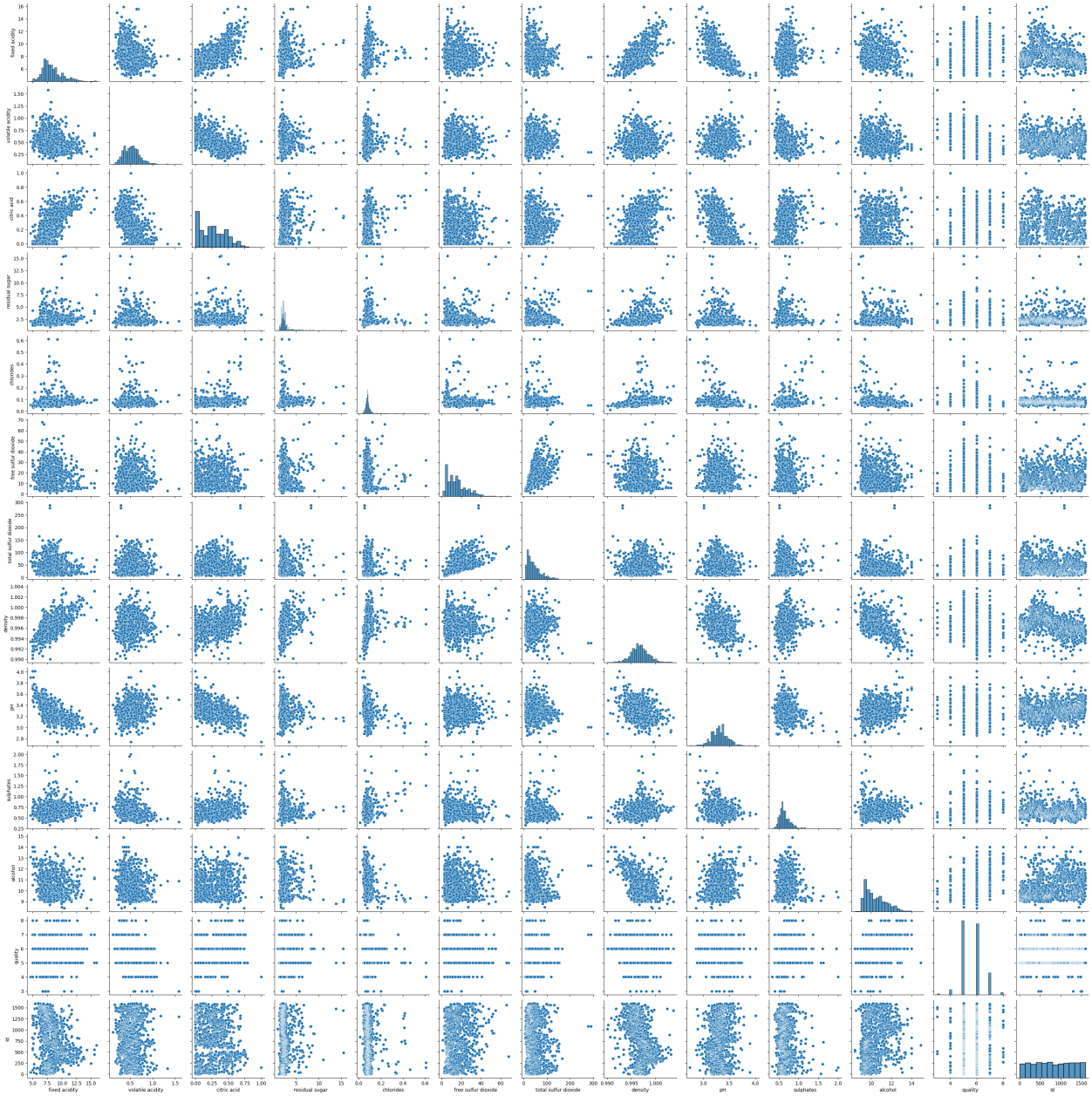
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

In [8]:
```
df.describe
```

Out[8]:
```
<bound method NDFrame.describe of       fixed acidity  volatile acidity  citr
ic acid  residual sugar  chlorides  \
0               7.4             0.700         0.00              1.9      0.076
1               7.8             0.880         0.00              2.6      0.098
2               7.8             0.760         0.04              2.3      0.092
3              11.2             0.280         0.56              1.9      0.075
4               7.4             0.700         0.00              1.9      0.076
...             ...               ...          ...               ...        ...
1138            6.3             0.510         0.13              2.3      0.076
1139            6.8             0.620         0.08              1.9      0.068
1140            6.2             0.600         0.08              2.0      0.090
1141            5.9             0.550         0.10              2.2      0.062
1142            5.9             0.645         0.12              2.0      0.075

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
...                   ...                   ...      ...   ...        ...
1138                 29.0                  40.0  0.99574  3.42       0.75
1139                 28.0                  38.0  0.99651  3.42       0.82
1140                 32.0                  44.0  0.99490  3.45       0.58
1141                 39.0                  51.0  0.99512  3.52       0.76
1142                 32.0                  44.0  0.99547  3.57       0.71

      alcohol  quality    Id
0         9.4        5     0
1         9.8        5     1
2         9.8        5     2
3         9.8        6     3
4         9.4        5     4
...       ...      ...   ...
1138     11.0        6  1592
1139      9.5        6  1593
1140     10.5        5  1594
1141     11.2        6  1595
1142     10.2        5  1597

[1143 rows x 13 columns]>
```

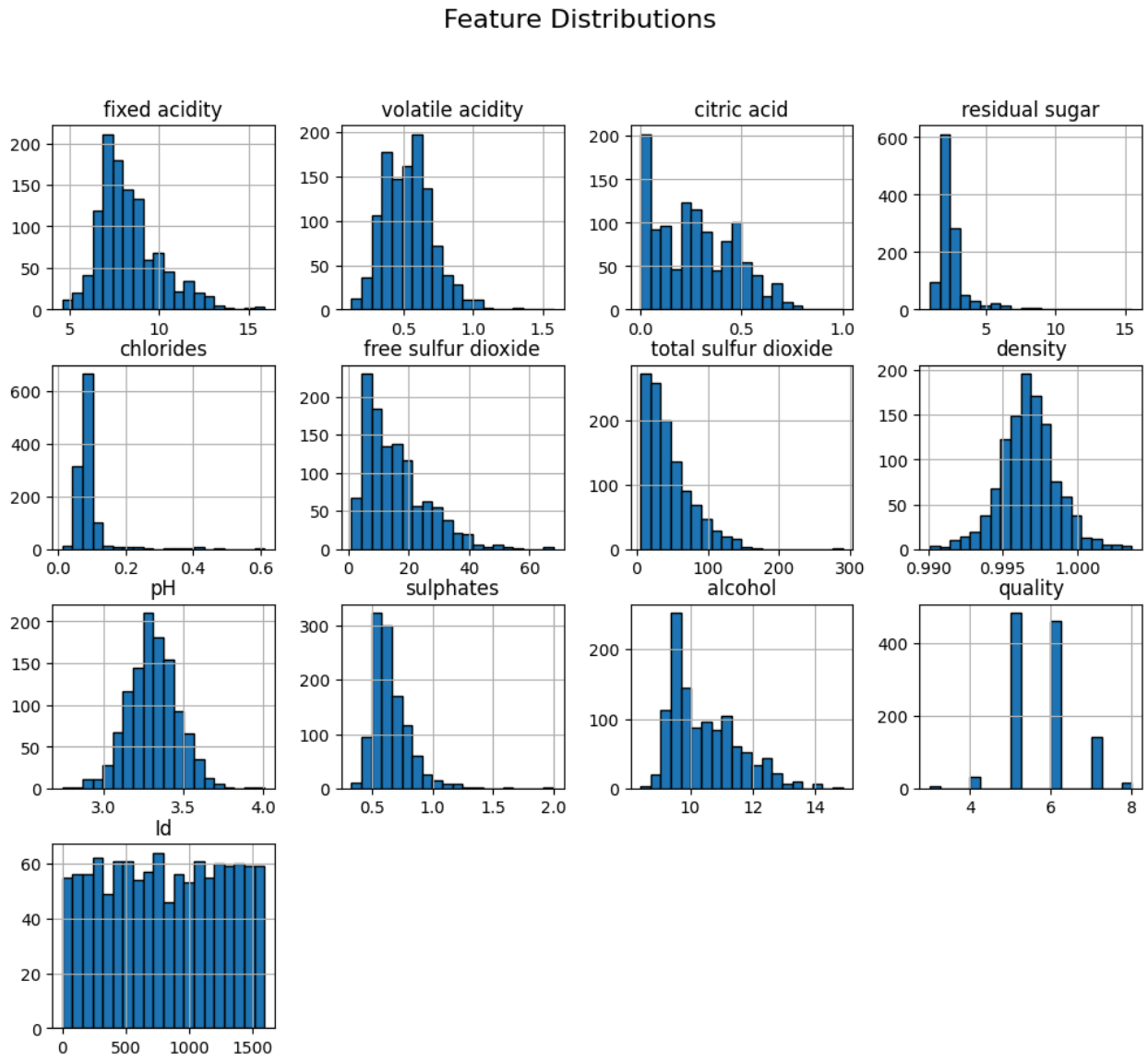
In [9]:
```
sns.pairplot(df)
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1b3d44435d0>
```



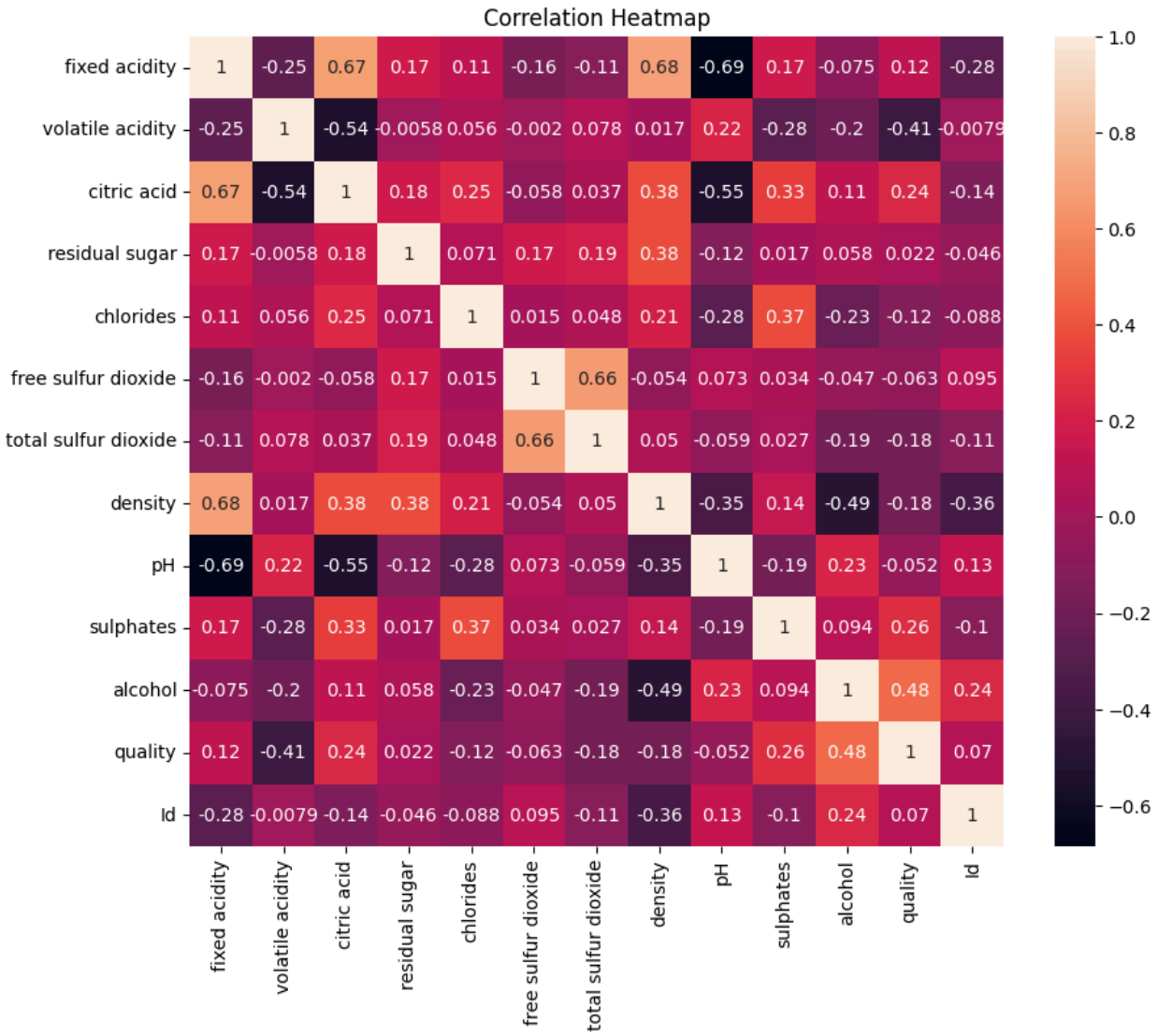
In [100…

```
df.hist(figsize=(12,10), bins=20, edgecolor="black")
plt.suptitle("Feature Distributions", fontsize=16)
plt.show()
```



In [106…

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True)
plt.title("Correlation Heatmap")
plt.show()
```



In [10]:
```
## Dependent and Independent Variables
x = df.drop(columns=['quality','Id'],axis=1)
y= df['quality']
```

In [ ]:
```
## SMOTE For the Imbalance dataset

from imblearn.over_sampling import SMOTE
oversample = SMOTE()
x,y = oversample.fit_resample(x,df['quality'])
```

In [103…
```
y.value_counts()
```

Out[103…
```
quality
5    483
6    483
7    483
4    483
8    483
3    483
Name: count, dtype: int64
```

In [105…
```
x.shape,y.shape
```

Out[105…
```
((2898, 11), (2898,))
```

In [60]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_sta
```

In [61]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [62]:
```
x_train_scaled = sc.fit_transform(x_train)
```

In [63]:
```
x_test_scaled = sc.transform(x_test)
```

In [119…
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
```

In [120…
```
model
```

Out[120…

▾ DecisionTreeClassifier

Parameters

criterion	'gini'
splitter	'best'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	None
random_state	None
max_leaf_nodes	None
min_impurity_decrease	0.0
class_weight	None
ccp_alpha	0.0
monotonic_cst	None

In [121…

```
model.fit(x_train_scaled,y_train)
```

Out[121…

▾ DecisionTreeClassifier

Parameters

criterion	'gini'
splitter	'best'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	None
random_state	None
max_leaf_nodes	None
min_impurity_decrease	0.0
class_weight	None
ccp_alpha	0.0
monotonic_cst	None

In [122…

```
y_pred = model.predict(x_test_scaled)
```

In [123…

```
y_pred
```

Out[123…
```
array([8, 3, 5, 4, 7, 3, 7, 8, 5, 4, 6, 4, 3, 6, 3, 4, 6, 8, 8, 8, 7, 5,
       3, 6, 3, 8, 4, 5, 4, 3, 7, 4, 3, 5, 7, 7, 3, 4, 4, 4, 3, 3, 3, 3,
       8, 4, 8, 5, 7, 8, 4, 3, 8, 6, 4, 4, 3, 3, 3, 4, 3, 5, 4, 6, 6, 3,
       6, 7, 4, 5, 3, 5, 5, 3, 7, 6, 8, 5, 4, 5, 3, 8, 3, 4, 3, 8, 5, 4,
       7, 3, 7, 5, 5, 7, 3, 7, 4, 3, 7, 7, 6, 4, 3, 3, 4, 7, 7, 8, 4, 7,
       7, 5, 4, 6, 6, 7, 5, 7, 8, 4, 7, 4, 7, 6, 4, 4, 4, 3, 7, 4, 4, 8,
       4, 3, 8, 7, 6, 4, 8, 3, 8, 8, 4, 6, 4, 3, 6, 4, 8, 5, 4, 7, 6, 7,
       4, 7, 8, 4, 4, 6, 3, 7, 4, 4, 8, 8, 8, 5, 5, 8, 3, 5, 3, 3, 5, 4,
       8, 5, 7, 7, 8, 5, 3, 7, 5, 6, 7, 7, 8, 3, 4, 6, 3, 6, 6, 6, 8, 6,
       6, 6, 5, 7, 6, 4, 5, 3, 3, 5, 7, 6, 8, 8, 4, 8, 5, 5, 7, 4, 4, 5,
       3, 8, 8, 5, 3, 7, 8, 7, 3, 3, 3, 8, 8, 4, 3, 7, 6, 6, 7, 4, 8, 5,
       4, 6, 7, 7, 5, 8, 7, 4, 7, 5, 4, 6, 4, 6, 7, 5, 4, 5, 4, 4, 6, 6,
       8, 5, 6, 3, 7, 7, 3, 6, 3, 3, 6, 5, 7, 4, 6, 4, 7, 6, 7, 6, 8, 8,
       4, 4, 5, 5, 4, 4, 6, 5, 5, 4, 6, 3, 5, 8, 5, 5, 8, 6, 6, 8, 3, 4,
       8, 8, 7, 6, 7, 3, 6, 7, 3, 6, 5, 8, 3, 7, 6, 6, 3, 7, 3, 8, 7, 6,
       3, 5, 5, 8, 4, 8, 5, 3, 3, 3, 6, 3, 4, 5, 3, 3, 5, 5, 4, 5, 8, 6,
       7, 3, 4, 3, 7, 3, 5, 7, 6, 8, 7, 5, 5, 4, 6, 3, 7, 6, 4, 7, 8, 6,
       4, 5, 4, 5, 8, 7, 7, 8, 7, 7, 6, 7, 7, 4, 8, 8, 4, 4, 5, 5, 3, 5,
       5, 7, 4, 6, 7, 3, 8, 5, 5, 6, 6, 5, 6, 3, 4, 3, 6, 5, 4, 3, 7, 5,
       4, 7, 7, 4, 7, 6, 8, 7, 5, 7, 8, 8, 3, 6, 5, 8, 7, 8, 5, 7, 5, 4,
       8, 8, 5, 6, 6, 3, 8, 4, 5, 5, 7, 8, 5, 8, 5, 6, 3, 7, 6, 6, 7, 8,
       6, 8, 8, 8, 8, 7, 3, 5, 8, 6, 4, 5, 5, 6, 6, 4, 7, 7, 5, 6, 8, 6,
       6, 5, 4, 6, 8, 6, 8, 7, 8, 7, 3, 3, 4, 6, 8, 5, 5, 6, 4, 7, 6, 8,
       4, 8, 3, 3, 6, 3, 5, 6, 7, 8, 4, 3, 3, 6, 8, 6, 8, 5, 4, 5, 5, 4,
       4, 6, 8, 5, 8, 7, 8, 7, 6, 7, 8, 6, 8, 7, 3, 6, 8, 7, 5, 5, 8, 7,
       6, 6, 3, 7, 8, 3, 7, 5, 6, 7, 7, 7, 8, 5, 4, 7, 5, 8, 3, 4, 8, 3,
       4, 7, 3, 6, 6, 7, 3, 4], dtype=int64)
```

In [124…
```
from sklearn.metrics import classification_report,accuracy_score,precision_scc
cr =classification_report(y_test,y_pred)
acc_dt = accuracy_score(y_test,y_pred)
re= recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
pr =precision_score(y_test,y_pred,average='micro')
```

In [125…
```
print(cr)
```

```
              precision    recall  f1-score   support

           3       0.95      0.91      0.93        97
           4       0.79      0.84      0.81        92
           5       0.68      0.59      0.63       108
           6       0.54      0.54      0.54        95
           7       0.79      0.84      0.81        96
           8       0.91      0.96      0.93        92

    accuracy                           0.77       580
   macro avg       0.77      0.78      0.78       580
weighted avg       0.77      0.77      0.77       580
```

In [126…
```
re
```

Out[126… 0.7741379310344828

In [127…
```
pr
```

Out[127… 0.7741379310344828

In [128…
```
acc_dt
```

Out[128… 0.7741379310344828

In [129…
```
f1
```

Out[129… 0.7741379310344828

In [130…
```
### Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rc = RandomForestClassifier()
```

In [131…
```
rc
```

Out[131…

▾ RandomForestClassifier

Parameters

n_estimators	100
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

In [132…

```
rc.fit(x_train_scaled,y_train)
```

Out[132…

▾ RandomForestClassifier

Parameters

n_estimators	100
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

In [133…

```
y_pred = rc.predict(x_test_scaled)
```

In [134…

```
y_pred
```

Out[134…
```
array([7, 3, 5, 4, 7, 3, 7, 8, 5, 4, 6, 7, 3, 5, 3, 6, 6, 8, 8, 8, 7, 5,
       3, 6, 3, 8, 5, 3, 4, 3, 7, 4, 3, 3, 6, 7, 3, 4, 4, 4, 3, 3, 3, 4,
       8, 4, 8, 6, 7, 8, 4, 3, 7, 6, 4, 4, 3, 3, 3, 4, 3, 5, 4, 6, 5, 3,
       5, 7, 4, 7, 3, 5, 4, 3, 7, 6, 8, 6, 4, 4, 3, 8, 3, 4, 3, 8, 4, 4,
       7, 3, 7, 6, 5, 7, 3, 7, 4, 3, 6, 7, 6, 4, 3, 3, 4, 7, 7, 8, 5, 7,
       6, 5, 6, 6, 5, 8, 3, 7, 8, 4, 7, 4, 7, 4, 4, 6, 4, 3, 7, 4, 4, 8,
       6, 4, 8, 7, 5, 4, 8, 3, 8, 8, 4, 4, 4, 3, 6, 4, 8, 4, 4, 7, 6, 7,
       4, 7, 8, 4, 4, 5, 3, 7, 6, 5, 8, 8, 8, 5, 6, 7, 3, 5, 3, 3, 5, 6,
       6, 5, 7, 7, 8, 6, 3, 7, 5, 6, 7, 4, 8, 3, 6, 5, 3, 6, 6, 6, 8, 4,
       6, 6, 5, 7, 5, 4, 3, 3, 3, 5, 7, 6, 8, 8, 5, 8, 5, 4, 7, 5, 5, 5,
       3, 8, 8, 5, 3, 7, 8, 7, 4, 3, 3, 8, 8, 4, 3, 7, 7, 6, 7, 4, 8, 5,
       4, 6, 7, 7, 3, 8, 7, 4, 7, 5, 4, 5, 4, 5, 7, 5, 4, 5, 3, 5, 6, 6,
       8, 5, 6, 3, 6, 7, 3, 5, 3, 3, 6, 6, 7, 4, 7, 4, 8, 7, 7, 6, 7, 8,
       5, 4, 6, 6, 5, 4, 6, 6, 5, 4, 6, 3, 5, 8, 4, 6, 8, 5, 6, 8, 3, 4,
       8, 8, 7, 6, 7, 3, 6, 7, 3, 5, 5, 6, 3, 7, 6, 5, 3, 7, 3, 8, 7, 5,
       3, 5, 5, 8, 4, 8, 5, 3, 3, 3, 6, 3, 4, 4, 3, 3, 5, 6, 4, 5, 8, 5,
       7, 3, 6, 3, 7, 3, 5, 7, 5, 8, 7, 5, 5, 4, 7, 3, 7, 6, 4, 7, 8, 6,
       4, 5, 4, 5, 6, 8, 7, 8, 7, 7, 6, 7, 6, 4, 8, 8, 5, 4, 3, 3, 3, 5,
       6, 7, 4, 7, 7, 3, 8, 5, 4, 6, 6, 6, 5, 3, 4, 3, 6, 5, 4, 4, 7, 5,
       4, 7, 7, 4, 7, 6, 7, 7, 5, 7, 8, 8, 3, 6, 5, 8, 7, 8, 5, 7, 4, 4,
       8, 8, 5, 5, 6, 3, 8, 4, 5, 5, 7, 8, 5, 8, 5, 6, 3, 6, 6, 7, 7, 8,
       6, 8, 8, 8, 8, 7, 3, 5, 8, 6, 4, 5, 5, 6, 7, 4, 8, 7, 6, 6, 8, 6,
       5, 7, 4, 7, 8, 5, 8, 7, 8, 7, 3, 3, 4, 5, 8, 5, 6, 7, 4, 7, 7, 8,
       4, 8, 3, 3, 5, 4, 5, 5, 7, 8, 4, 3, 3, 6, 8, 5, 8, 5, 4, 3, 4, 4,
       4, 6, 8, 5, 8, 7, 8, 7, 6, 6, 8, 4, 8, 6, 3, 7, 8, 7, 5, 6, 8, 7,
       6, 6, 3, 7, 8, 3, 7, 5, 4, 7, 7, 7, 8, 5, 4, 7, 5, 8, 3, 4, 8, 3,
       4, 7, 3, 5, 6, 7, 3, 5], dtype=int64)
```

In [135…
```
from sklearn.metrics import classification_report,accuracy_score,precision_scc
cr =classification_report(y_test,y_pred)
acc_rf = accuracy_score(y_test,y_pred)
re= recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
pr =precision_score(y_test,y_pred,average='micro')
cm = confusion_matrix(y_test,y_pred)
```

In [136…
```
print(cr)
```

```
              precision    recall  f1-score   support

           3       1.00      1.00      1.00        97
           4       0.91      0.97      0.94        92
           5       0.80      0.70      0.75       108
           6       0.62      0.58      0.60        95
           7       0.81      0.92      0.86        96
           8       0.99      1.00      0.99        92

    accuracy                           0.86       580
   macro avg       0.85      0.86      0.86       580
weighted avg       0.85      0.86      0.85       580
```

In [137…
```
acc_rf
```

Out[137… 0.8568965517241379

In [138… re

Out[138… 0.8568965517241379

In [139… f1

Out[139… 0.8568965517241379

In [140… pr

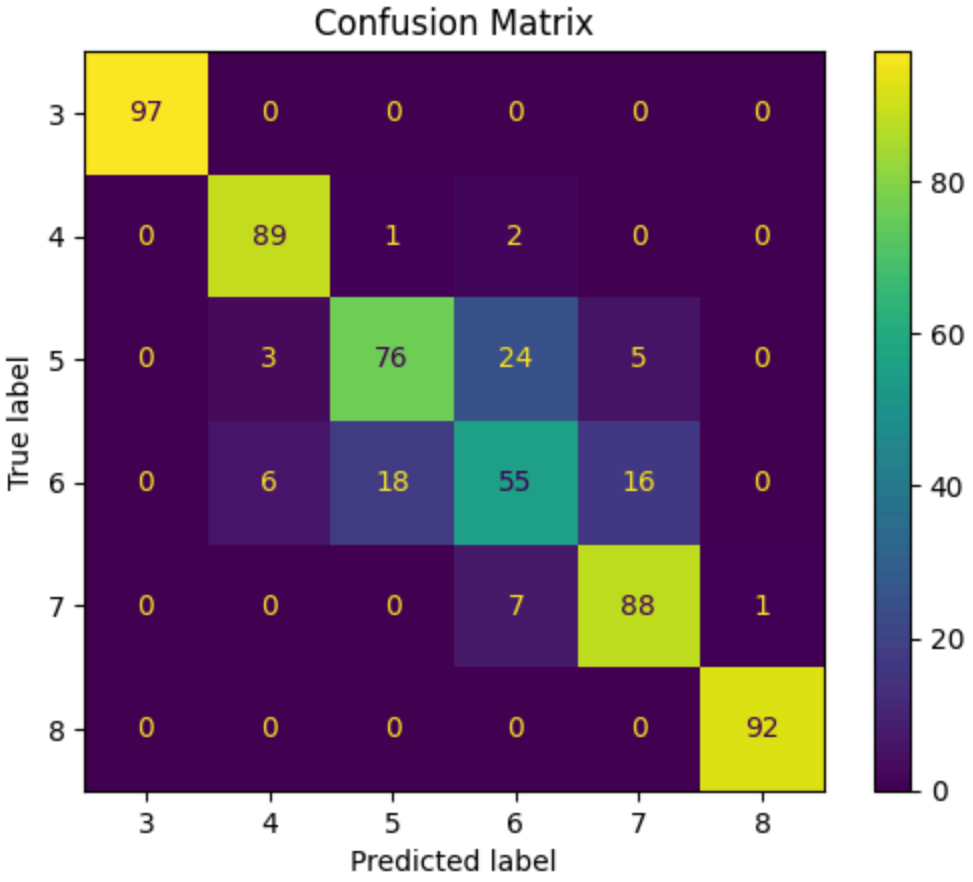
Out[140… 0.8568965517241379

In [141… print(cm)

```
[[97  0  0  0  0  0]
 [ 0 89  1  2  0  0]
 [ 0  3 76 24  5  0]
 [ 0  6 18 55 16  0]
 [ 0  0  0  7 88  1]
 [ 0  0  0  0  0 92]]
```

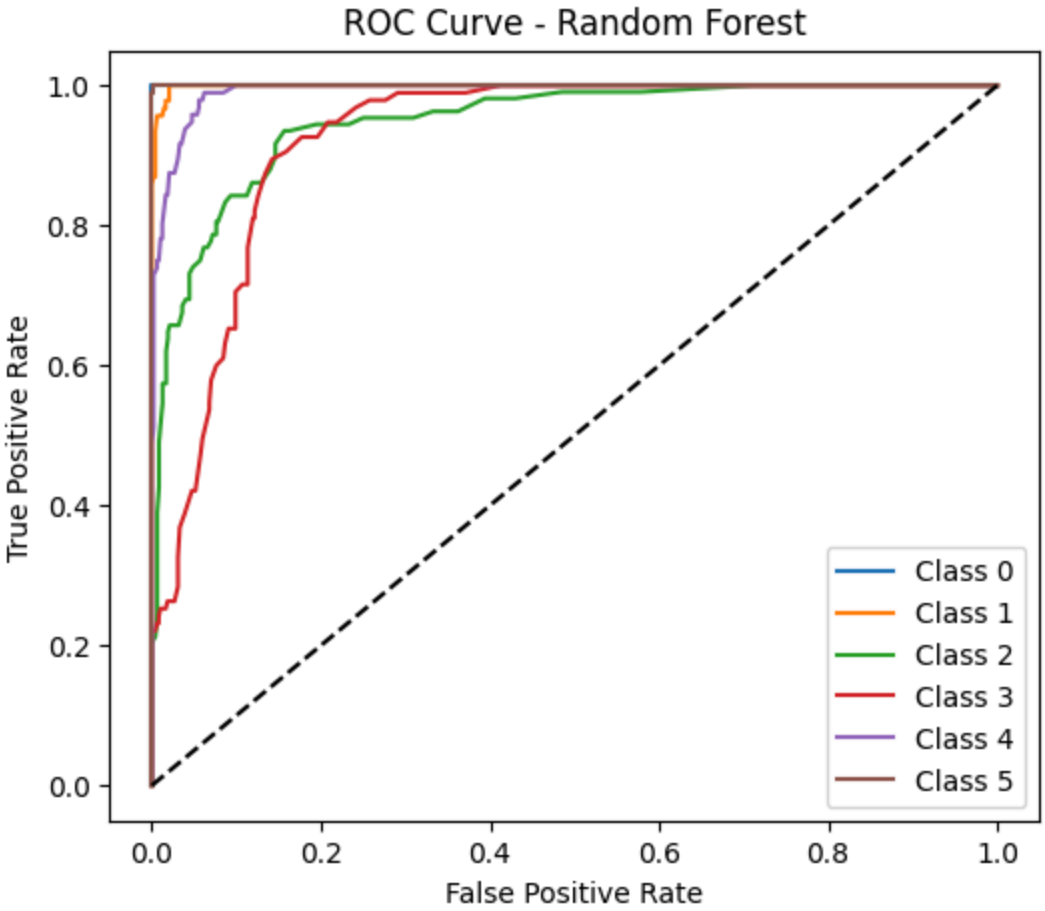
In [142…

```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classe
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```



In [143…

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

y_test_bin = label_binarize(y_test, classes=sorted(df['quality'].unique()))

y_score = rc.predict_proba(x_test_scaled)

plt.figure(figsize=(6,5))
for i in range(y_test_bin.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    plt.plot(fpr, tpr, label=f"Class {i}")

plt.plot([0,1], [0,1], "k--")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC - Random Forest")
plt.legend()
plt.show()
```



In [145…

```
plt.figure(figsize=(6,5))
plt.bar(["Decision Tree", "Random Forest"], [acc_dt, acc_rf], color=["blue", "
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison")
plt.show()
```


Model Accuracy Comparison
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Decision Tree
Random Forest

In [ ]:

Assignment No.4

Develop a Support Vector Machine (SVM) classification model to predict whether a tumor is

malignant or benign based on various features derived from digitized images of fine needle

aspirate (FNA) of breast mass. Dataset: Breast Cancer Wisconsin (Diagnostic) Dataset (UCI

Machine Learning Repository).

Name: Wavhal Prathmesh Navnath

Roll No:23107137

Class : TY-B

In [16]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
import seaborn as sns
```

In [18]:
```
df = pd.read_csv("C:/Users/prath/Downloads/data.csv")
```

In [20]:
```
df
```

Out[20]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_m
0	842302	M	17.99	10.38	122.80	10
1	842517	M	20.57	17.77	132.90	13
2	84300903	M	19.69	21.25	130.00	12
3	84348301	M	11.42	20.38	77.58	3
4	84358402	M	20.29	14.34	135.10	12
...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	14
565	926682	M	20.13	28.25	131.20	12
566	926954	M	16.60	28.08	108.30	8
567	927241	M	20.60	29.33	140.10	12
568	92751	B	7.76	24.54	47.92	1

569 rows × 33 columns

In [22]:
```
df.isnull().sum()
```

Out[22]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

In [24]:
```
df = df.drop("Unnamed: 32",axis=1)
```

In [26]:
```
df
```

Out[26]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_m
0	842302	M	17.99	10.38	122.80	10
1	842517	M	20.57	17.77	132.90	13
2	84300903	M	19.69	21.25	130.00	12
3	84348301	M	11.42	20.38	77.58	3
4	84358402	M	20.29	14.34	135.10	12
...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	14
565	926682	M	20.13	28.25	131.20	12
566	926954	M	16.60	28.08	108.30	8
567	927241	M	20.60	29.33	140.10	12
568	92751	B	7.76	24.54	47.92	1

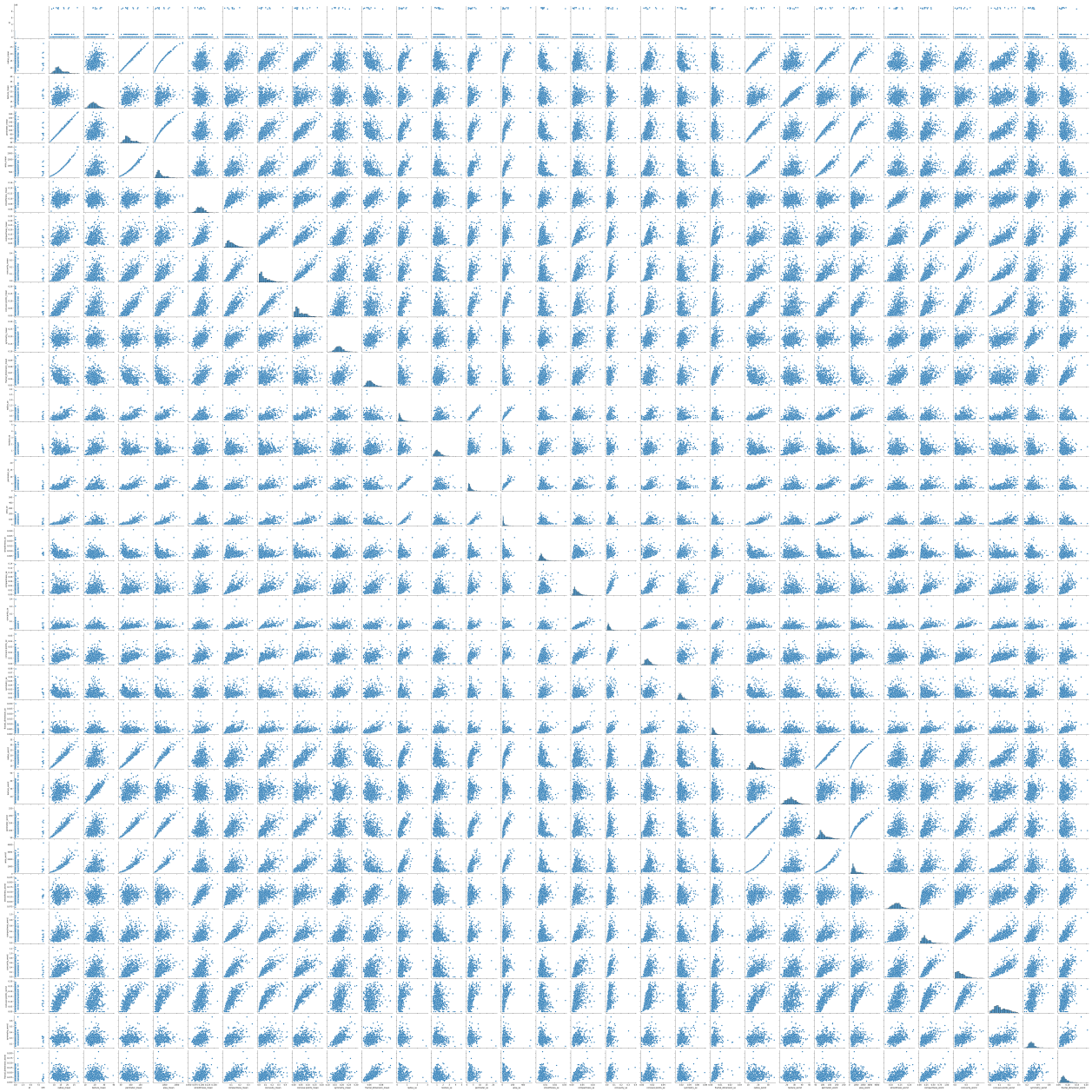
569 rows × 32 columns

In [28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

In [30]:
```
sns.pairplot(df)
```

Out[30]:
```
<seaborn.axisgrid.PairGrid at 0x1f276c5ed10>
```



In [32]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [34]:
```
df['diagnosis'] = le.fit_transform(df['diagnosis'])
```

In [36]:
```
df
```

Out[36]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_m
0	842302	1	17.99	10.38	122.80	10
1	842517	1	20.57	17.77	132.90	13
2	84300903	1	19.69	21.25	130.00	12
3	84348301	1	11.42	20.38	77.58	3
4	84358402	1	20.29	14.34	135.10	12
...	...	...	...	...	...	
564	926424	1	21.56	22.39	142.00	14
565	926682	1	20.13	28.25	131.20	12
566	926954	1	16.60	28.08	108.30	8
567	927241	1	20.60	29.33	140.10	12
568	92751	0	7.76	24.54	47.92	1

569 rows × 32 columns

In [38]:
```
df['diagnosis'].value_counts()
```

Out[38]:
```
diagnosis
0    357
1    212
Name: count, dtype: int64
```

In [40]:
```
x = df.drop("diagnosis",axis=1)
y = df['diagnosis']
```

In [41]:
```
x
```

Out[41]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoot
0	842302	17.99	10.38	122.80	1001.0	
1	842517	20.57	17.77	132.90	1326.0	
2	84300903	19.69	21.25	130.00	1203.0	
3	84348301	11.42	20.38	77.58	386.1	
4	84358402	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	
564	926424	21.56	22.39	142.00	1479.0	
565	926682	20.13	28.25	131.20	1261.0	
566	926954	16.60	28.08	108.30	858.1	
567	927241	20.60	29.33	140.10	1265.0	
568	92751	7.76	24.54	47.92	181.0	

569 rows × 31 columns

In [42]:
```
y
```

Out[42]:
```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int32
```

In [43]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_sta
```

In [44]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [45]:
```
x_train = sc.fit_transform(x_train)
```

In [46]:
```
x_test = sc.transform(x_test)
```

In [47]:
```
from sklearn.svm import SVC
model = SVC(probability=True)
```

In [48]:
```
model
```

Out[48]:

SVC

Parameters

C	1.0
kernel	'rbf'
degree	3
gamma	'scale'
coef0	0.0
shrinking	True
probability	True
tol	0.001
cache_size	200
class_weight	None
verbose	False
max_iter	-1
decision_function_shape	'ovr'
break_ties	False
random_state	None

In [49]:
```
model.fit(x_train,y_train)
```

Out[49]:

SVC

Parameters

C	1.0
kernel	'rbf'
degree	3
gamma	'scale'
coef0	0.0
shrinking	True
probability	True
tol	0.001
cache_size	200
class_weight	None
verbose	False
max_iter	-1
decision_function_shape	'ovr'
break_ties	False
random_state	None

In [50]:
```
y_pred = model.predict(x_test)
```

In [51]:
```
y_pred
```

Out[51]:
```
array([0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1])
```

In [52]:
```
from sklearn.metrics import classification_report,accuracy_score,precision_scc
cr =classification_report(y_test,y_pred)
acc_svc = accuracy_score(y_test,y_pred)
re= recall_score(y_test,y_pred)
f1 = f1_score(y_test,y_pred)
pr =precision_score(y_test,y_pred)
cm = confusion_matrix(y_test,y_pred)
```

In [53]:
```
print(cr)
```

```
                precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       1.00      0.95      0.98        43

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

In [55]:
```
acc_svc
```

Out[55]:
```
0.9824561403508771
```

In [56]:
```
re
```

Out[56]:
```
0.9534883720930233
```

In [57]:
```
f1
```

Out[57]:
```
0.9761904761904762
```

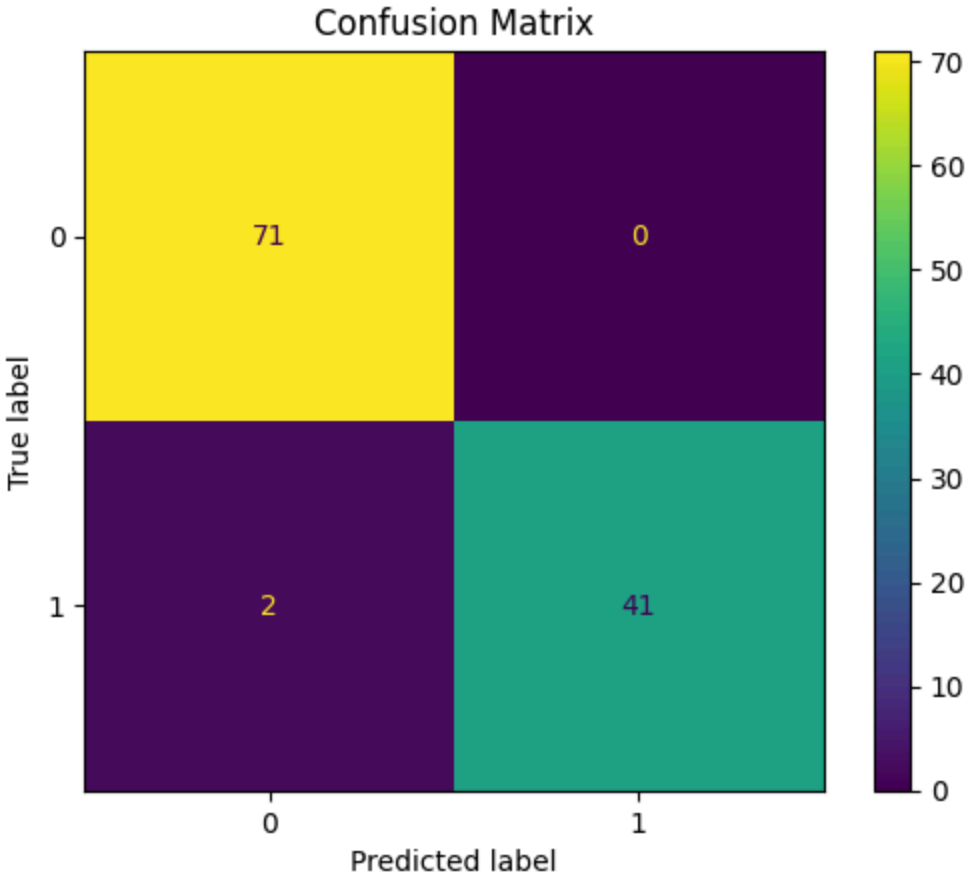
In [58]:
```
pr
```

Out[58]:
```
1.0
```

In [59]:
```
cm
```

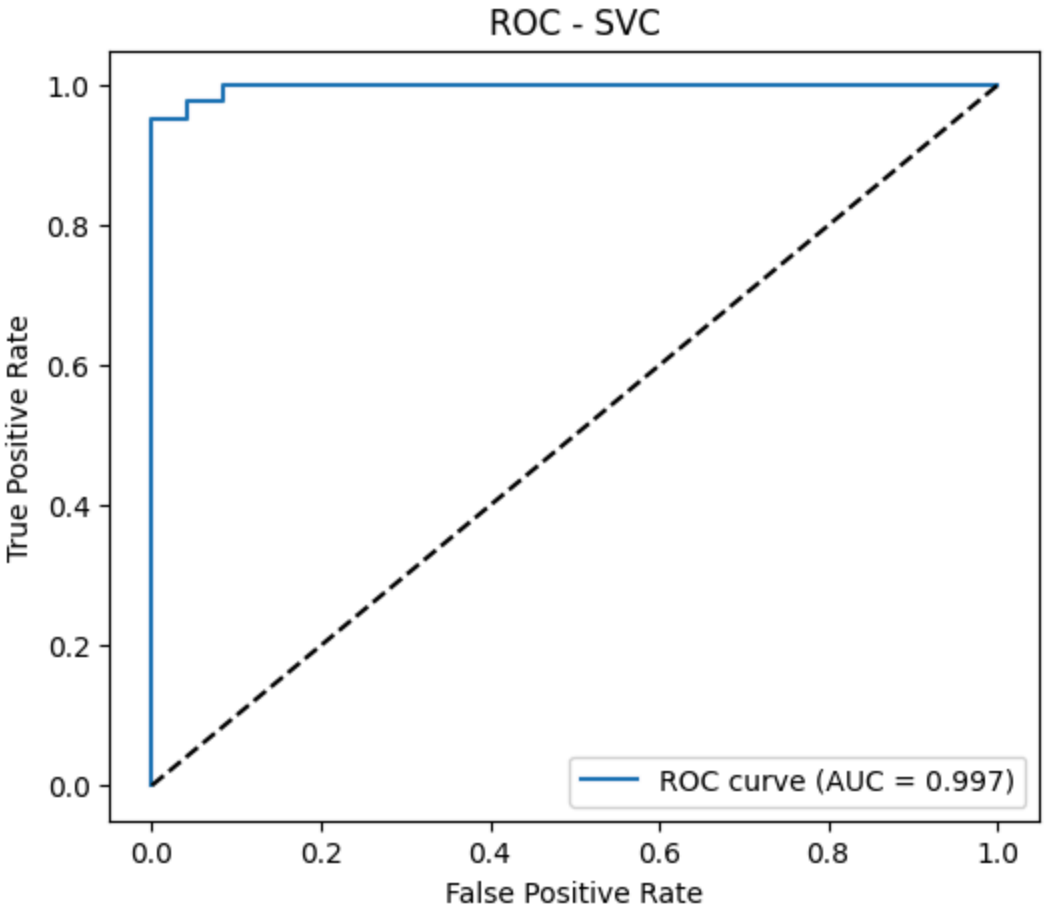
Out[59]:
```
array([[71,  0],
       [ 2, 41]], dtype=int64)
```

In [60]:
```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classe
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```



In [61]:

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get probability scores for the positive class (benign = 1 in sklearn's datas
y_score = model.predict_proba(x_test)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.3f})")
plt.plot([0,1], [0,1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC - SVC")
plt.legend(loc="lower right")
plt.show()
```



Hyperparameter Tunning

In [62]:
```
param_grid = {'C':[1,10],
              'kernel':['rbf','linear'],
              'gamma':['scale']
             }
```

In [63]:
```
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
```

In [64]:
```
grid.fit(x_train, y_train)
```

Out[64]:

▸ **GridSearchCV**

▸ **best_estimator_: SVC**

▸ SVC

In [65]:
```
grid.best_estimator_
```

Out[65]:

SVC

Parameters

C	10
kernel	'rbf'
degree	3
gamma	'scale'
coef0	0.0
shrinking	True
probability	True
tol	0.001
cache_size	200
class_weight	None
verbose	False
max_iter	-1
decision_function_shape	'ovr'
break_ties	False
random_state	None

In [66]:
```
best = grid.best_estimator_
```

In [67]:
```
y_pred1 = best.predict(x_test)
```

In [68]:
```
 grid.best_score_
```

Out[68]:
```
0.9758241758241759
```

In [69]:
```
from sklearn.metrics import classification_report,accuracy_score,precision_scc
cr =classification_report(y_test,y_pred1)
acc_hyp = accuracy_score(y_test,y_pred1)
re= recall_score(y_test,y_pred1)
f1 = f1_score(y_test,y_pred1)
pr =precision_score(y_test,y_pred1)
cm = confusion_matrix(y_test,y_pred1)
```

In [70]:
```
print(cr)
```

```
                   precision    recall  f1-score   support

                0       0.97      0.99      0.98        71
                1       0.98      0.95      0.96        43

         accuracy                           0.97       114
        macro avg       0.97      0.97      0.97       114
     weighted avg       0.97      0.97      0.97       114
```

In [71]:
```
acc_hyp
```

Out[71]:
```
0.9736842105263158
```

In [72]:
```
re
```

Out[72]:
```
0.9534883720930233
```

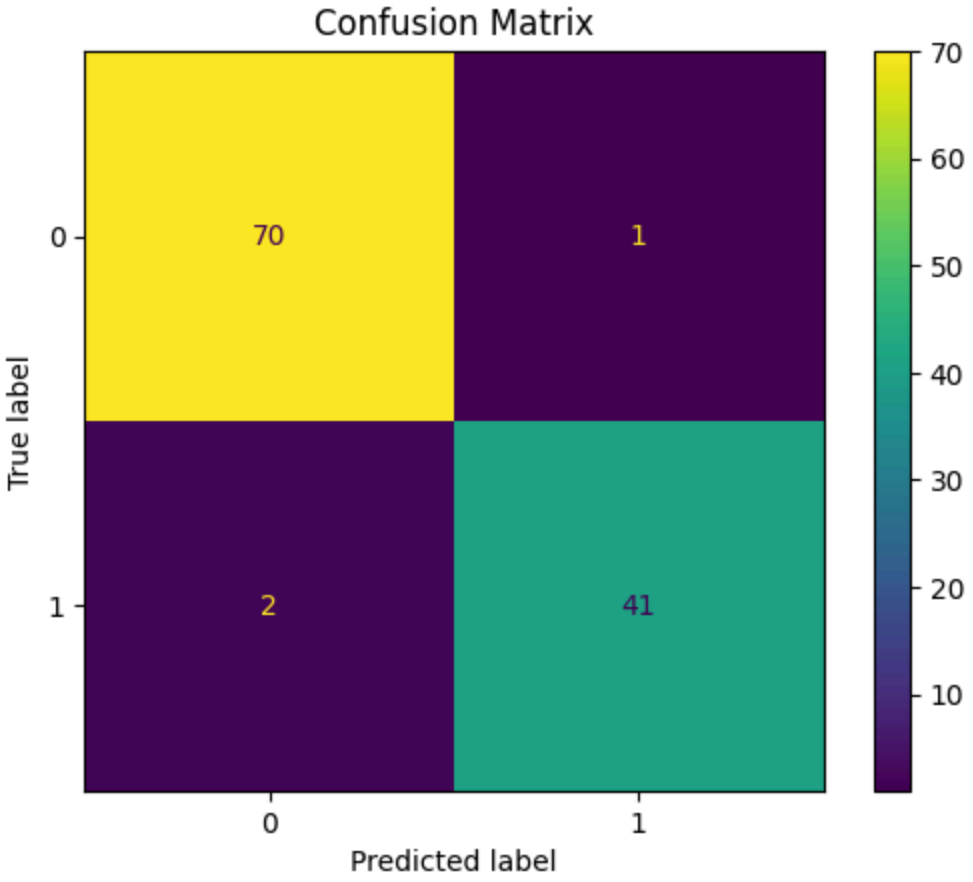
In [73]:
```
f1
```

Out[73]:
```
0.9647058823529412
```

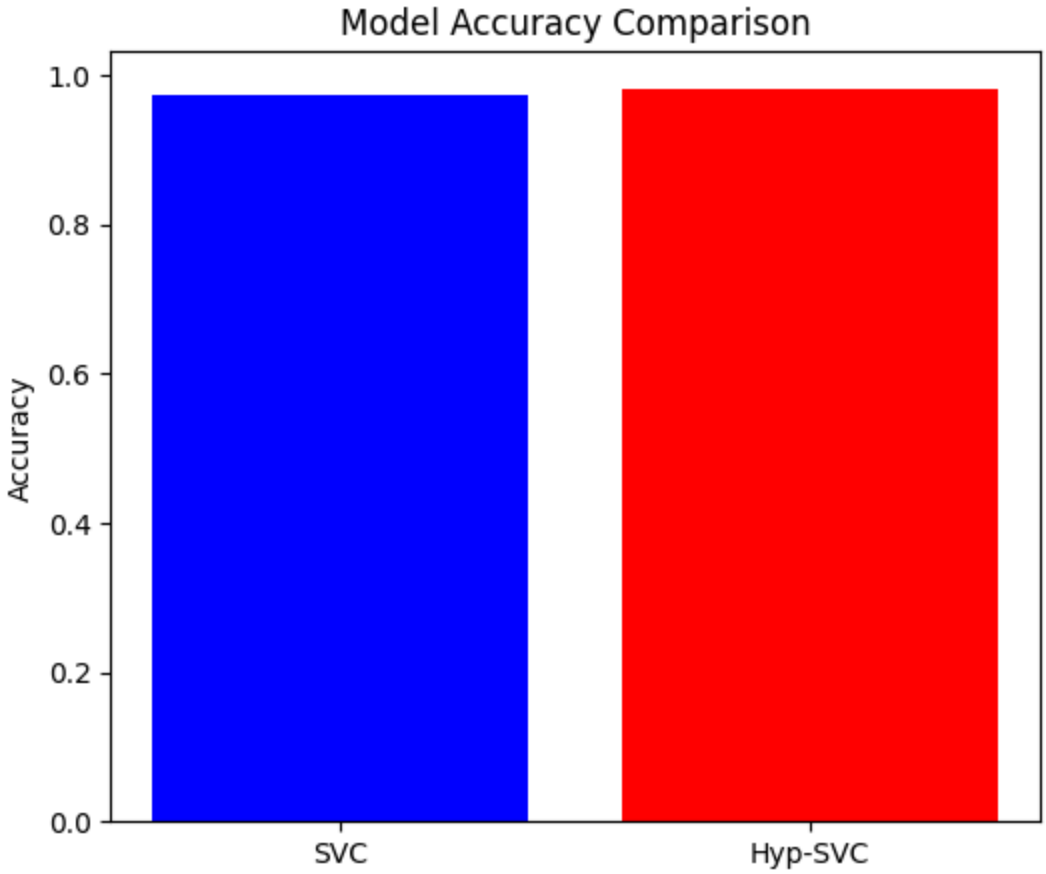
In [74]:
```
cm
```

Out[74]:
```
array([[70,  1],
       [ 2, 41]], dtype=int64)
```

In [76]:
```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best.classes
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```



In [75]:
```
plt.figure(figsize=(6,5))
plt.bar(["SVC", "Hyp-SVC"], [acc_hyp, acc_svc], color=["blue", "red"])
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison")
plt.show()
```


Model Accuracy Comparison
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
SVC
Hyp-SVC

In [ ]:

In [ ]:

In [ ]:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
import seaborn as sns
```

In [2]:
```
df = pd.read_csv("C:/Users/prath/Downloads/covtype.csv")
```

In [3]:
```
df
```

Out[3]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Di
0	2596	51	3	258	
1	2590	56	2	212	
2	2804	139	9	268	
3	2785	155	18	242	
4	2595	45	2	153	
...	...	...	...	...	
581007	2396	153	20	85	
581008	2391	152	19	67	
581009	2386	159	17	60	
581010	2384	170	15	60	
581011	2383	165	13	60	

581012 rows × 55 columns

In [4]:
```
df.isnull().sum()
```

Out[4]:

Elevation	0
Aspect	0
Slope	0
Horizontal_Distance_To_Hydrology	0
Vertical_Distance_To_Hydrology	0
Horizontal_Distance_To_Roadways	0
Hillshade_9am	0
Hillshade_Noon	0
Hillshade_3pm	0
Horizontal_Distance_To_Fire_Points	0
Wilderness_Area1	0
Wilderness_Area2	0
Wilderness_Area3	0
Wilderness_Area4	0
Soil_Type1	0
Soil_Type2	0
Soil_Type3	0
Soil_Type4	0
Soil_Type5	0
Soil_Type6	0
Soil_Type7	0
Soil_Type8	0
Soil_Type9	0
Soil_Type10	0
Soil_Type11	0
Soil_Type12	0
Soil_Type13	0
Soil_Type14	0
Soil_Type15	0
Soil_Type16	0
Soil_Type17	0
Soil_Type18	0
Soil_Type19	0
Soil_Type20	0
Soil_Type21	0
Soil_Type22	0
Soil_Type23	0
Soil_Type24	0
Soil_Type25	0
Soil_Type26	0
Soil_Type27	0
Soil_Type28	0
Soil_Type29	0
Soil_Type30	0
Soil_Type31	0
Soil_Type32	0
Soil_Type33	0
Soil_Type34	0
Soil_Type35	0
Soil_Type36	0
Soil_Type37	0
Soil_Type38	0
Soil_Type39	0
Soil_Type40	0

```
Cover_Type                          0
dtype: int64
```

In [6]:
```
df['Cover_Type'].value_counts()
```

Out[6]:
```
Cover_Type
2    283301
1    211840
3     35754
7     20510
6     17367
5      9493
4      2747
Name: count, dtype: int64
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 581012 entries, 0 to 581011
Data columns (total 55 columns):
 #   Column                              Non-Null Count   Dtype
---  ------                              --------------   -----
 0   Elevation                           581012 non-null  int64
 1   Aspect                              581012 non-null  int64
 2   Slope                               581012 non-null  int64
 3   Horizontal_Distance_To_Hydrology    581012 non-null  int64
 4   Vertical_Distance_To_Hydrology      581012 non-null  int64
 5   Horizontal_Distance_To_Roadways     581012 non-null  int64
 6   Hillshade_9am                       581012 non-null  int64
 7   Hillshade_Noon                      581012 non-null  int64
 8   Hillshade_3pm                       581012 non-null  int64
 9   Horizontal_Distance_To_Fire_Points  581012 non-null  int64
 10  Wilderness_Area1                    581012 non-null  int64
 11  Wilderness_Area2                    581012 non-null  int64
 12  Wilderness_Area3                    581012 non-null  int64
 13  Wilderness_Area4                    581012 non-null  int64
 14  Soil_Type1                          581012 non-null  int64
 15  Soil_Type2                          581012 non-null  int64
 16  Soil_Type3                          581012 non-null  int64
 17  Soil_Type4                          581012 non-null  int64
 18  Soil_Type5                          581012 non-null  int64
 19  Soil_Type6                          581012 non-null  int64
 20  Soil_Type7                          581012 non-null  int64
 21  Soil_Type8                          581012 non-null  int64
 22  Soil_Type9                          581012 non-null  int64
 23  Soil_Type10                         581012 non-null  int64
 24  Soil_Type11                         581012 non-null  int64
 25  Soil_Type12                         581012 non-null  int64
 26  Soil_Type13                         581012 non-null  int64
 27  Soil_Type14                         581012 non-null  int64
 28  Soil_Type15                         581012 non-null  int64
 29  Soil_Type16                         581012 non-null  int64
 30  Soil_Type17                         581012 non-null  int64
 31  Soil_Type18                         581012 non-null  int64
 32  Soil_Type19                         581012 non-null  int64
 33  Soil_Type20                         581012 non-null  int64
 34  Soil_Type21                         581012 non-null  int64
 35  Soil_Type22                         581012 non-null  int64
 36  Soil_Type23                         581012 non-null  int64
 37  Soil_Type24                         581012 non-null  int64
 38  Soil_Type25                         581012 non-null  int64
 39  Soil_Type26                         581012 non-null  int64
 40  Soil_Type27                         581012 non-null  int64
 41  Soil_Type28                         581012 non-null  int64
 42  Soil_Type29                         581012 non-null  int64
 43  Soil_Type30                         581012 non-null  int64
 44  Soil_Type31                         581012 non-null  int64
 45  Soil_Type32                         581012 non-null  int64
 46  Soil_Type33                         581012 non-null  int64
 47  Soil_Type34                         581012 non-null  int64
 48  Soil_Type35                         581012 non-null  int64
```

```
 49  Soil_Type36                             581012 non-null  int64
 50  Soil_Type37                             581012 non-null  int64
 51  Soil_Type38                             581012 non-null  int64
 52  Soil_Type39                             581012 non-null  int64
 53  Soil_Type40                             581012 non-null  int64
 54  Cover_Type                              581012 non-null  int64
dtypes: int64(55)
memory usage: 243.8 MB
```

In [10]:
```
x = df.drop("Cover_Type",axis=1)
y = df["Cover_Type"]
```

In [11]:
```
x
```

Out[11]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Di
0	2596	51	3	258	
1	2590	56	2	212	
2	2804	139	9	268	
3	2785	155	18	242	
4	2595	45	2	153	
...	...	...	...	...	
581007	2396	153	20	85	
581008	2391	152	19	67	
581009	2386	159	17	60	
581010	2384	170	15	60	
581011	2383	165	13	60	

581012 rows × 54 columns

In [12]:
```
y
```

Out[12]:
```
0         5
1         5
2         2
3         2
4         5
         ..
581007    3
581008    3
581009    3
581010    3
581011    3
Name: Cover_Type, Length: 581012, dtype: int64
```

In [13]:
```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_sta
```

In [14]:
```
x_train
```

Out[14]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Di
519924	3289	22	19	240	
318451	2963	21	18	134	
22325	3037	185	9	127	
449376	3113	203	13	190	
482753	3128	346	9	120	
...	...	...	...	...	
110268	3182	70	13	362	
259178	3172	156	29	716	
365838	3153	287	17	335	
131932	3065	348	21	124	
121958	3021	26	16	60	

464809 rows × 54 columns

In [17]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [18]:
```
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [19]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
```

In [20]:
```
model
```

Out[20]:

▾ RandomForestClassifier

Parameters

n_estimators	100
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

In [21]:

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

Out[21]:

RandomForestClassifier

Parameters

n_estimators	100
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

In [22]:
```
y_pred = model.predict(x_test)
```

In [24]:
```
from sklearn.metrics import classification_report,accuracy_score,precision_scc
cr =classification_report(y_test,y_pred)
acc_rf = accuracy_score(y_test,y_pred)
re= recall_score(y_test,y_pred,average='micro')
f1 = f1_score(y_test,y_pred,average='micro')
pr =precision_score(y_test,y_pred,average='micro')
cm = confusion_matrix(y_test,y_pred)
```

In [25]:
```
print(cr)
```

```
                precision    recall  f1-score   support

           1       0.97      0.94      0.95     42557
           2       0.95      0.97      0.96     56500
           3       0.95      0.97      0.96      7121
           4       0.93      0.85      0.88       526
           5       0.94      0.77      0.85      1995
           6       0.94      0.91      0.92      3489
           7       0.97      0.95      0.96      4015

    accuracy                           0.96    116203
   macro avg       0.95      0.91      0.93    116203
weighted avg       0.96      0.96      0.96    116203
```

In [26]:
```
acc_rf
```

Out[26]:
```
0.9555519220674165
```

In [27]:
```
re
```

Out[27]:
```
0.9555519220674165
```

In [28]:
```
f1
```

Out[28]:
```
0.9555519220674165
```

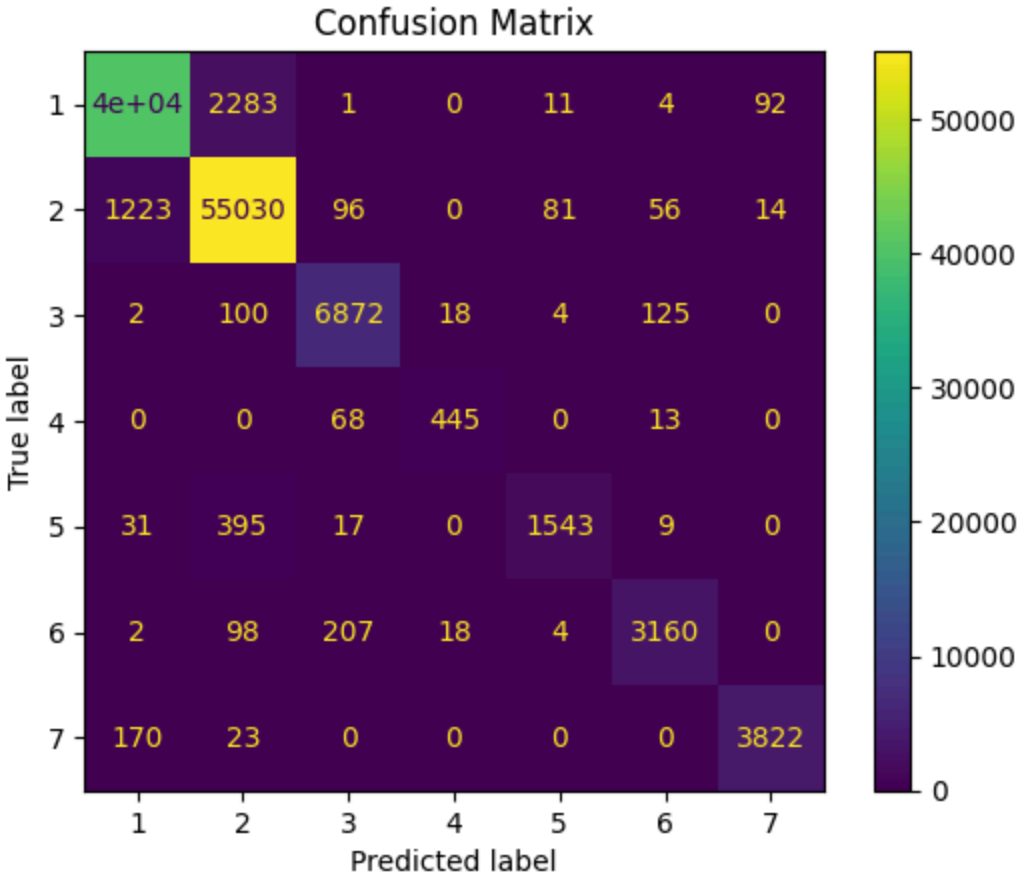
In [29]:
```
pr
```

Out[29]:
```
0.9555519220674165
```

In [30]:
```
cm
```

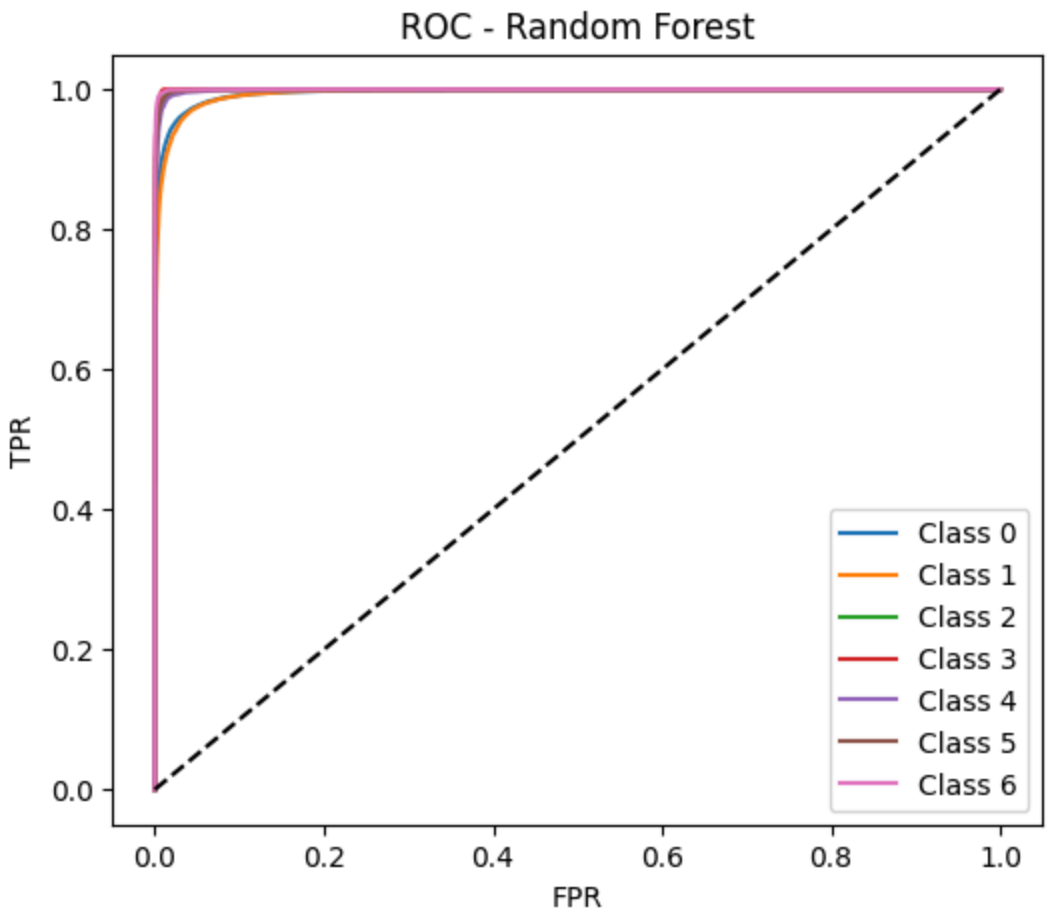
Out[30]:
```
array([[40166,  2283,     1,     0,    11,     4,    92],
       [ 1223, 55030,    96,     0,    81,    56,    14],
       [    2,   100,  6872,    18,     4,   125,     0],
       [    0,     0,    68,   445,     0,    13,     0],
       [   31,   395,    17,     0,  1543,     9,     0],
       [    2,    98,   207,    18,     4,  3160,     0],
       [  170,    23,     0,     0,     0,     0,  3822]], dtype=int64)
```

In [31]:
```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classe
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```



In [35]:

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

y_test_bin = label_binarize(y_test, classes=sorted(df['Cover_Type'].unique()))

y_score = model.predict_proba(x_test)

plt.figure(figsize=(6,5))
for i in range(y_test_bin.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    plt.plot(fpr, tpr, label=f"Class {i}")

plt.plot([0,1], [0,1], "k--")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC - Random Forest")
plt.legend()
plt.show()
```



Hyperparmeter Tunning

In [36]:
```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 'log2']
}
```

In [37]:
```
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1,ver
```

In [38]:
```
grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[38], line 1
----> 1 grid.fit(x_train, y_train)

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.p
y:1365, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args,
**kwargs)
   1358     estimator._validate_params()
   1360 with config_context(
   1361     skip_parameter_validation=(
   1362         prefer_skip_nested_validation or global_skip_validation
   1363     )
   1364 ):
-> 1365     return fit_method(estimator, *args, **kwargs)

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\mode
l_selection\_search.py:1051, in BaseSearchCV.fit(self, X, y, **params)
   1045     results = self._format_results(
   1046         all_candidate_params, n_splits, all_out, all_more_results
   1047     )
   1049     return results
-> 1051 self._run_search(evaluate_candidates)
   1053 # multimetric is determined here because in the case of a callable
   1054 # self.scoring the return type is only known after calling
   1055 first_test_score = all_out[0]["test_scores"]

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\mode
l_selection\_search.py:1605, in GridSearchCV._run_search(self, evaluate_candida
tes)
   1603 def _run_search(self, evaluate_candidates):
   1604     """Search all candidates in param_grid"""
-> 1605     evaluate_candidates(ParameterGrid(self.param_grid))

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\mode
l_selection\_search.py:997, in BaseSearchCV.fit.<locals>.evaluate_candidates(ca
ndidate_params, cv, more_results)
    989 if self.verbose > 0:
    990     print(
    991         "Fitting {0} folds for each of {1} candidates,"
    992         " totalling {2} fits".format(
    993             n_splits, n_candidates, n_candidates * n_splits
    994         )
    995     )
--> 997 out = parallel(
    998     delayed(_fit_and_score)(
    999         clone(base_estimator),
   1000         X,
   1001         y,
   1002         train=train,
   1003         test=test,
   1004         parameters=parameters,
   1005         split_progress=(split_idx, n_splits),
   1006         candidate_progress=(cand_idx, n_candidates),
```

```
   1007             **fit_and_score_kwargs,
   1008         )
   1009         for (cand_idx, parameters), (split_idx, (train, test)) in product(
   1010             enumerate(candidate_params),
   1011             enumerate(cv.split(X, y, **routed_params.splitter.split)),
   1012         )
   1013     )
   1015     if len(out) < 1:
   1016         raise ValueError(
   1017             "No fits were performed. "
   1018             "Was the CV iterator empty? "
   1019             "Were there no candidates?"
   1020         )

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\
parallel.py:82, in Parallel.__call__(self, iterable)
     73 warning_filters = warnings.filters
     74 iterable_with_config_and_warning_filters = (
     75     (
     76         _with_config_and_warning_filters(delayed_func, config, warnin
g_filters),
   (...)     80     for delayed_func, args, kwargs in iterable
     81 )
---> 82 return super().__call__(iterable_with_config_and_warning_filters)

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\joblib\paralle
l.py:2072, in Parallel.__call__(self, iterable)
   2066 # The first item from the output is blank, but it makes the interpreter
   2067 # progress until it enters the Try/Except block of the generator and
   2068 # reaches the first `yield` statement. This starts the asynchronous
   2069 # dispatch of the tasks to the workers.
   2070 next(output)
-> 2072 return output if self.return_generator else list(output)

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\joblib\paralle
l.py:1682, in Parallel._get_outputs(self, iterator, pre_dispatch)
   1679     yield
   1681     with self._backend.retrieval_context():
-> 1682         yield from self._retrieve()
   1684 except GeneratorExit:
   1685     # The generator has been garbage collected before being fully
   1686     # consumed. This aborts the remaining tasks if possible and warn
   1687     # the user if necessary.
   1688     self._exception = True

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\joblib\paralle
l.py:1800, in Parallel._retrieve(self)
   1789 if self.return_ordered:
   1790     # Case ordered: wait for completion (or error) of the next job
   1791     # that have been dispatched and not retrieved yet. If no job
   (...)   1795     # control only have to be done on the amount of time the ne
xt
   1796     # dispatched job is pending.
   1797     if (nb_jobs == 0) or (
```

```
   1798         self._jobs[0].get_status(timeout=self.timeout) == TASK_PENDING
   1799     ):
-> 1800         time.sleep(0.01)
   1801         continue
   1803 elif nb_jobs == 0:
   1804     # Case unordered: jobs are added to the list of jobs to
   1805     # retrieve `self._jobs` only once completed or in error, which
   (...)   1811     # timeouts before any other dispatched job has completed an
d
   1812     # been added to `self._jobs` to be retrieved.

KeyboardInterrupt:
```

In [ ]:
```
grid.best_score_
```

In [ ]:
```
best = grid.best_estimator_
```

In [ ]:
```
y_pred1 = best.predict(x_test)
```

In [ ]:
```
from sklearn.metrics import classification_report,accuracy_score,precision_scc
cr =classification_report(y_test,y_pred1)
acc_hyp = accuracy_score(y_test,y_pred1)
re= recall_score(y_test,y_pred1)
f1 = f1_score(y_test,y_pred1)
pr =precision_score(y_test,y_pred1)
cm = confusion_matrix(y_test,y_pred1)
```

In [ ]:
```
print(cr)
```

In [ ]:
```
acc_hyp
```

In [ ]:
```
re
```

In [ ]:
```
f1
```

In [ ]:
```
pr
```

In [ ]:
```
cm
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best.classes
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```

In [ ]:
```
plt.figure(figsize=(6,5))
plt.bar(["SVC", "Hyp-SVC"], [acc_hyp, acc_rf], color=["blue", "red"])
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison")
plt.show()
```

Assignment No.6

Utilize the K-Means clustering algorithm to segment customers based on their purchasing

behavior. Apply the Elbow Method to determine the optimal number of clusters that best

represent distinct customer segments for targeted marketing strategies. Dataset: UCI Machine

Learning Repository - Wholesale Customers Data Set

Name: Wavhal Prathmesh Navnath

Roll No:23107137

Class : TY-B

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
import seaborn as sns
```

In [2]:
```
df = pd.read_csv("C:/Users/prath/Downloads/Wholesale customers data.csv")
```

In [3]:
```
df
```

Out[3]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delica
0	2	3	12669	9656	7561	214	2674	
1	2	3	7057	9810	9568	1762	3293	
2	2	3	6353	8808	7684	2405	3516	
3	1	3	13265	1196	4221	6404	507	
4	2	3	22615	5410	7198	3915	1777	
...	...	...	...	...	...	...	...	
435	1	3	29703	12051	16027	13135	182	
436	1	3	39228	1431	764	4510	93	
437	2	3	14531	15488	30243	437	14841	
438	1	3	10290	1981	2232	1038	168	
439	1	3	2787	1698	2510	65	477	

440 rows × 8 columns

In [4]:
```
df = df.drop(columns=["Channel","Region"],axis=1)
```

In [5]:
```
df
```

Out[5]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	12669	9656	7561	214	2674	1338
1	7057	9810	9568	1762	3293	1776
2	6353	8808	7684	2405	3516	7844
3	13265	1196	4221	6404	507	1788
4	22615	5410	7198	3915	1777	5185
...	...	...	...	...	...	...
435	29703	12051	16027	13135	182	2204
436	39228	1431	764	4510	93	2346
437	14531	15488	30243	437	14841	1867
438	10290	1981	2232	1038	168	2125
439	2787	1698	2510	65	477	52

440 rows × 6 columns

In [6]:
```
## Standardization = feature scaling technique
```

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

In [7]:
```
x =df
```

Outlier Detection using IQR

In [8]:
```
Q1 = x.quantile(0.25)
Q3 = x.quantile(0.75)
IQR = Q3 - Q1

# Find outlier rows
outlier_condition = ((x< (Q1 - 1.5 * IQR)) | (x > (Q3 + 1.5 * IQR))).any(axis=
outliers = df[outlier_condition]

print(f"Number of outliers: {len(outliers)}")
print(outliers)
```

```
Number of outliers: 108
     Fresh   Milk  Grocery  Frozen  Detergents_Paper  Delicassen
2     6353   8808     7684    2405              3516        7844
4    22615   5410     7198    3915              1777        5185
17    5876   6157     2933     839               370        4478
22   31276   1917     4469    9408              2381        4334
23   26373  36423    22019    5154              4337       16523
..     ...    ...      ...     ...               ...         ...
427  31012  16687     5429   15082               439        1163
431   8533   5506     5160   13486              1377        1498
435  29703  12051    16027   13135               182        2204
436  39228   1431      764    4510                93        2346
437  14531  15488    30243     437             14841        1867

[108 rows x 6 columns]
```

In [9]:
```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
x = scaler.fit_transform(x)
```

In [10]:
```
x = scalar.fit_transform(x)
```

In [11]:
```
x
```

Out[11]:
```
array([[ 0.05293319,  0.52356777, -0.04111489, -0.58936716, -0.04356873,
        -0.06633906],
       [-0.39130197,  0.54445767,  0.17031835, -0.27013618,  0.08640684,
         0.08915105],
       [-0.44702926,  0.40853771, -0.0281571 , -0.13753572,  0.13323164,
         2.24329255],
       ...,
       [ 0.20032554,  1.31467078,  2.34838631, -0.54337975,  2.51121768,
         0.12145607],
       [-0.13538389, -0.51753572, -0.60251388, -0.41944059, -0.56977032,
         0.21304614],
       [-0.72930698, -0.5559243 , -0.57322717, -0.62009417, -0.50488752,
        -0.52286938]])
```

Elbow Method to select K-Value

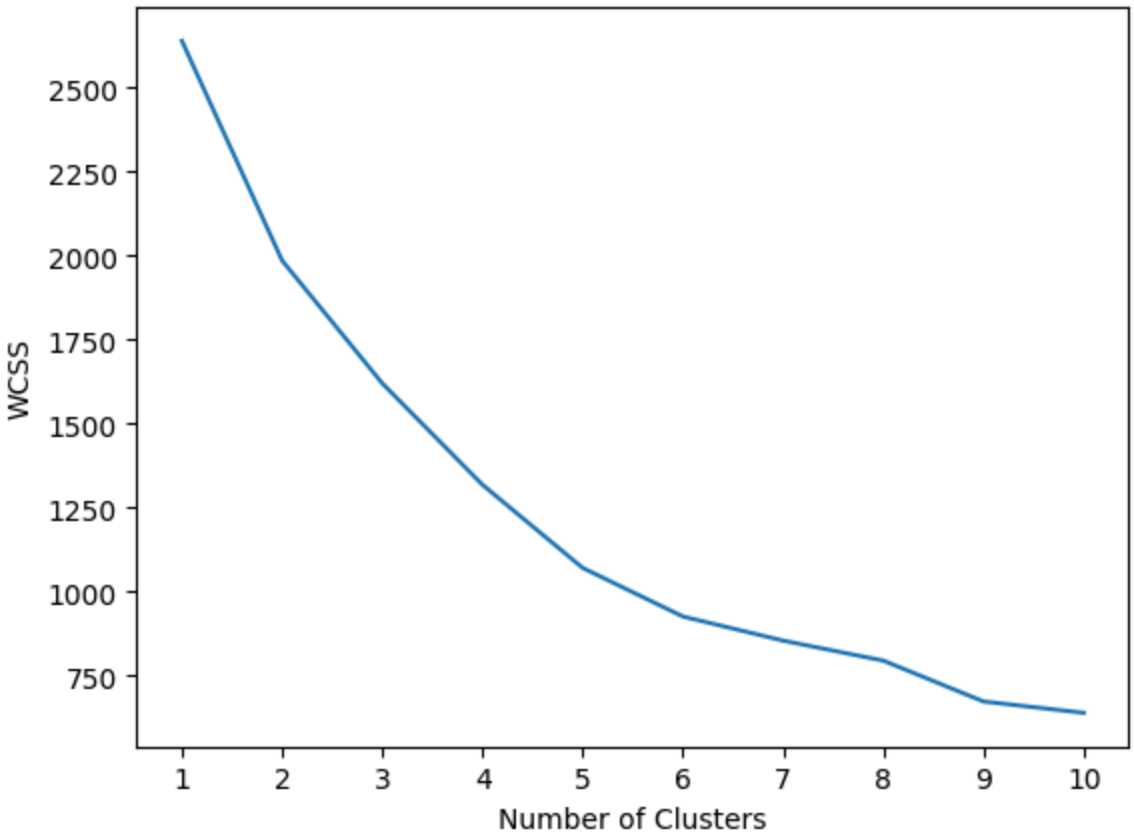
In [12]:
```
from sklearn.cluster import KMeans
```

In [13]:
```
wcss = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k,init="k-means++")
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
```

In [14]:
```
wcss
```

Out[14]:
```
[2640.0,
 1985.8310672356574,
 1619.952782172456,
 1317.8973573404485,
 1070.2551995135616,
 925.5621320487785,
 853.7550323759655,
 794.1649663000028,
 672.5116366417119,
 638.6802117739219]
```

In [15]:
```
## plot the ELBOW Curve
plt.plot(range(1,11),wcss)
plt.xticks(range(1,11))
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.show()
```



In [31]:
```
model = KMeans(n_clusters=3, random_state=42, n_init=10)
```

In [32]:
```
model.fit(x)
```

Out[32]:

KMeans

Parameters

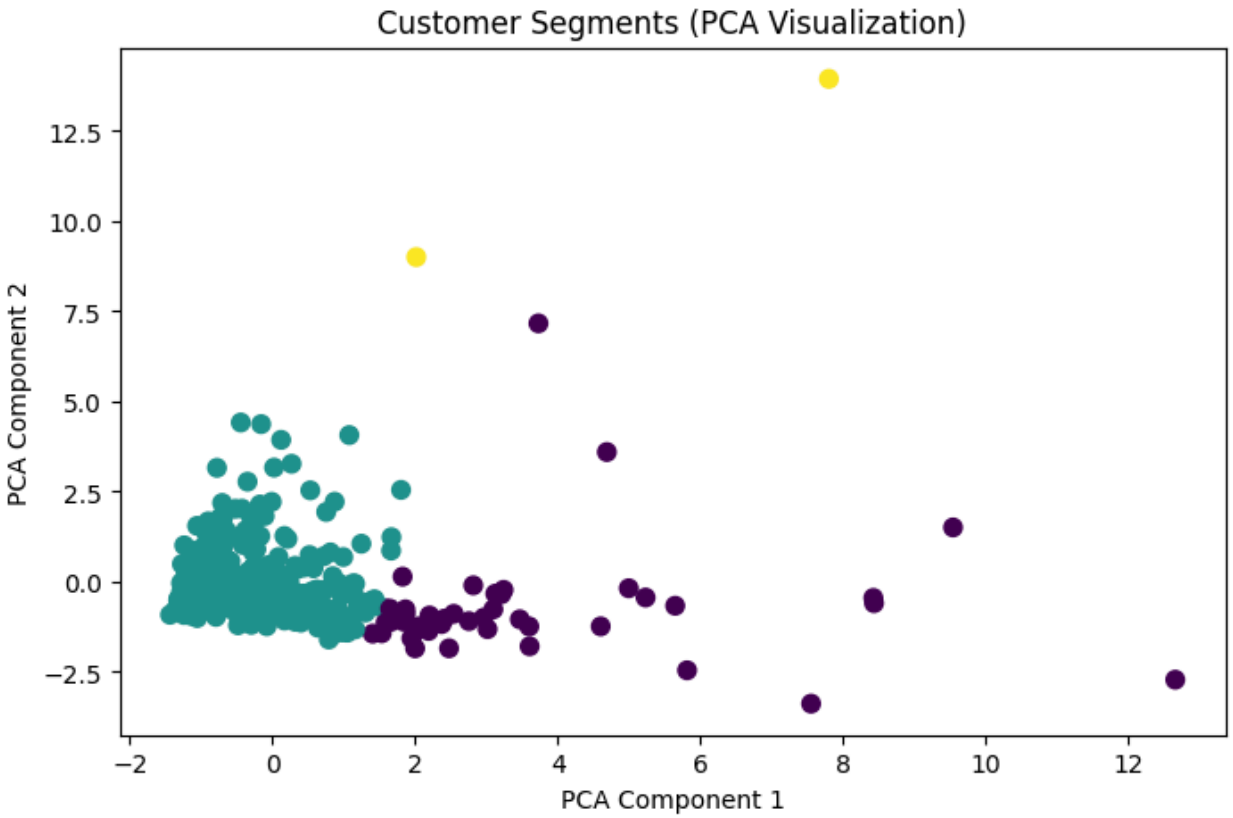
n_clusters	3
init	'k-means++'
n_init	10
max_iter	300
tol	0.0001
verbose	0
random_state	42
copy_x	True
algorithm	'lloyd'

In [33]:
```
clusters= model.fit_predict(x)
```

In [34]:
```
df['Cluster'] = clusters
```

In [36]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(x)
```

In [37]:
```
plt.figure(figsize=(8,5))
plt.scatter(X_pca[:,0], X_pca[:,1], c=clusters, cmap='viridis', s=50)
plt.title('Customer Segments (PCA Visualization)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```



Validating k-value diff Method

kneelocator

In [38]:
```
from kneed import KneeLocator
kl = KneeLocator(range(1,11),wcss,curve="convex",direction="decreasing")
```

In [39]:
```
kl.elbow
```

Out[39]: 5

Sillihoute Scoring

In [40]:
```
from sklearn.metrics import silhouette_score
```

In [41]:
```
sillhoute_coef = []
for k in range(2,11):
    kmeans = KMeans(n_clusters=k,init="k-means++")
    kmeans.fit(x)
    score = silhouette_score(x,kmeans.labels_)
    sillhoute_coef.append(score)
```

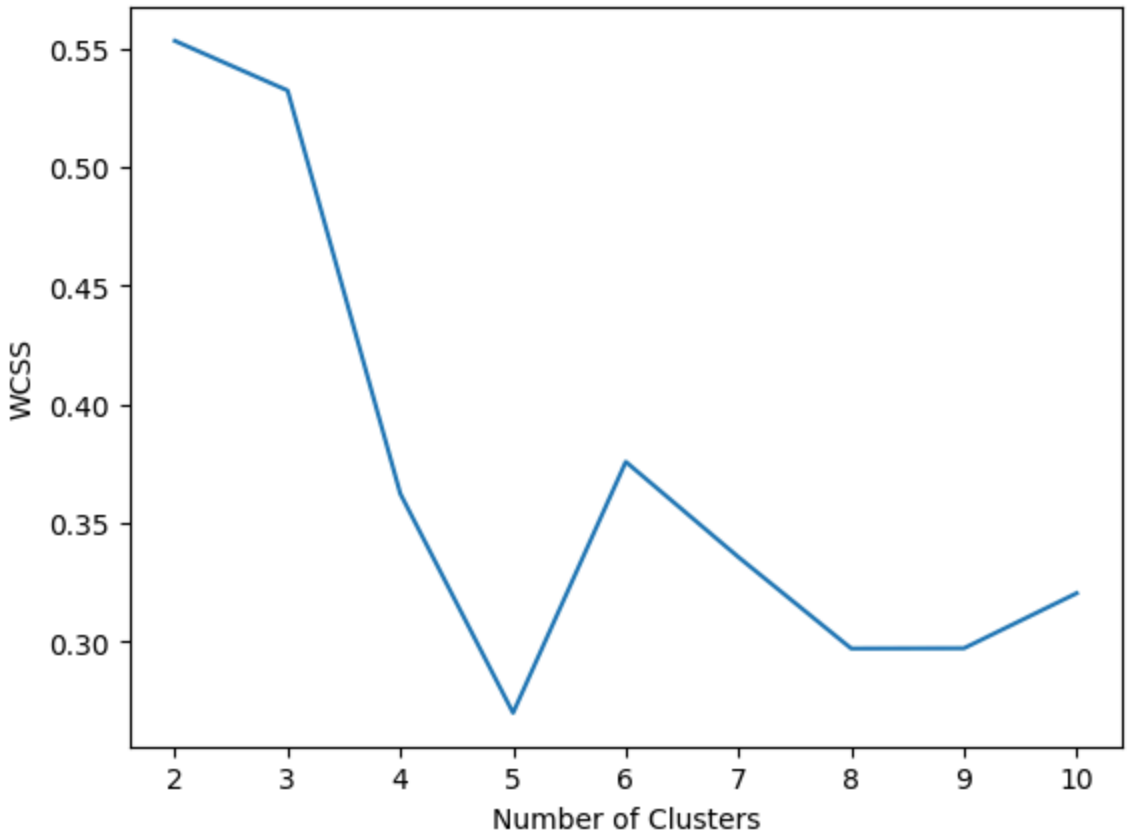
In [42]:
```
sillhoute_coef
```

Out[42]:
```
[0.5532526964184259,
 0.5323210929297174,
 0.36232371550681225,
 0.2699786894327585,
 0.3758033637245776,
 0.3356008212733988,
 0.29709019883625454,
 0.2972001998245971,
 0.32040877415020924]
```

In [43]:
```
from sklearn.metrics import silhouette_score

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    clusters = kmeans.fit_predict(x)
    score = silhouette_score(x, clusters)
    print(f"k = {k}, Silhouette Score = {score:.3f}")
```

```
k = 2, Silhouette Score = 0.400
k = 3, Silhouette Score = 0.458
k = 4, Silhouette Score = 0.349
k = 5, Silhouette Score = 0.369
k = 6, Silhouette Score = 0.276
k = 7, Silhouette Score = 0.277
k = 8, Silhouette Score = 0.324
k = 9, Silhouette Score = 0.295
k = 10, Silhouette Score = 0.229
```

In [44]:
```
## plot the ELBOW Curve
plt.plot(range(2,11),sillhoute_coef)
plt.xticks(range(2,11))
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.show()           ## choose the max value in graph        ## i.e. is 5.
```



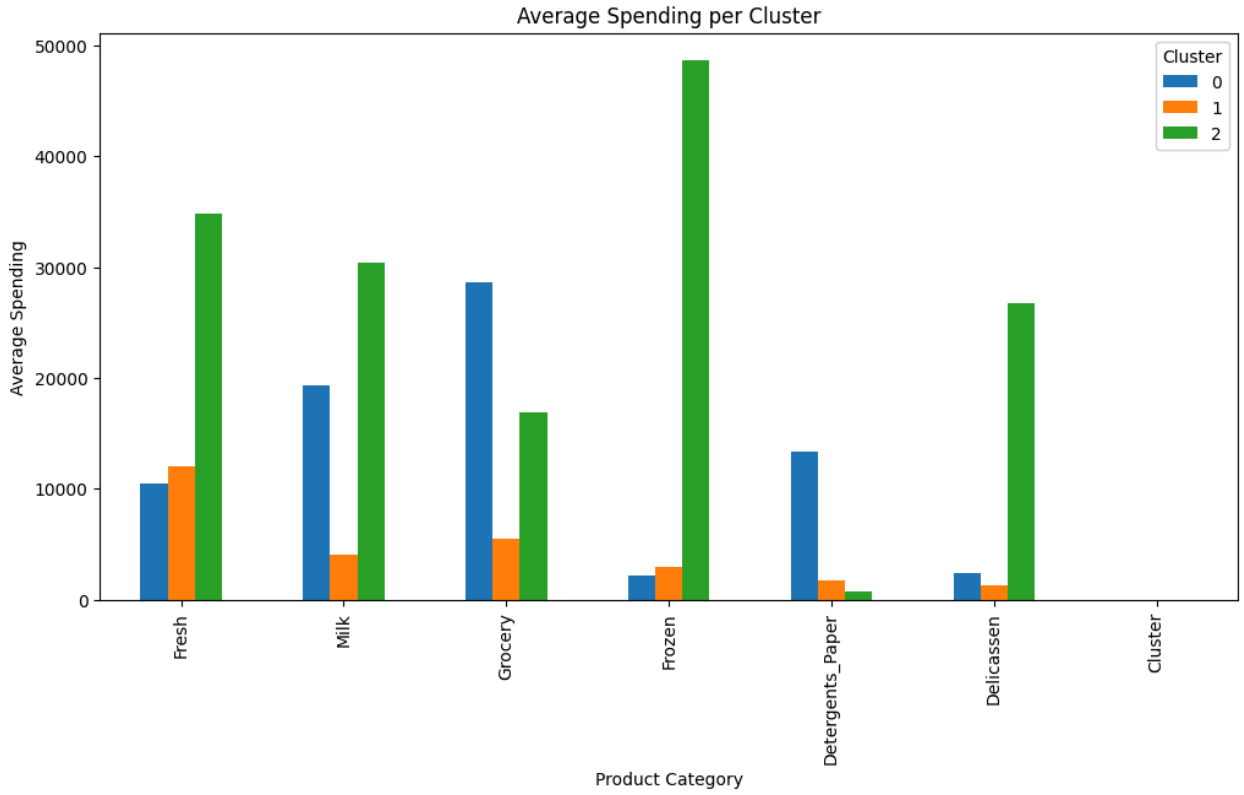
In [46]:
```
from sklearn.metrics import silhouette_score

# Calculate silhouette score
score = silhouette_score(x, clusters)
print("Silhouette Score:", score)
```

```
Silhouette Score: 0.22876866444760022
```

In [47]:
```
cluster_summary = df.groupby('Cluster')[df.columns].mean()
print(cluster_summary)
```

```
                Fresh          Milk       Grocery        Frozen  \
Cluster
0        10440.933333  19386.422222  28656.088889   2190.244444
1        12062.913486   4115.099237   5534.966921   2940.676845
2        34782.000000  30367.000000  16898.000000  48701.500000

         Detergents_Paper    Delicassen  Cluster
Cluster
0            13327.800000   2374.200000      0.0
1             1696.170483   1299.114504      1.0
2              755.500000  26776.000000      2.0
```

In [48]:
```
cluster_summary[df.columns].T.plot(kind='bar', figsize=(12,6))
plt.title("Average Spending per Cluster")
plt.xlabel("Product Category")
plt.ylabel("Average Spending")
plt.show()
```


Average Spending per Cluster
Cluster
0
1
2
Average Spending
50000
40000
30000
20000
10000
0
Fresh
Milk
Grocery
Frozen
Detergents_Paper
Delicassen
Cluster
Product Category

In [ ]:

In [ ]:

Assignment No.7

Use K-Medoids clustering to segment wholesale customers based on their annual spending on

different product categories. Identify the optimal number of clusters to effectively group

customers with similar purchasing behaviours. Dataset: Wholesale Customers Dataset (UCI).

Name: Wavhal Prathmesh Navnath

Roll No:23107137

Class : TY-B

In [24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
import seaborn as sns
```

In [25]:
```
df = pd.read_csv("C:/Users/prath/Downloads/Wholesale customers data.csv")
```

In [26]:
```
df
```

Out[26]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delica
0	2	3	12669	9656	7561	214	2674	
1	2	3	7057	9810	9568	1762	3293	
2	2	3	6353	8808	7684	2405	3516	
3	1	3	13265	1196	4221	6404	507	
4	2	3	22615	5410	7198	3915	1777	
...	...	...	...	...	...	...	...	
435	1	3	29703	12051	16027	13135	182	
436	1	3	39228	1431	764	4510	93	
437	2	3	14531	15488	30243	437	14841	
438	1	3	10290	1981	2232	1038	168	
439	1	3	2787	1698	2510	65	477	

440 rows × 8 columns

In [43]:
```
df.isnull().sum()
```

Out[43]:
```
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicassen          0
Cluster             0
dtype: int64
```

In [27]:
```
df = df.drop(columns=["Channel","Region"],axis=1)
```

In [28]:
```
df
```

Out[28]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	12669	9656	7561	214	2674	1338
1	7057	9810	9568	1762	3293	1776
2	6353	8808	7684	2405	3516	7844
3	13265	1196	4221	6404	507	1788
4	22615	5410	7198	3915	1777	5185
...	...	...	...	...	...	...
435	29703	12051	16027	13135	182	2204
436	39228	1431	764	4510	93	2346
437	14531	15488	30243	437	14841	1867
438	10290	1981	2232	1038	168	2125
439	2787	1698	2510	65	477	52

440 rows × 6 columns

In [29]:
```
x = df
```

Outlier Detection using IQR

In [30]:
```
Q1 = x.quantile(0.25)
Q3 = x.quantile(0.75)
IQR = Q3 - Q1

# Find outlier rows
outlier_condition = ((x< (Q1 - 1.5 * IQR)) | (x > (Q3 + 1.5 * IQR))).any(axis=
outliers = df[outlier_condition]

print(f"Number of outliers: {len(outliers)}")
```

```
print(outliers)
```

```
Number of outliers: 108
     Fresh   Milk  Grocery  Frozen  Detergents_Paper  Delicassen
2     6353   8808     7684    2405              3516        7844
4    22615   5410     7198    3915              1777        5185
17    5876   6157     2933     839               370        4478
22   31276   1917     4469    9408              2381        4334
23   26373  36423    22019    5154              4337       16523
..     ...    ...      ...     ...               ...         ...
427  31012  16687     5429   15082               439        1163
431   8533   5506     5160   13486              1377        1498
435  29703  12051    16027   13135               182        2204
436  39228   1431      764    4510                93        2346
437  14531  15488    30243     437             14841        1867

[108 rows x 6 columns]
```

Robust Scaler

In [31]:
```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
x = scaler.fit_transform(x)                ### it uses the Median and IQR for
```

In [32]:
```
## Standardization = feature scaling technique
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
x = scalar.fit_transform(x)
```

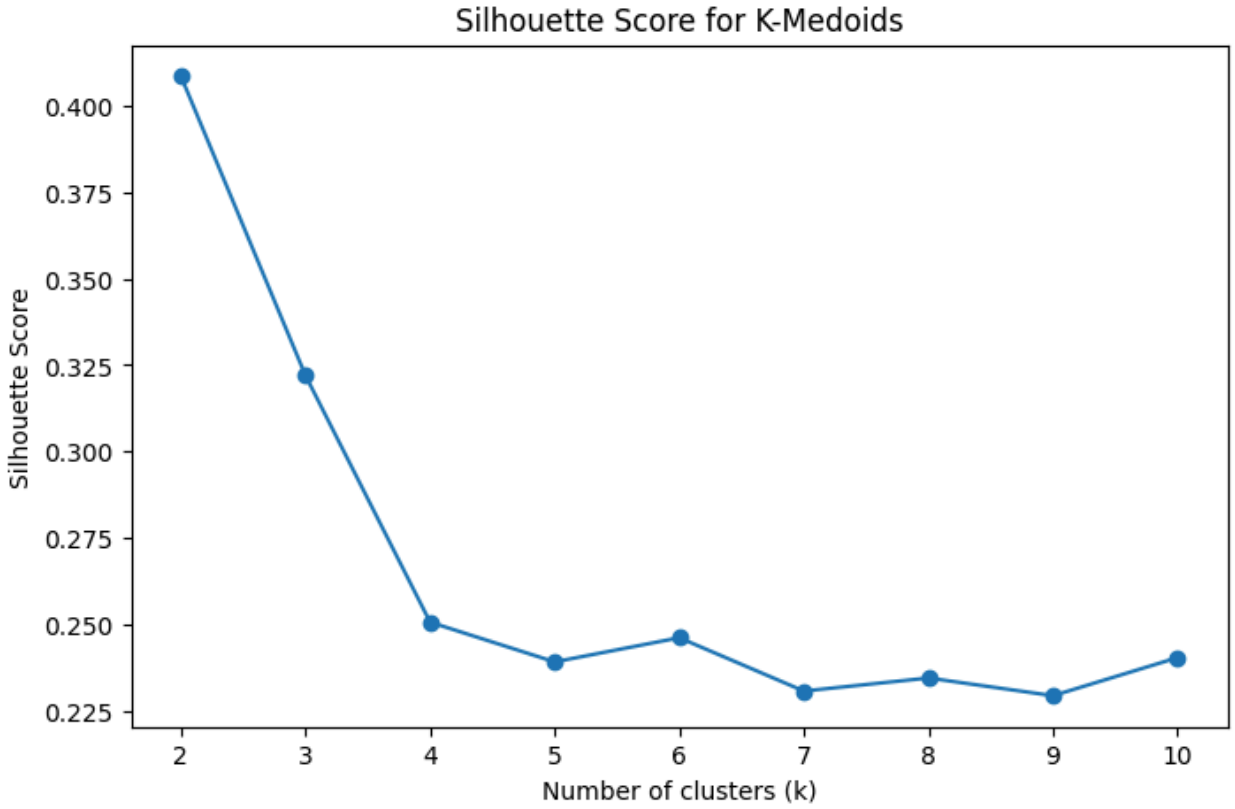
Determine optimal number of clusters using Silhouette Score

In [33]:
```
from sklearn_extra.cluster import KMedoids
from sklearn.metrics import silhouette_score
```

In [34]:
```
sil_scores = []
k_range = range(2, 11)                                                    #
for k in k_range:
    kmedoids = KMedoids(n_clusters=k, random_state=42, method='pam')
    labels = kmedoids.fit_predict(x)
    score = silhouette_score(x, labels)
    sil_scores.append(score)
    print(f"k = {k}, Silhouette Score = {score:.3f}")
```

```
k = 2, Silhouette Score = 0.408
k = 3, Silhouette Score = 0.322
k = 4, Silhouette Score = 0.251
k = 5, Silhouette Score = 0.239
k = 6, Silhouette Score = 0.246
k = 7, Silhouette Score = 0.231
k = 8, Silhouette Score = 0.235
k = 9, Silhouette Score = 0.229
k = 10, Silhouette Score = 0.240
```

In [35]:
```python
# Plot Silhouette Scores
plt.figure(figsize=(8,5))
plt.plot(k_range, sil_scores, marker='o')
plt.title("Silhouette Score for K-Medoids")
plt.xlabel("Number of clusters (k)")
plt.ylabel("Silhouette Score")
plt.show()
```



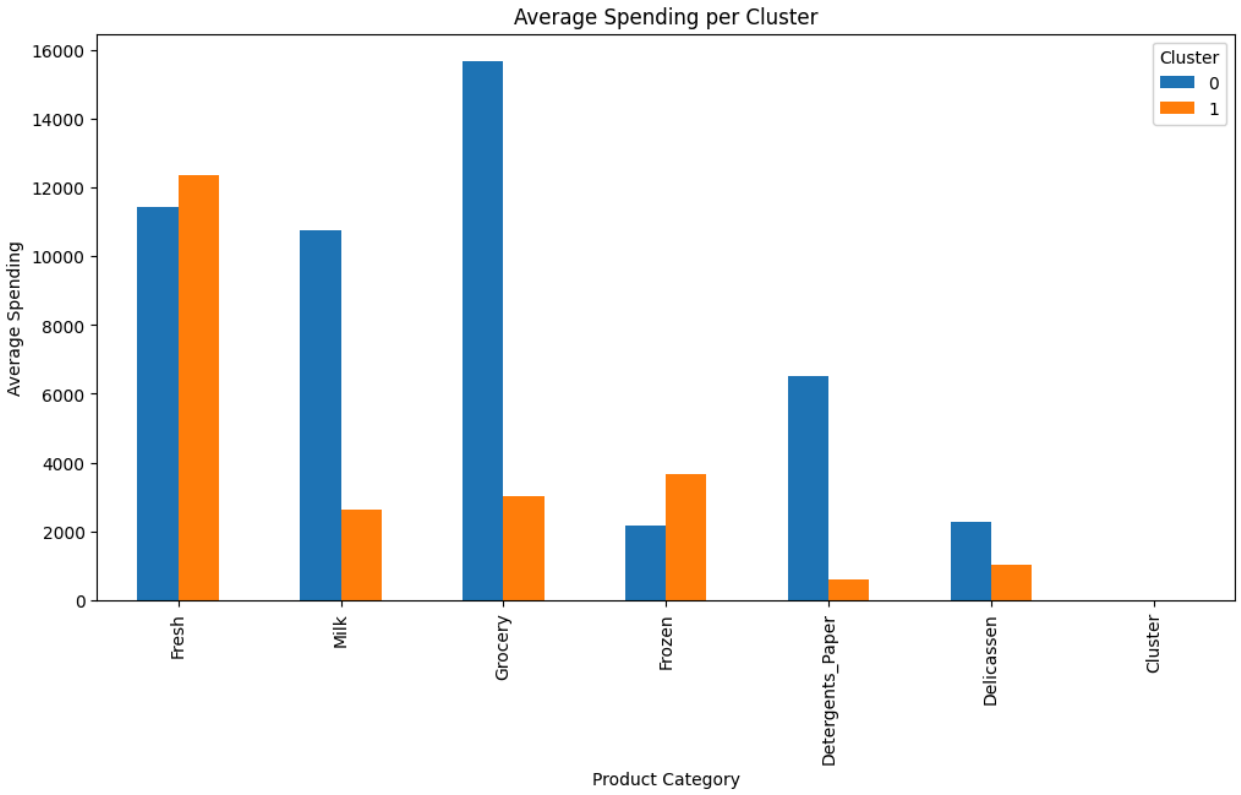
In [36]:
```python
## max value score is at K=2
kmedoids = KMedoids(n_clusters=2, random_state=42)
clusters = kmedoids.fit_predict(x)
```

In [37]:
```python
df['Cluster'] = clusters
```

In [38]:
```python
cluster_summary = df.groupby('Cluster')[df.columns].mean()
print("\nCluster Summary:\n", cluster_summary)
```

```
Cluster Summary:
                Fresh          Milk       Grocery      Frozen  \
Cluster
0        11422.467836  10762.111111  15691.064327  2153.22807
1        12367.617100   2639.539033   3031.189591  3655.94052

         Detergents_Paper   Delicassen  Cluster
Cluster
0             6502.637427  2285.269006      0.0
1              579.576208  1041.494424      1.0
```

In [39]:
```
cluster_summary[df.columns].T.plot(kind='bar', figsize=(12,6))
plt.title("Average Spending per Cluster")
plt.xlabel("Product Category")
plt.ylabel("Average Spending")
plt.show()
```



Evaluation

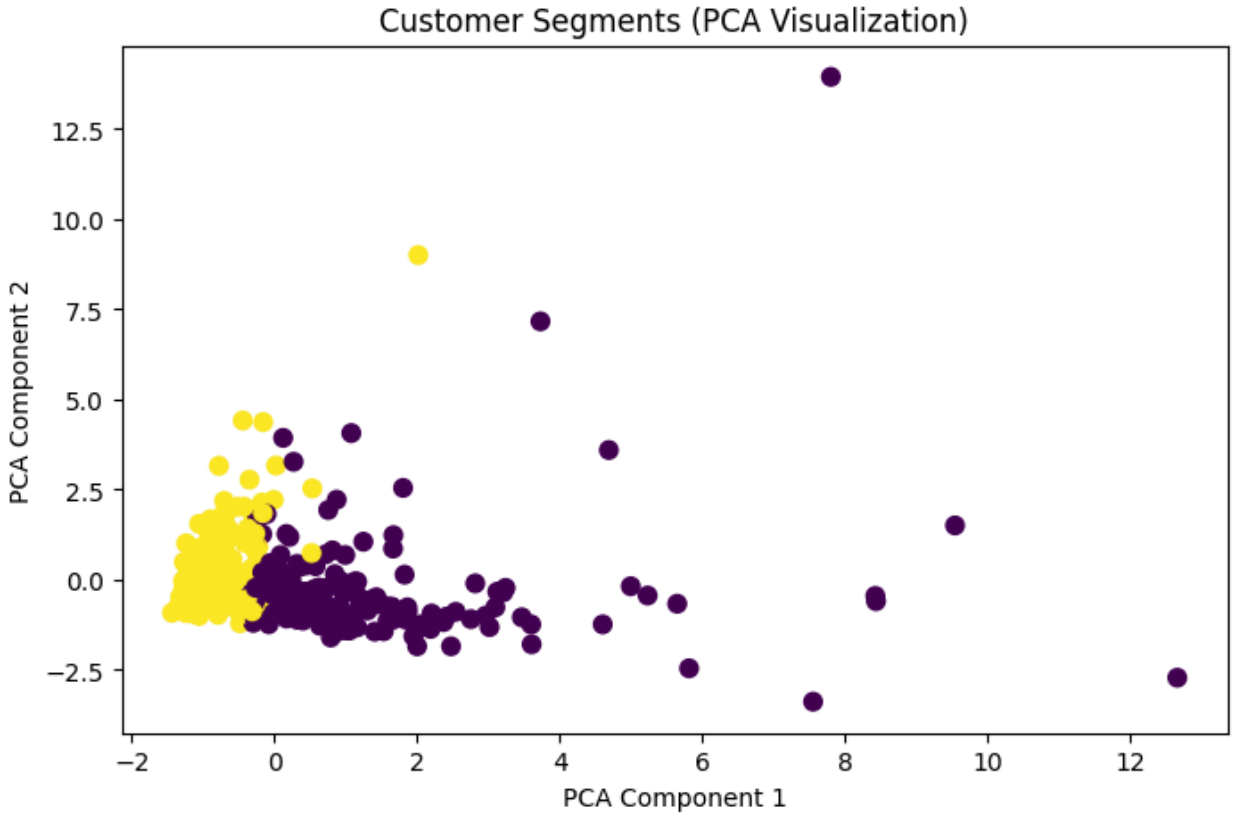
In [40]:
```
from sklearn.metrics import silhouette_score

# Calculate silhouette score
score = silhouette_score(x, clusters)
print("Silhouette Score:", score)
```

```
Silhouette Score: 0.29290996355033305
```

PCA to merge all 7 in 2 colmns and to visualize the Clusters

In [41]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(x)
```

In [42]:
```
plt.figure(figsize=(8,5))
plt.scatter(X_pca[:,0], X_pca[:,1], c=clusters, cmap='viridis', s=50)
plt.title('Customer Segments (PCA Visualization)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```



In [ ]:

Assignment No.8

Develop a Gradient Boosting regression model to predict wine quality based

on physicochemical properties. Handle missing values, perform feature scaling, and evaluate

the model using MSE, MAE, RMSE, and R-squared. Dataset: Wine Quality Dataset (UCI).

Name: Wavhal Prathmesh Navnath

Roll No:23107137

Class : TY-B

In [1]:
```
import pandas as pd
import seaborn as sns
import sklearn as sk
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
df = pd.read_csv("C:/Users/prath/Downloads/WineQT.csv")
```

In [3]:
```
df
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51
...	...	...	...	...	...	...	...	...	..
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57

1143 rows × 13 columns

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
```

In [5]:
```
df.info()
```

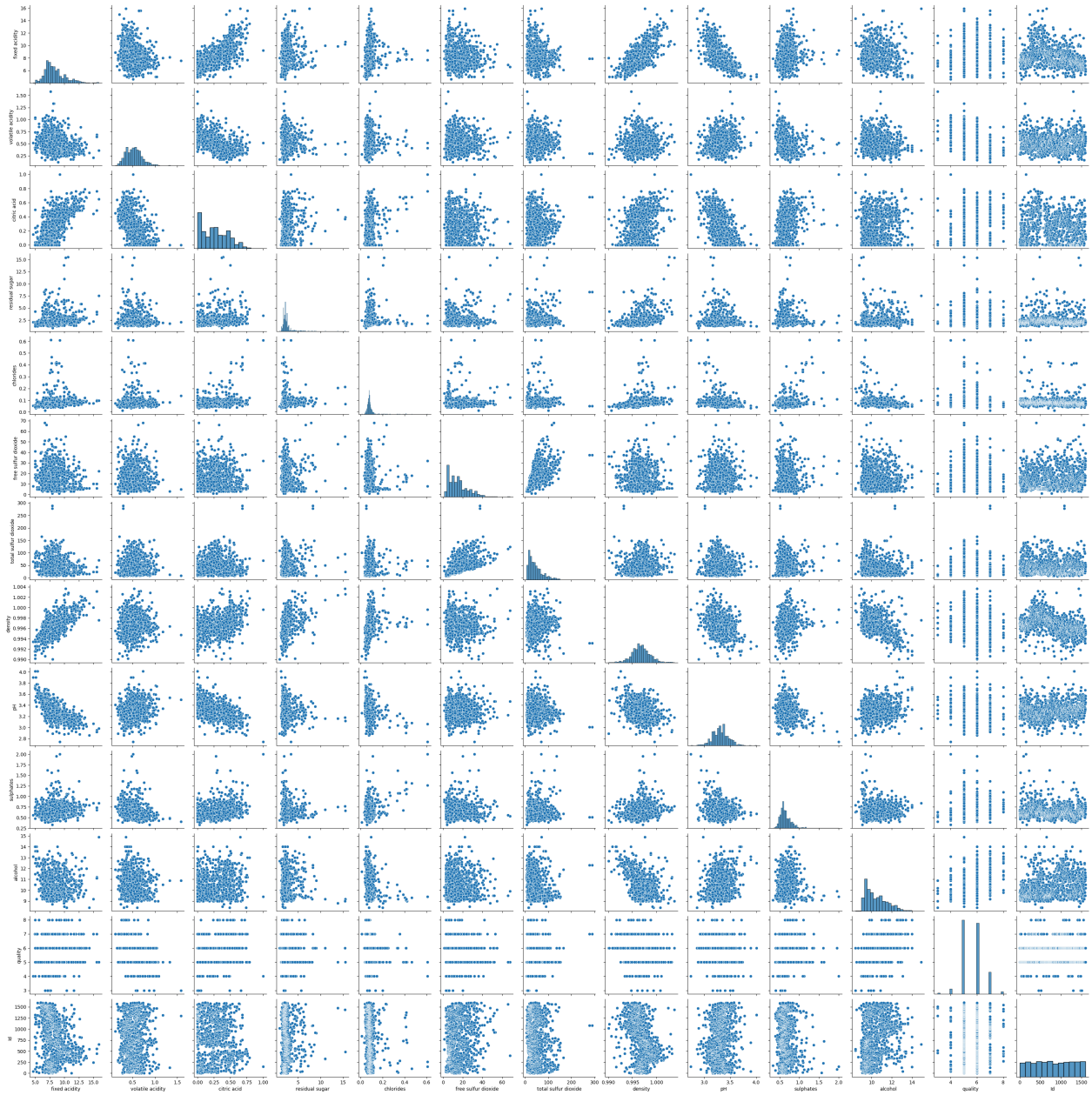
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

In [6]:
```
df['quality'].value_counts()
```

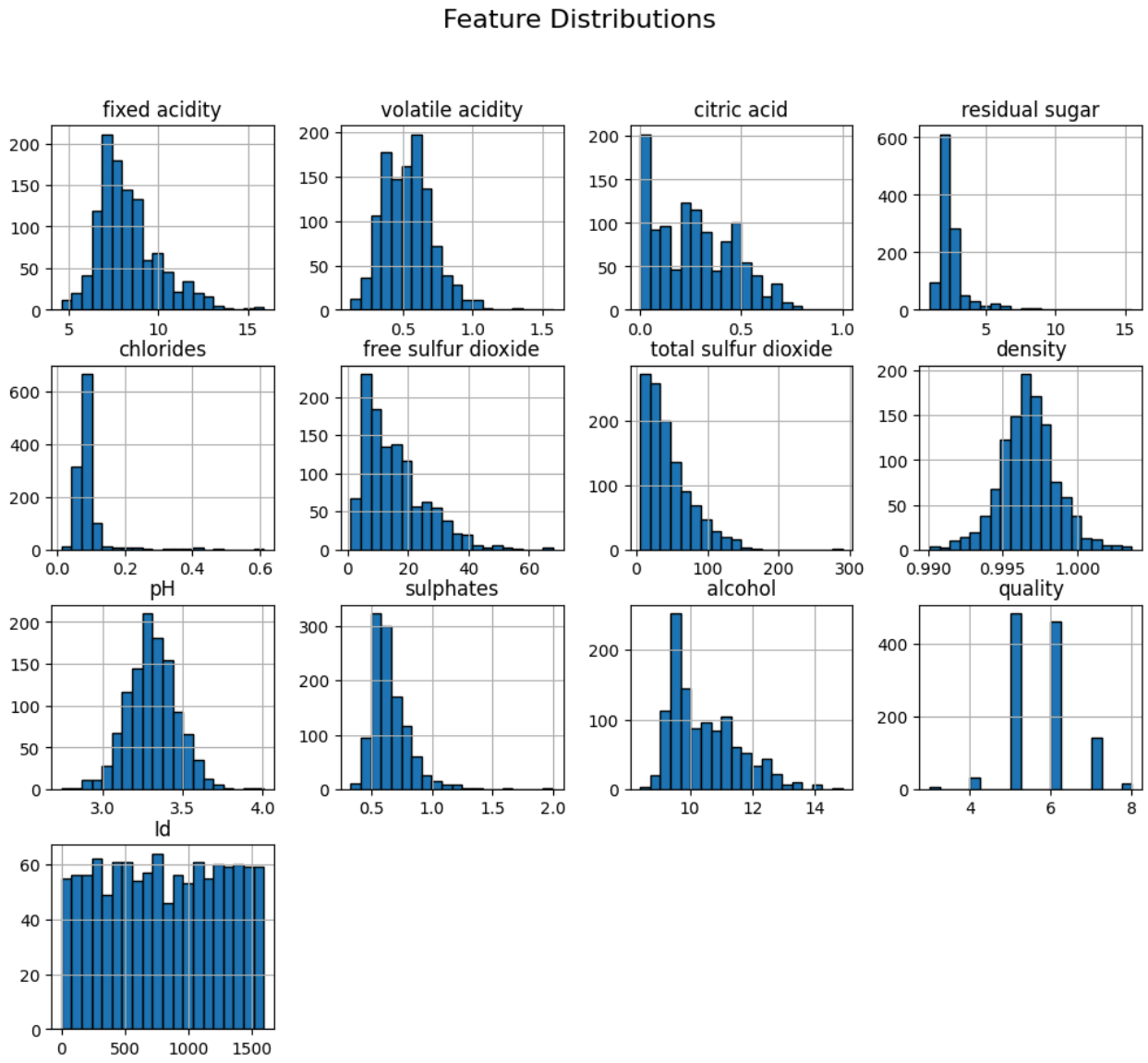
Out[6]:
```
quality
5    483
6    462
7    143
4     33
8     16
3      6
Name: count, dtype: int64
```

In [7]:
```
sns.pairplot(df)
```

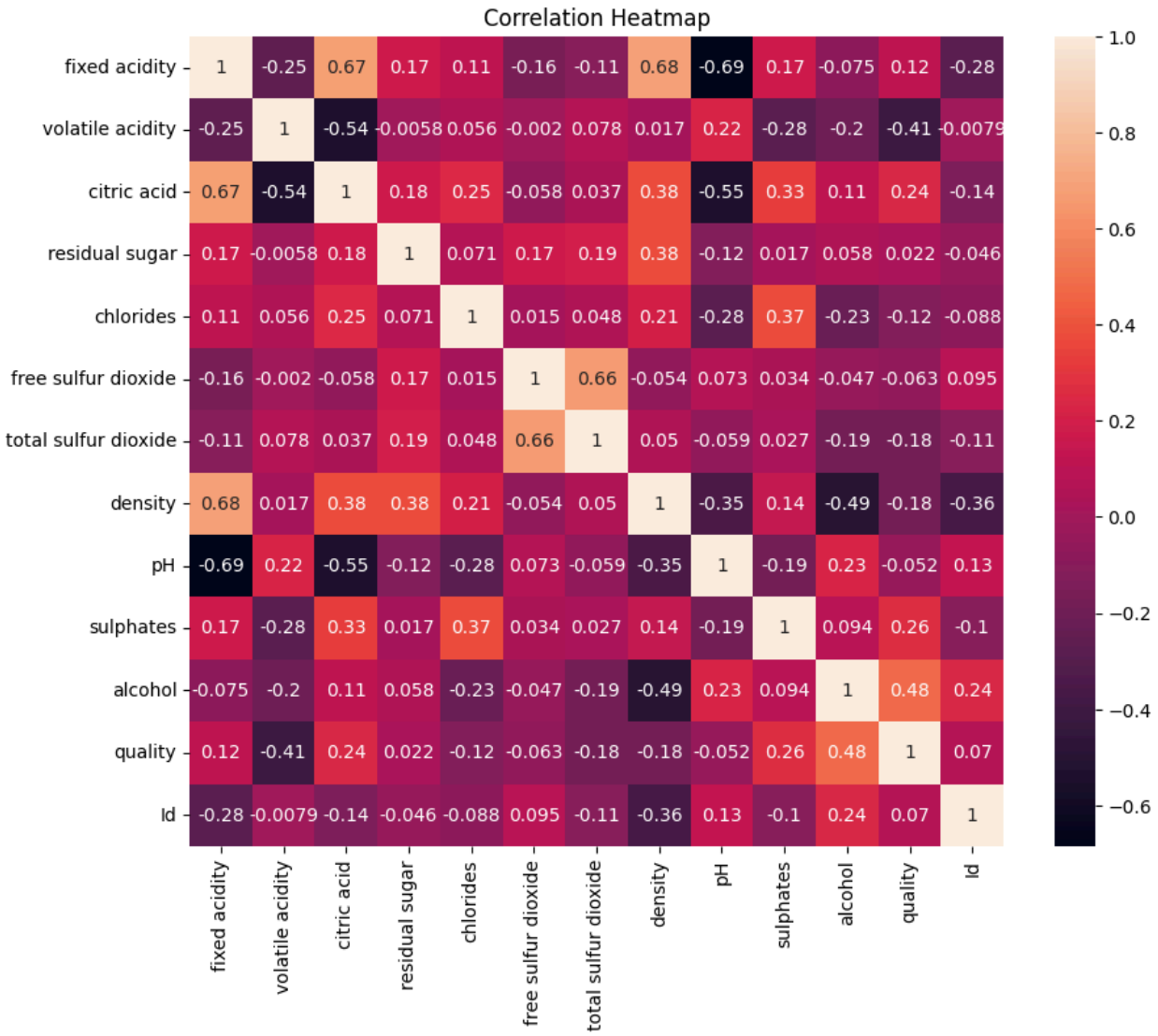
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x204ee115710>
```



In [8]:
```
df.hist(figsize=(12,10), bins=20, edgecolor="black")
plt.suptitle("Feature Distributions", fontsize=16)
plt.show()
```



In [9]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True)
plt.title("Correlation Heatmap")
plt.show()
```



In [10]:
```
### Getting All diff types of features
## get all numeric featuers
num_features = [feature for feature in df.columns if df[feature].dtype != 'O']
print("Numerical Features :",len(num_features))
```

```
Numerical Features : 13
```

In [11]:
```
# categorical features
cat_features = [feature for feature in df.columns if df[feature].dtype == 'O']
print("Categorical Features are:",len(cat_features))
```

```
Categorical Features are: 0
```

In [19]:
```
## Discrete Features       ## more no. of the categories more than 10
dis_features = [feature for feature in num_features if len(df[feature].unique(
print("Num of the Dis :",len(dis_features))
print(dis_features)
```

```
Num of the Dis : 1
['quality']
```

In [13]:
```
## Continuos features
cont_features = [feature for feature in num_features if feature not in dis_fea
print("Num of the Dis :",len(cont_features))
```

```
Num of the Dis : 12
```

In [14]:
```
## Independent and Dependent features
x = df.drop("quality",axis=1)
y = df['quality']
```

In [15]:
```
x = x.drop("Id",axis=1)
```

In [16]:
```
x
```

Out[16]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51
...	...	...	...	...	...	...	...	...	..
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57

1143 rows × 11 columns

In [17]:
```
y
```

Out[17]:
```
0       5
1       5
2       5
3       6
4       5
       ..
1138    6
1139    6
1140    5
1141    6
1142    5
Name: quality, Length: 1143, dtype: int64
```

In [21]:
```
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_stat
```

In [22]:
```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
```

In [23]:
```
x_train = sc.fit_transform(x_train)
```

In [24]:
```
x_test = sc.transform(x_test)
```

In [25]:
```
from sklearn.ensemble import GradientBoostingRegressor
model = GradientBoostingRegressor()
```

In [26]:
```
model
```

Out[26]:

GradientBoostingRegressor

Parameters

loss	'squared_error'
learning_rate	0.1
n_estimators	100
subsample	1.0
criterion	'friedman_mse'
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_depth	3
min_impurity_decrease	0.0
init	None
random_state	None
max_features	None
alpha	0.9
verbose	0
max_leaf_nodes	None
warm_start	False
validation_fraction	0.1
n_iter_no_change	None
tol	0.0001
ccp_alpha	0.0

In [27]:
```
model.fit(x_train,y_train)
```

Out[27]:

GradientBoostingRegressor

Parameters

loss	'squared_error'
learning_rate	0.1
n_estimators	100
subsample	1.0
criterion	'friedman_mse'
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_depth	3
min_impurity_decrease	0.0
init	None
random_state	None
max_features	None
alpha	0.9
verbose	0
max_leaf_nodes	None
warm_start	False
validation_fraction	0.1
n_iter_no_change	None
tol	0.0001
ccp_alpha	0.0

In [28]:
```
y_pred = model.predict(x_test)
```

In [38]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
mae =mean_absolute_error(y_test,y_pred)
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
print(mse)
print(mae)
print(rmse)
```

```
r2 = r2_score(y_test, y_pred)
```

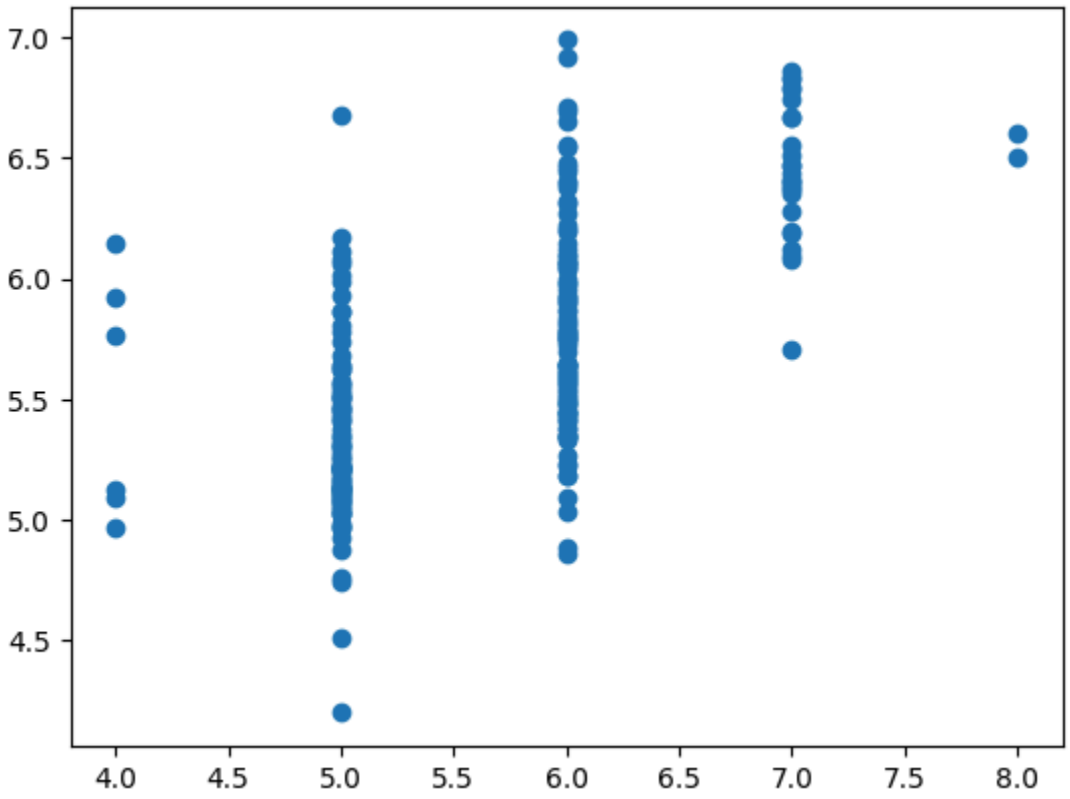
```
0.32780602358491967
0.4453997110497018
0.5725434687295976
```

In [39]:
```
r2
```

Out[39]:
```
0.4109219490502103
```

In [32]:
```
plt.scatter(y_test,y_pred)
```

Out[32]:
```
<matplotlib.collections.PathCollection at 0x20480f5ab50>
```



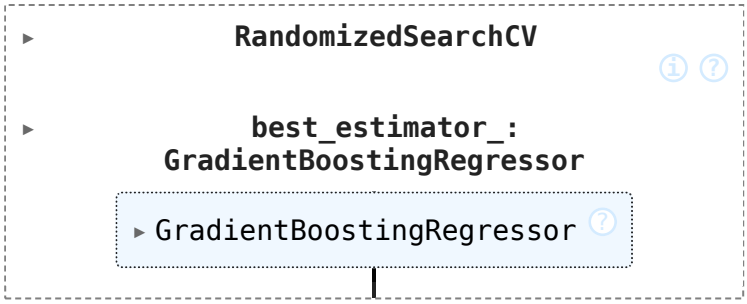
In [49]:
```
grad_param = {
    "max_depth": [3, 5, 8, 10, None],
    "n_estimators": [100, 200, 500, 1000],
    "loss": ['squared_error', 'absolute_error', 'huber'],
    "min_samples_split": [2, 5, 10, 15, 20],
    "learning_rate": [0.1, 0.05, 0.01, 0.001],

}
```

In [50]:
```
from sklearn.model_selection import RandomizedSearchCV
rc = RandomizedSearchCV(model,param_distributions=grad_param,n_iter=100,n_jobs
```

In [51]:
```
rc.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[51]:



In [52]:
```
best= rc.best_estimator_
```

In [53]:
```
best
```

Out[53]:

GradientBoostingRegressor

Parameters

loss	'absolute_error'
learning_rate	0.1
n_estimators	100
subsample	1.0
criterion	'friedman_mse'
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_depth	10
min_impurity_decrease	0.0
init	None
random_state	None
max_features	None
alpha	0.9
verbose	0
max_leaf_nodes	None
warm_start	False
validation_fraction	0.1
n_iter_no_change	None
tol	0.0001
ccp_alpha	0.0

In [54]:
```
rc.best_params_
```

Out[54]:
```
{'n_estimators': 100,
 'min_samples_split': 2,
 'max_depth': 10,
 'loss': 'absolute_error',
 'learning_rate': 0.1}
```

In [55]:
```
rc.best_score_
```

Out[55]: 0.39693087655319226

In [56]:
```
y_pred= best.predict(x_test)
```

In [57]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
mae =mean_absolute_error(y_test,y_pred)
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
print(mse)
print(mae)
print(rmse)
r2 = r2_score(y_test, y_pred)
```

```
0.34053571376244934
0.41503855861495104
0.5835543794390111
```

In [58]:
```
r2
```

Out[58]: 0.3880462831397232

In [ ]:
```

```

Assignment No.9

Build an XGBoost classification model to predict whether a customer will take the travel package (ProdTaken) using

demographic and marketing features.

The process includes handling missing values (e.g., Age, Income, DurationOfPitch),

encoding categorical variables (Occupation, Gender, TypeofContact, etc.),

feature engineering, and balancing class distribution.

The model will be trained and evaluated using AUC-ROC and F1 score to ensure robust prediction of customer decisions.

Name: Wavhal Prathmesh Navnath

Roll No:23107137

Class : TY-B

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
```

In [3]:
```
df = pd.read_csv("C:/Users/prath/Downloads/Travel.csv")
```

Handellling Missing Values

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
CustomerID                   0
ProdTaken                    0
Age                        226
TypeofContact               25
CityTier                     0
DurationOfPitch            251
Occupation                   0
Gender                       0
NumberOfPersonVisiting       0
NumberOfFollowups           45
ProductPitched               0
PreferredPropertyStar       26
MaritalStatus                0
NumberOfTrips              140
Passport                     0
PitchSatisfactionScore       0
OwnCar                       0
NumberOfChildrenVisiting    66
Designation                  0
MonthlyIncome              233
dtype: int64
```

In [5]:
```
df['Gender']= df['Gender'].replace("Fe Male","Female")
df['MaritalStatus']= df['MaritalStatus'].replace("Single","Unmarried")
```

In [6]:
```
## Checking Missing Values
## these are NaN values
na_values = [features for features in df.columns if df[features].isnull().sum(
for feature in na_values:
    print(feature,np.round(df[feature].isnull().mean()*100,5),'% missing Value
```

```
Age 4.62357 % missing Values
TypeofContact 0.51146 % missing Values
DurationOfPitch 5.13502 % missing Values
NumberOfFollowups 0.92062 % missing Values
PreferredPropertyStar 0.53191 % missing Values
NumberOfTrips 2.86416 % missing Values
NumberOfChildrenVisiting 1.35025 % missing Values
MonthlyIncome 4.76678 % missing Values
```

In [7]:
```
df[na_values].select_dtypes(exclude = 'object').describe()
```

Out[7]:

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertySta
count	4662.000000	4637.000000	4843.000000	4862.00000
mean	37.622265	15.490835	3.708445	3.58103
std	9.316387	8.519643	1.002509	0.79800
min	18.000000	5.000000	1.000000	3.00000
25%	31.000000	9.000000	3.000000	3.00000
50%	36.000000	13.000000	4.000000	3.00000
75%	44.000000	20.000000	4.000000	4.00000
max	61.000000	127.000000	6.000000	5.00000

In [8]:
```
df.Age.fillna(df.Age.median(),inplace = True)
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_1252\2067532558.py:1: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df.Age.fillna(df.Age.median(),inplace = True)
```

In [9]:
```
df.TypeofContact.fillna(df.TypeofContact.mode()[0],inplace = True)
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_1252\1271724004.py:1: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df.TypeofContact.fillna(df.TypeofContact.mode()[0],inplace = True)
```

In [10]:
```
df.DurationOfPitch.fillna(df.DurationOfPitch.median(),inplace=True)
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_1252\1100695732.py:1: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df.DurationOfPitch.fillna(df.DurationOfPitch.median(),inplace=True)
```

In [11]:
```
df.NumberOfFollowups.fillna(df.NumberOfFollowups.mode()[0],inplace = True)   #
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_1252\821710093.py:1: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df.NumberOfFollowups.fillna(df.NumberOfFollowups.mode()[0],inplace = True)
## for the categorical features use the mode
```

In [12]:
```
df.PreferredPropertyStar.fillna(df.PreferredPropertyStar.mode()[0],inplace = T
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_1252\4219815442.py:1: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df.PreferredPropertyStar.fillna(df.PreferredPropertyStar.mode()[0],inplace =
True)
```

In [13]:
```
df.NumberOfTrips.fillna(df.NumberOfTrips.median(),inplace=True)  # here no val
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_1252\610309013.py:1: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df.NumberOfTrips.fillna(df.NumberOfTrips.median(),inplace=True)  # here no va
lues
```

In [14]:
```
df.NumberOfChildrenVisiting.fillna(df.NumberOfChildrenVisiting.mode()[0],inpla
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_1252\3127042307.py:1: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df.NumberOfChildrenVisiting.fillna(df.NumberOfChildrenVisiting.mode()[0],inpl
ace=True)
```

In [15]:
```
df.MonthlyIncome.fillna(df.MonthlyIncome.median(),inplace=True)
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_1252\3937586248.py:1: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df.MonthlyIncome.fillna(df.MonthlyIncome.median(),inplace=True)
```

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
CustomerID                0
ProdTaken                 0
Age                       0
TypeofContact             0
CityTier                  0
DurationOfPitch           0
Occupation                0
Gender                    0
NumberOfPersonVisiting    0
NumberOfFollowups         0
ProductPitched            0
PreferredPropertyStar     0
MaritalStatus             0
NumberOfTrips             0
Passport                  0
PitchSatisfactionScore    0
OwnCar                    0
NumberOfChildrenVisiting  0
Designation               0
MonthlyIncome             0
dtype: int64
```

In [17]:
```
# cust id is not imp so remove it
df.drop('CustomerID',inplace=True,axis=1)
```

Feature Engineering

In [20]:
```
### create new column
df['Total_Visiting'] = df['NumberOfChildrenVisiting'] + df['NumberOfPersonVisi
df.drop(columns=['NumberOfChildrenVisiting','NumberOfPersonVisiting'],inplace=
```

In [21]:
```
## get all numeric featuers
num_features = [feature for feature in df.columns if df[feature].dtype != 'O']
print("Numerical Features :",len(num_features))
```

```
Numerical Features : 12
```

In [22]:
```
# categorical features
cat_features = [feature for feature in df.columns if df[feature].dtype == 'O']
print("Categorical Features are:",len(cat_features))
```

```
Categorical Features are: 6
```

In [23]:
```
## Discrete Features       ## more no. of the categories more than 10
dis_features = [feature for feature in num_features if len(df[feature].unique(
print("Num of the Dis :",len(dis_features))
```

```
Num of the Dis : 9
```

In [24]:
```
## Continuos features
cont_features = [feature for feature in num_features if feature not in dis_fea
print("Num of the Dis :",len(cont_features))
```

```
Num of the Dis : 3
```

MODEL TRAINING

In [25]:
```
from sklearn.model_selection import train_test_split
x= df.drop('ProdTaken',axis = 1)
y = df['ProdTaken']
```

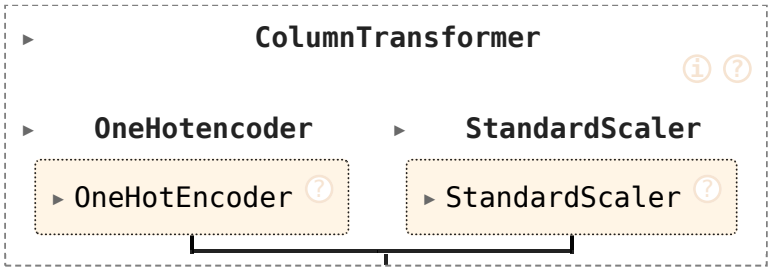
In [26]:
```
x_train,x_test,y_train,y_test  =train_test_split(x,y,train_size=0.8,random_sta
x_train.shape ,x_test.shape
```

Out[26]:
```
((3910, 17), (978, 17))
```

In [27]:
```
## Create column transformer with 3 types of transformers
cat_features = x.select_dtypes(include='object').columns
num_features = x.select_dtypes(exclude='object').columns

from sklearn.preprocessing import OneHotEncoder,StandardScaler
from sklearn.compose import ColumnTransformer

num_transformer = StandardScaler()
oh_transformer = OneHotEncoder(drop='first')

preprocesser = ColumnTransformer(
    [
        ("OneHotencoder",oh_transformer,cat_features),
        ("StandardScaler",num_transformer,num_features)
    ]
)       ## combine Both in one it is columntransformer
```

In [28]:
```
preprocesser
```

Out[28]:



In [29]:
```
## Apply for the training data (Fit Transform)
x_train = preprocesser.fit_transform(x_train)
```

In [30]:
```
x_test = preprocesser.transform(x_test)
```

In [33]:
```
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,classification_report,f1_score,prec
```

In [34]:

```
models = {
    "Random Forest" :RandomForestClassifier(),
    "Decision tree" :DecisionTreeClassifier(),
    "Logistic Regression":LogisticRegression(),
    "Gradient Boosting":GradientBoostingClassifier(),
    "Adaboost Boosting":AdaBoostClassifier(),
    "XgBoost Classifier":XGBClassifier()                    ## we can add any a
}
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(x_train,y_train) # Train

    ## Predictions
    y_train_pred = model.predict(x_train)
    y_test_pred = model.predict(x_test)

    ## Performance for Y_train(TRAIN)
    model_train_acc = accuracy_score(y_train,y_train_pred)
    model_train_f1 = f1_score(y_train,y_train_pred, average='weighted')
    model_train_precision = precision_score(y_train,y_train_pred)
    model_train_recall = recall_score(y_train,y_train_pred)
    model_train_roc = roc_auc_score(y_train,y_train_pred)

    ## Performance for y_test(TEST)
    model_test_acc = accuracy_score(y_test,y_test_pred)
    model_test_f1 = f1_score(y_test,y_test_pred, average='weighted')
    model_test_precision = precision_score(y_test,y_test_pred)
    model_test_recall = recall_score(y_test,y_test_pred)
    model_test_roc = roc_auc_score(y_test,y_test_pred)

    print(list(models.keys())[i])

    print("model Performance for the Training Set")
    print(" - Accuracy:{:.4f}".format(model_train_acc))
    print(" - F1 Score:{:.4f}".format(model_train_f1))

    print(" - Precision:{:.4f}".format(model_train_precision))
    print(" - Recall:{:.4f}".format(model_train_recall))
    print(" - ROC Score:{:.4f}".format(model_train_roc))

    print("----------------------------------------------")

    print("model Performance for the Testing Set")
    print(" - Accuracy:{:.4f}".format(model_test_acc))
    print(" - F1 Score:{:.4f}".format(model_test_f1))

    print(" - Precision:{:.4f}".format(model_test_precision))
    print(" - Recall:{:.4f}".format(model_test_recall))
    print(" - ROC Score:{:.4f}".format(model_test_roc))

    print('='*35)
    print('\n')
```

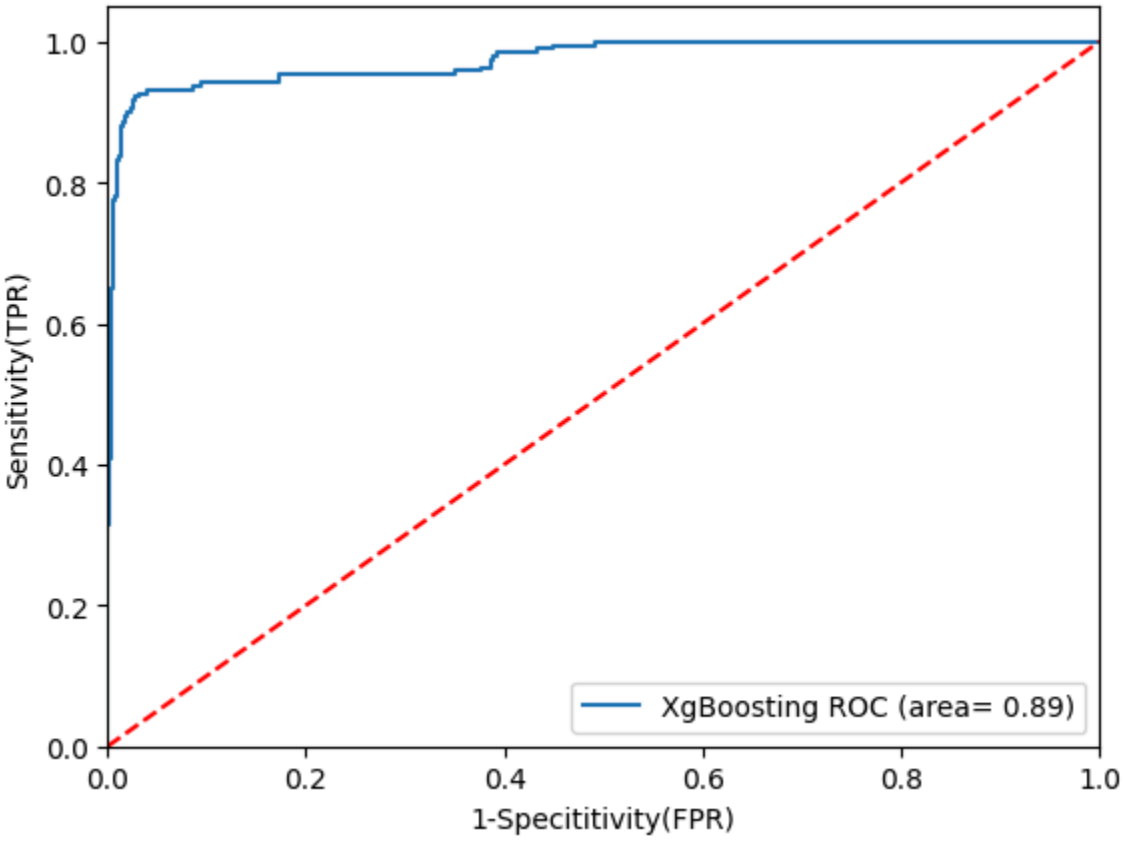
```
Random Forest
model Performance for the Training Set
 - Accuracy:0.9997
 - F1 Score:0.9997
 - Precision:1.0000
 - Recall:0.9986
 - ROC Score:0.9993
--------------------------------------------
model Performance for the Testing Set
 - Accuracy:0.9294
 - F1 Score:0.9241
 - Precision:0.9692
 - Recall:0.6597
 - ROC Score:0.8273
====================================


Decision tree
model Performance for the Training Set
 - Accuracy:1.0000
 - F1 Score:1.0000
 - Precision:1.0000
 - Recall:1.0000
 - ROC Score:1.0000
--------------------------------------------
model Performance for the Testing Set
 - Accuracy:0.9243
 - F1 Score:0.9240
 - Precision:0.8128
 - Recall:0.7958
 - ROC Score:0.8757
====================================


Logistic Regression
model Performance for the Training Set
 - Accuracy:0.8460
 - F1 Score:0.8202
 - Precision:0.7016
 - Recall:0.3032
 - ROC Score:0.6368
--------------------------------------------
model Performance for the Testing Set
 - Accuracy:0.8364
 - F1 Score:0.8087
 - Precision:0.6914
 - Recall:0.2932
 - ROC Score:0.6307
====================================


Gradient Boosting
model Performance for the Training Set
 - Accuracy:0.8939
```

```
 - F1 Score:0.8819
 - Precision:0.8756
 - Recall:0.5021
 - ROC Score:0.7429
----------------------------------------------
model Performance for the Testing Set
 - Accuracy:0.8589
 - F1 Score:0.8398
 - Precision:0.7732
 - Recall:0.3927
 - ROC Score:0.6824
==================================


Adaboost Boosting
model Performance for the Training Set
 - Accuracy:0.8478
 - F1 Score:0.8146
 - Precision:0.7815
 - Recall:0.2551
 - ROC Score:0.6194
----------------------------------------------
model Performance for the Testing Set
 - Accuracy:0.8354
 - F1 Score:0.7987
 - Precision:0.7500
 - Recall:0.2356
 - ROC Score:0.6083
==================================


XgBoost Classifier
model Performance for the Training Set
 - Accuracy:0.9992
 - F1 Score:0.9992
 - Precision:1.0000
 - Recall:0.9959
 - ROC Score:0.9979
----------------------------------------------
model Performance for the Testing Set
 - Accuracy:0.9356
 - F1 Score:0.9318
 - Precision:0.9507
 - Recall:0.7068
 - ROC Score:0.8490
==================================
```

In [48]:

```python
## Plot AUC ROC Curve

from sklearn.metrics import roc_auc_score,roc_curve
plt.figure()
```

```
# Add models to the list that you want to view on ROC
auc_models = [
    {
        'label':'XgBoosting',
        'model':XGBClassifier(n_estimators=200,max_depth=12,learning_rate=0.21
        'auc': 0.8935
    }]
for algo in auc_models:
    model = algo['model']  # select model
    model.fit(x_train,y_train)

    ## Calculate FPR , TPR
    fpr,tpr,thrs = roc_curve(y_test,model.predict_proba(x_test)[:,1])
    #calculate AUC

    plt.plot(fpr, tpr, label='%s ROC (area= %0.2f)' % (algo['label'], algo['au

plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel("1-Specititivity(FPR)")
plt.ylabel("Sensitivity(TPR)")
plt.legend(loc = 'lower right')
plt.show()
```



In [49]:

In [ ]:

In [2]:
```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df = pd.read_csv("C:/Users/prath/Downloads/Breast_cancer.csv")
```

In [4]:
```
df
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_m
0	842302	M	17.99	10.38	122.80	10
1	842517	M	20.57	17.77	132.90	13
2	84300903	M	19.69	21.25	130.00	12
3	84348301	M	11.42	20.38	77.58	3
4	84358402	M	20.29	14.34	135.10	12
...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	14
565	926682	M	20.13	28.25	131.20	12
566	926954	M	16.60	28.08	108.30	8
567	927241	M	20.60	29.33	140.10	12
568	92751	B	7.76	24.54	47.92	1

569 rows × 33 columns

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
```

In [20]:
```
df = df.drop("Unnamed: 32",axis=1)
```

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    int32  
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int32(1), int64(1)
memory usage: 140.2 KB
```

In [22]:
```
df.describe()
```

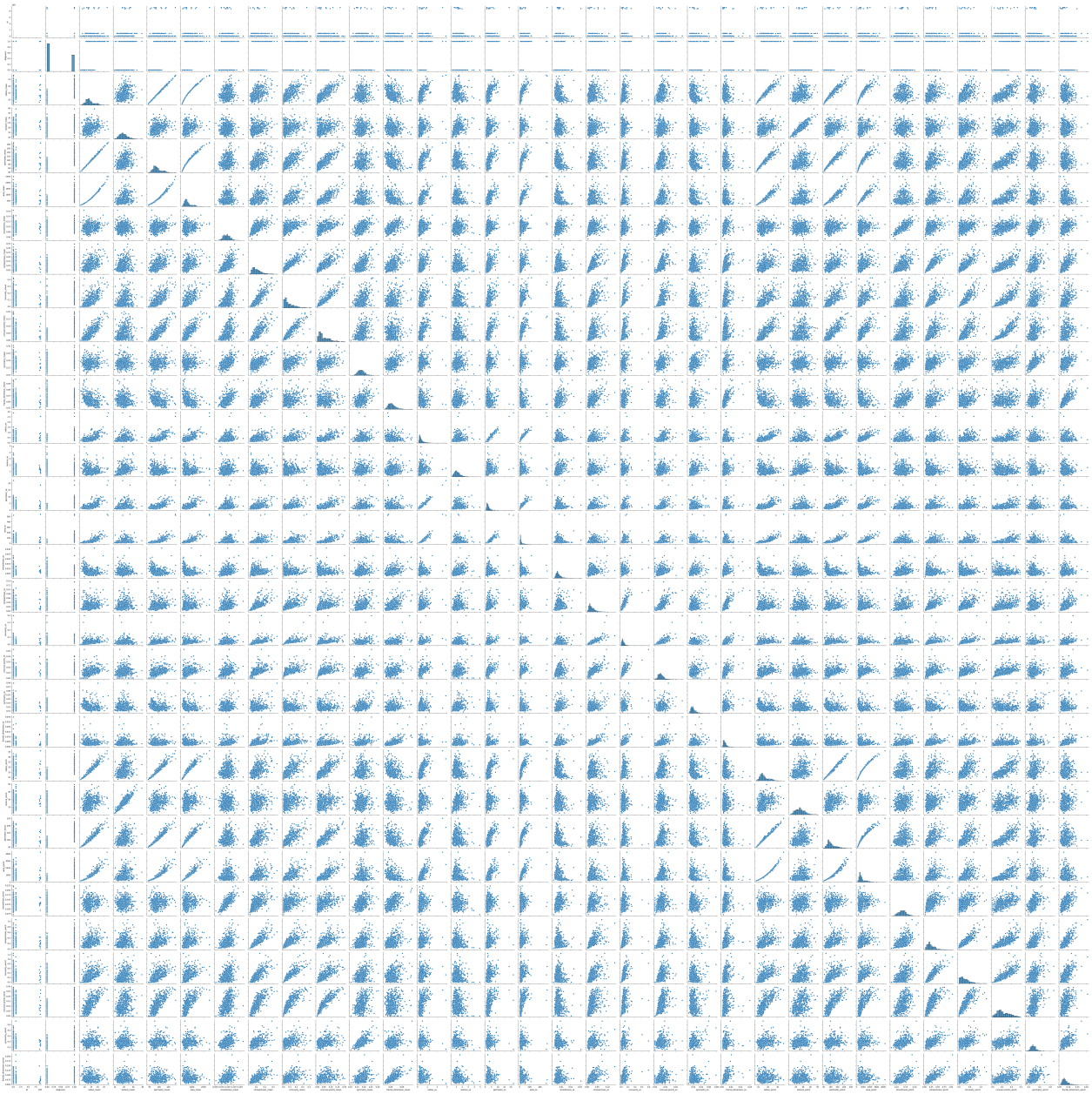
Out[22]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	0.372583	14.127292	19.289649	91.969033
std	1.250206e+08	0.483918	3.524049	4.301036	24.298981
min	8.670000e+03	0.000000	6.981000	9.710000	43.790000
25%	8.692180e+05	0.000000	11.700000	16.170000	75.170000
50%	9.060240e+05	0.000000	13.370000	18.840000	86.240000
75%	8.813129e+06	1.000000	15.780000	21.800000	104.100000
max	9.113205e+08	1.000000	28.110000	39.280000	188.500000

8 rows × 32 columns

In [23]:
```
sns.pairplot(df)
```

Out[23]:
```
<seaborn.axisgrid.PairGrid at 0x284db309990>
```



In [24]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [25]:
```
df['diagnosis'] = le.fit_transform(df['diagnosis'])
```

In [26]:
```
df
```

Out[26]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_m
0	842302	1	17.99	10.38	122.80	10
1	842517	1	20.57	17.77	132.90	13
2	84300903	1	19.69	21.25	130.00	12
3	84348301	1	11.42	20.38	77.58	3
4	84358402	1	20.29	14.34	135.10	12
...	...	...	...	...	...	
564	926424	1	21.56	22.39	142.00	14
565	926682	1	20.13	28.25	131.20	12
566	926954	1	16.60	28.08	108.30	8
567	927241	1	20.60	29.33	140.10	12
568	92751	0	7.76	24.54	47.92	1

569 rows × 32 columns

In [27]:
```
df['diagnosis'].value_counts()
```

Out[27]:
```
diagnosis
0    357
1    212
Name: count, dtype: int64
```

In [28]:
```
x = df.drop("diagnosis",axis=1)
y = df['diagnosis']
```

In [29]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8,random_sta
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [31]:
```
x_train = sc.fit_transform(x_train)
```

In [32]:
```
x_test = sc.transform(x_test)
```

Model Training

In [33]:
```
# Initialize KNN model
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()  # you can tune k later
```

In [34]:
```
model.fit(x_train, y_train)
```

Out[34]:

KNeighborsClassifier

Parameters

n_neighbors	5
weights	'uniform'
algorithm	'auto'
leaf_size	30
p	2
metric	'minkowski'
metric_params	None
n_jobs	None

In [35]:
```
y_pred = model.predict(x_test)
```

In [36]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
```

In [37]:
```
accuracy
```

Out[37]:
```
0.9473684210526315
```

In [38]:
```
precision
```

Out[38]:
```
0.9302325581395349
```

In [39]:
```
recall
```

Out[39]:
```
0.9302325581395349
```

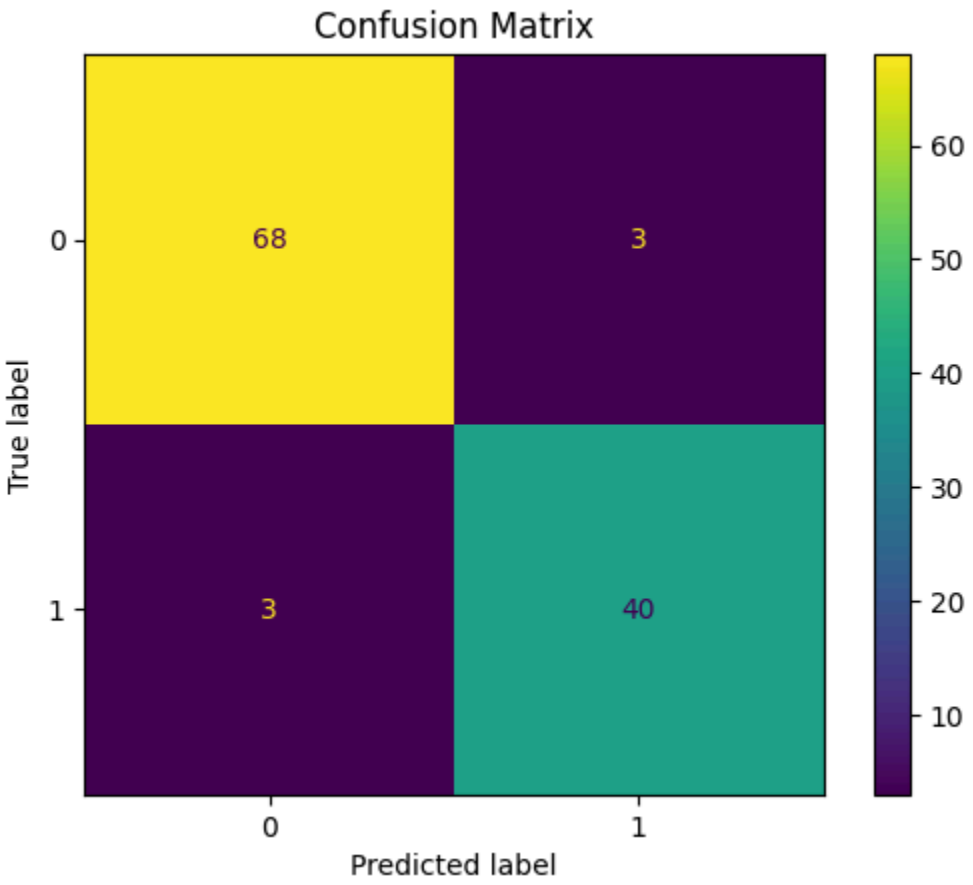
In [40]:
```
f1
```

Out[40]:
```
0.9302325581395349
```

In [41]:
```
cm
```

Out[41]:
```
array([[68,  3],
       [ 3, 40]], dtype=int64)
```

In [42]:
```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classe
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```



In [ ]:

In [2]:
```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df = pd.read_csv("C:/Users/prath/Downloads/online_shoppers_intention.csv")
```

In [5]:
```
df
```

Out[5]:

	Administrative	Administrative_Duration	Informational	Informational_Du
0	0	0.0	0	
1	0	0.0	0	
2	0	0.0	0	
3	0	0.0	0	
4	0	0.0	0	
...	...	...	...	
12325	3	145.0	0	
12326	0	0.0	0	
12327	0	0.0	0	
12328	4	75.0	0	
12329	0	0.0	0	

12330 rows × 18 columns

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

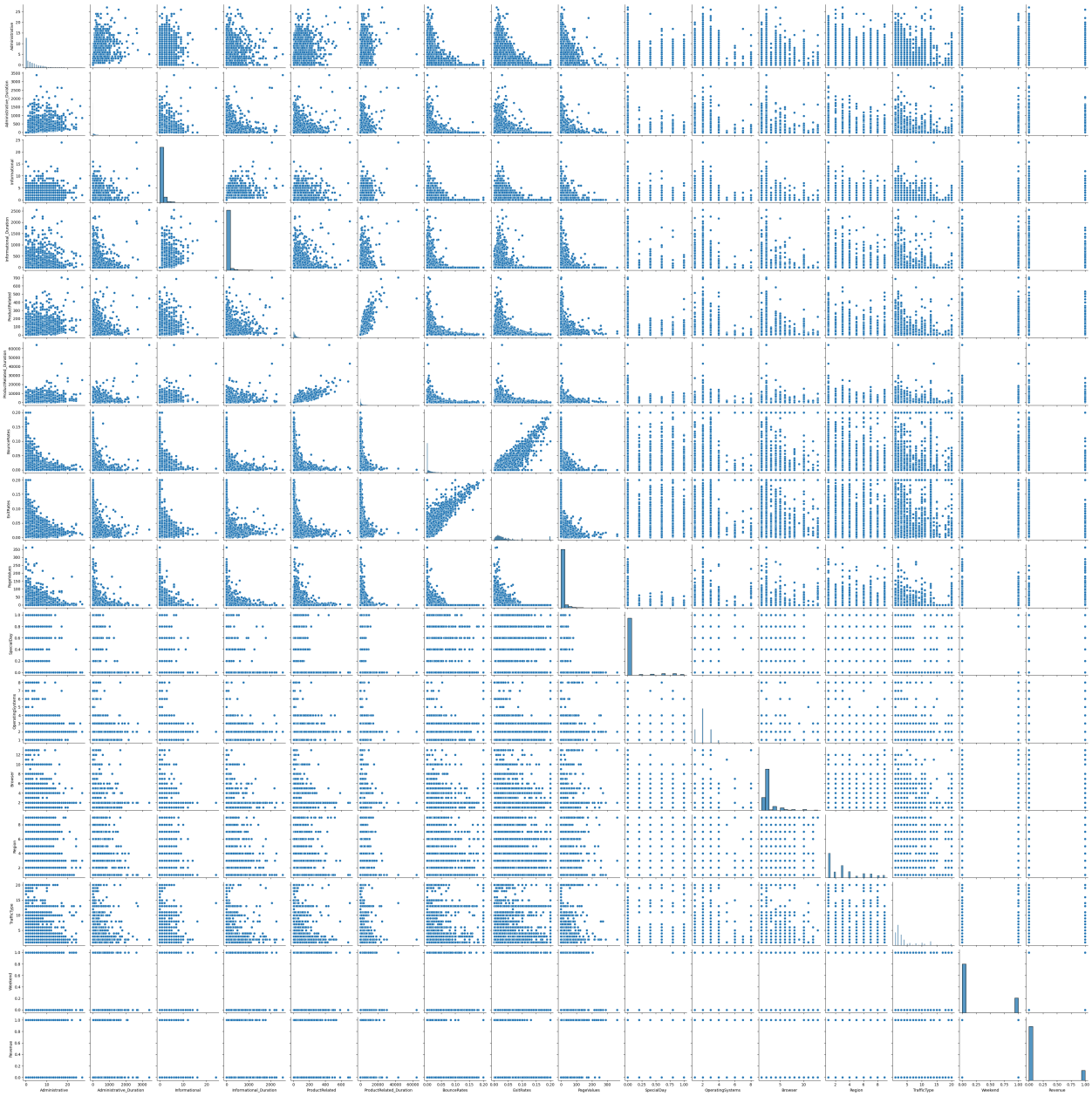
In [8]:
```
df.describe()
```

Out[8]:

	Administrative	Administrative_Duration	Informational	Informational_Du
count	12330.000000	12330.000000	12330.000000	12330.0
mean	2.315166	80.818611	0.503569	34.4
std	3.321784	176.779107	1.270156	140.7
min	0.000000	0.000000	0.000000	0.0
25%	0.000000	0.000000	0.000000	0.0
50%	1.000000	7.500000	0.000000	0.0
75%	4.000000	93.256250	0.000000	0.0
max	27.000000	3398.750000	24.000000	2549.3

In [9]:
```
sns.pairplot(df)
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1e44a067e90>
```



In [26]:
```
df['Revenue'].value_counts()
```

Out[26]:
```
Revenue
False    10422
True      1908
Name: count, dtype: int64
```

In [10]:
```
from sklearn.preprocessing import LabelEncoder
```

In [11]:
```
le = LabelEncoder()
categorical_cols = ['Month', 'VisitorType', 'Weekend']
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])
```

In [13]:
```
# Features and Target
```

```
x = df.drop('Revenue', axis=1)
y = df['Revenue']
```

In [14]:
```
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, randc
```

In [27]:
```
print("Before SMOTE:")
print(y_train.value_counts())
```

```
Before SMOTE:
Revenue
False    8367
True     1497
Name: count, dtype: int64
```

In [28]:
```
# Apply SMOTE for balancing the classes
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
x_train, y_train = sm.fit_resample(x_train, y_train)
```

In [30]:
```
print("After SMOTE:")
print(y_train.value_counts())
```

```
After SMOTE:
Revenue
False    8367
True     8367
Name: count, dtype: int64
```

Model Training

In [31]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
model = DecisionTreeClassifier(random_state=42, max_depth=5)
```

In [32]:
```
model.fit(x_train, y_train)
```

Out[32]:

DecisionTreeClassifier

Parameters

criterion	'gini'
splitter	'best'
max_depth	5
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	None
random_state	42
max_leaf_nodes	None
min_impurity_decrease	0.0
class_weight	None
ccp_alpha	0.0
monotonic_cst	None

In [33]:
```
y_pred = model.predict(x_test)
```

In [34]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
```

In [35]:
```
accuracy
```

Out[35]: 0.8694241686942417

In [36]:
```
precision
```

Out[36]: 0.5779334500875657

In [37]:
```
recall
```

Out[37]: 0.8029197080291971

In [38]:
```
f1
```

Out[38]:
```
0.6720977596741344
```

In [39]:
```
cm
```

Out[39]:
```
array([[1814,  241],
       [  81,  330]], dtype=int64)
```

In [40]:
```
import numpy as np

cm = np.array([[1814, 241],
               [81, 330]])

TN, FP, FN, TP = cm.ravel()

accuracy = (TP + TN) / cm.sum()
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1 = 2 * (precision * recall) / (precision + recall)

print(f"Accuracy:  {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")
```

```
Accuracy:  0.8694
Precision: 0.5779
Recall:    0.8029
F1 Score:  0.6721
```

In [ ]:
```

```

137-ds-6-1

March 27, 2025

Name: Wavhal Prathmesh Navnath

Div:SY-B

Roll No:23107137

Assignment No.6

Data Analytics III

[1]:
```
import pandas as pd
```

[5]:
```
df=pd.read_csv(r"../admin1/iris (2).csv")
```

[7]:
```
df
```

[7]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
..	…	…	…	…	…
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

[150 rows x 5 columns]

[9]:
```
df.isnull()
```

[9]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
..	…	…	…	…	…

```
145         False        False         False        False    False
146         False        False         False        False    False
147         False        False         False        False    False
148         False        False         False        False    False
149         False        False         False        False    False

[150 rows x 5 columns]
```

[11]:
```
df.isna()
```

[11]:
```
     sepal_length  sepal_width  petal_length  petal_width  species
0           False        False         False        False    False
1           False        False         False        False    False
2           False        False         False        False    False
3           False        False         False        False    False
4           False        False         False        False    False
..            …            …             …            …        …
145         False        False         False        False    False
146         False        False         False        False    False
147         False        False         False        False    False
148         False        False         False        False    False
149         False        False         False        False    False

[150 rows x 5 columns]
```

[13]:
```
df.notnull()
```

[13]:
```
     sepal_length  sepal_width  petal_length  petal_width  species
0            True         True          True         True     True
1            True         True          True         True     True
2            True         True          True         True     True
3            True         True          True         True     True
4            True         True          True         True     True
..            …            …             …            …        …
145          True         True          True         True     True
146          True         True          True         True     True
147          True         True          True         True     True
148          True         True          True         True     True
149          True         True          True         True     True

[150 rows x 5 columns]
```

[15]:
```
df.isnull().sum()
```

[15]:
```
sepal_length    0
sepal_width     0
petal_length    0
```

```
petal_width     0
species         0
dtype: int64
```

[17]:
```
df.describe()
```

[17]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

[19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

[21]:
```
import sklearn as sk
```

[23]:
```
df.dropna()
```

[23]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
..	…	…	…	…	…
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

```
[150 rows x 5 columns]
```

[25]:
```
df.replace(["setosa","versicolor","virginica"],[1,2,3],inplace=True)
```

[27]:
```
df
```

[27]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1
..	…	…	…	…	…
145	6.7	3.0	5.2	2.3	3
146	6.3	2.5	5.0	1.9	3
147	6.5	3.0	5.2	2.0	3
148	6.2	3.4	5.4	2.3	3
149	5.9	3.0	5.1	1.8	3

```
[150 rows x 5 columns]
```

[33]:
```
x=df[["sepal_length","sepal_width","petal_length","petal_width"]]
y=df["species"]
```

[35]:
```
x
```

[35]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	…	…	…	…
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

```
[150 rows x 4 columns]
```

[37]:
```
y
```

[37]:
```
0      1
1      1
2      1
```

```
3      1
4      1
      ..
145    3
146    3
147    3
148    3
149    3
Name: species, Length: 150, dtype: int64
```

[39]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.8)
```

[41]:
```
x_train
```

[41]:

	sepal_length	sepal_width	petal_length	petal_width
22	4.6	3.6	1.0	0.2
8	4.4	2.9	1.4	0.2
63	6.1	2.9	4.7	1.4
108	6.7	2.5	5.8	1.8
115	6.4	3.2	5.3	2.3
..	…	…	…	…
19	5.1	3.8	1.5	0.3
68	6.2	2.2	4.5	1.5
89	5.5	2.5	4.0	1.3
70	5.9	3.2	4.8	1.8
88	5.6	3.0	4.1	1.3

[120 rows x 4 columns]

[43]:
```
x_test
```

[43]:

	sepal_length	sepal_width	petal_length	petal_width
83	6.0	2.7	5.1	1.6
81	5.5	2.4	3.7	1.0
41	4.5	2.3	1.3	0.3
6	4.6	3.4	1.4	0.3
57	4.9	2.4	3.3	1.0
110	6.5	3.2	5.1	2.0
4	5.0	3.6	1.4	0.2
49	5.0	3.3	1.4	0.2
117	7.7	3.8	6.7	2.2
100	6.3	3.3	6.0	2.5
10	5.4	3.7	1.5	0.2
37	4.9	3.1	1.5	0.1
52	6.9	3.1	4.9	1.5
98	5.1	2.5	3.0	1.1

```
47          4.6         3.2          1.4         0.2
42          4.4         3.2          1.3         0.2
46          5.1         3.8          1.6         0.2
95          5.7         3.0          4.2         1.2
138         6.0         3.0          4.8         1.8
148         6.2         3.4          5.4         2.3
0           5.1         3.5          1.4         0.2
34          4.9         3.1          1.5         0.1
136         6.3         3.4          5.6         2.4
84          5.4         3.0          4.5         1.5
111         6.4         2.7          5.3         1.9
45          4.8         3.0          1.4         0.3
53          5.5         2.3          4.0         1.3
5           5.4         3.9          1.7         0.4
17          5.1         3.5          1.4         0.3
91          6.1         3.0          4.6         1.4
```

[45]:
```
y_train
```

[45]:
```
22     1
8      1
63     2
108    3
115    3
      ..
19     1
68     2
89     2
70     2
88     2
Name: species, Length: 120, dtype: int64
```

[47]:
```
y_test
```

[47]:
```
83     2
81     2
41     1
6      1
57     2
110    3
4      1
49     1
117    3
100    3
10     1
37     1
52     2
```

```
98     2
47     1
42     1
46     1
95     2
138    3
148    3
0      1
34     1
136    3
84     2
111    3
45     1
53     2
5      1
17     1
91     2
Name: species, dtype: int64
```

[51]:
```
from sklearn.naive_bayes import GaussianNB
model =GaussianNB()
```

[53]:
```
model
```

[53]:
```
GaussianNB()
```

[55]:
```
model.fit(x_train,y_train)
```

[55]:
```
GaussianNB()
```

[57]:
```
y_pred=model.predict(x_test)
```

[59]:
```
y_pred
```

[59]:
```
array([3, 2, 1, 1, 2, 3, 1, 1, 3, 3, 1, 1, 3, 2, 1, 1, 1, 2, 3, 3, 1, 1,
       3, 2, 3, 1, 2, 1, 1, 2])
```

[61]:
```
model.score(x_test,y_pred)
```

[61]:
```
1.0
```

[63]:
```
from sklearn.metrics import mean_absolute_error
sk.metrics.mean_squared_error(y_test,y_pred)
```

[63]:
```
0.06666666666666667
```

[71]:
```
mse=mean_absolute_error(y_test,y_pred)
```

[73]:
```
mse
```

[73]: 0.06666666666666667

[75]:
```
import numpy as np
```

[77]:
```
np.sqrt(mse)
```

[77]: 0.2581988897471611

[79]:
```
from sklearn.metrics import accuracy_score
acc =accuracy_score(y_test,y_pred)
```

[145]:
```
print("Accuracy Of Gaussian Model is:",acc*100,"%")
```

```
Accuracy Of Gaussian Model is: 90.0 %
```

[91]:
```
Error_rate = 1-acc
```

[97]:
```
print("Error_rate is:",Error_rate*100,"%")
```

```
Error_rate is: 6.666666666666665 %
```

[101]:
```
from sklearn.metrics import precision_score
pr=precision_score(y_test,y_pred,average="macro")
```

[103]:
```
pr
```

[103]: 0.9259259259259259

[105]:
```
from sklearn.metrics import recall_score
re=recall_score(y_test,y_pred,average="macro")
```

[107]:
```
re
```

[107]: 0.9259259259259259

[113]:
```
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
```

[115]:
```
print(cr)
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        14
           2       1.00      0.78      0.88         9
           3       0.78      1.00      0.88         7
```

```
    accuracy                           0.93        30
   macro avg       0.93      0.93      0.92        30
weighted avg       0.95      0.93      0.93        30
```

[167]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
```

[169]:
```
cm
```

[169]:
```
array([[14,  0,  0],
       [ 0,  6,  3],
       [ 0,  0,  7]])
```

[171]:
```
cm.ravel()
```

[171]:
```
array([14,  0,  0,  0,  6,  3,  0,  0,  7])
```

[ ]:
```

```

Multinomial

[202]:
```
from sklearn.naive_bayes import MultinomialNB
model1 =MultinomialNB()
```

[204]:
```
model1
```

[204]:
```
MultinomialNB()
```

[206]:
```
model1.fit(x_train,y_train)
```

[206]:
```
MultinomialNB()
```

[208]:
```
y_pred=model1.predict(x_test)
```

[210]:
```
y_pred
```

[210]:
```
array([3, 2, 1, 1, 2, 3, 1, 1, 3, 3, 1, 1, 2, 2, 1, 1, 1, 2, 3, 3, 1, 1,
       3, 3, 3, 1, 3, 1, 1, 2])
```

[212]:
```
model1.score(x_test,y_pred)
```

[212]:
```
1.0
```

[214]:
```
from sklearn.metrics import mean_absolute_error
sk.metrics.mean_squared_error(y_test,y_pred)
```

[214]:
```
0.1
```

[216]:
```
mse=mean_absolute_error(y_test,y_pred)
```

[218]:
```
mse
```

[218]: 0.1

[220]:
```
np.sqrt(mse)
```

[220]: 0.31622776601683794

[222]:
```
from sklearn.metrics import accuracy_score
acc =accuracy_score(y_test,y_pred)
```

[224]:
```
print("Accuracy Of Multinomial Model is:",acc*100,"%")
```

```
Accuracy Of Multinomial Model is: 90.0 %
```

[226]:
```
Error_rate = 1-acc
```

[228]:
```
print("Error_rate is:",Error_rate*100,"%")
```

```
Error_rate is: 9.999999999999998 %
```

[230]:
```
from sklearn.metrics import precision_score
pr=precision_score(y_test,y_pred,average="macro")
```

[232]:
```
pr
```

[232]: 0.9

[234]:
```
from sklearn.metrics import recall_score
re=recall_score(y_test,y_pred,average="macro")
```

[236]:
```
re
```

[236]: 0.8888888888888888

[238]:
```
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
```

[240]:
```
print(cr)
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        14
           2       1.00      0.67      0.80         9
           3       0.70      1.00      0.82         7
```

```
    accuracy                           0.90        30
   macro avg       0.90      0.89      0.87        30
weighted avg       0.93      0.90      0.90        30
```

[242]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
```

[244]:
```
cm
```

[244]:
```
array([[14,  0,  0],
       [ 0,  6,  3],
       [ 0,  0,  7]])
```

[246]:
```
cm.ravel()
```

[246]:
```
array([14,  0,  0,  0,  6,  3,  0,  0,  7])
```

[260]:
```
from sklearn.naive_bayes import BernoulliNB
model2 =BernoulliNB()
```

[264]:
```
model2
```

[264]:
```
BernoulliNB()
```

[266]:
```
model2.fit(x_train,y_train)
```

[266]:
```
BernoulliNB()
```

[268]:
```
y_pred=model2.predict(x_test)
```

[256]:
```
y_pred
```

[256]:
```
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3])
```

[270]:
```
model2.score(x_test,y_pred)
```

[270]:
```
1.0
```

[272]:
```
from sklearn.metrics import mean_absolute_error
sk.metrics.mean_squared_error(y_test,y_pred)
```

[272]:
```
2.1666666666666665
```

[274]:
```
mse=mean_absolute_error(y_test,y_pred)
```

[276]:
```
mse
```

[276]: 1.2333333333333334

[278]:
```
np.sqrt(mse)
```

[278]: 1.1105554165971787

[280]:
```
from sklearn.metrics import accuracy_score
acc =accuracy_score(y_test,y_pred)
```

[282]:
```
print("Accuracy Of Bernoulli Model is:",acc*100,"%")
```

```
Accuracy Of Bernoulli Model is: 23.333333333333332 %
```

[284]:
```
Error_rate = 1-acc
```

[286]:
```
print("Error_rate is:",Error_rate*100,"%")
```

```
Error_rate is: 76.66666666666666 %
```

[288]:
```
from sklearn.metrics import precision_score
pr=precision_score(y_test,y_pred,average="macro")
```

```
/home/admin1/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[294]:
```
pr
```

[294]: 0.07777777777777778

[290]:
```
from sklearn.metrics import recall_score
re=recall_score(y_test,y_pred,average="macro")
```

[292]:
```
re
```

[292]: 0.3333333333333333

[296]:
```
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
```

```
/home/admin1/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
/home/admin1/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/admin1/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[298]:
```
print(cr)
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        14
           2       0.00      0.00      0.00         9
           3       0.23      1.00      0.38         7

    accuracy                           0.23        30
   macro avg       0.08      0.33      0.13        30
weighted avg       0.05      0.23      0.09        30
```

[300]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
```

[302]:
```
cm
```

[302]:
```
array([[ 0,  0, 14],
       [ 0,  0,  9],
       [ 0,  0,  7]])
```

[304]:
```
cm.ravel()
```

[304]:
```
array([ 0,  0, 14,  0,  0,  9,  0,  0,  7])
```

[ ]:
```

```