

INDEX**PSIT303A: Machine Learning**

Sr. No.	Title/Aim	Date	Sign
1	Design a simple machine learning model to train the training instances and test the same.	30/08/2023	
2	Implement and demonstrate the find-s algorithm for finding the most specific.	08/09/2023	
3	Support Vector Machine Algorithm for Multiclass classification using Iris.csv & wine dataset from sklearn.	15/09/2023	
4	For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate-elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.	27/09/2023	
5	Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering few test data sets.	04/10/2023	
6	Decision Tree classifier & Random Forest Classifier.	13/10/2023	
7	Data loading, feature scoring and ranking, feature selection (principal component analysis).	09/11/2023	
8	A) For a given set of training data examples stored in a .csv file implement Least Square Regression algorithm. B) For a given set of training data examples stored in a .csv file implement Logistic Regression algorithm.	09/11/2023	
9	A) Build an artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. B) Perform Text pre-processing, Text clustering, classification with Prediction, Test Score and Confusion Matrix.	29/11/2023	
10	A) Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix. B) Implement the classification model using K means clustering with Prediction, Test Score and Confusion Matrix.	01/12/2023	

Practical No: 1**Date:****Design the machine learning model.****Aim:** Design a simple machine learning model to train the training instances and test the same.**Description:****1. Training Data:**

Training data is the subset of original data that is used to train the machine learning model.

The first 80 data points are used for training (train_x and train_y).

```
train_x = x[:80] train_y  
= y[:80]
```

2. Test Data:

Testing data is used to check the accuracy of the model.

The remaining 20 data points are used for testing (test_x and test_y).

```
test_x = x[80:] test_y  
= y[80:]
```

The train_test_split function from sklearn.model_selection is used to split the data.

It randomly shuffles the data and splits it into training and testing sets.

test_size=0.3 specifies that 30% of the data should be used for testing, and the remaining 70% for training.

train_x, test_x, train_y, and test_y are assigned accordingly.

3. Learning:

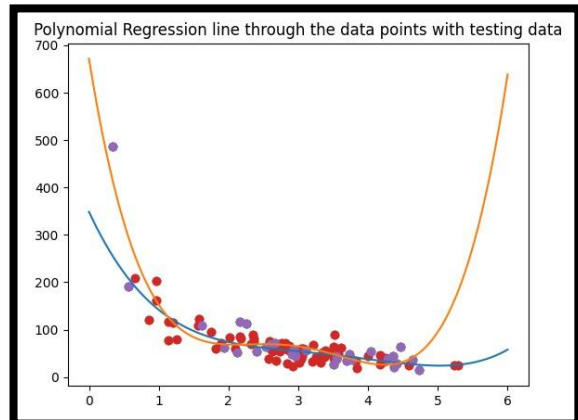
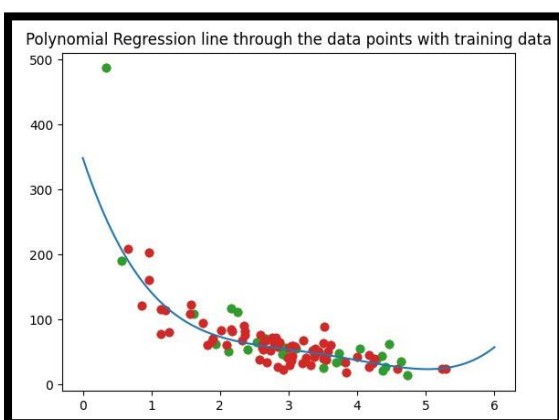
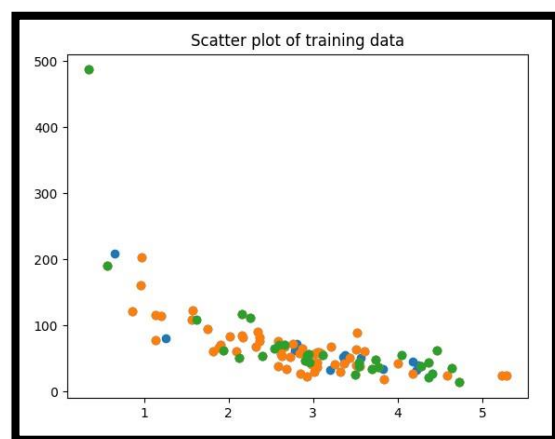
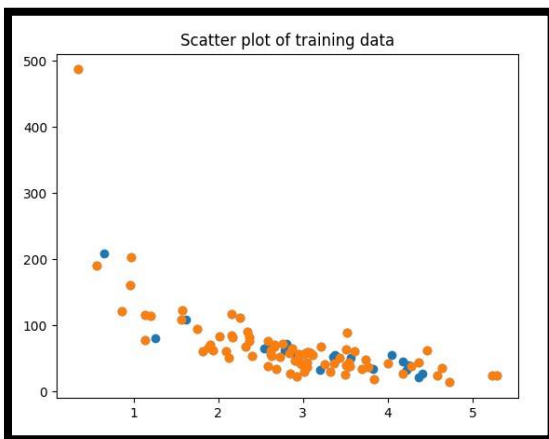
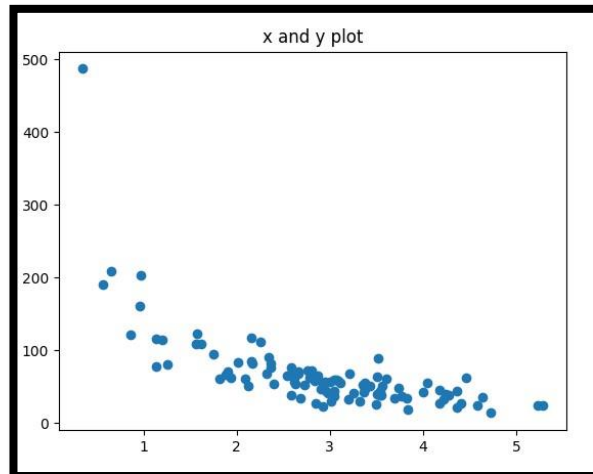
- Polynomial regression of degree 4 was applied to capture complex patterns.
- Plots showcased how well the polynomial curve aligned with the data points.
- R2 score evaluated model fit on training data, quantifying its performance.
- Model predictions demonstrated its capability to estimate outcomes.
- Data was split into training and testing sets for model assessment.
- Plots with regression lines displayed how well the model generalized to new data.
- The code illustrated data manipulation, polynomial regression, and visualization for model understanding and evaluation.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2)
from sklearn.model_selection import train_test_split
x=np.random.normal(3,1,100)
print(x)
y=np.random.normal(150,40,100)/x
print(y)
plt.scatter(x, y)
plt.show()
train_x=x[:80]
train_y=y[:80]
test_x=x[80:]
test_y=y[80:]
print(train_x, train_y, test_x, test_y)
plt.scatter(train_x, train_y)
plt.show()
train_x, train_y, test_x, test_y=train_test_split(x, y, test_size=0.3)
plt.scatter(test_x, test_y)
plt.show()
#Draw polynomial Regression line through the data points with training data
mymodel=np.poly1d(np.polyfit(train_x, train_y, 4))
myline=np.linspace(0, 6, 100)
plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
mymodel=np.poly1d(np.polyfit(test_x, test_y, 4))
myline=np.linspace(0, 6, 100)
plt.scatter(test_x, test_y)
plt.plot(myline, mymodel(myline))
plt.show()
#Measures the relationship between x and y axis
from sklearn.metrics import r2_score
numpy.random.seed(2)
r2=r2_score(train_y, mymodel(train_x))
print(r2)
#Predict the values
print(mymodel(5))
```

Output:

```
[2.58324215 2.94373317 0.8635039 4.44027051 1.20656441 2.15825263
3.50288142 1.75471191 1.94204778 2.09099239 3.55145404 5.29220801
3.04153939 1.88207455 3.53905832 2.4038403 2.9808695 4.17500122
2.25212905 3.00902525 2.12189211 2.84356583 3.25657045 2.01122095
2.66117803 2.76381597 2.36234499 1.81238771 1.57878277 2.8465048
2.73094304 5.23136679 0.56523242 3.1127265 3.37044454 4.35963386
3.50185721 2.1557863 3.00000976 3.54235257 2.4864918 3.77101174
1.13190935 4.73119467 4.46767801 2.46432266 3.61134078 3.04797059
2.17086471 3.08771022 4.00036589 2.61890748 2.62433058 2.92552924
3.43349633 4.27837923 2.36532069 3.50839624 3.21611601 1.14138761
2.58068352 2.8676711 2.96042976 3.32600343 0.95967695 3.04625552
2.32232442 1.56056097 3.52429643 3.73527958 2.34674973 3.84245628
2.61848352 3.06648901 1.90126105 4.58448706 0.34055054 2.90854738
3.69511961 0.94653345 2.81053074 2.92278133 3.02470301 4.24821292
2.59610773 1.61548133 4.36723542 4.21788563 2.53799465 3.35088845
3.38186623 3.56627544 3.20420798 4.40669624 1.2620405 4.04082395
3.38047197 2.78286473 4.1735315 0.65639681]
[ 76.05204933 56.20180641 121.17874037 36.05903817 114.23885932
117.41526024 63.77986643 95.52998052 62.4237197 60.57574247
38.57519009 24.10914678 37.45148152 67.13926856 39.26245343
53.79918302 40.94657678 27.02857247 111.90190427 30.26643537
51.4368334 58.83311239 42.08623741 83.01076429 68.37843958
72.54627253 76.22874513 60.83111238 123.11113005 27.89501382
53.25015791 24.86406278 190.30762228 55.79245737 42.32964984
43.76381026 25.90093643 85.28325651 56.63901768 43.77321677
34.70979433 37.10649657 77.86225629 14.08666443 62.93869329
70.87521926 41.39097018 43.58292288 81.92492065 57.61442568
43.5111781 57.3316853 53.67848811 22.97550427 50.79538368
39.01941998 82.32095959 39.62788318 68.30365792 115.73628743
38.66530343 65.39332448 44.34023444 30.00934597 161.50533328
59.1743156 68.74904453 108.8692008 89.19445659 48.95077634
90.02681869 18.36485932 62.86162946 59.01318439 71.22685026
25.07604874 487.03726791 47.24533754 34.16662793 202.76589695]
```



```
R2 Score: 0.1983529435994391
95.12966899799278
```

Learning:

- i. Using NumPy, generate a set of 100 random numbers following a normal distribution with mean 3 and standard deviation 1, assigned to variable x. You also generate another set of 100 random numbers following a normal distribution with mean 150 and standard deviation 40, divided element-wise by x, assigned to variable y.
- ii. Split the data into training and testing sets. train_x and train_y contain the first 80 elements of x and corresponding y, while test_x and test_y contain the next 20 elements of x and corresponding y.
- iii. Create a scatter plot to visualize the training data.
- iv. Split the data into training and testing sets again, this time using train_test_split from sklearn.model_selection. 70% of the data is used for training (train_x, train_y), and 30% for testing (test_x, test_y).
- v. Fit a polynomial regression model of degree 4 (polyfit) on the training data (train_x, train_y). Create a set of x-values (myline) to draw the fitted curve and then plot the training data points along with the polynomial regression line.
- vi. Similarly, fit a polynomial regression model of degree 4 on the testing data (test_x, test_y) and plot the testing data points along with the polynomial regression line.
- vii. You calculate the R-squared score (r2_score) to measure the goodness of fit of the polynomial regression model on the training data. It tells us how well the model fits the variability of the data.
- viii. Use the trained polynomial regression model to predict the y-value for a given x-value of 5.

Practical No: 2**Date:****Concept learning****Aim:** Implement and demonstrate the find-s algorithm for finding the most specific.**Description:****Training dataset table (input data):**

- Make .csv datafile including 6 columns + 1 column whether a boy enjoy sport or not
- EnjoySportOrNot.csv file is a training dataset, it contains parameters for Sky, AirTemp, Humidity, Wind, Water, Forecast, EnjoySport
- Training dataset table contains combinations of rows and columns to teach a model for predicting weather report(enjoy sport or not).

EnjoySportOrNot.csv

	A	B	C	D	E	F	G
1	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes
3	Sunny	Warm	High	Strong	Warm	Same	Yes
4	Rainy	Cold	High	Strong	Warm	Change	No
5	Sunny	Warm	High	Strong	Cool	Change	Yes
6							

2. Write the right hypothesis/function from historical data

- Right hypothesis is a tentative assumption made to draw out and test its logical.
- H_0 is a null statement which is to be tested.

3. How Does It Work?

- Initialize H
- Identify a positive sample(rows), where predicting “Yes” for enjoying a sport or not.
- Define x_1 and h_1
- Check for attributes in x_1 and h_1
- If $\text{Attributevalue}(x_1, x_2, \dots)$ is equal to $\text{hypothesis}(h_1, h_2, \dots)$ value, then take that as a final

- If Attributevalue(x1, x2,...) is equal to hypothesis(h1, h2,...) value, then replace the value by “?”

Code: import csv

num_attributes=6 a=[]

print("\n Given dataset: \n")

with

open(r'C:\Users\admin\Downloads\EnjoySportOrNot\EnjoySportOrNot.csv', 'r')

as csvfile: reader=csv.reader(csvfile) count=0 for row in reader: if

count==0: print(row) count+=1 else:

a.append(row) print(row)

count+=1 print("\n The initial value of

hypothesis: ") hyp=["0"]*num_attributes

print(hyp) for j in range(0,

num_attributes):

hyp[j]=a[0][j] print(hyp) print("\n Find S: finding a
maximally specific hypothesis \n") for i in range(0, len(a)):

if a[i][num_attributes]=='Yes':

for j in range(0, num_attributes):

if a[i][j]!=hyp[j]:

hyp[j]='?'

else:

hyp[j]=a[i][j] print(" For training example no: {0} the
hypothesis is ".format(i), hyp) print(hyp)

Output:

```

Given dataset:

['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
['Sunny', '0', '0', '0', '0', '0']
['Sunny', 'Warm', '0', '0', '0', '0']
['Sunny', 'Warm', 'Normal', '0', '0', '0']
['Sunny', 'Warm', 'Normal', 'Strong', '0', '0']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', '0']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

Find S: finding a maximally specific hypothesis

For training example no: 0 the hypothesis is ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
For training example no: 1 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For training example no: 2 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For training example no: 3 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
['Sunny', 'Warm', '?', 'Strong', '?', '?']

```

Input Dataset table:

BookBuyOrNot.csv

	A	B	C	D	E	F	G
1	citations	size	inLibrary	price	editions	buy	
2	some	small	no	affordable	many	no	
3	many	big	no	expensive	one	yes	
4	some	big	always	expensive	few	no	
5	many	medium	no	expensive	many	yes	
6	many	small	no	affordable	many	yes	
7							

Code: import csv

num_attributes=5 a=[]

print("\n Given dataset: \n")

with

open(r"C:\Users\admin\Downloads\EnjoySportOrNot\BookBuyOrNot.csv", 'r')

as csvfile: reader=csv.reader(csvfile) count=0 for row in reader: if

count==0: print(row) count+=1 else:


```

        a.append(row)        print(row)
count+=1 print("\n The initial value of
hypothesis: ") hyp=["0"]*num_attributes
print(hyp) for j in range(0,
num_attributes):
    hyp[j]=a[0][j]    print(hyp) print("\n Find S: finding a
maximally specific hypothesis \n") for i in range(0, len(a)):
    if a[i][num_attributes]=='yes':
for j in range(0, num_attributes):
if a[i][j]!=hyp[j]:

    hyp[j]='?'
else:

    hyp[j]=a[i][j]    print(" For training example no: {0} the
hypothesis is ".format(i), hyp) print("\nOutput: ") print(hyp)

```

Output:

```

Given dataset:

['citations', 'size', 'inLibrary', 'price', 'editions', 'buy']
['some', 'small', 'no', 'affordable', 'many', 'no']
['many', 'big', 'no', 'expensive', 'one', 'yes']
['some', 'big', 'always', 'expensive', 'few', 'no']
['many', 'medium', 'no', 'expensive', 'many', 'yes']
['many', 'small', 'no', 'affordable', 'many', 'yes']

The initial value of hypothesis:
['0', '0', '0', '0', '0']
['some', '0', '0', '0', '0']
['some', 'small', '0', '0', '0']
['some', 'small', 'no', '0', '0']
['some', 'small', 'no', 'affordable', '0']
['some', 'small', 'no', 'affordable', 'many']

Find S: finding a maximally specific hypothesis

For training example no: 0 the hypothesis is ['some', 'small', 'no', 'affordable', 'many']
For training example no: 1 the hypothesis is ['?', '?', 'no', '?', '?']
For training example no: 2 the hypothesis is ['?', '?', 'no', '?', '?']
For training example no: 3 the hypothesis is ['?', '?', 'no', '?', '?']
For training example no: 4 the hypothesis is ['?', '?', 'no', '?', '?']

Output:
['?', '?', 'no', '?', '?']

```

Learning:

- The dataset consists of seven columns: 'Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', and 'EnjoySport'.
- Each row represents a training example, where the first row is the column header, and subsequent rows contain attribute values and the target attribute 'EnjoySport'.
- For each example, the algorithm checks if the 'EnjoySport' attribute is "Yes," indicating a positive example.
- If the example is positive, it compares each attribute in the example with the current hypothesis.
- If an attribute in the example differs from the hypothesis, it is replaced with a '?'.
- If an attribute in the example matches the hypothesis, it is retained in the hypothesis. → After processing all training examples, the final hypothesis is:

['Sunny', 'Warm', '?', 'Strong', 'Warm', '?'].

The final hypothesis suggests that certain conditions (e.g., 'Sunny' sky, 'Warm' air temperature, 'Strong' wind) are associated with the enjoyment of a sport. The '?' symbolizes uncertainty.

Practical No: 3**Date:****Multi-class classification Aim: Support vector machine algorithm for multiclass classification using iris.csv and wine dataset from sklearn****Description:**

Confusion matrix is a way of assessing the performance of a classification model. It is a comparison between the ground truth (actual values) and the predicted values emitted by the model for the target variable.

A confusion matrix is useful in the [supervised learning](#) category of machine learning using a labelled data set. As shown below, it is represented by a table

True Positive (TP) is an outcome where the model correctly predicts the positive class.

True Negative (TN) is an outcome where the model correctly predicts the negative class.

False Positive (FP) is an outcome where the model incorrectly predicts the positive class.

False Negative (FN) is an outcome where the model incorrectly predicts the negative class.

Code:

```
from sklearn import svm, datasets
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, train_size=0.80,
test_size=0.20, random_state=101)
rbf = svm.SVC(kernel='rbf', gamma=0.5, C=0.1).fit(X_train, y_train)
poly = svm.SVC(kernel='poly', degree=3, C=1).fit(X_train, y_train)
poly_pred = poly.predict(X_test)
rbf_pred = rbf.predict(X_test)
poly_accuracy = accuracy_score(y_test, poly_pred)
poly_f1 = f1_score(y_test, poly_pred, average='weighted')
print('Accuracy (Polynomial Kernel): ', "%.2f" % (poly_accuracy*100))
print('F1 (Polynomial Kernel): ', "%.2f" % (poly_f1*100))
rbf_accuracy = accuracy_score(y_test, rbf_pred)
rbf_f1 = f1_score(y_test, rbf_pred, average='weighted')
print('Accuracy (RBF Kernel): ', "%.2f" % (rbf_accuracy*100))
print('F1 (RBF Kernel): ', "%.2f" % (rbf_f1*100))
```

Output:

```
>>> ===== RESTART: C:/Users/admin/Downloads/IrisCm-P3-1.py =====
Accuracy (Polynomial Kernel): 70.00
F1 (Polynomial Kernel): 69.67
Accuracy (RBF Kernel): 76.67
F1 (RBF Kernel): 76.36
>>>
```

For wine dataset from sklearn: Code:

```
from sklearn import svm, datasets
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score from
sklearn.metrics import f1_score wine =
datasets.load_wine() X = wine.data[:, :2] y =
wine.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, train_size=0.80,
test_size=0.20, random_state=101)
rbf = svm.SVC(kernel='rbf', gamma=0.5, C=0.1).fit(X_train, y_train)
poly = svm.SVC(kernel='poly', degree=3, C=1).fit(X_train, y_train)
poly_pred = poly.predict(X_test) rbf_pred = rbf.predict(X_test)
poly_accuracy = accuracy_score(y_test, poly_pred) poly_f1 =
f1_score(y_test, poly_pred, average='weighted') print('Accuracy
(Polynomial Kernel): ', "%.2f" % (poly_accuracy*100)) print('F1
(Polynomial Kernel): ', "%.2f" % (poly_f1*100)) rbf_accuracy =
accuracy_score(y_test, rbf_pred) rbf_f1 = f1_score(y_test, rbf_pred,
average='weighted') print('Accuracy (RBF Kernel): ', "%.2f" %
(rbf_accuracy*100)) print('F1 (RBF Kernel): ', "%.2f" % (rbf_f1*100))
```

Output:

The screenshot shows a Jupyter Notebook interface. The top part displays the first five rows of a dataset with 14 features: Wine, Alcohol, Malic.acid, Ash, Alc1, Mg, Phenols, Flavanoids, Nonflavanoid.phenols, Proanth, Color.int, Hue, OD, and Proline. The bottom part shows the data types for these features.

	Wine	Alcohol	Malic.acid	Ash	Alc1	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.05	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Feature	dtype
Wine	int64
Alcohol	float64
Malic.acid	float64
Ash	float64
Alc1	float64
Mg	int64
Phenols	float64
Flavanoids	float64
Nonflavanoid.phenols	float64
Proanth	float64
Color.int	float64
Hue	float64
OD	float64

Learning:

1. The code loads the wine dataset, selects the first two features, and splits it into training and testing sets.
2. It utilizes Support Vector Machines (SVM) with Polynomial and Radial Basis Function (RBF) kernels for classification.
3. The models predict the test set, and accuracy along with F1 scores are calculated for both Polynomial and RBF kernels.
4. The code prints the accuracy and F1 scores for each kernel type, providing insights into the models' performance.

Practical No: 4**Date:****Candidate-elimination algorithm**

Aim: For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate-elimination algorithm to output a description of the set of all hypotheses consistent with the training examples

Description:

- i. Candidate Elimination Learning Algorithm is a method for learning concepts from data that is supervised.
- ii. The Candidate Elimination Algorithm (CEA) is an improvement over the Find-S algorithm for classification tasks.
- iii. CEA considers both positive and negative examples to generate the hypothesis, which can result in higher accuracy when dealing with noisy data.
 - 1) General Hypothesis:
 - Not Specifying features to learn the machine.
 - $G = \{ '?', '?', '?', '?' \dots \}$: Number of attributes
 - 2) Specific Hypothesis:
 - Specifying features to learn machine (Specific feature)
 - $S = \{ 'pi', 'pi', 'pi' \dots \}$: The number of pi depends on a number of attributes.
 - 3) Version Space:
 - It is an intermediate of general hypothesis and Specific hypothesis.
 - It not only just writes one hypothesis but a set of all possible hypotheses based on training data-set.
 - iv. Examples, recognizing spam emails or classifying customer behavior, predicting house prices or estimating stock prices.

Code:

```
import numpy as np
import pandas as pd

#Loading data from a csv file. data =
pd.DataFrame(data=pd.read_csv('enjoysport.csv'))

print(data)
```

```
>>>
= RESTART: C:/Users/Aditi/OneDrive/Documents/Python Scripts/ML_Prac4_
sky air_temp humidity wind water forecast enjoy_sport
0 sunny warm normal strong warm same yes
1 sunny warm high strong warm same yes
2 rainy cold high strong warm change no
3 sunny warm high strong cool change yes
```

```
#Separating concept features from Target
```

```
concepts = np.array(data.iloc[:,0:6])
```

```
print(concepts)
```

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

```
#Isolating target into a separate DataFrame
```

```
#Copying last column to target array target
```

```
= np.array(data.iloc[:,6]) print(target)
```

```
['yes' 'yes' 'no' 'yes']
```

```
def learn(concepts, target):
```

```
#Initialise S0 with the first instance from concepts.
```

```
#.copy()makes sure a new list is created instead of just pointing to the same memory location.
```

```
specific_h = concepts[0].copy()
```

```
print("\nInitialization of specific_h and general_h")
```

```
print("\nSpecific Boundary: ", specific_h)
```

```
general_h = [['?' for i in range(len(specific_h))] for i in range(len(specific_h))]
```

```
print("\nGeneric Boundary: ",general_h)
```

```
Initialization of specific_h and general_h
Specific Boundary: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

```
# The learning iterations.    for i, h
```

```
in enumerate(concepts):
```

```
print("\nInstance", i+1 , "is ", h)
```

```
# Checking if the hypothesis has a positive target.
```

```
if target[i] == "yes":
```

```
    print("Instance is Positive ")
```

```
    for x in range(len(specific_h)):
```

Change values in S & G only if values change.

```

        if h[x] != specific_h[x]:
specific_h[x] = '?'                                general_h[x][x]
                                                    = '?'

```

Checking if the hypothesis has a positive target.

```
if target[i] == "no":
```

```
    print("Instance is Negative ")
```

```
for x in range(len(specific_h)):
```

For negative hypothesis change values only in G.

```

        if h[x] != specific_h[x]:
            general_h[x][x] = specific_h[x]
        else:
            general_h[x][x] = '?'          print("Specific Boundary after ",
i+1, "Instance is ", specific_h)          print("Generic Boundary after ",
i+1, "Instance is ", general_h) # find indices where we have empty rows,
meaning those that are unchanged.

```

```
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
```

```
for i in indices:
```

```
# remove those rows from general_h
```

```
    general_h.remove(['?', '?', '?', '?', '?', '?'])
```

```
# Return final values    return
```

```
specific_h, general_h s_final, g_final =
```

```
learn(concepts, target) print("Final
```

```
Specific_h: ", s_final, sep="\n")
```

```
print("Final General_h: ", g_final, sep="\n")
```


Output:

```
Initialization of specific_h and general_h

Specific Boundary: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Boundary after 1 Instance is ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Boundary after 2 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

>
```

```
Instance 3 is ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Boundary after 3 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after 3 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
 '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', 'same']]

Instance 4 is ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Boundary after 4 Instance is ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after 4 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
 '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?']]
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

>
```

Learnings:

1. Initialize the specific hypothesis (specific_h) with the first instance in the dataset.
2. Iterate through each instance in the dataset and adjust specific_h and general hypothesis (general_h) based on positive and negative instances.
3. If an instance is positive, update specific_h and reset inconsistent values in general_h and if an instance is negative, update general_h while keeping specific_h unchanged.
4. Dynamically adjust hypotheses based on each instance to accurately classify positive and negative cases.
5. Handle changes in specific_h and general_h to adapt to the dataset's patterns.
6. Identify rows in general_h that remain unchanged (no adjustment in values) and remove these unchanged rows from general_h to eliminate redundant information.
7. Return the final specific_h and general_h as the learned hypotheses.
8. These hypotheses capture patterns in the dataset and aid in classifying future instances, showcasing an incremental learning approach.

Practical No: 5**Date:****Naïve Bayes & Gaussian Classification**

Aim: Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Description:**Naïve Bayesian classifier:**

- It is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- It is a probabilistic classifier, means it predicts based on the probability of an object.
- Examples: Spam filtration, Sentimental analysis, and classifying articles.

Gaussian Classifier:

- It is a classification technique used in Machine Learning (ML) based on the probabilistic approach and Gaussian distribution.
- Gaussian classifier assumes that each parameter (also called features or predictors) has an independent capacity of predicting the output variable.
- The combination of the prediction for all parameters is the final prediction, that returns a probability of the dependent variable to be classified in each group.
- The final classification is assigned to the group with the higher probability.

Code:

```
import pandas as pd from
sklearn import tree
from sklearn.preprocessing import LabelEncoder from
sklearn.naive_bayes import GaussianNB

data=pd.read_csv('tennisdata.csv')
print("The first 5 values of data is: \n", data.head())

x=data.iloc[:, :-1]
print("\nThe First 5 values of train data is: \n", x.head())
```

```
y = data.iloc[:,-1]
print("\nThe first 5 values of Train output is: \n",y.head())

le_outlook = LabelEncoder()
x.Outlook = le_outlook.fit_transform(x.Outlook)

le_Temperature = LabelEncoder()
x.Temperature = le_Temperature.fit_transform(x.Temperature)

le_Humidity = LabelEncoder()
x.Humidity = le_Humidity.fit_transform(x.Humidity)

le_Windy = LabelEncoder()
x.Windy = le_Windy.fit_transform(x.Windy)

print("\nNow the Train data is : \n",x.head())

le_PlayTennis = LabelEncoder() y =
le_PlayTennis.fit_transform(y)
print("\nNow the Train output is: \n",y)

from sklearn.model_selection import train_test_split x_train,
x_test, y_train, y_test = train_test_split(x,y, test_size=0.20)

classifier = GaussianNB() classifier.fit(x_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is: ",accuracy_score(classifier.predict(x_test),y_test))
```

Output:

```

>>>
===== REST
/Downloads/Naive_Bayes_ML_P5.py =====
=====
The first 5 values of data is:
      Outlook Temperature Humidity Windy PlayTennis
0      Sunny          Hot      High  False         No
1      Sunny          Hot      High   True         No
2  Overcast          Hot      High  False         Yes
3      Rainy          Mild      High  False         Yes
4      Rainy          Cool     Normal False         Yes

The First 5 values of train data is:
      Outlook Temperature Humidity Windy
0      Sunny          Hot      High  False
1      Sunny          Hot      High   True
2  Overcast          Hot      High  False
3      Rainy          Mild      High  False
4      Rainy          Cool     Normal False

The first 5 values of Train output is:
0      No
1      No
2      Yes
3      Yes
4      Yes
Name: PlayTennis, dtype: object

Now the Train data is :
      Outlook Temperature Humidity Windy
0           2           1           0           0
1           2           1           0           1
2           0           1           0           0
3           1           2           0           0
4           1           0           1           0

Now the Train output is:
[0 0 1 1 1 0 1 0 1 1 1 1 0]
Accuracy is:  1.0
>>>

```

Analysis of Confusion Matrix

```

y_pred = classifier.predict(x_test) confusion
= confusion_matrix(y_test, y_pred)
print("Confusion Matrix: \n", confusion)

```

```

Confusion Matrix:
[[0 0]
 [1 2]]
>>>

```

Confusion matrix values: TP= 0, TN=0, FP=1, FN=2

Learnings:

- 1) Load a CSV file named 'tennisdata.csv' into a Pandas DataFrame called 'data'.
- 2) Print the first 5 rows of the loaded data to get a glimpse of its content.
- 3) Extract the feature variables (attributes) from the DataFrame 'data' and assign them to the variable 'x'.
- 4) Extract the target variable (output) from the DataFrame 'data' and assign it to the variable 'y'.
- 5) Encode categorical features in 'x' (Outlook, Temperature, Humidity, Windy) into numerical values using LabelEncoder.
- 6) Encode the target variable 'y' (PlayTennis) into numerical values using LabelEncoder.
- 7) Split the data into training and testing sets, with 80% for training and 20% for testing, using the 'train_test_split' function from Scikit-Learn.
- 8) Create a Gaussian Naive Bayes classifier ('classifier'), fit it to the training data, and train the model.
- 9) Calculate and print the accuracy of the model by comparing its predictions on the test data with the actual test labels.

Practical No: 6**Date:****Design the machine learning model.**

Aim: Design a simple machine learning model to train the training instances and test the same.

Description:**Decision Tree Classifier:-**

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- Decision trees are a prime example of grouping models.
- In a decision tree, the instance space is recursively divided into segments (nodes), ultimately leading to leaves where decisions are made.
- All instances within a leaf receive the same prediction, which may be the majority class.

Random Forest Classifier:-

- The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement.
- Random forest algorithms have three main hyperparameters, which need to be set before training.
- These include node size, the number of trees, and the number of features sampled. From there, the random forest classifier can be used to solve for regression or classification problems.
- It can be used to evaluate customers with high credit risk, to detect fraud, and option pricing problems.
- The random forest algorithm has applications, allowing doctors to tackle problems such as gene expression classification, biomarker discovery, and sequence annotation.

Decision tree: Code:

```
import pandas as pd
```

```
import numpy as np
```

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

```
import seaborn as sns
```

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

```
%matplotlib inline
```

```

df = pd.read_csv(r"C:\Users\admin\Downloads\WA_Fn-UseC_-HR-Employee-Attrition.csv")
df.head()

# Exploratory Data Analysis

sns.countplot(x='Attrition', data=df) plt.show()

from pandas.core.arrays import categorical

df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'], axis="columns",
inplace=True) categorical_col = [] for column in df.columns: if df[column].dtype ==
object:

    categorical_col.append(column)

df['Attrition'] = df.Attrition.astype("category").cat.codes

from sklearn.preprocessing import LabelEncoder for
column in categorical_col:

    df[column] = LabelEncoder().fit_transform(df[column])

from sklearn.model_selection import train_test_split X =
df.drop('Attrition', axis=1) y = df.Attrition

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

def print_score(clf, X_train, y_train, X_test, y_test, train=True): if train:

    pred = clf.predict(X_train)

    clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))

print("Train Result:\n=====")

print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%") print(" ")

    print(f"CLASSIFICATION REPORT:\n{clf_report}")

print(" ")

    print(f"Confusion Matrix: \n{confusion_matrix(y_train, pred)}\n")

elif train==False:

    pred = clf.predict(X_test)

    clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))

print("Test Result:\n=====")

print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%") print(" ")

    print(f"CLASSIFICATION REPORT:\n{clf_report}")

print(" ")

```

```
print(f'Confusion Matrix: \n{confusion_matrix(y_test, pred)}\n")
from sklearn.tree import DecisionTreeClassifier from pickle import
TRUE
from sklearn.tree import DecisionTreeClassifier

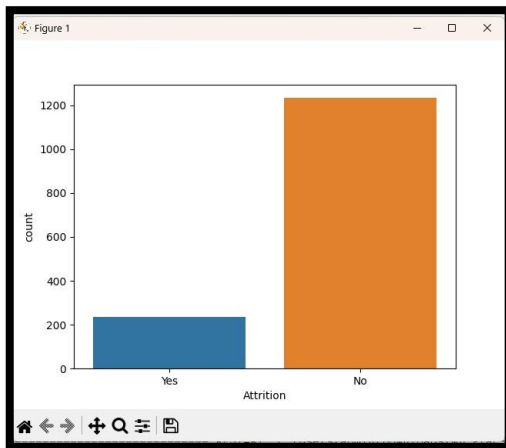
tree_clf = DecisionTreeClassifier(random_state=42) tree_clf.fit(X_train,
y_train)

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True) print_score(tree_clf,
X_train, y_train, X_test, y_test, train=False)

from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=42) rf_clf.fit(X_train,
y_train)

print_score(rf_clf, X_train, y_train, X_test, y_test, train=True) print_score(rf_clf,
X_train, y_train, X_test, y_test, train=False)
```


Output:

```
>>>
===== RESTART: C:/Users/admin/Downloads/ML_Prac-6 Decision Tree.py
=====
Train Result:
Accuracy Score: 100.00%

CLASSIFICATION REPORT:
      0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0      1.0      1.0      1.0
recall       1.0    1.0      1.0      1.0      1.0
f1-score     1.0    1.0      1.0      1.0      1.0
support     853.0  176.0      1.0    1029.0    1029.0

Confusion Matrix:
[[853   0]
 [  0 176]]

Test Result:
=====
Accuracy Score: 77.78%

CLASSIFICATION REPORT:
      0      1  accuracy  macro avg  weighted avg
precision    0.887363  0.259740  0.777778  0.573551  0.800549
recall       0.850000  0.327869  0.777778  0.588534  0.777778
f1-score     0.866280  0.289555  0.777778  0.579047  0.788271
support     380.000000  61.000000  0.777778  441.000000  441.000000

Confusion Matrix:
[[323  57]
 [ 41  20]]
```

```
Train Result:
=====
Accuracy Score: 100.00%

CLASSIFICATION REPORT:
      0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0      1.0      1.0      1.0
recall       1.0    1.0      1.0      1.0      1.0
f1-score     1.0    1.0      1.0      1.0      1.0
support     853.0  176.0      1.0    1029.0    1029.0

Confusion Matrix:
[[853   0]
 [  0 176]]

Test Result:
=====
Accuracy Score: 86.17%

CLASSIFICATION REPORT:
      0      1  accuracy  macro avg  weighted avg
precision    0.871795  0.500000  0.861678  0.685897  0.820367
recall       0.984211  0.098361  0.861678  0.541286  0.861678
f1-score     0.924598  0.164384  0.861678  0.544491  0.819444
support     380.000000  61.000000  0.861678  441.000000  441.000000

Confusion Matrix:
[[374   6]
 [ 55   6]]

>>>
```

Random Forest classifier:**Code:**

```
import pandas as pd
import
```

```
numpy as np
```

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

```
import seaborn as sns
```

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

```
%matplotlib inline
```

```
df = pd.read_csv(r"C:\Users\admin\Downloads\WA_Fn-UseC_-HR-Employee-Attrition.csv")
df.head()
```

```
# Exploratory Data Analysis
```

```
sns.countplot(x='Attrition', data=df)
plt.show()
```

```
from pandas.core.arrays import categorical
```

```
df.drop(['EmployeeCount','EmployeeNumber', 'Over18', 'StandardHours'], axis="columns",
inplace=True) categorical_col = [] for column in df.columns: if df[column].dtype ==
object:
```

```
    categorical_col.append(column)
```

```
df['Attrition'] = df.Attrition.astype("category").cat.codes
```

```
from sklearn.preprocessing import LabelEncoder for
column in categorical_col:
```

```
    df[column] = LabelEncoder().fit_transform(df[column])
```

```
from sklearn.model_selection import train_test_split X =
```

```
df.drop('Attrition', axis=1) y = df.Attrition
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
def print_score(clf, X_train, y_train, X_test, y_test, train=True): if train:
```

```
    pred = clf.predict(X_train)
```

```
    clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
```

```
print("Train Result:\n=====")
```

```
print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%") print(" ")
```

```
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

```
print(" ")
```

```
    print(f"Confusion Matrix: \n{confusion_matrix(y_train, pred)}\n")
```

```
elif train==False:
```

```
    pred = clf.predict(X_test)
```

```
    clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
```

```
print("Test Result:\n=====")
```

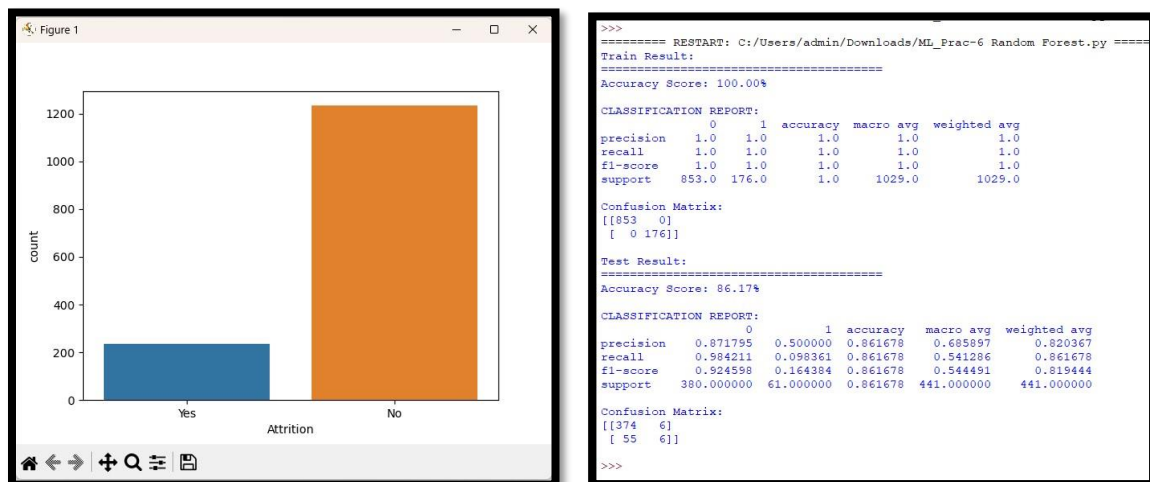
```
print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%") print(" ")
```

```
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

```
print(" ")
```

```
    print(f"Confusion Matrix: \n{confusion_matrix(y_test, pred)}\n")
```

Output:



Learnings:

- 1) This Python script analyzes employee attrition in a Human Resources dataset. After loading the data, it conducts exploratory data analysis, visualizing attrition counts. Irrelevant columns are dropped, and categorical features are encoded.
- 2) The dataset is split into training and testing sets.
- 3) Two machine learning models, Decision Tree and Random Forest classifiers, are trained and evaluated using accuracy scores, classification reports, and confusion matrices.
- 4) The script provides a practical application of machine learning for HR analytics, aiding in predicting and understanding employee attrition factors.
- 5) Organizations can use such models to proactively manage human resources and implement strategies to improve employee retention.

Practical No: 7**Date:****Feature Selection****Aim: Data loading, feature scoring and ranking, feature selection (principal component analysis)****Description:****Principal Component Analysis:**

- i. It is an unsupervised learning algorithm technique used to examine the interrelations among a set of variables.
- ii. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components.
- iii. It works on the condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum. **iv.** Example, image processing, movie recommendation system, etc.

Code:

```
import numpy as np
import pandas as pd
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
dataset = pd.read_csv(url, names=names)
dataset.head()
x = dataset.drop('Class',1)
y = dataset['Class']
y.head()
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train1 = sc.fit_transform(x_train)
x_test1 = sc.transform(x_test)
y_train1 = y_train
y_test1 = y_test
from sklearn.decomposition import PCA
pca = PCA()
x_train1 = pca.fit_transform(x_train1)
x_test1 = pca.transform(x_test1)
explained_variance = pca.explained_variance_ratio_
print("Explained variance: ",
```

```
explained_variance) from sklearn.ensemble import  
  
RandomForestClassifier classifier =  
  
RandomForestClassifier(max_depth=2, random_state=0)  
  
classifier.fit(x_train1,y_train1) y_pred = classifier.predict(x_test1)  
  
#Confusion matrix from sklearn.metrics import  
  
confusion_matrix from sklearn.metrics import  
  
accuracy_score cm = confusion_matrix(y_test,  
y_pred) print(cm) print('Accuracy',  
accuracy_score(y_test1, y_pred))
```

Output:

```
Explained variance: [0.72226528 0.23974795 0.03338117 0.0046056 ]  
[[11  0  0]  
 [ 0 11  2]  
 [ 0  1  5]]  
Accuracy 0.9  
>
```

Learnings:

1. The code loads the Iris dataset, separates features and target variable, and splits the data into training and testing sets.
2. StandardScaler is applied to standardize feature values, ensuring uniform contributions to the model and PCA is used to reduce feature dimensions while retaining maximum variance.
3. A Random Forest classifier with a defined maximum depth is instantiated and classifier is trained on the standardized and dimensionality-reduced training data.
4. The model predicts the target variable for the test set and confusion matrix is generated to evaluate classification performance.
5. The accuracy score is computed to quantify the model's predictive accuracy and explained variance ratio after PCA transformation is printed.
6. The confusion matrix and accuracy score are displayed, providing insights into the model's effectiveness.

Practical No: 8**Date:****Linear & Logistic Regression****8 A) Aim: For a given set of training data examples stored in a CSV. File implement Least Square Regression Algorithm.****Description:****Least Square Method:**

- The least-squares method can be defined as a statistical method that is used to find the equation of the line of best fit related to the given data.
- The Least Squares Method is used to derive a generalized linear equation between two variables, one of which is independent and the other dependent on the former.
- This method is called so as it aims at reducing the sum of squares of deviations as much as possible. The line obtained from such a method is called a regression line.
- Example, For financial analysts, the method can help to quantify the relationship between two or more variables, such as a stock's share price and its earnings per share (EPS). By performing this type of analysis investors often try to predict the future behavior of stock prices **Code:**

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

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

```
plt.rcParams['figure.figsize'] = (12.0, 9.0)
```

```
# Preprocessing Input data
data =
```

```
pd.read_csv(r"C:\Users\Aditi\OneDrive\Documents\Python Scripts\data.csv")
```

```
X = data.iloc[:, 0]
Y =
```

```
data.iloc[:, 1]
```

```
plt.scatter(X, Y)
```

```
plt.show()
```

```
# Building the model
```

```
X_mean = np.mean(X)
```

```
Y_mean = np.mean(Y)
```

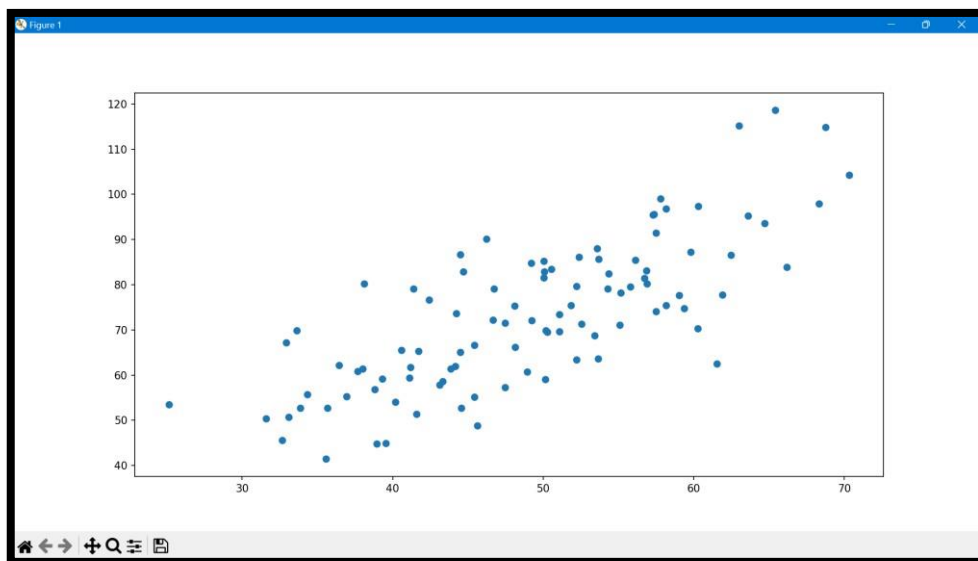
```
num = 0
```

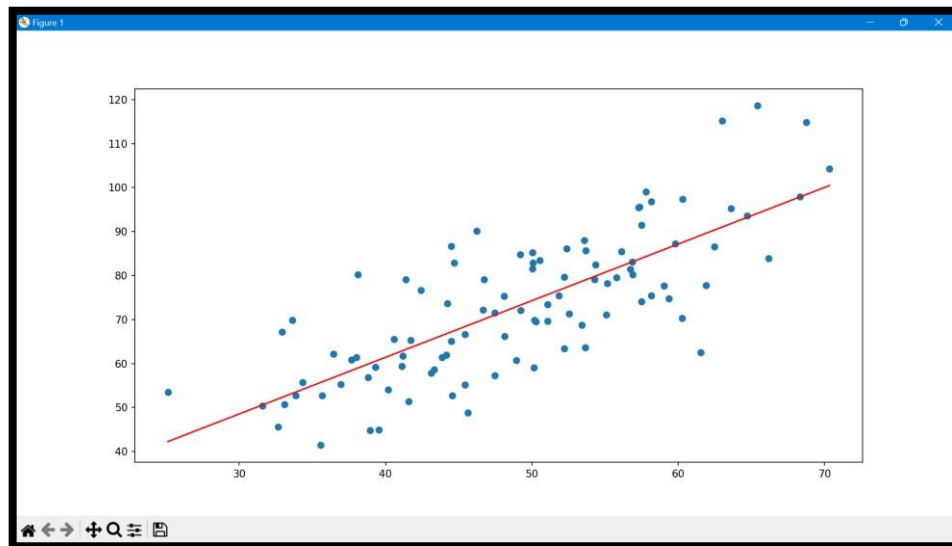
```
den = 0
for i in range(len(X)):
    num += (X[i] - X_mean)*(Y[i] - Y_mean)
    den += (X[i] - X_mean)**2
m = num / den
c = Y_mean - m*X_mean

# Making predictions Y_pred
Y_pred = m*X + c

plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)],
         color='red') # predicted

plt.show()
```

Output:



Learnings:

1. The code loads a dataset and visualizes it using a scatter plot with matplotlib.
2. Calculates the slope (m) and intercept (c) for a simple linear regression model using the least squares method.
3. Iterates through the dataset to compute the slope and intercept based on the relationship between input variable (X) and target variable (Y).
4. Utilizes the obtained slope and intercept to make predictions for the target variable (Y_{pred}) based on the input variable (X).
5. Plots the original data points along with the fitted regression line, demonstrating the linear relationship between X and Y .

8 B) Aim: For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm.

Logistic Regression:

- Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class or not.
- It is used for predicting the categorical dependent variable using a given set of independent variables.
- Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc, it gives the probabilistic values which lie between 0 and 1.
- Example, In the financial industry, logistic regression can be used to predict if a transaction is fraudulent or not. In marketing, logistic regression can be used to predict if a targeted audience will respond or not.

Code and output:

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

dataset = pd.read_csv(r"C:\Users\Aditi\OneDrive\Documents\Python Scripts\DMVWrittenTests.csv")
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, 2].values

dataset.head(5)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

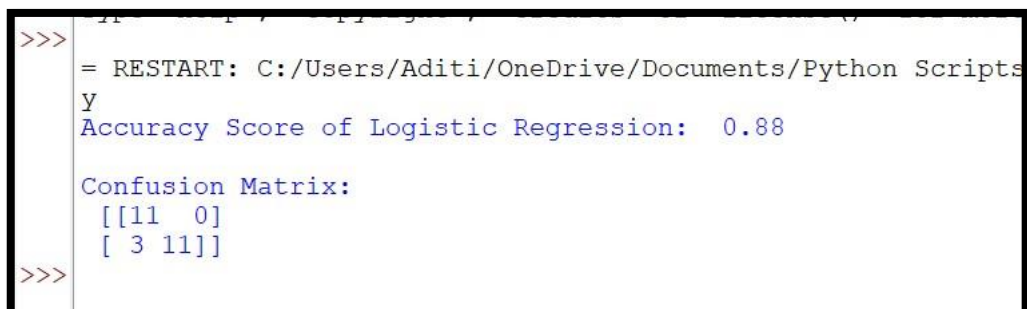
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import accuracy_score
print("Accuracy Score of Logistic Regression: ", accuracy_score(y_test, y_pred), "\n")
print("Confusion Matrix: \n", cm)
```

Output:

```
>>> = RESTART: C:/Users/Aditi/OneDrive/Documents/Python Scripts
y
Accuracy Score of Logistic Regression: 0.88

Confusion Matrix:
[[11  0]
 [ 3 11]]
>>>
```

Learnings:

1. The dataset and splits it into features (X) and target variable (y) using pandas.
2. Utilizes StandardScaler from scikit-learn to normalize the features (X) to a smaller range, aiding in computational efficiency and model performance.
3. Implements logistic regression using scikit-learn, training the model on the normalized training set (X_train, y_train).
4. Predicts the target variable for the test set (y_pred) and calculates the confusion matrix to evaluate the model's performance.
5. Computes and prints the accuracy score of the logistic regression model, indicating the proportion of correctly classified instances in the test set.

Practical No: 9**Date:****Backpropagation algorithm and Text pre-processing, Text clustering, classification**

A- Aim: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Description:**Backpropagation algorithm:**

- Backpropagation is an algorithm that backpropagates the errors from the output nodes to the input nodes. Therefore, it is simply referred to as the backward propagation of errors.
- Neural networks use supervised learning to generate output vectors from input vectors that the network operates on.
- It Compares generated output to the desired output and generates an error report if the result does not match the generated output vector.
- Then it adjusts the weights according to the bug report to get your desired output.
- Applications, Character recognition, Signature verification, etc.

Code:

```
import numpy as np

X=np.array([[2,9],[1,5],[3,6]],dtype=float)
Y=np.array([[92],[86],[89]],dtype=float)
X=X/np.amax(X,axis=0)
Y=Y/100;

class NN(object):
    def __init__(self):
        self.inputsiz=2
        self.outputsiz=1
        self.hiddensiz=3
        self.W1=np.random.randn(self.inputsiz,self.hiddensiz)
        self.W2=np.random.randn(self.hiddensiz,self.outputsiz)
    def forward(self,X):
        self.z=np.dot(X,self.W1)
        self.z2=self.sigmoidal(self.z)
        self.z3=np.dot(self.z2,self.W2)
```

```

op=self.sigmoidal(self.z3) return
op; def sigmoidal(self,s):
return 1/(1+np.exp(-s)) def
sigmoidalprime(self,s):
return s* (1-s) def
backward(self,X,Y,o):
self.o_error=Y-o self.o_delta=self.o_error *
self.sigmoidalprime(o)
self.z2_error=self.o_delta.dot(self.W2.T)
self.z2_delta=self.z2_error * self.sigmoidalprime(self.z2)
self.W1 = self.W1 + X.T.dot(self.z2_delta) self.W2=
self.W2+ self.z2.T.dot(self.o_delta) def train(self,X,Y):
o=self.forward(X) self.backward(X,Y,o)
obj=NN()
for i in range(4):
print("input"+str(X)) print("Actual output"+str(Y))
print("Predicted output"+str(obj.forward(X)))
print("loss"+str(np.mean(np.square(Y-obj.forward(X)))))
obj.train(X,Y)

```

Output:

Learnings:

- The neural network has an input layer of size 2, a hidden layer with 3 neurons, and an output layer with 1 neuron.
- Sigmoidal activation function is used throughout the network. It converts the weighted sum of inputs into the range (0, 1).
- The 'forward' method computes the predicted output by passing the input through the network, applying the sigmoid activation function at each layer.

- Backpropagation is implemented in the 'backward' method, updating weights based on the calculated errors. The mean squared loss is used for optimization.
- The network is trained for 4 iterations, and at each iteration, it prints the input, actual output, predicted output, and the mean squared loss. The weights are updated using the backpropagation algorithm to minimize the loss.

B- Aim: Perform Text pre-processing, Text clustering, classification with Prediction, Test Score and Confusion Matrix

Description:

Text pre-processing:

- Text pre-processing is a crucial step for tasks involving natural language processing (NLP).
- It involves cleaning and transforming raw text data into a format suitable for machine learning models to learn from.
- This process ensures data is consistent and relevant, improving the accuracy and efficiency of machine learning algorithms.
- Reduces the complexity of the data by removing stop words and stemming or lemmatizing words.

Text Clustering:

- Text clustering is an unsupervised machine learning technique that groups similar texts together based on their content.
- It doesn't require any pre-defined labels for the data, automatically identifying groups based on their shared characteristics.
- It is a tool for discovering hidden patterns and themes within large amounts of text data.
- Example, gain valuable insights from unstructured text data, enabling a variety of applications in areas like information retrieval, document classification, and topic modelling.

Code and output:

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

dataset = pd.read_csv('/content/Restaurant_Reviews.tsv', delimiter = '\t', quoting = 3)

import re
import nltk

nltk.download('stopwords') from nltk.corpus

import stopwords from nltk.stem.porter
import
```

```

PorterStemmer corpus = [] for i in range(0,1000):
review = re.sub('[^a-zA-Z]','',dataset['Review'][i])
review = review.lower() review = review.split()
ps = PorterStemmer()
review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
review = ".join(review) corpus.append(review)

from sklearn.feature_extraction.text import CountVectorizer cv
= CountVectorizer(max_features=1500)
X = cv.fit_transform(corpus).toarray() Y
= dataset.iloc[:,1].values

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.25, random_state=100)
from sklearn.naive_bayes import GaussianNB classifier = GaussianNB()
classifier.fit(X_train, Y_train)
Y_pred = classifier.predict(X_test)

from sklearn import metrics
from sklearn.metrics import confusion_matrix
print("Accuracy:",metrics.accuracy_score(Y_test, Y_pred))

from sklearn.metrics import confusion_matrix cm =
confusion_matrix(Y_test, Y_pred) print(cm)

```

Output:

```

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
Accuracy: 0.54
[[ 1 115]
 [ 0 134]]

```

Learnings:

- Splitting reviews into words and converting them to lowercase for uniformity.
- Eliminating common words and reducing words to their base form.
- Using CountVectorizer to convert pre-processed text into a numerical matrix (bag of words).

- Dividing the dataset into training and testing sets (75% training, 25% testing) using 'train_test_split'.
- Choosing the Gaussian Naive Bayes classifier for text classification.
- Fitting the classifier to the training set using 'fit'.
- Using 'accuracy_score' from 'sklearn.metrics' to measure the accuracy of the model on the test set.
- Creating a confusion matrix with 'confusion_matrix' to evaluate classification performance.
- Analyzing the accuracy score provides an overall measure of classification correctness.
- Examining the confusion matrix reveals true positives, true negatives, false positives, and false negatives, offering insights into model performance.

Practical No: 10**Date:****Euclidean distance method Aim:****A. Implement the different distance methods (Euclidean) with prediction, test score & confusion Matrix.****Description:**

- Euclidean distance is a measure of the straight-line distance between two points in a geometric space.
- For geometrical problems Euclidean distance is used as the standard metric.
- It is simply the ordinary distance between two points.
- Euclidean distance is mainly extensively used in clustering problems.
- In K-means algorithms by default Euclidean distance is used as distance measure.
- The Euclidean distance is calculated by taking the root of square differences between the coordinates of a pair of objects (x1, y1) and (x2, y2) as:

$$\text{Distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Prediction: In machine learning, prediction refers to the model's output or estimate for a given input.

Test Score: The test score, often represented as accuracy, indicates the proportion of correctly classified instances out of the total test instances.

Confusion Matrix: Table that describes the performance of a classification model. It compares the predicted classifications to the actual classifications.

Code:

```
import numpy as np import pandas as pd from sklearn.model_selection import
train_test_split from sklearn.neighbors import KNeighborsClassifier from
sklearn.metrics import accuracy_score df =
pd.read_csv(r"C:\Users\Aditi\OneDrive\Documents\Python Scripts\iris.csv")
df.head(5)

X = df.drop(['variety'], axis=1)
y = df['variety']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

knn = KNeighborsClassifier(n_neighbors= 6, p = 2, metric='minkowski')
```



```

knn.fit(X_train, y_train) print(knn.score(X_test,y_test)) y_pred = knn.predict(X_test)

from sklearn.metrics import confusion_matrix

cm=np.array(confusion_matrix(y_test,y_pred)) print(cm) knn =

KNeighborsClassifier(n_neighbors= 6, p = 1, metric='minkowski') knn.fit(X_train,
y_train) print(knn.score(X_test,y_test)) from sklearn.metrics import confusion_matrix

cm=np.array(confusion_matrix(y_test,y_pred)) print(cm)

```

Output:

```

>>> = RESTART: C:/Users/Aditi/OneDrive/Documents/Py
ist.py
0.9777777777777777
[[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
0.9555555555555556
[[16  0  0]
 [ 0 17  1]
 [ 0  0 11]]
>>> |

```

Learnings:

- The code loads the Iris dataset, splits it into training and testing sets (70-30 split).
- It employs the KNN algorithm, training a model with 6 neighbors and Euclidean distance.
- The code calculates and prints the accuracy score for model evaluation on the test set.
- It computes and displays the confusion matrix for detailed result analysis.
- An alternative KNN model is shown with Manhattan distance and p=1.
- It evaluates and prints the accuracy score for the alternative model.
- The confusion matrix for the alternative model is calculated and displayed.

B. Implement the classification model using k means clustering with prediction, test score and confusion matrix.

Code:

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn

dataset = pd.read_csv(r"C:\Users\Aaditi\OneDrive\Documents\Python Scripts\Mall_Customers.csv")
X = dataset.iloc[:, [3,4]].values

from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):

    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

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

plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters= 5, init= "k-means++", random_state=42)
y_kmeans = kmeans.fit_predict(X)

plt.scatter(X[y_kmeans == 0,0], X[y_kmeans == 0,1], s = 60, c='red', label = 'Cluster1')
plt.scatter(X[y_kmeans == 1,0], X[y_kmeans == 1,1], s = 60, c='blue', label = 'Cluster2')
plt.scatter(X[y_kmeans == 2,0], X[y_kmeans == 2,1], s = 60, c='green', label = 'Cluster3')
plt.scatter(X[y_kmeans == 3,0], X[y_kmeans == 3,1], s = 60, c='violet', label = 'Cluster4')
plt.scatter(X[y_kmeans == 4,0], X[y_kmeans == 4,1], s = 60, c='yellow', label = 'Cluster5')

plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=100, c='black', label = 'Centroids')

plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')

plt.legend()
plt.show()

plt.scatter(X[y_kmeans == 0,0], X[y_kmeans == 0,1], s=100, c='red', label = 'Careful')
plt.scatter(X[y_kmeans == 1,0], X[y_kmeans == 1,1], s=100, c='blue', label = 'Standard')
plt.scatter(X[y_kmeans == 2,0], X[y_kmeans == 2,1], s=100, c='green', label = 'Target')
plt.scatter(X[y_kmeans == 3,0], X[y_kmeans == 3,1], s=100, c='violet', label = 'Careless')
plt.scatter(X[y_kmeans == 4,0], X[y_kmeans == 4,1], s=100, c='yellow', label = 'Sensible')

```

```
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[1], s=100, c='black', label =
```

```
'Centroids') plt.xlabel('Annual
```

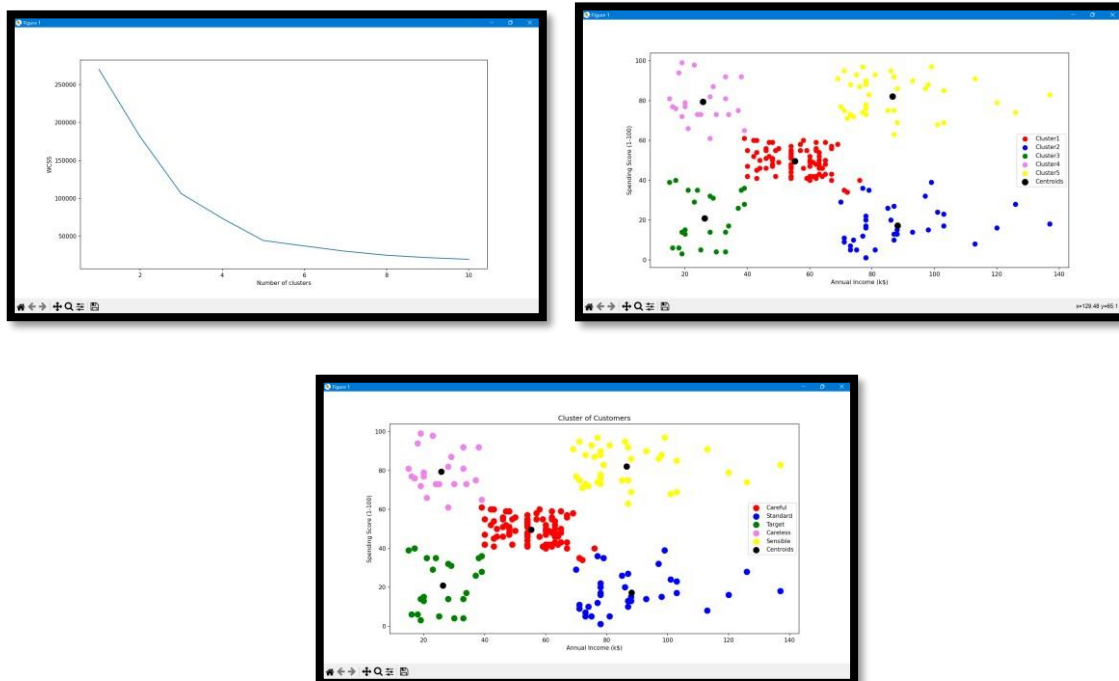
```
Income (k$)) plt.ylabel('Spending
```

```
Score (1-100)) plt.title('Cluster of
```

```
Customers') plt.legend()
```

```
plt.show()
```

Output:



Learnings:

1. The code loads a Mall Customers dataset and selects two features (Annual Income and Spending Score) for clustering.
2. It employs the KMeans algorithm with the Elbow Method to determine the optimal number of clusters. The within-cluster sum of squares (WCSS) is plotted against the number of clusters.
3. Based on the Elbow Method, the code performs K-Means clustering with five clusters and visualizes the clusters along with centroids.
4. It creates a scatter plot with different colors for each cluster and highlights cluster centroids.
5. The code provides a more detailed scatter plot, labeling each cluster as 'Careful,' 'Standard,' 'Target,' 'Careless,' and 'Sensible' based on customer characteristics.

