Ex no: 1	Perform Prediction using Regression Algorithm
Date:	

Aim:

To write a python programming using linear regression algorithm for prediction Application.

Algorithm:

Step 1: Load the dataset

Step 2: Split dataset int training set and test set.

Step 3: Fit simple linear regression model

Step 4: Finding there is any correlation between 2 variables

Step 5: Finding the best fit line for dataset.

Step 6: Dependent variable is changing into independent variable.

Step 7: Predict the test set.

Step 8: Visualizing the test set.

Step 9: Make new predictions

Implementation:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset=pd.read_csv('Salary_Data.csv')
dataset.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
print(dataset)
dataset.tail()
```

dataset.shape

```
(30, 2)
```

dataset.info()

```
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

dataset.describe()

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

dataset.size

dataset.isnull().sum()

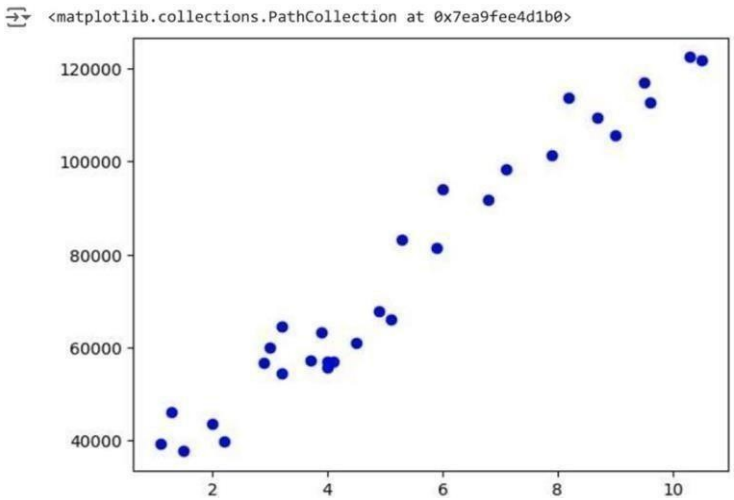
	0
YearsExperience	0
Salary	0

dtype: int64

```
plt.scatter(dataset['YearsExperience'],dataset['Salary'],color='blue')
plt.scatter(dataset['YearsExperience'],dataset['Salary'],color='blue')
plt.title('Comparsion chart')
plt.xlabel('Experience of year')
plt.ylabel('Salary')
plt.show()
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
```

x=dataset[['YearsExperience']]

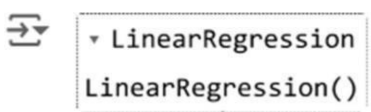
y=dataset['Salary']



x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)

model=LinearRegression()

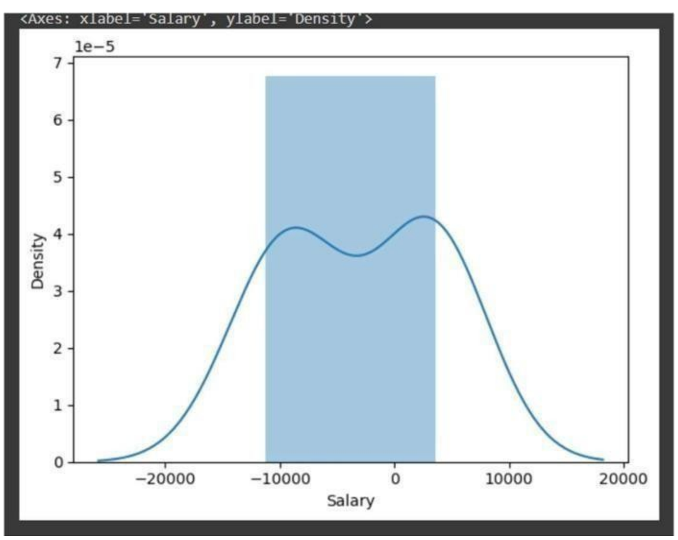
model.fit(x_train,y_train)



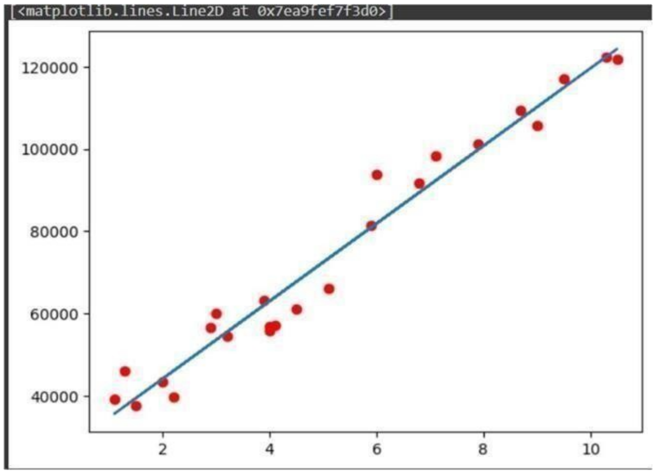
predictions = model.predict(x_test)

import seaborn as sns

sns.distplot(predictions-y_test)



```
plt.scatter(x_train, y_train, color='red')
plt.plot(x_train, model.predict(x_train))
```



Result:

Thus, the implementation of python programming using linear regression algorithm for prediction application has been completed successfully

Ex no: 2	Data Classification using Decision Trees
Date:	

Aim:

To write a python programming using data classification using tree for car safety application.

Algorithm:

Step 1: Begin the tree with the root node, which contains the complete dataset.

Step 2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Step 3: Divide the data set into subsets that contains possible values for the best attributes.which is determined using information gain entrophy & gain of the attribute.

Step 4: Generate the decision tree node, which contains the best attribute

Step 5: Recursively make new decision trees using the subsets of the dataset created in step 3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Implementation:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

!pip install category_encoders

import category_encoders as ce
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn import tree

import graphviz

from sklearn.metrics import confusion_matrix

dataset=pd.read_csv('car_evaluation.csv')

dataset.head()
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

dataset.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   vhigh    1727 non-null   object
 1   vhigh.1  1727 non-null   object
 2   2        1727 non-null   object
 3   2.1      1727 non-null   object
 4   small    1727 non-null   object
 5   low      1727 non-null   object
 6   unacc    1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

dataset.tail()

	vhigh	vhigh.1	2	2.1	small	low	unacc
1722	low	low	5more	more	med	med	good
1723	low	low	5more	more	med	high	vgood
1724	low	low	5more	more	big	low	unacc
1725	low	low	5more	more	big	med	good
1726	low	low	5more	more	big	high	vgood

dataset.isnull().sum()

#renaming columns
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']

dataset.columns = col_names

dataset.describe()

	buying	maint	doors	persons	lug_boot	safety	class
count	1727	1727	1727	1727	1727	1727	1727
unique	4	4	4	3	3	3	4
top	high	high	3	4	med	med	unacc
freq	432	432	432	576	576	576	1209

X = dataset.drop("class", axis = 1)
y = dataset["class"]

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

X_train.shape, X_test.shape

```
((1208, 6), (519, 6))
```

encoder = ce.OrdinalEncoder(cols = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])

X_train = encoder.fit_transform(X_train)

```
X_test = encoder.transform(X_test)

giniclf = DecisionTreeClassifier(criterion = "gini", max_depth = 3, random_state = 0)

giniclf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=0)
```

```
ypred = giniclf.predict(X_test)

ypredtrain = giniclf.predict(X_train) #accuracy

print('Model accuracy score for test data with criterion gini index: {0:0.4f}'.
format(accuracy_score(y_test, ypred)))
```

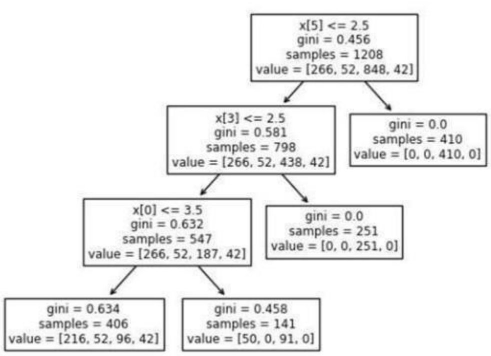
```
Model accuracy for training data: 0.8013
Model accuracy for test data: 0.8150
```

```
print(print('Model accuracy score for training data with criterion gini index: {0:0.4f}'.
format(accuracy_score(y_train, ypredtrain))))

print('Training set score: {:.4f}'.format(giniclf.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(giniclf.score(X_test, y_test)))
```

```
Training set score:0.8013
Test set score:0.8150
```

```
plt.figure(figsize = (12,8))
tree.plot_tree(giniclf.fit(X_train, y_train))

tree.plot_tree(giniclf.fit(X_train, y_train))
```

```
[Text(0.6666666666666666, 0.875, 'x[5] <= 2.5\ngini = 0.456\nsamples = 1208\nvalue = [266, 52, 848, 42]'),
 Text(0.5, 0.625, 'x[3] <= 2.5\ngini = 0.581\nsamples = 798\nvalue = [266, 52, 438, 42]'),
 Text(0.3333333333333333, 0.375, 'x[0] <= 3.5\ngini = 0.632\nsamples = 547\nvalue = [266, 52, 187, 42]'),
 Text(0.16666666666666666, 0.125, 'gini = 0.634\nsamples = 406\nvalue = [216, 52, 96, 42]'),
 Text(0.5, 0.125, 'gini = 0.458\nsamples = 141\nvalue = [50, 0, 91, 0]'),
 Text(0.6666666666666666, 0.375, 'gini = 0.0\nsamples = 251\nvalue = [0, 0, 251, 0]'),
 Text(0.8333333333333334, 0.625, 'gini = 0.0\nsamples = 410\nvalue = [0, 0, 410, 0]')]
```




```
newtree = tree.export_graphviz(giniclf,out_file = None, feature_names = X_train.columns,
class_names = y_train, filled = True, rounded = True, special_characters = True)

graph = graphviz.Source(newtree)

from sklearn.tree import DecisionTreeClassifier

enclf = DecisionTreeClassifier(criterion = "entropy", max_depth = 3, random_state = 0)

enclf.fit(X_train,y_train)
```

```
ypreden = enclf.predict(X_test)
ypredten = enclf.predict(X_train)


print('Model accuracy for training data:  {0:0.4f}'.format(accuracy_score(y_train, ypredten)))
print('Model accuracy for  test data:  {0:0.4f}'.format(accuracy_score(y_test,ypreden)))


print('Training set score:{:.4f}'.format(enclf.score(X_train,y_train)))
print('Test set score:{:.4f}'.format(enclf.score(X_test,y_test)))

plt.figure(figsize = (12,8))
tree.plot_tree(enclf.fit(X_train,y_train))

newtreeen = tree.export_graphviz(enclf, out_file=None, feature_names=X_train.columns,
class_names=y_train, filled=True,  rounded=True,  special_characters=True)

graph = graphviz.Source(newtreeen)

cm = confusion_matrix(y_test,ypreden)

print('Confusion Matrix\n',cm)
```

```
Confusion Matrix
 [[ 96   0  22   0]
 [ 17   0   0   0]
 [ 34   0 327   0]
 [ 23   0   0   0]]
```

Result:

Thus, the implementation of python programming using data classification using tree has been executed successfully.

Ex no: 3	Data Classification using Bayesian learning method for income prediction
Date:	

Aim:

To write a python program using data classification by Bayesian learning method for income prediction

Algorithm:

Step 1: Importing all the necessary libraries.

Step 2: Load the dataset.

Step 3: Bayesian learning classifier determines the probability of hypothesis with prior knowledge.

Step 4: Convert the given dataset into frequency tables.

Step 5: Generate likelihood table by finding the probabilities of given features.

Step 6: Apply Baye's theorem to calculate the posterior probability for income predictions.

Step 7: Thus, income prediction is Implemented by Bayesian learning.

Implementation:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset = pd.read_csv('heart.csv')
dataset.head(10)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1

dataset.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

X = dataset[['age']]

y = dataset['fbs']

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

model = GaussianNB()

model.fit(X_train, y_train)

```
▾ GaussianNB
GaussianNB()
```

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:\n", report)

```
Accuracy: 0.80

Classification Report:
               precision    recall  f1-score   support

           0       0.80      1.00      0.89        73
           1       0.00      0.00      0.00        18

    accuracy                           0.80        91
   macro avg       0.40      0.50      0.45        91
weighted avg       0.64      0.80      0.71        91
```

Result:

Thus, the implementation of python program using Bayesian learning for income prediction has been executed successfully

Ex no: 4	Data Classification using Support Vector Machine for Credit Card Fraud Detection
Date:	

Aim:

To wire a python program using data Classification using Support Vector Machine for Credit Card Fraud Detection.

Algorithm:

Step 1: Importing all the necessary Libraries.

Step 2: Load the dataset from the csv file.

Step 3: The sum classifier classifies the dataset by linear separable method

to find the best line or decision boundary.

Step 4: Sum classifier finds the closet point of the lines from the different classes.

Step 5: Train the dataset using sum classifier.

Step 6: Test the dataset.

Step 7: Thus, credit card fraud detection is implemented by sum.

Implementation:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split

from  sklearn.preprocessing  import  StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, classification_report

dataset =  pd.read_csv('heart.csv')

dataset.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
X = dataset[['cp']]
y = dataset['slope']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

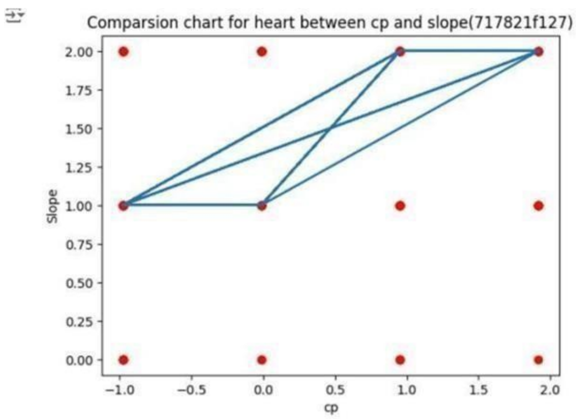
```
SVC
SVC(kernel='linear', random_state=42)
```

```
X_test = scaler.transform(X_test)
model = SVC(kernel='linear', C=1.0, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", report)
```

```
Accuracy: 0.51

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         7
           1       0.52      0.75      0.61        44
           2       0.48      0.33      0.39        40

    accuracy                           0.51        91
   macro avg       0.33      0.36      0.33        91
weighted avg       0.46      0.51      0.47        91
```

```
def plot_decision_boundaries(X, y, model):
    h = .02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
```

```
np.arange(y_min, y_max, h))
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, model.predict(X_train))
plt.title('Comparsion chart cp for heart between and slope(717821f127)')
plt.xlabel('cp')
plt.ylabel('Slope')
plt.show()
```



Result:

Thus ,the implementation of python program using data Classification using Support Vector Machine for Credit Card Fraud Detection has been executed successfully.

Ex no: 5	Implementation of Bagging Ensemble Method
Date:	

Aim:

To write a python programming for implementation of bagging ensemble method.

Algorithm:

Step 1: Importing all the necessary libraries

Step 2: Load the dataset

Step 3: Multiple subsets are created from the original dataset with equal tuples selecting

observation with replacement.

Step 4: A Base model is created on each of these subsets.

Step 5: Each model is learned in parallel with each training set and independent of each other.

Step 6: The final predictions are determined by combining the prediction using voting from all the model.

Step 7: Thus the implementation of bagging method is implemented.

Implementation:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_classification from

sklearn.model_selection import train_test_splitfrom

sklearn.ensemble import BaggingClassifier from

sklearn.tree import DecisionTreeClassifier from

sklearn.metrics import accuracy_score

from mlxtend.plotting import plot_decision_regions

X, y = make_classification(n_samples=300, n_features=2, n_informative=2,n_redundant=0,

                    n_clusters_per_class=1, random_state=42)

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

= DecisionTreeClassifier()

bagging_clf = BaggingClassifier(estimator=base_clf, n_estimators=50,random_state=42)
```

bagging_clf.fit(X_train, y_train) y_pred =

bagging_clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy * 100:.2f}%')

plt.figure(figsize=(8, 6))

plot_decision_regions(X_test, y_test, clf=bagging_clf, legend=2)

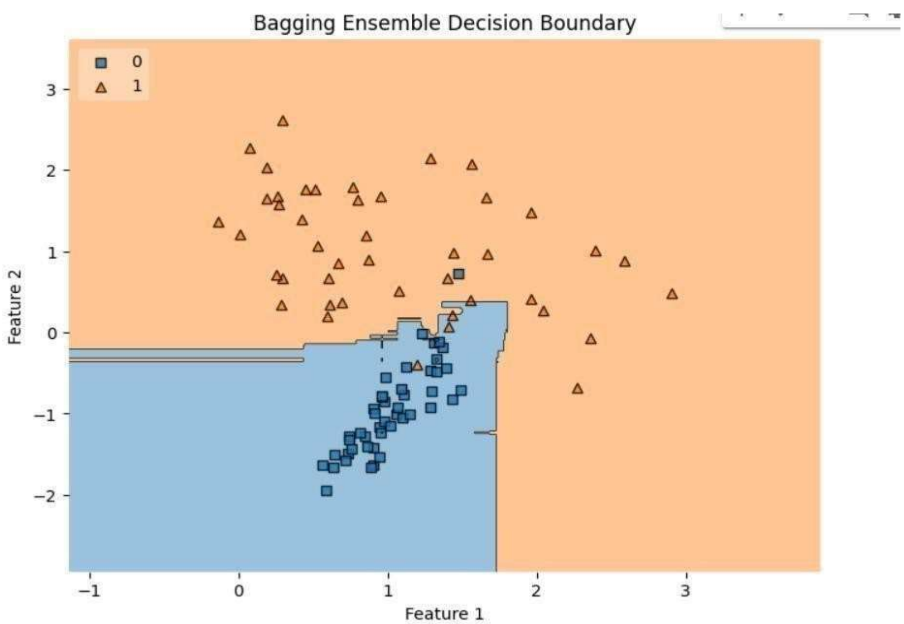
plt.title("Bagging Ensemble Decision Boundary") plt.xlabel("Feature

1")

plt.ylabel("Feature 2")

plt.show()

Output:



Result:

Thus, the implementation of bagging ensemble method has been completed successfully and the

Output has been verified.

Ex no: 6	BAGGING, BOOSTING APPLICATIONS USING REGRESSION TREES
Date:	

Aim:

To write an python program for bagging, boosting applications using regression trees.

Algorithm:

BEGIN

Step 1: Import the libraries of all model predictions by combining them into ensemble predictions.

Step 2: Load the data and pre-process the data loaded in.

Step 3: Apply the bagging decision trees for the each training and testing results.

Step 4: Same as before, apply the boosting algorithm's decision trees for the each training and test results.

Step 5: Now ensemble the voting process and execute for it's training and test results.

Step 6: Compare all of the regression trees of bagging, boosting and voting through an model chart.

END

Implementation:

Bagging:

```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_splitfrom
sklearn.metrics import mean_squared_error
X, y = make_regression(n_samples=500, n_features=4, noise=0.3)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
base_regressor = DecisionTreeRegressor()
bagging_regressor = BaggingRegressor(base_estimator=base_regressor, n_estimators=50,random_state=42)
bagging_regressor.fit(X_train, y_train) y_pred =
bagging_regressor.predict(X_test) mse =
mean_squared_error(y_test, y_pred)
print(f"Bagging Regressor MSE: {mse:.4f}")
```

Output:

```
Bagging Regressor MSE: 547.2237
```

Boosting:

From sklearn.ensemble import AdaBoostRegressor base_regressor =

DecisionTreeRegressor(max_depth=4)

boosting_regressor = AdaBoostRegressor(base_estimator=base_regressor, n_estimators=50,

random_state=42)

boosting_regressor.fit(X_train, y_train) y_pred_boost =

boosting_regressor.predict(X_test)

mse_boost = mean_squared_error(y_test, y_pred_boost)

print(f"AdaBoost Regressor MSE: {mse_boost:.4f}")

Output:

```
AdaBoost Regressor MSE: 16583.7937
```

Result:

Thus, to write a python program for bagging, boosting applications using regression tree has been verified and executed successfully.

Ex no: 7	Data and Text Classification Using Neural Networks
Date:	

Aim:

To write an python program for Data and Text Classification Using Neural Networks.

Algorithm:

BEGIN

Step 1:Importing Necessary Libraries

Step 2: Prepare a small synthetic dataset consisting of text samples and their corresponding sentiment labels (positive/negative).

Step 3: Split the dataset into training and testing subsets using train_test_split.

Step 4: Compile the model

Step 5: Train the model on the training data with specified epochs and batch size, while validating on a portion of the training set.

Step 6: Evaluate the model's performance on the test set and print the test accuracy.

Step 7: Print the predicted sentiment for each new text sample based on the model's output.

END

Implementation:

```
pip install tensorflow

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_splitfrom

sklearn.preprocessing import LabelEncoder

# Corrected import statements

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequencesfrom

tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense#

Correct import statement
```

```
from tensorflow.keras.utils import to_categorical

# Let's create a small synthetic dataset for illustration.texts =
[
    "I love programming", "Python is great for data science",
    "I hate bugs in the code", "Machine learning is fascinating","I
    enjoy debugging", "Data analysis with Python is fun", "Deep
    learning is the future", "I dislike syntax errors"
]

# Labels (positive: 1, negative: 0)
labels = [1, 1, 0, 1, 1, 1, 1, 0]

# Convert the labels to categorical values for classificationle =
LabelEncoder()
labels = to_categorical(le.fit_transform(labels))

# Tokenize the text data
max_words = 1000 # Maximum number of words in vocabulary
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
# Pad sequences to ensure equal length for all input data
max_len = 10 # Maximum length of each text sequence X =
pad_sequences(sequences, maxlen=max_len)


# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)from
tensorflow.keras.layers import GlobalAveragePooling1D
# Build a simple neural network model for text classificationmodel =
Sequential()
model.add(Embedding(input_dim=max_words, output_dim=16, input_length=max_len))
```

model.add(GlobalAveragePooling1D()) # Flatten the output of the embedding layer

model.add(Dense(16, activation='relu')) # Hidden layer with ReLU activation

model.add(Dense(2, activation='softmax')) # Output layer for binary classification

Compile the model

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

Train the model

history = model.fit(X_train, y_train, epochs=10, batch_size=4, validation_split=0.2, verbose=2)

```
Epoch 1/10
1/1 - 1s - 1s/step - accuracy: 0.5000 - loss: 0.6927 - val_accuracy: 1.0000 - val_loss: 0.6851
Epoch 2/10
1/1 - 0s - 61ms/step - accuracy: 0.5000 - loss: 0.6912 - val_accuracy: 1.0000 - val_loss: 0.6853
Epoch 3/10
1/1 - 0s - 58ms/step - accuracy: 0.5000 - loss: 0.6900 - val_accuracy: 1.0000 - val_loss: 0.6857
Epoch 4/10
1/1 - 0s - 56ms/step - accuracy: 0.7500 - loss: 0.6890 - val_accuracy: 1.0000 - val_loss: 0.6861
Epoch 5/10
1/1 - 0s - 36ms/step - accuracy: 0.7500 - loss: 0.6881 - val_accuracy: 1.0000 - val_loss: 0.6864
Epoch 6/10
1/1 - 0s - 40ms/step - accuracy: 0.7500 - loss: 0.6872 - val_accuracy: 1.0000 - val_loss: 0.6869
Epoch 7/10
1/1 - 0s - 33ms/step - accuracy: 1.0000 - loss: 0.6863 - val_accuracy: 1.0000 - val_loss: 0.6873
Epoch 8/10
1/1 - 0s - 34ms/step - accuracy: 1.0000 - loss: 0.6854 - val_accuracy: 1.0000 - val_loss: 0.6878
Epoch 9/10
1/1 - 0s - 58ms/step - accuracy: 1.0000 - loss: 0.6845 - val_accuracy: 1.0000 - val_loss: 0.6882
Epoch 10/10
1/1 - 0s - 36ms/step - accuracy: 1.0000 - loss: 0.6835 - val_accuracy: 1.0000 - val_loss: 0.6881
```

:.4# Evaluate the model on the test set

loss, accuracy = model.evaluate(X_test, y_test, verbose=2)

print(f"Test Accuracy: {accuracy f}")

```
1/1 - 0s - 16ms/step - accuracy: 0.0000e+00 - loss: 0.6975
Test Accuracy: 0.0000
```

Predict on new data

new_texts = ["I love learning new things", "Syntax errors are frustrating"]new_sequences =

tokenizer.texts_to_sequences(new_texts)

new_X = pad_sequences(new_sequences, maxlen=max_len)

predictions = model.predict(new_X)

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
```

```
# Output the predictions

for i, text in enumerate(new_texts):

print(f"Text: '{text}' - Predicted Sentiment: {'Positive' if np.argmax(predictions[i]) == 1 else
'Negative'}")
```

```
Text: 'I love learning new things' - Predicted Sentiment: Positive
Text: 'Syntax errors are frustrating' - Predicted Sentiment: Positive
```

Result:

Thus, to write a python program for Data and Text Classification Using Neural Networks has been verified and executed successfully.

Ex. No: 08	Data And Text Clustering Using K Means Clustering
Date:	

Aim:

To write an python program on data and text clustering using k means clustering.

Algorithm:

Step 1: Applying the k-means clustering algorithm and import the libraries for the execution.

Step 2: Ignore the warnings when importing the dataset and check, preview and view the summary of the loaded dataset.

Step 3: Drop the redundant columns and view the dataset again for explore and drop the desired variables from the set.

Step 4: Declare the feature vector and target variable and convert the categorical variable into integers.

Step 5: Apply the k-means algorithm with two clusters for checking the quality of the dataset. If not, apply the elbow method and apply different clusters.

Step 6: Compare all the cluster's accurate values and find the highest accurate value to conclude.

Implementation:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from IPython.display import clear_output
from sklearn.cluster import KMeans

players = pd.read_csv("players_22.csv")
players
```

	sofifa_id	player_url	short_name	long_name	player_positions	overall	potential	value_eur	wage_eur	age	...	lcb	cb	rcb	rb	gk
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	RW, ST, CF	93	93	78000000.0	320000.0	34	...	50+3	50+3	50+3	61+3	19+3
1	188545	https://sofifa.com/player/188545/robert-lewand...	R. Lewandowski	Robert Lewandowski	ST	92	92	119500000.0	270000.0	32	...	60+3	60+3	60+3	61+3	19+3
2	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	ST, LW	91	91	45000000.0	270000.0	36	...	53+3	53+3	53+3	60+3	20+3
3	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Júnior	LW, CAM	91	91	129000000.0	270000.0	29	...	50+3	50+3	50+3	62+3	20+3
4	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevin De Bruyne	CM, CAM	91	91	125500000.0	350000.0	30	...	69+3	69+3	69+3	75+3	21+3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19234	261962	https://sofifa.com/player/261962/defu-song/220002	Song Defu	宋德福	CDM	47	52	70000.0	1000.0	22	...	46+2	46+2	46+2	48+2	15+2
19235	262040	https://sofifa.com/player/262040/caoimhin-port...	C. Porter	Caoimhin Porter	CM	47	59	110000.0	500.0	19	...	44+2	44+2	44+2	48+2	14+2
19236	262760	https://sofifa.com/player/262760/nathan-logue/...	N. Logue	Nathan Logue-Cunningham	CM	47	55	100000.0	500.0	21	...	45+2	45+2	45+2	47+2	12+2
19237	262820	https://sofifa.com/player/262820/luke-rudden/2...	L. Rudden	Luke Rudden	ST	47	60	110000.0	500.0	19	...	26+2	26+2	26+2	32+2	15+2
19238	264540	https://sofifa.com/player/264540/emanuel-lalch...	E. Lalchhanchhuaha	Emanuel Lalchhanchhuaha	CAM	47	60	110000.0	500.0	19	...	41+2	41+2	41+2	45+2	16+2

19239 rows × 110 columns

```
features = ["overall", "potential", "wage_eur", "value_eur", "age"]
players = players.dropna(subset=features)
data = players[features].copy()
```

data

	overall	potential	wage_eur	value_eur	age
0	93	93	320000.0	78000000.0	34
1	92	92	270000.0	119500000.0	32
2	91	91	270000.0	45000000.0	36
3	91	91	270000.0	129000000.0	29
4	91	91	350000.0	125500000.0	30
...	...	...	...	...	...
19234	47	52	1000.0	70000.0	22
19235	47	59	500.0	110000.0	19
19236	47	55	500.0	100000.0	21
19237	47	60	500.0	110000.0	19
19238	47	60	500.0	110000.0	19

19165 rows × 5 columns

```
data = ((data- data.min()) / (data.max()- data.min())) * 9 + 1
data.describe()
```

	overall	potential	wage_eur	value_eur	age
count	19165.000000	19165.000000	19165.000000	19165.000000	19165.000000
mean	4.670472	5.319998	1.219443	1.131826	4.063345
std	1.346635	1.191076	0.501528	0.353229	1.575838
min	1.000000	1.000000	1.000000	1.000000	1.000000
25%	3.739130	4.521739	1.012876	1.021620	2.666667
50%	4.717391	5.304348	1.064378	1.044817	4.000000
75%	5.500000	6.086957	1.193133	1.092370	5.333333
max	10.000000	10.000000	10.000000	10.000000	10.000000

```
data.head()
```

	overall	potential	wage_eur	value_eur	age
0	10.000000	9.608696	9.227468	4.618307	7.000000
1	9.804348	9.413043	7.939914	6.543654	6.333333
2	9.608696	9.217391	7.939914	3.087308	7.666667
3	9.608696	9.217391	7.939914	6.984396	5.333333
4	9.608696	9.217391	10.000000	6.822018	5.666667

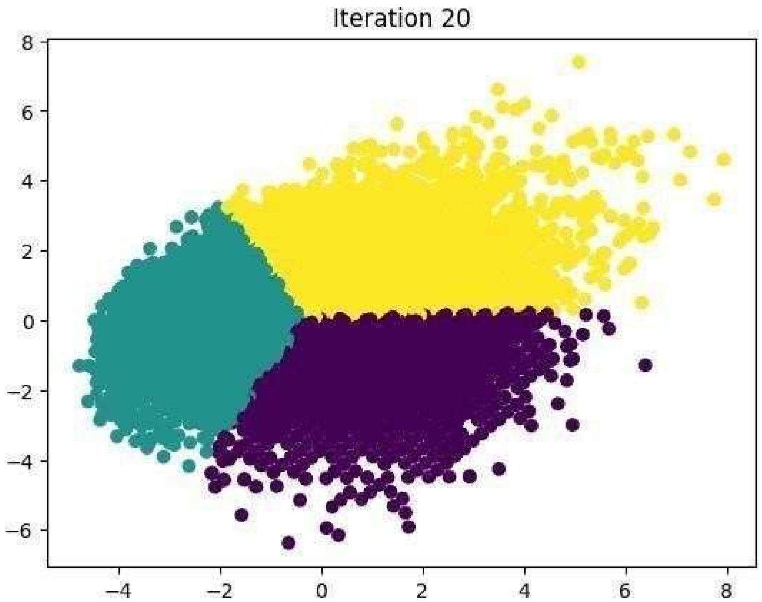
```
def random_centroid(data, k):
    centroids = []
    for i in range(k):
        centroid = data.apply(lambda x: float(x.sample()))
        centroids.append(centroid)
    return pd.concat(centroids, axis=1)
centroids = random_centroid(data, 5)
centroids
```

	0	1	2	3	4
overall	5.500000	4.326087	6.282609	3.739130	2.956522
potential	5.108696	5.695652	4.913043	4.326087	6.282609
wage_eur	1.000000	1.000000	1.270386	1.090129	1.193133
value_eur	1.021620	1.055255	1.050616	1.019300	1.059895
age	2.333333	1.333333	5.000000	4.000000	3.666667

```
def get_labels(data, centroids):
    distances = centroids.apply(lambda x: np.sqrt(((data- x) ** 2).sum(axis=1)))return
    distances.idxmin(axis=1)

labels = get_labels(data, centroids)
labels.value_counts()
```

```
2    6585
3    6282
1    3002
0    2634
4     662
Name: count, dtype: int64
```

```
def new_centroids(data, labels, k):
    return data.groupby(labels).apply(lambda x: np.exp(np.log(x).mean())).T

def plot_clusters(data, labels, centroids, iteration):pca
    = PCA(n_components=2)
    data_2d = pca.fit_transform(data)
    centroids_2d = pca.transform(centroids.T)
    clear_output(wait=True) plt.title(f'Iteration
    {iteration}')
    plt.scatter(x=data_2d[:, 0], y=data_2d[:,1], c=labels)
    plt.show()

max_iterations = 100
k =3
centroids = random_centroid(data, k)
old_centroids = pd.DataFrame()
iteration = 1
while iteration < max_iterations and not centroids.equals(old_centroids):
    old_centroids = centroids
    labels = get_labels(data, centroids) centroids =
    new_centroids(data, labels, k)
    plot_clusters(data, labels, centroids, iteration)
    iteration += 1
```



Centroids

	0	1	2
overall	4.781960	3.205672	5.807503
potential	4.506813	4.930905	6.497870
wage_eur	1.118498	1.028564	1.420500
value_eur	1.044909	1.026655	1.285685
age	5.467648	2.514741	3.598215

players[labels == 1][["short_name"] + features]

	short_name	overall	potential	wage_eur	value_eur	age
7025	Sandeiro Leal	68	68	7000.0	1400000.0	21
8028	Narcisso Mau	67	67	4000.0	1100000.0	21
8029	Botelhinonsa	67	67	4000.0	1100000.0	21
8030	Edenildo Lagoas	67	67	3000.0	1100000.0	21
8040	Dener Rolim	67	67	4000.0	1200000.0	21
...	...	...	...	...	...	...
19234	Song Defu	47	52	1000.0	70000.0	22
19235	C. Porter	47	59	500.0	110000.0	19
19236	N. Logue	47	55	500.0	100000.0	21
19237	L. Rudden	47	60	500.0	110000.0	19
19238	E. Lalchhanchhuaha	47	60	500.0	110000.0	19

6209 rows × 6 columns

kmeans = KMeans(3)
kmeans.fit(data)

```
▾ KMeans
KMeans(n_clusters=3)
```

centroids = kmeans.cluster_centers_
pd.DataFrame(centroids, columns=features).T

	0	1	2
overall	3.583634	4.807911	6.215275
potential	5.197250	4.512451	6.619682
wage_eur	1.039293	1.113912	1.650920
value_eur	1.035237	1.040393	1.410811
age	2.706599	5.606693	4.116439

Result:

Thus, the implementation of python programming for data and text clustering using Kmeans clustering has been completed and verified successfully.

Ex. No: 09	**Data and Text Clustering using Gaussian Mixture models**
Date:	

Aim:

To write a python programming for Data and Text clustering using Gaussian Mixture models.

Algorithm:

Step 1: Load the dataset.

Step 2: Split dataset into training set and Test set.

Step 3: Fit simple Gaussian mixture model.

Step4: Initialize the mean, the covariance matrix and the mixing coefficients by some random values.

Step 5: Compute the ck values for all k.

Step6: Again, estimate all the parameters using the current ck values.

Step7: Compute log-likelihood function and put some convergence criterior.

Implementation:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt from
pandas import DataFrame
from sklearn.preprocessing import StandardScaler, normalizefrom
sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
from sklearn.model_selection import train_test_splitfrom
sklearn import metrics

raw_df = pd.read_csv('dataset.csv')
raw_df = raw_df.drop('CUST_ID', axis = 1)
raw_df.fillna(method ='ffill', inplace = True)
raw_df.head(2)
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INST
0	40.900749	0.818182	95.4	0.0	95.4	0.000000	0.166667	0.0	
1	3202.467416	0.909091	0.0	0.0	0.0	6442.945483	0.000000	0.0	

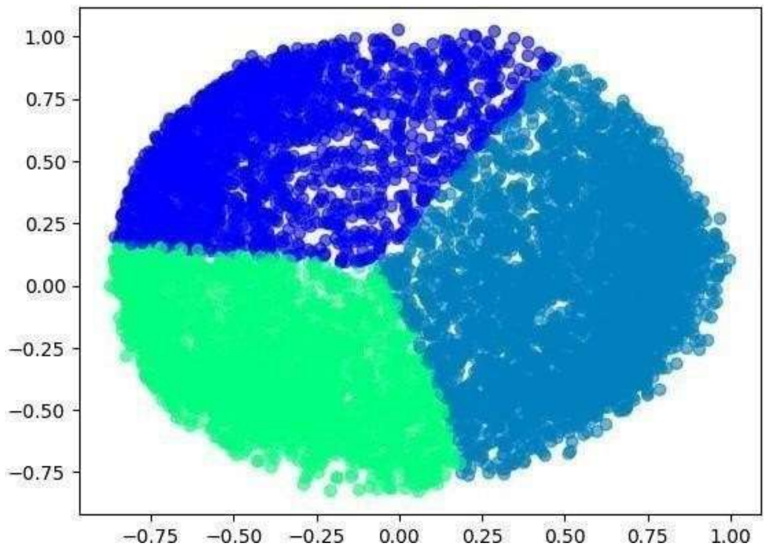
```
scaler = StandardScaler()
scaled_df = scaler.fit_transform(raw_df)
normalized_df = normalize(scaled_df)
normalized_df = pd.DataFrame(normalized_df)pca
= PCA(n_components = 2)
X_principal = pca.fit_transform(normalized_df)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
X_principal.head(2)
```

	P1	P2
0	-0.489949	-0.679976
1	-0.519099	0.544827

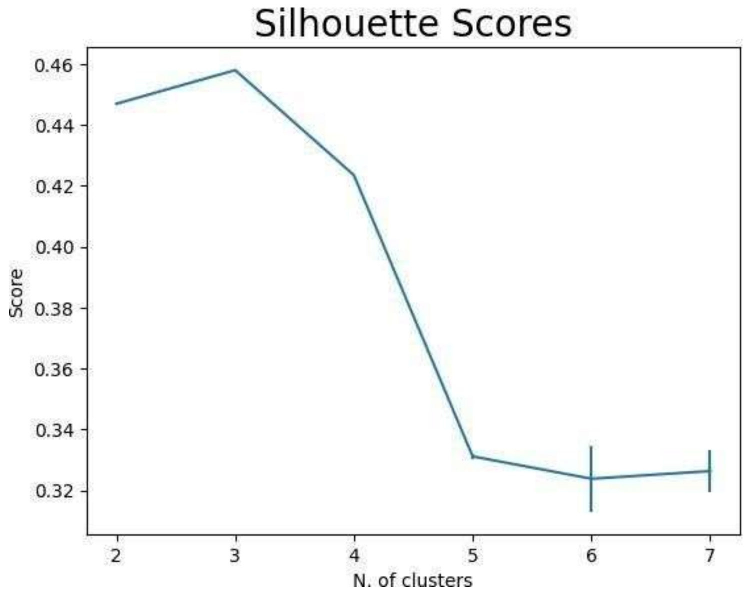
```
gmm=GaussianMixture(n_components = 3)
gmm.fit(X_principal)
```

```
▾ GaussianMixture
GaussianMixture(n_components=3)
```

```
plt.scatter(X_principal['P1'], X_principal['P2'],
c = GaussianMixture(n_components = 3).fit_predict(X_principal), cmap =plt.cm.winter, alpha =0.6)
plt.show()
```

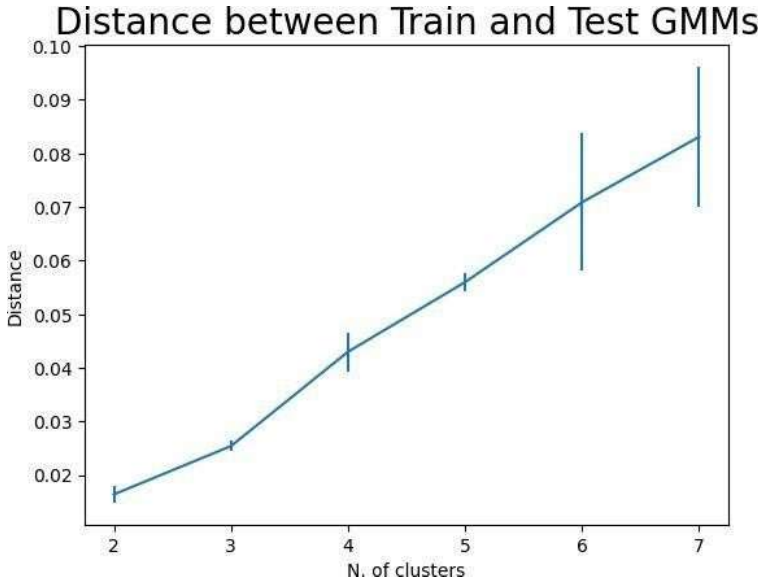



```
 def SelBest(arr:list, X:int)->list:
    dx=np.argsort(arr)[:X]
    return arr[dx]
n_clusters=np.arange(2, 8)
sils=[]
sils_err=[]
iterations=20
for n in n_clusters:
    tmp_sil=[]
    for _ in range(iterations):
        gmm=GaussianMixture(n, n_init=2).fit(X_principal)
        labels=gmm.predict(X_principal)
        sil=metrics.silhouette_score(X_principal, labels, metric='euclidean')
        tmp_sil.append(sil)
    val=np.mean(SelBest(np.array(tmp_sil), int(iterations/5)))
    err=np.std(tmp_sil)
    sils.append(val)
    sils_err.append(err)
plt.errorbar(n_clusters, sils, yerr=sils_err)
plt.title("Silhouette Scores", fontsize=20)
plt.xticks(n_clusters)
plt.xlabel("N. of clusters")
plt.ylabel("Score")
```

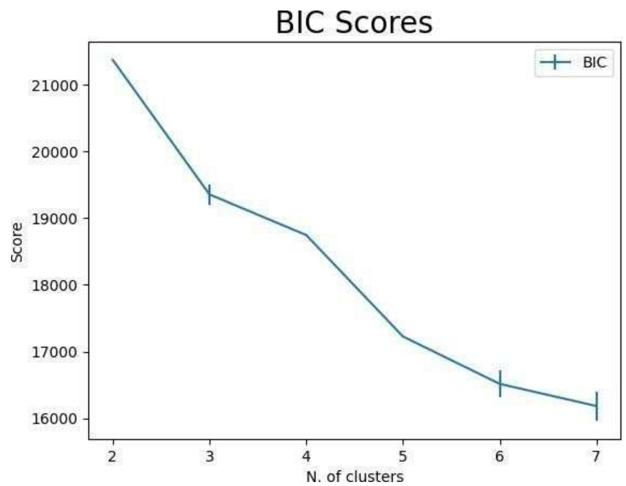



```
def gmm_js(gmm_p, gmm_q, n_samples=10**5):
    X=gmm_p.sample(n_samples)[0]
    log_p_X = gmm_p.score_samples(X)
    log_q_X = gmm_q.score_samples(X)
    log_mix_X = np.logaddexp(log_p_X, log_q_X)

    Y=gmm_q.sample(n_samples)[0]
    log_p_Y = gmm_p.score_samples(Y)
    log_q_Y = gmm_q.score_samples(Y)
    log_mix_Y = np.logaddexp(log_p_Y, log_q_Y)
    return np.sqrt((log_p_X.mean()- (log_mix_X.mean()- np.log(2))
    + log_q_Y.mean()- (log_mix_Y.mean()- np.log(2))) / 2)
n_clusters=np.arange(2, 8)
iterations=20
results=[]
res_sigs=[]
for n in n_clusters:
    dist=[]
    for iteration in range(iterations):
        train, test=train_test_split(X_principal, test_size=0.5)
        gmm_train=GaussianMixture(n, n_init=2).fit(train)
        gmm_test=GaussianMixture(n, n_init=2).fit(test)
        dist.append(gmm_js(gmm_train, gmm_test))
    selec=SelBest(np.array(dist), int(iterations/5))
    result=np.mean(selec)
    res_sig=np.std(selec)
    results.append(result)
    res_sigs.append(res_sig)

plt.errorbar(n_clusters, results, yerr=res_sigs)
plt.title("Distance between Train and Test GMMs", fontsize=20)
plt.xticks(n_clusters)
plt.xlabel("N. of clusters")
```

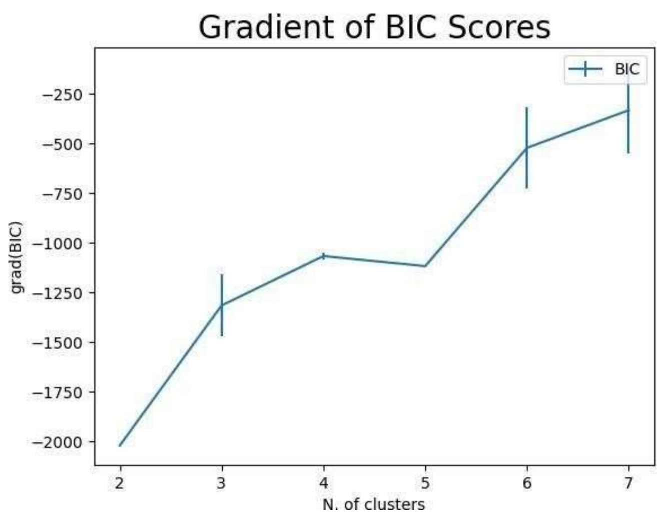
```
plt.ylabel("Distance")
plt.show()
```




```
n_clusters=np.arange(2, 8)
bics=[]
bics_err=[]
iterations=20
for n in n_clusters:
    tmp_bic=[]
    for _ in range(iterations):
        gmm=GaussianMixture(n, n_init=2).fit(X_principal)
        tmp_bic.append(gmm.bic(X_principal))
    val=np.mean(SelBest(np.array(tmp_bic), int(iterations/5)))
    err=np.std(tmp_bic)
    bics.append(val)
    bics_err.append(err)
plt.errorbar(n_clusters,bics, yerr=bics_err, label='BIC')
plt.title("BIC Scores", fontsize=20) plt.xticks(n_clusters)
plt.xlabel("N. of clusters")
plt.ylabel("Score")
plt.legend()
```




```
plt.errorbar(n_clusters, np.gradient(bics), yerr=bics_err, label='BIC')
plt.title("Gradient of BIC Scores", fontsize=20) plt.xticks(n_clusters)
plt.xlabel("N. of clusters")
plt.ylabel("grad(BIC)")
plt.legend()
```



Result:

Thus ,to write a python program for data and text clustering using Gaussian mixture model has been verified and executed successfully.

Ex. No: 10	Dimensionality Reduction Using Image Processing Applications
Date:	

Aim:

To write a python programming for Dimensionality reductional algorithms using Image processing Applications.

Algorithm:

Step 1: Load the dataset.

Step 2: Compute the means of the variables.

Step 3: Calculate the covariance variable and matrix by ordered pairs(xi,yi).

Step4: Compute the curse of dimensionality.

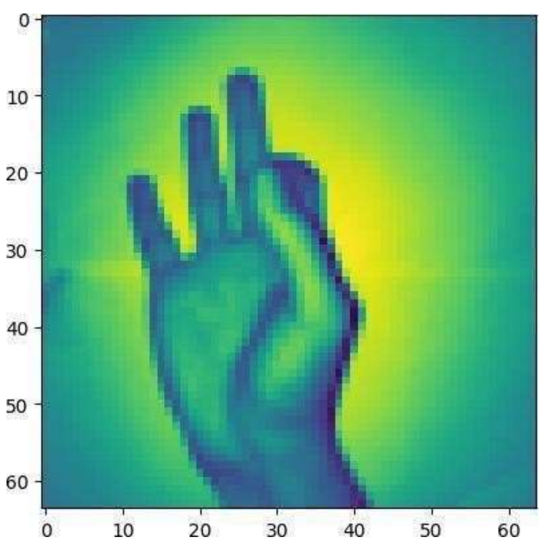
Step 5: Derive new dataset by calculating

Implementation:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_splitfrom
sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrixfrom
sklearn.decomposition import PCA

X=np.load('X.npy')
Y=np.load('Y.npy')
X.shape
```

```
(2062, 64, 64)
```

```
plt.imshow(X[0])
```




```
9- np.argmax(Y[0])
```

```
9
```

```
X_flat = np.array(X).reshape((2062, 64*64))
X_train, X_test, y_train, y_test = train_test_split(X_flat, Y, test_size=0.3, random_state=42)clf =
MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(20, 20, 20), random_state=1)
clf.fit(X_train, y_train)
```

```
MLPClassifier
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(20, 20, 20), random_state=1)
```

```
y_hat = clf.predict(X_test)
print("accuracy: " + str(accuracy_score(y_test, y_hat)))
```

```
accuracy: 0.34733441033925 69
```

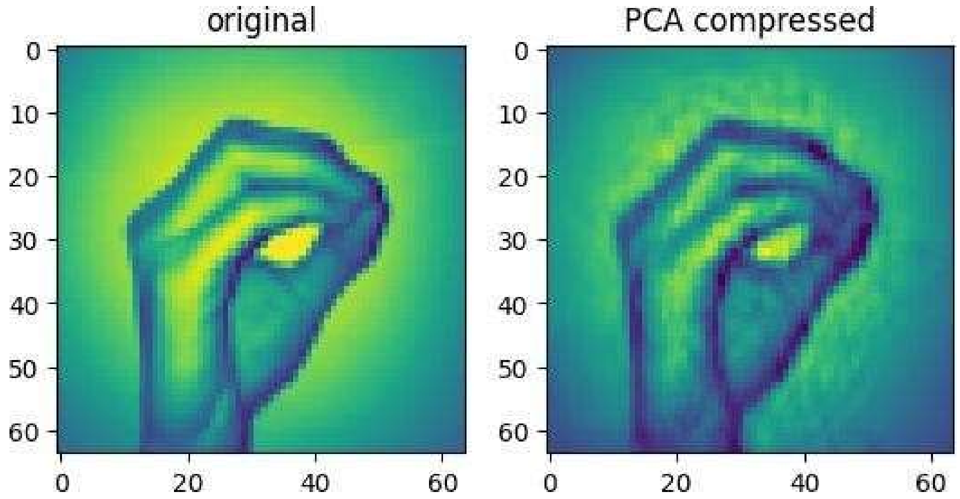
```
pca_dims = PCA()
pca_dims.fit(X_train)
cumsum= np.cumsum(pca_dims.explained_variance_ratio_)d
=np.argmax(cumsum >= 0.95) + 1
d
```

```
292
```

```
pca = PCA(n_components=d)
X_reduced = pca.fit_transform(X_train) X_recovered =
pca.inverse_transform(X_reduced) print("reduced shape: " +
str(X_reduced.shape))
print("recovered shape: " + str(X_recovered.shape))
```

```
reduced shape: (1443, 292)
recovered shape: (1443, 4096)
```

```
f = plt.figure()
f.add_subplot(1,2, 1)
plt.title("original")
plt.imshow(X_train[0].reshape((64,64)))f.add_subplot(1,2, 2)
plt.title("PCA compressed")
plt.imshow(X_recovered[0].reshape((64,64)))plt.show(block=True)
```




```
clf_reduced = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(20, 20,20))
clf_reduced.fit(X_reduced, y_train)
```

```
MLPClassifier
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(20, 20, 20))
```

```
X_test_reduced = pca.transform(X_test) y_hat_reduced =
clf_reduced.predict(X_test_reduced)
print("accuracy: " + str(accuracy_score(y_test, y_hat_reduced)))
```

```
accuracy: 0.630048465266559
```

Result:

Thus, the implementation of python programming for Dimensionality reduction algorithms using Image processing Applications has been completed and verified successfully.

Ex. No: 11	Implementation of Sampling methods
Date:	

Aim:

To write a python programming for Implementation of Sampling methods.

Algorithm:

Step 1: Import all the libraries.

Step 2: Calculate Setup the model by identifying the dependent variable.

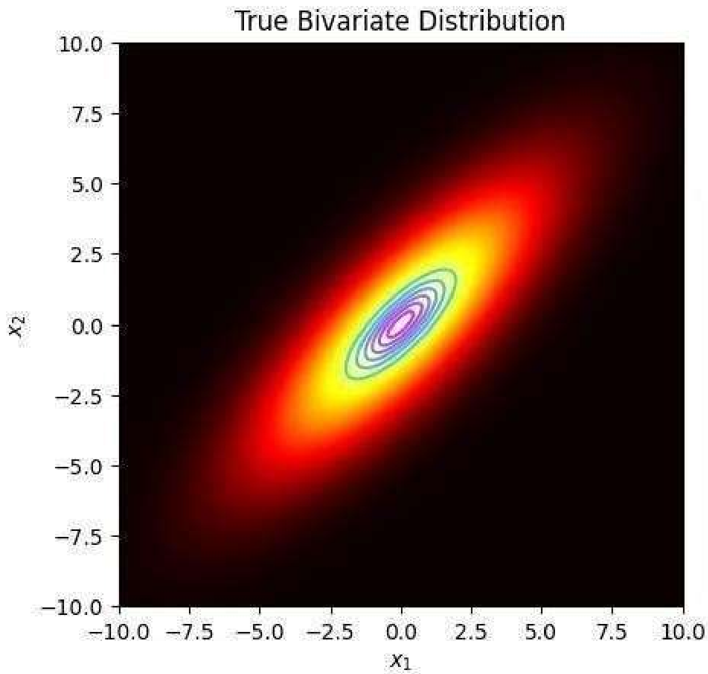
Step 3: Specify the probability distribution for independent variables .

Step 4: Run iterative simulatins by generating enough possible values for independent variables.

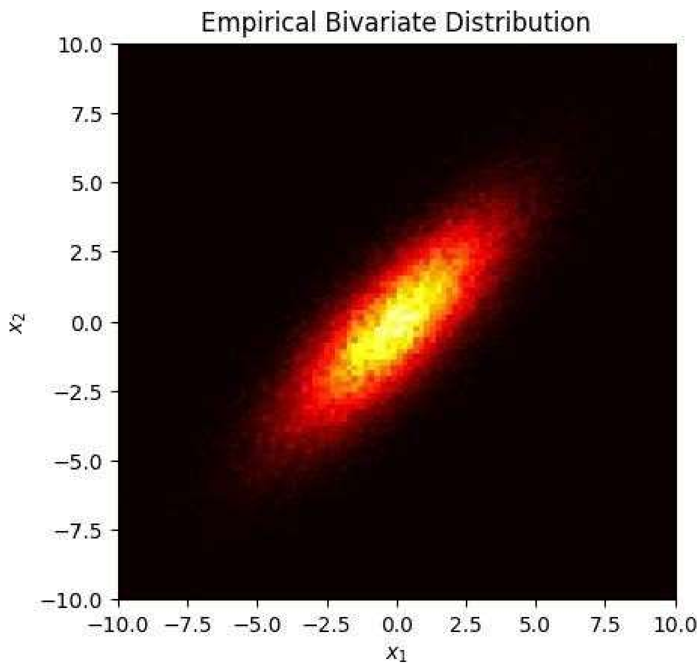
Step 5: Visualize the results in the plot.

Implementation:

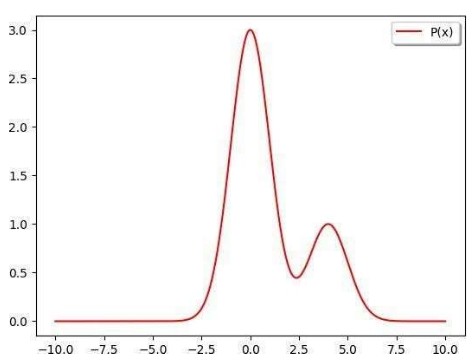
```
%matplotlib inline
from___future____import print_function
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as LA
def multivariate_normal(X, mu=np.array([[0, 0]]), sig=np.array([[1, 0.8], [0.8, 1]])):
    sqrt_det_2pi_sig = np.sqrt(2 * np.pi * LA.det(sig))
    sig_inv = LA.inv(sig)
    X = X[:, None, :] - mu[None, :, :]
    return np.exp(-np.matmul(np.matmul(X, np.expand_dims(sig_inv, 0)), (X.transpose(0, 2,
1)))/2)/sqrt_det_2pi_sig
x = np.linspace(-3, 3, 1000)
X = np.array(np.meshgrid(x, x)).transpose(1, 2, 0)X =
np.reshape(X, [X.shape[0] * X.shape[1], -1]) z =
multivariate_normal(X)
plt.imshow(z.squeeze().reshape([x.shape[0], -1]), extent=[-10, 10, -10, 10], cmap='hot',
origin='lower')
plt.contour(x, x, z.squeeze().reshape([x.shape[0], -1]), cmap='cool')
plt.title('True Bivariate Distribution')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.show()
```



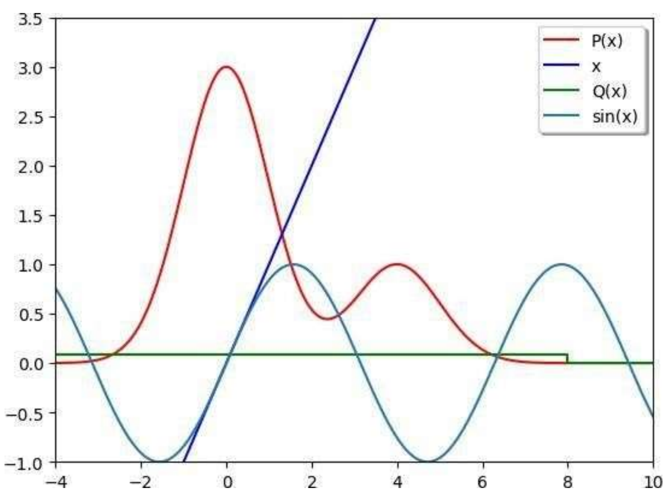

```
x0 = [0, 0]
xt = x0
b = 0.8
samples = []
for i in range(100000):
    x1_t   =   np.random.normal(b*xt[1],   1-b*b)
    x2_t = np.random.normal(b*x1_t, 1-b*b)xt =
    [x1_t, x2_t]
    samples.append(xt)
burn_in = 1000
samples = np.array(samples[burn_in:])
im, x_, y_ = np.histogram2d(samples[:, 0], samples[:, 1], bins=100)
plt.imshow(im,   extent=[-10,   10,   -10,   10],   cmap='hot',   origin='lower',   interpolation='nearest')
plt.title('Empirical Bivariate Distribution')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.show()
```




```
%matplotlib inline
from___future____import print_function
import numpy as np
import matplotlib.pyplot as plt
P = lambda x: 3 * np.exp(-x*x/2) + np.exp(-(x - 4)**2/2)
Z = 10.0261955464
x_vals = np.linspace(-10, 10, 1000)
y_vals = P(x_vals)
plt.figure(1)
plt.plot(x_vals, y_vals, 'r', label='P(x)')
plt.legend(loc='upper right', shadow=True)
plt.show()
```



```
f_x = lambda x: x
g_x = lambda x: np.sin(x) true_expected_fx
= 10.02686647165
true_expected_gx = -1.15088010640
a, b = -4, 8
uniform_prob = 1./(b - a)
plt.figure(2)
plt.plot(x_vals, y_vals, 'r', label='P(x)')
plt.plot(x_vals, f_x(x_vals), 'b', label='x')
plt.plot([-10, a, a, b, b, 10], [0, 0, uniform_prob, uniform_prob, 0, 0], 'g', label='Q(x)')
plt.plot(x_vals, np.sin(x_vals), label='sin(x)')
plt.xlim(-4, 10)
plt.ylim(-1, 3.5)
plt.legend(loc='upper right', shadow=True)
plt.show()
```




```
expected_f_x = 0.
expected_g_x = 0.
n_samples = 100000
den = 0.
for i in range(n_samples):
    sample = np.random.uniform(a, b)
    importance = P(sample) / uniform_probden
    += importance
```

```
expected_f_x += importance * f_x(sample)
expected_g_x += importance * g_x(sample)
expected_f_x /= den
expected_g_x /= den
expected_f_x *= Z
expected_g_x *= Z
print('E[f(x)] = %.5f, Error = %.5f' % (expected_f_x, abs(expected_f_x - true_expected_fx)))
print('E[g(x)] = %.5f, Error = %.5f' % (expected_g_x, abs(expected_g_x - true_expected_gx)))
```

```
E[f(x)] = 10.14677, Error = 0.11990
E[g(x)] = -1.16472, Error = 0.01384
```

Result:

Thus, the implementation of python programming for Implementation of sampling methods has been completed and verified successfully.

Ex.No: 12	**Application of Hidden Markov Model for Weather prediction**
Date:	

Aim:

To write a python programming for Implementing the application of Hidden Markov model for weather prediction.

Algorithm:

Step 1: Define the state space and observation space.

Step 2: Define the initial state distribution.

Step 3: Define the state transition probabilities and observation likelihoods.

Step 4: Train the model.

Step 5: Decode the most likely sequence of hidden states.

Step 6: Evaluate the model.

Implementation:

```
import numpy as np
P_transition = np.array([[0.75, 0.15, 0.10],
                         [0.25, 0.55, 0.20],
                         [0.30, 0.30, 0.40]])
P_emission = np.array([[0.75, 0.15, 0.65],
                       [0.25, 0.85, 0.35]])
P_init = [0.65, 0.20, 0.15]
T = 3
hidden_states = np.zeros((T,), dtype=np.int32)
probs = np.zeros((T, 3))
for t in range(T):if
    t == 0:
        probs[t, :] = P_init * P_transition[1, :]
    else:
        probs[t, :] = np.max(probs[t-1, :, None] * P_transition, axis=0)
    hidden_states[t] = np.argmax(probs[t, :])
state_names = ['Sunny', 'Rainy', 'Foggy']
forecast = [state_names[s] for s in hidden_states]
print(forecast)
```

```
['Sunny', 'Sunny', 'Sunny']
```

```
P_transition = np.array([[0.75, 0.15, 0.10],
                         [0.25, 0.55, 0.20],
                         [0.30, 0.30, 0.40]])

P_emission = np.array([[0.75, 0.15, 0.65],
```

```
                      [0.25, 0.85, 0.35],
                      [0.0, 0.0, 0.0]])
P_init = np.array([0.65, 0.20, 0.15])
P_hidden_init = P_transition[1, :]
alpha = np.zeros((3,))
for i in range(3):
    alpha[i] = P_emission[i, 0] * P_init[i]for
t in range(1, 3):
    alpha = np.dot(alpha, P_transition) * P_emission[:, t]
P_evidence = np.sum(alpha)
posteriors = alpha / P_evidence state_names
= ['Sunny', 'Rainy', 'Foggy']for i in range(3):
    print("Probability of being in state %s: %.4f" % (state_names[i], posteriors[i]))
most_likely_weather = state_names[np.argmax(posteriors)]
print("The most likely weather is %s." % most_likely_weather)
```

```
Probability of being in state Sunny: 0.6812
Probability of being in state Rainy: 0.3188
Probability of being in state Foggy: 0.0000
The most likely weather is Sunny.
```

```
E = ["no umbrella", "umbrella", "umbrella", "no umbrella", "umbrella", "no umbrella"]alpha =
np.zeros((3,))
for i in range(3):
    alpha[i] = P_emission[i, 0] * P_init[i]for
t in range(1, len(E)):
    alpha = np.dot(alpha, P_transition) * P_emission[:, t % 2]
final_state = np.argmax(alpha)
state_names = ['Sunny', 'Rainy', 'Foggy']
most_likely_weather = state_names[final_state]
print("Final state:", state_names[final_state])
print("Most likely weather:", most_likely_weather)
```

```
Final state: Rainy
Most likely weather: Rainy
```

```
P_emission = np.array([[0.6, 0.4, 0.0, 0.0],
                      [0.0, 0.0, 0.5, 0.5],
                      [0.3, 0.3, 0.2, 0.2]])
P_transition = np.array([[0.7, 0.2, 0.1],
                        [0.2, 0.5, 0.3],
                        [0.3, 0.3, 0.4]])

E = [0, 1, 2, 2, 1, 0]
T = len(E)
trellis = np.zeros((T, 3))
backpointers = np.zeros((T-1, 3), dtype=np.int64)for i
in range(3):
    trellis[0, i] = P_emission[i, E[0]] * P_transition[i, 0]for t
in range(1, T):
    for j in range(3):
        prob_transitions = trellis[t - 1, :] * P_transition[:, j]
```

```
        trellis[t, j] = P_emission[j, E[t]] * np.max(prob_transitions)
        backpointers[t - 1, j] = np.argmax(prob_transitions)
hidden_states = [np.argmax(trellis[-1])]for
i in range(T-2, -1, -1):
    hidden_states.append(backpointers[i, hidden_states[-1]])
hidden_states.reverse()
print("The most likely sequence of hidden states is:", hidden_states)
```

```
The most likely sequence of hidden states is: [0, 0, 1, 1, 0, 0]
```

Result:

Thus, the implementation of python programming for Implementing the application of Hidden Markov Model for weather prediction has been completed and verified successfully.