INDEX

Exp No	List of experiments	Marks	Sign
1	Load Real Time data Set and Python Libraries, Installing Libraries through Anaconda Prompt, Perform data pre-processing through Pandas Library.		
2	Implement the Naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets		
3	Implement decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample		
4	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs		
5	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering		
6	Implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem		
7	Implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem		
8	Implement Q Learning with Linear Function Approximation.		
9	Implement the Policy Gradient concept in Reinforcement learning. Compare the Reinforce with Baseline and Actor Critic with Baseline		
10	Consider a time series data set. Plot the data, identify the components of the Time Series data, calculate the seasonality and stationarity and Identify the trend pattern present in the time series data. Remove the white noise if available in the time series data.		

Ex No: 1 Date :	Installation of Anaconda Python

AIM

Load Real Time data Set and Python Libraries, Installing Libraries through Anaconda Prompt, Perform data pre-processing through Pandas Library.

ALGORITHM

Step 1: In your browser, download the Anaconda installer for Linux.

Step 2: Search for “terminal” in your applications and click to open.

Step 3: Verify the installer’s data integrity with SHA-256. For more information on hash verification, see cryptographic hash validation.

In the terminal, run the following:

shasum -a 256 /PATH/FILENAME

Replace /PATH/FILENAME with your installation's path and filename.

Step 4: Install for Python 3.7 or 2.7 in the terminal:

For Python 3.7, enter the following:

Include the bash command regardless of whether or not you are using the Bash shell

bash ~/Downloads/Anaconda3-2020.05-Linux-x86_64.sh

Replace ~/Downloads with your actual path

Replace the .sh file name with the name of the file you downloaded

For Python 2.7, enter the following:

Include the bash command regardless of whether or not you are using the Bash shell

bash ~/Downloads/Anaconda2-2019.10-MacOSX-x86_64.sh

Replace ~/Downloads with your actual path

Replace the .sh file name with the name of the file you downloaded

Step 5: Press Enter to review the license agreement. Then press and hold Enter to scroll.

Step 6: Enter “yes” to agree to the license agreement.

Step 7: Use Enter to accept the default install location, use CTRL+C to cancel the installation, or enter another file path to specify an alternate installation directory. If you accept the default install location, the installer displays PREFIX=/home/<USER>/anaconda<2/3> and continues the installation. It may take a few minutes to complete.

. Step 8: The installer prompts you to choose whether to initialize Anaconda Distribution by running conda init. Anaconda recommends entering “yes”.

If you enter “no”, then conda will not modify your shell scripts at all. In order to initialize after the installation process is done, first run source [PATH TO CONDA]/bin/activate and then run conda init. See FAQ.

Step 9: The installer finishes and displays, “Thank you for installing Anaconda<2/3>!”

Step 10: Close and re-open your terminal window for the installation to take effect, or enter the command source ~/.bashrc to refresh the terminal.

Step 11: You can also control whether or not your shell has the base environment activated each time it opens.

```
# The base environment is activated by default
conda config --set auto_activate_base True
# The base environment is not activated by default
conda config --set auto_activate_base False
# The above commands only work if conda init has been run first
# conda init is available in conda versions 4.6.12 and later
```

Step 12: Verify your installation.

OUTPUT

Anaconda installed and verified

RESULT

The installation process is completed successfully.

Ex No: 2 Date :	Naive Bayesian classifier

AIM

Implement the Naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few tests data sets

ALGORITHM

Step 1: Load the training data set from the CSV file into a pandas Data Frame.

Step 2: Separate the feature columns and the target column in the Data Frame.

Step 3: Calculate the prior probabilities of each class (target variable) in the training data set.

Step 4: For each feature column, calculate the conditional probabilities for each class.

Step 5: Store the probabilities calculated in steps 3 and 4 in a dictionary for later use.

Step 6: Load the test data set from the CSV file into a new panda Data Frame.

Step 7: For each row in the test data set, calculate the probability of the row belonging to each class using the probabilities calculated in step 5.

Step 8: choose the class with the highest probability as the predicted class for each row in the test data set.

Step 9: Compare the predicted classes with the actual classes in the test data set to calculate the accuracy of the classifier.

PROGRAM

```
def main():
population = 1000000
P_ill = 0.01
P_positive_if_ill = 0.99 # sensitivity
P_negative_if_healthy = 0.99 # specificity
calculate_without_bayes(population, P_ill, P_positive_if_ill, P_negative_if_healthy)

print()
calculate_with_bayes(P_ill, P_positive_if_ill, P_negative_if_healthy)
```

```
def calculate_without_bayes(population, P_ill, P_positive_if_ill, P_negative_if_healthy):
heading = "Calculate P(ill | positive) without Bayes' Theorem"
print(heading)
print("=" * len(heading) + "\n")

percent_ill = P_ill * 100
number_ill = population * P_ill
number_healthy = population * (1 - P_ill)
ill_positive = number_ill * P_positive_if_ill
healthy_positive = number_healthy * (1 - P_negative_if_healthy)
P_ill_if_positive = ill_positive / (ill_positive + healthy_positive)

print(f"Population:              {population}")
print(f"Percent ill:             {percent_ill}%")
print(f"Number ill:              {number_ill:>.0f}")
print(f"Number healthy:          {number_healthy:>.0f}")
print(f"P(positive if ill):      {P_positive_if_ill}")
print(f"P(negative if healthy):  {P_negative_if_healthy}")
print(f"Ill and test positive:   {ill_positive:>.0f}")
print(f"Healthy but test positive: {healthy_positive:>.0f}")
print(f"P(ill | positive):       {P_ill_if_positive:>.2f}")

def calculate_with_bayes(P_ill, P_positive_if_ill, P_negative_if_healthy):
P_healthy = 1 - P_ill
P_positive_if_healthy = 1 - P_negative_if_healthy
P_ill_if_positive = (P_positive_if_ill * P_ill) / ((P_healthy * P_positive_if_healthy) + (P_ill *
P_positive_if_ill))

heading = "Calculate P(ill | positive) with Bayes' Theorem"
```

```
print(heading)
print("=" * len(heading) + "\n")

print(f"P(ill):                  {P_ill}")
print(f"P(healthy):              {P_healthy}")
print(f"P(positive if ill):      {P_positive_if_ill}")
print(f"P(positive if healthy): {P_positive_if_healthy:>.2f}\n")

print("   P(positive if ill) * P(ill)")
print("P(ill | positive) = ------------------------------------------------------------------")
print("   P(healthy) * P(positive if healthy) + P(ill) * P(positive if ill)")

print("\n")

print(f" {P_positive_if_ill} * {P_ill}")
print("  = ------------------------------------------------------------------")
print(f"  {P_healthy} * {P_positive_if_healthy:>.2f} + {P_ill} * {P_positive_if_ill}")

print("\n")

print(f   = {P_ill_if_positive:>.2f}")

main()
```

OUTPUT

```
 Population:               1000000
Percent ill:               1.0%
Number ill:                10000
Number healthy:            990000
P(positive if ill):        0.99
P(negative if healthy):    0.99
Ill and test positive:     9900
Healthy but test positive: 9900
P(ill | positive):         0.50


Calculate P(ill | positive) with Bayes' Theorem
================================================
P(ill):                    0.01
P(healthy):                0.99
P(positive if ill):        0.99
P(positive if healthy): 0.01


                         P(positive if ill) * P(ill)
P(ill | positive) = -----------------------------------------------------------------
              P(healthy) * P(positive if healthy) + P(ill) * P(positive if ill)


                             0.99 * 0.01
            = -----------------------------------------------------------------
                          0.99 * 0.01 + 0.01 * 0.99


            = 0.50
```

RESULT

The given python program is executed successfully

Ex No: 3 Date :	ID3 algorithm

AIM

Implement decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample

ALGORITHM

Step 1: Define a function to calculate the entropy of a given dataset.

Step 2: Define a function to calculate the information gain of a given dataset and its features.

Step 3: Define a function to recursively build the decision tree based on the information gain of each feature.

Step 4: Define a function to classify a new sample based on the decision tree.

Step 5: Load the dataset from a CSV file.

Step 6: Split the dataset into training and testing sets.

Step 7: Build the decision tree based on the training set.

Step 8: Classify the testing set using the decision tree.

Step 9: Evaluate the accuracy of the classifier.

PROGRAM

```
import pandas as pd

import math

import numpy as np


data = pd.read_csv("Dataset/4-dataset.csv")

features = [feat for feat in data]

features.remove("answer")
```

Create a class named Node with four members children, value, isLeaf and pred.

```
class Node:
```

```
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
```

Define a function called entropy to find the entropy oof the dataset.

```
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
```

Define a function named info_gain to find the gain of the attribute

```
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
```

```
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
```

Define a function named ID3 to get the decision tree for the given dataset

```
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
```

```
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

    return root
```

Define a function named printTree to draw the decision tree

```
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
```

```
    print()
    for child in root.children:
        printTree(child, depth + 1)
```

Define a function named classify to classify the new example

```
def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
                exit
            else:
                classify (child.children[0], new)
```

Finally, call the ID3, printTree and classify functions

```
root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}
classify (root, new)
```

DATASET

	A	B	C	D	E
1	outlook	temperature	humidity	wind	answer
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no

OUTPUT

Decision Tree is:

```
outlook
    overcast ->  ['yes']

    rain
        wind
            strong ->  ['no']

            weak ->  ['yes']

    sunny
        humidity
            high ->  ['no']

            normal ->  ['yes']

------------------
Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind': 'strong'}  is: ['yes']
```

RESULT

The given python program is executed successfully

Ex No: 4 Date :	Weighted Regression algorithm

AIM

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

ALGORITHM

Step 1: Load the dataset from a CSV file.

Step 2: Define a function to calculate the weights for each training example based on its distance from the test example.

Step 3: Define a function to fit a test example using the locally weighted regression algorithm.

Step 4: Define a function to fit all test examples using the locally weighted regression algorithm.

Step 5: Draw a graph to show the fitted curve for the entire dataset.

PROGRAM

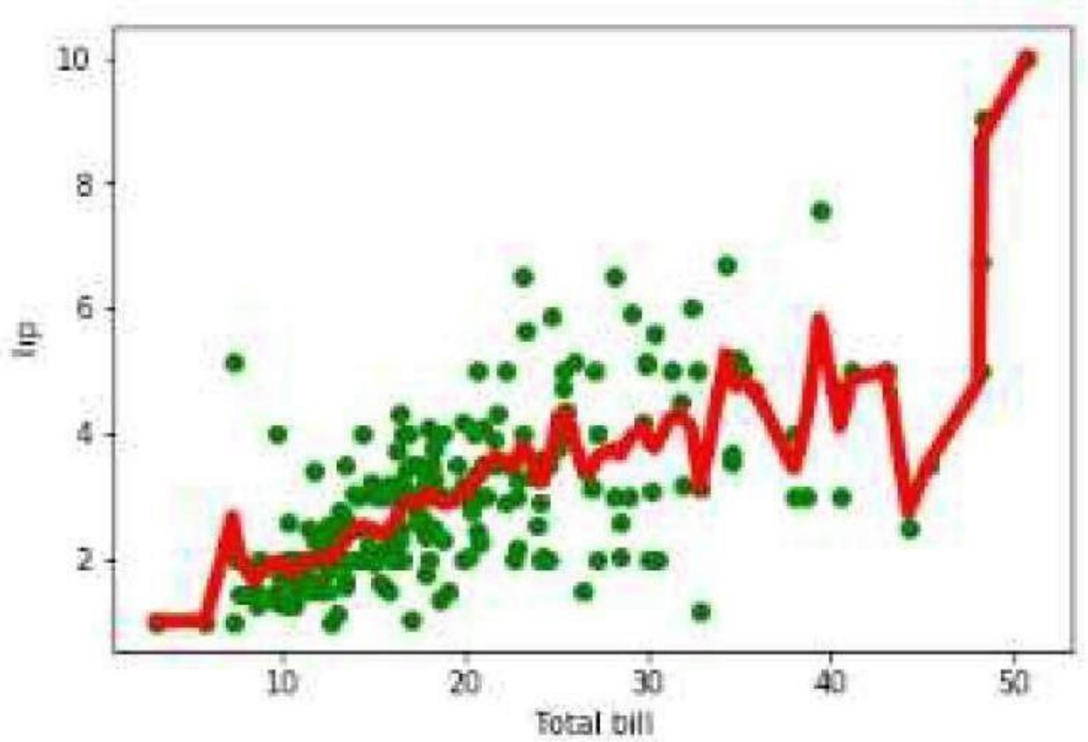
```
from numpy import *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W=(X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
```

```
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

# load data points
data = pd.read_csv('data10.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)
m= np1.shape(mbill)[1]
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,2)


SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]


fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],
  ypred[SortIndex], color = 'red',
linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

DATASET

	A	B	C	D	E	F	G
1	total_bill	tip	sex	smoker	day	time	size
2	16.99	1.01	Female	No	Sun	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3
4	21.01	3.5	Male	No	Sun	Dinner	3
5	23.68	3.31	Male	No	Sun	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4
7	25.29	4.71	Male	No	Sun	Dinner	4
8	8.77	2	Male	No	Sun	Dinner	2
9	26.88	3.12	Male	No	Sun	Dinner	4
10	15.04	1.96	Male	No	Sun	Dinner	2
11	14.78	3.23	Male	No	Sun	Dinner	2
12	10.27	1.71	Male	No	Sun	Dinner	2
13	35.26	5	Female	No	Sun	Dinner	4
14	15.42	1.57	Male	No	Sun	Dinner	2
15	18.43	3	Male	No	Sun	Dinner	4
16	14.83	3.02	Female	No	Sun	Dinner	2
17	21.58	3.92	Male	No	Sun	Dinner	2
18	10.33	1.67	Female	No	Sun	Dinner	3
19	16.29	3.71	Male	No	Sun	Dinner	3
20	16.97	3.5	Female	No	Sun	Dinner	3
21	20.65	3.35	Male	No	Sat	Dinner	3
22	17.92	4.08	Male	No	Sat	Dinner	2
23	20.29	2.75	Female	No	Sat	Dinner	2
24	15.77	2.23	Female	No	Sat	Dinner	2
25	39.42	7.58	Male	No	Sat	Dinner	4
26	19.82	3.18	Male	No	Sat	Dinner	2
27	17.81	2.34	Male	No	Sat	Dinner	4
28	13.37	2	Male	No	Sat	Dinner	2
29	12.69	2	Male	No	Sat	Dinner	2
30	21.7	4.3	Male	No	Sat	Dinner	2
31	19.65	3	Female	No	Sat	Dinner	2
32	9.55	1.45	Male	No	Sat	Dinner	2
33	18.35	2.5	Male	No	Sat	Dinner	4
34	15.06	3	Female	No	Sat	Dinner	2
35	20.69	2.45	Female	No	Sat	Dinner	4
36	17.78	3.27	Male	No	Sat	Dinner	2
37	24.06	3.6	Male	No	Sat	Dinner	3
38	16.31	2	Male	No	Sat	Dinner	3
39	16.93	3.07	Female	No	Sat	Dinner	3
40	18.69	2.31	Male	No	Sat	Dinner	3

OUTPUT



RESULT

The given python program is executed successfully

Ex No: 5 Date :	EM algorithm

AIM

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering

ALGORITHM

Step 1: Load the dataset from a CSV file.

Step 2: Define the number of clusters **K**.

Step 3: Initialize the means, covariances, and mixing coefficients of the Gaussian mixture model (GMM) using the k-Means algorithm.

Step 4: Implement the EM algorithm to update the means, covariances, and mixing coefficients until convergence.

Step 5: Use the GMM to cluster the data.

Step 6: Calculate the clustering performance metrics, such as the silhouette score, for the GMM and k-Means algorithms.

Step 7: Compare the clustering results and comment on the quality of clustering.

PROGRAM

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("8-dataset.csv", names=names)
```

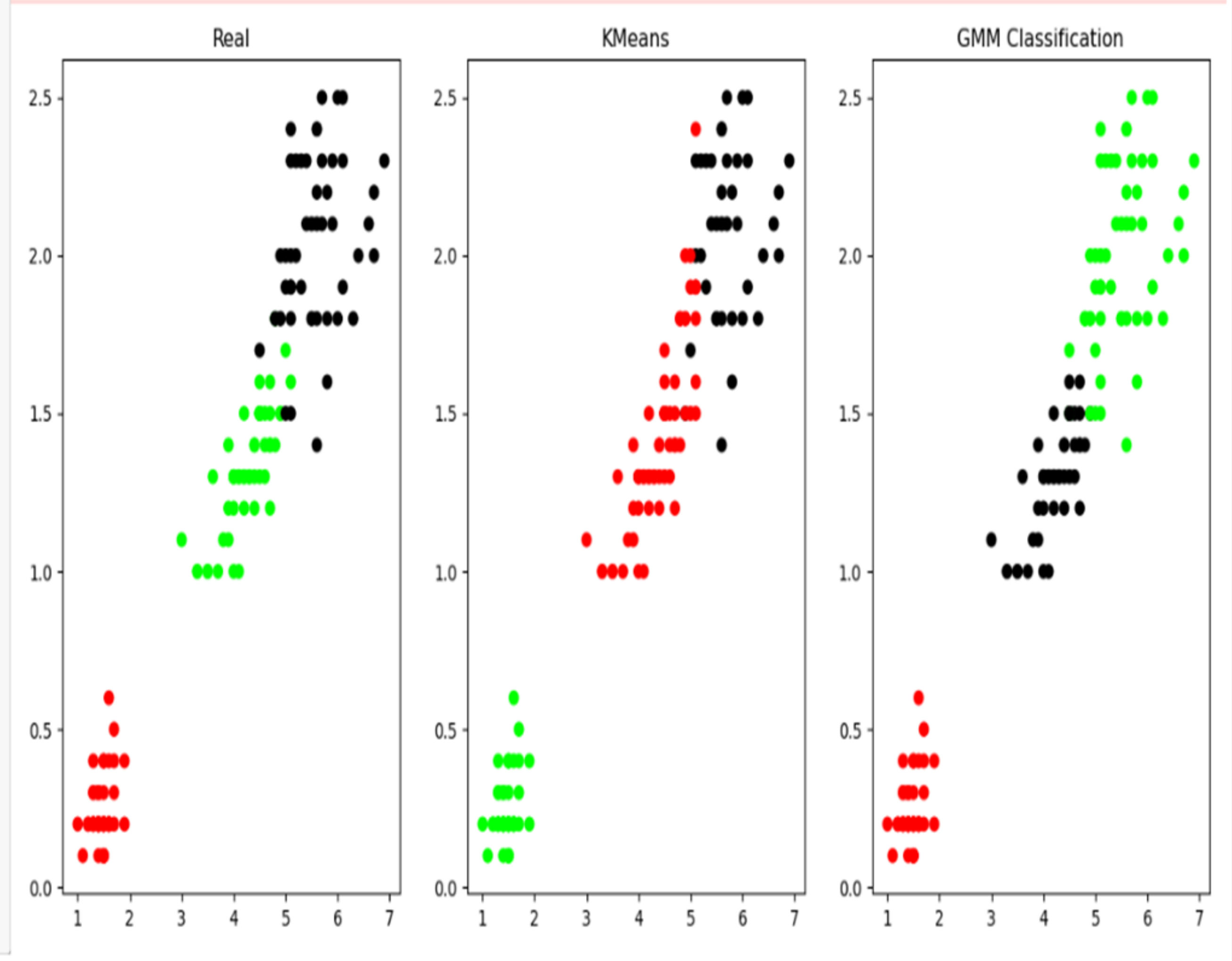
```
X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])


# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

DATASET

	A	B	C	D	E
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3	1.4	0.1	Iris-setosa
14	4.3	3	1.1	0.1	Iris-setosa
15	5.8	4	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa
20	5.1	3.8	1.5	0.3	Iris-setosa
21	5.4	3.4	1.7	0.2	Iris-setosa
22	5.1	3.7	1.5	0.4	Iris-setosa
23	4.6	3.6	1	0.2	Iris-setosa
24	5.1	3.3	1.7	0.5	Iris-setosa
25	4.8	3.4	1.9	0.2	Iris-setosa
26	5	3	1.6	0.2	Iris-setosa
27	5	3.4	1.6	0.4	Iris-setosa
28	5.2	3.5	1.5	0.2	Iris-setosa
29	5.2	3.4	1.4	0.2	Iris-setosa
30	4.7	3.2	1.6	0.2	Iris-setosa
31	4.8	3.1	1.6	0.2	Iris-setosa
32	5.4	3.4	1.5	0.4	Iris-setosa
33	5.2	4.1	1.5	0.1	Iris-setosa
34	5.5	4.2	1.4	0.2	Iris-setosa
35	4.9	3.1	1.5	0.1	Iris-setosa
36	5	3.2	1.2	0.2	Iris-setosa
37	5.5	3.5	1.3	0.2	Iris-setosa
38	4.9	3.1	1.5	0.1	Iris-setosa
39	4.4	3	1.3	0.2	Iris-setosa
40	5.1	3.4	1.5	0.2	Iris-setosa

OUTPUT

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:
 [[ 0 50  0]
 [48  0  2]
```



RESULT

The given python program is executed successfully

Ex No:6 Date :	k-Nearest Neighbor algorithm

AIM

Implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem

ALGORITHM

Step 1: Load the iris data set from a library or CSV file.
Step 2: Split the data set into a training set and a test set.
Step 3: Choose the value of k.
Step 4: For each test instance, calculate the distances to all the training instances.
Step 5: Select the k instances with the smallest distances.
Step 6: Classify the test instance based on the majority class of the k nearest neighbors.
Step 7: Calculate the accuracy of the classifier.
Step 8: Print the correct and wrong predictions.

PROGRAM

```
import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn import metrics


names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']


# Read dataset to pandas dataframe

dataset = pd.read_csv("9-dataset.csv", names=names)

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

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

```
print(X.head())

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)


classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)


ypred = classifier.predict(Xtest)


i = 0

print ("\n---------------------------------------------------------------------------")

print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))

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

for label in ytest:

    print ('%-25s %-25s' % (label, ypred[i]), end="")

    if (label == ypred[i]):

        print (' %-25s' % ('Correct'))

    else:

        print (' %-25s' % ('Wrong'))

    i = i + 1

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

print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))

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

print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))

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

print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))

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

DATASET

	A	B	C	D	E
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3	1.4	0.1	Iris-setosa
14	4.3	3	1.1	0.1	Iris-setosa
15	5.8	4	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa
20	5.1	3.8	1.5	0.3	Iris-setosa
21	5.4	3.4	1.7	0.2	Iris-setosa
22	5.1	3.7	1.5	0.4	Iris-setosa
23	4.6	3.6	1	0.2	Iris-setosa
24	5.1	3.3	1.7	0.5	Iris-setosa
25	4.8	3.4	1.9	0.2	Iris-setosa
26	5	3	1.6	0.2	Iris-setosa
27	5	3.4	1.6	0.4	Iris-setosa
28	5.2	3.5	1.5	0.2	Iris-setosa
29	5.2	3.4	1.4	0.2	Iris-setosa
30	4.7	3.2	1.6	0.2	Iris-setosa
31	4.8	3.1	1.6	0.2	Iris-setosa
32	5.4	3.4	1.5	0.4	Iris-setosa
33	5.2	4.1	1.5	0.1	Iris-setosa
34	5.5	4.2	1.4	0.2	Iris-setosa
35	4.9	3.1	1.5	0.1	Iris-setosa
36	5	3.2	1.2	0.2	Iris-setosa
37	5.5	3.5	1.3	0.2	Iris-setosa
38	4.9	3.1	1.5	0.1	Iris-setosa
39	4.4	3	1.3	0.2	Iris-setosa
40	5.1	3.4	1.5	0.2	Iris-setosa

OUTPUT

Confusion matrix is as follows

[[4 0 0]

[0 4 0]

[0 2 5]]

Classification report

	precision	recall	Fl-score	Support
Iris setosa	1.00	1.00	1.00	4
Iris versicolor	0.67	1.00	0.80	4
Iris verginica	1.00	0.71	0.83	7
Avg/total	0.91	0.87	0.87	15

Accuracy of the classifier is: 0.87

RESULT

The given python program is executed successfully.

Ex No:7 Date :	Semi Supervised Classifier

AIM

Assuming a set of documents that need to be classified, use the Semi Supervised Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

ALGORITHM

Step 1: Load the set of documents from a library or CSV file.

Step 2: Split the set of documents into a labelled set and an unlabelled set.

Step 3: Train a classifier on the labelled set using a supervised learning algorithm, such as Naive Bayes or Support Vector Machines.

Step 4: Use the trained classifier to classify the unlabelled set.

Step 5: For each document in the unlabelled set, assign a label based on the majority class of its k nearest labelled neighbours.

Step 6: Repeat steps 3-5 until convergence or a maximum number of iterations is reached.

Step 7: Calculate the accuracy, precision, and recall of the classifier.

Step 8: Print the evaluation metrics.

PROGRAM

```
import pandas as pd

msg=pd.read_csv('naivetext1.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
print(X)
print(y)
```

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

from sklearn import metrics
print('Accuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('Recall and Precison ')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
```

DATASET

1	I love this	pos
2	This is an	pos
3	I feel very	pos
4	This is my	pos
5	What an a	pos
6	I do not lik	neg
7	I am tired	neg
8	I canâ€™t	neg
9	He is my s	neg
10	My boss is	neg
11	This is an	pos
12	I do not lik	neg
13	I love to d	pos
14	I am sick a	neg
15	What a gre	pos
16	That is a b	neg
17	We will ha	pos
18	I went to	neg

OUTPUT

The dimensions of the dataset (18, 2)

0 I love this sandwich

1 This is an amazing place

2 I feel very good about these beers

3 This is my best work

4 What an awesome view

5 I do not like this restaurant

6 I am tired of this stuff

7 I can't deal with this

8 He is my sworn enemy

9 My boss is horrible

10 This is an awesome place

11 I do not like the taste of this juice

12 I love to dance

13 I am sick and tired of this place

14 What a great holiday

15 That is a bad locality to stay

16 We will have good fun tomorrow

17 I went to my enemy's house today

Name: message, dtype: object

0 1

1 1

2 1

3 1

4 1

5 0

6 0

7 0

8 0

9 0

10 1

11 0

12 1

13 0

14 1

15 0

16 1

17 0

Name: labelnum, dtype: int64

(5,)

(13,)

(5,)

(13,)

Accuracy metrics

Accuracy of the classifer is 0.8

Confusion matrix

[[3 1]

[0 1]]

Recall and Precison

1.0

0.5

RESULT

The given python program is executed successfully

Ex No:8 Date :	Q Learning with Linear Function Approximation.

AIM

Implement Q Learning with Linear Function Approximation.

ALGORITHM

Step 1: Initialize Q-table with random values

Step 2: Set learning rate, discount factor, exploration rate, and number of episodes

Step 3: For each episode:

a. Initialize the state

b. While the game is not over:

i. Choose an action based on the state and exploration rate

ii. Take the chosen action and observe the next state and reward

iii. Update Q-table using the linear function approximation method

iv. Set the current state to the next state c. Reduce exploration rate

Step 4: Return the updated Q-table

PROGRAM

```
import numpy as np
import random
import matplotlib.pyplot as plt



#set the rows and columns length
BOARD_ROWS = 5
BOARD_COLS = 5

#initalise start, win and lose states
START = (0, 0)
WIN_STATE = (4, 4)
HOLE_STATE = [(1,0),(3,1),(4,2),(1,3)]
```

```
#class state defines the board and decides reward, end and next position
class State:
    def _init_(self, state=START):
        #initalise the state to start and end to false
        self.state = state
        self.isEnd = False

    def getReward(self):
        #give the rewards for each state -5 for loss, +1 for win, -1 for others
        for i in HOLE_STATE:
            if self.state == i:
                return -5
        if self.state == WIN_STATE:
            return 1

        else:
            return -1

    def isEndFunc(self):
        #set state to end if win/loss
        if (self.state == WIN_STATE):
            self.isEnd = True

        for i in HOLE_STATE:
            if self.state == i:
                self.isEnd = True

    def nxtPosition(self, action):
        #set the positions from current action - up, down, left, right
        if action == 0:
            nxtState = (self.state[0] - 1, self.state[1]) #up
        elif action == 1:
            nxtState = (self.state[0] + 1, self.state[1]) #down
        elif action == 2:
            nxtState = (self.state[0], self.state[1] - 1) #left
```

```
        else:
            nxtState = (self.state[0], self.state[1] + 1) #right


        #check if next state is possible
        if (nxtState[0] >= 0) and (nxtState[0] <= 4):
            if (nxtState[1] >= 0) and (nxtState[1] <= 4):
                    #if possible change to next state
                    return nxtState
        #Return current state if outside grid
        return self.state


#class agent to implement reinforcement learning through grid
class Agent:

    def _init_(self):
        #inialise states and actions
        self.states = []
        self.actions = [0,1,2,3]    # up, down, left, right
        self.State = State()
        #set the learning and greedy values
        self.alpha = 0.5
        self.gamma = 0.9
        self.epsilon = 0.1
        self.isEnd = self.State.isEnd

        # array to retain reward values for plot
        self.plot_reward = []

        #initalise Q values as a dictionary for current and new
        self.Q = {}
        self.new_Q = {}
        #initalise rewards to 0
        self.rewards = 0
```

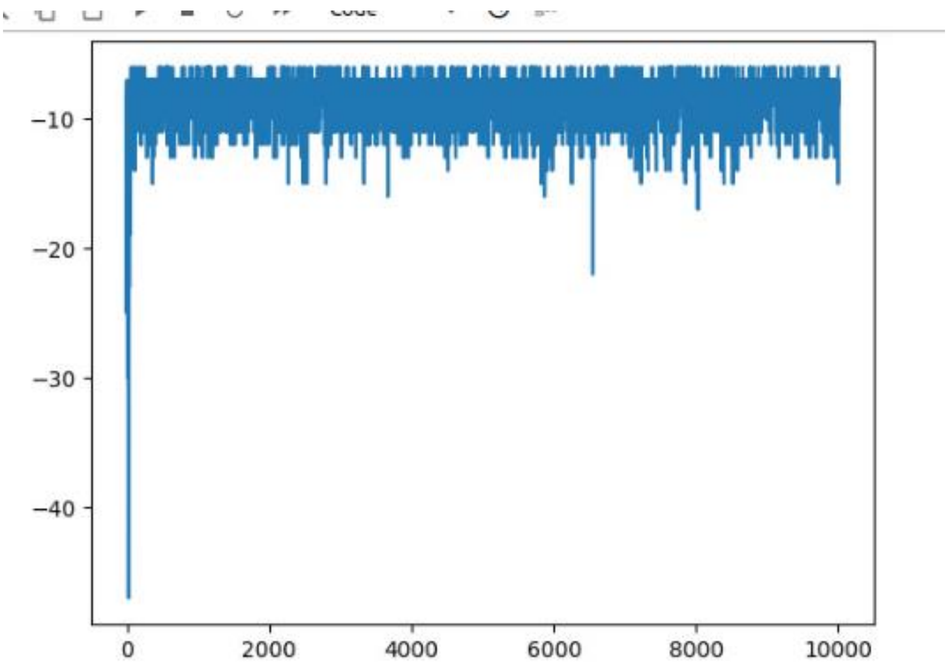
```
    #initalise all Q values across the board to 0, print these values
    for i in range(BOARD_ROWS):
        for j in range(BOARD_COLS):
            for k in range(len(self.actions)):
                self.Q[(i, j, k)] =0
                self.new_Q[(i, j, k)] = 0

    print(self.Q)



#method to choose action with Epsilon greedy policy, and move to next state
def Action(self):
    #random value vs epsilon
    rnd = random.random()
    #set arbitraty low value to compare with Q values to find max
    mx_nxt_reward =-10
    action = None

    #9/10 find max Q value over actions
    if(rnd >self.epsilon) :
        #iterate through actions, find Q  value and choose best
        for k in self.actions:

            i,j = self.State.state

            nxt_reward = self.Q[(i,j, k)]

            if nxt_reward >= mx_nxt_reward:
                action = k
                mx_nxt_reward = nxt_reward

    #else choose random action
    else:
        action = np.random.choice(self.actions)

    #select the next state based on action chosen
```

```
    position = self.State.nxtPosition(action)
    return position,action


#Q-learning Algorithm
def Q_Learning(self,episodes):
    x = 0
    #iterate through best path for each episode
    while(x < episodes):
        #check if state is end
        if self.isEnd:
            #get current rewrard and add to array for plot
            reward = self.State.getReward()
            self.rewards += reward
            self.plot_reward.append(self.rewards)

            #get state, assign reward to each Q_value in state
            i,j = self.State.state
            for a in self.actions:
                self.new_Q[(i,j,a)] = round(reward,3)

            #reset state
            self.State = State()
            self.isEnd = self.State.isEnd

            #set rewards to zero and iterate to next episode
            self.rewards = 0
            x+=1
        else:
            #set to arbitrary low value to compare net state actions
            mx_nxt_value = -10
            #get current state, next state, action and current reward
            next_state, action = self.Action()
            i,j = self.State.state
            reward = self.State.getReward()
            #add reward to rewards for plot
```

```
            self.rewards +=reward

            #iterate through actions to find max Q value for action based on next state action
            for a in self.actions:
                nxtStateAction = (next_state[0], next_state[1], a)
                q_value = (1-self.alpha)self.Q[(i,j,action)] + self.alpha(reward + self.gamma*self.Q[nxtStateAction])

                #find largest Q value
                if q_value >= mx_nxt_value:
                    mx_nxt_value = q_value

            #next state is now current state, check if end state
            self.State = State(state=next_state)
            self.State.isEndFunc()
            self.isEnd = self.State.isEnd

            #update Q values with max Q value for next state
            self.new_Q[(i,j,action)] = round(mx_nxt_value,3)

        #copy new Q values to Q table
        self.Q = self.new_Q.copy()
    #print final Q table output
    print(self.Q)

#plot the reward vs episodes
def plot(self,episodes):

    plt.plot(self.plot_reward)
    plt.show()



#iterate through the board and find largest Q value in each, print output
def showValues(self):
    for i in range(0, BOARD_ROWS):
        print('----------------------------------------------')
```

```
        out = '| '
        for j in range(0, BOARD_COLS):
            mx_nxt_value = -10
            for a in self.actions:
                nxt_value = self.Q[(i,j,a)]
                if nxt_value >= mx_nxt_value:
                    mx_nxt_value = nxt_value
            out += str(mx_nxt_value).ljust(6) + ' | '
        print(out)
    print('----------------------------------------------')



if _name_ == "_main_":
    #create agent for 10,000 episdoes implementing a Q-learning algorithm plot and show values.
    ag = Agent()
    episodes = 10000
    ag.Q_Learning(episodes)
    ag.plot(episodes)
    ag.showValues()
```

OUTPUT



-5.262	-4.736	-4.152	-3.503	-2.782
-5	-4.152	-3.503	-5	-1.98
-3.529	-3.503	-2.782	-1.98	-1.089
-3.284	-5	-1.98	-1.089	-0.099
-3.227	-2.649	-5	-0.1	1

RESULT

The given python program is executed successfully

Ex No:9 Date :	Policy Gradient

AIM

Implement the Policy Gradient concept in Reinforcement learning. Compare the Reinforce with Baseline and Actor Critic with Baseline

ALGORITHM

Step 1: Initialize the policy parameters randomly
Step 2: Initialize the environment
Step 3: Set the discount factor gamma and the learning rate alpha
Step 4: Repeat until convergence:
  Generate a set of trajectories using the current policy
  Compute the total reward for each trajectory
  Compute the gradients of the policy with respect to the total reward for each trajectory
  Update the policy parameters using the gradients and the learning rate

PROGRAM

```
import gym
import torch
import matplotlib.pyplot as plt
env = gym.make('CartPole-v0')
n_state = env.observation_space.shape[0]
n_action = env.action_space.n

def run_episode(env, weight):
    state = env.reset()
    grads = []
    total_reward = 0
```

```
    is_done = False
    while not is_done:
        state = torch.from_numpy(state).float()
        z = torch.matmul(state, weight)
        probs = torch.nn.Softmax()(z)
        action = int(torch.bernoulli(probs[1]).item())
        d_softmax = torch.diag(probs) -probs.view(-1, 1) * probs
        d_log = d_softmax[action] / probs[action]
        grad = state.view(-1, 1) * d_log
        grads.append(grad)
        state, reward, is_done, _ = env.step(action)
        total_reward += reward
        if is_done:
            break
    return total_reward, grads


n_episode = 10
weight = torch.rand(n_state, n_action)
total_rewards = []
learning_rate = 0.001
for episode in range(n_episode):
    total_reward, gradients = run_episode(env, weight)
    print('Episode {}: {}'.format(episode + 1, total_reward))
    for i, gradient in enumerate(gradients):
        weight += learning_rate * gradient * (total_reward - i)
        total_rewards.append(total_reward)


plt.plot(total_rewards)
```

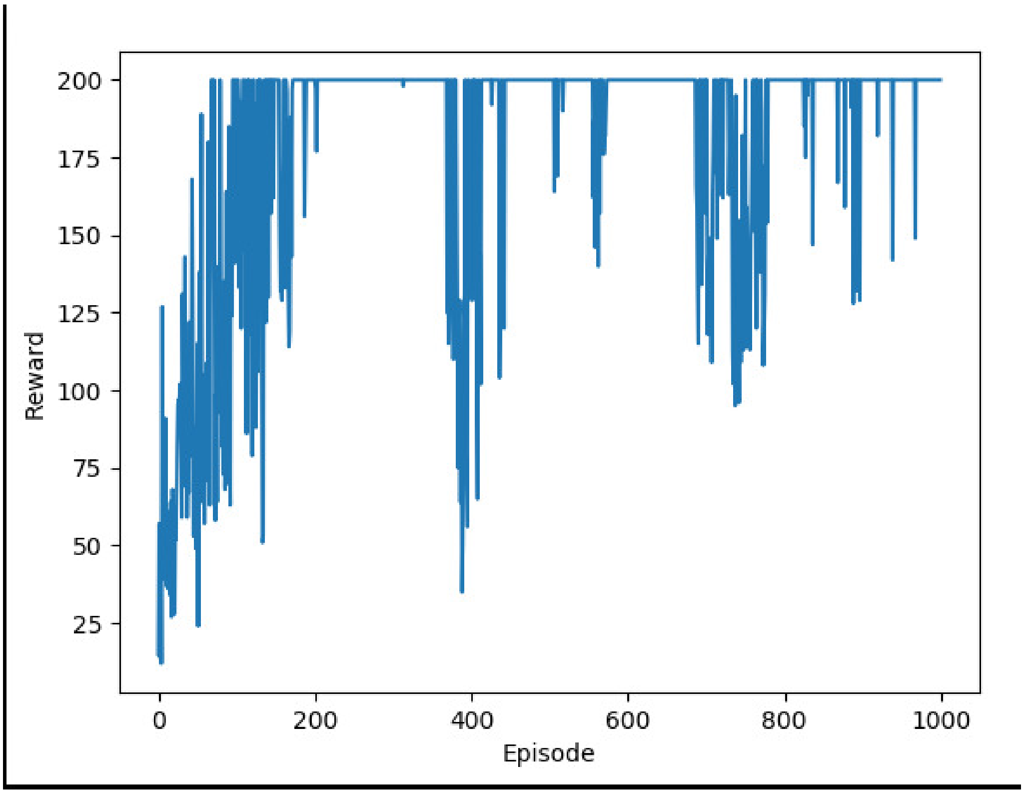
plt.xlabel('Episode')

plt.ylabel('Reward')

plt.show()

OUTPUT

```
Episode 101: 200.0
Episode 102: 200.0
Episode 103: 200.0
Episode 104: 190.0
Episode 105: 133.0
......
......
Episode 996: 200.0
Episode 997: 200.0
Episode 998: 200.0
Episode 999: 200.0
Episode 1000: 200.0
```



RESULT

The given python program is executed successfully

Ex No:10 Date:	Time Series data

AIM

Consider a time series data set. Plot the data, identify the components of the Time Series data, calculate the seasonality and stationarity and Identify the trend pattern present in the time series data. Remove the white noise if available in the time series data.

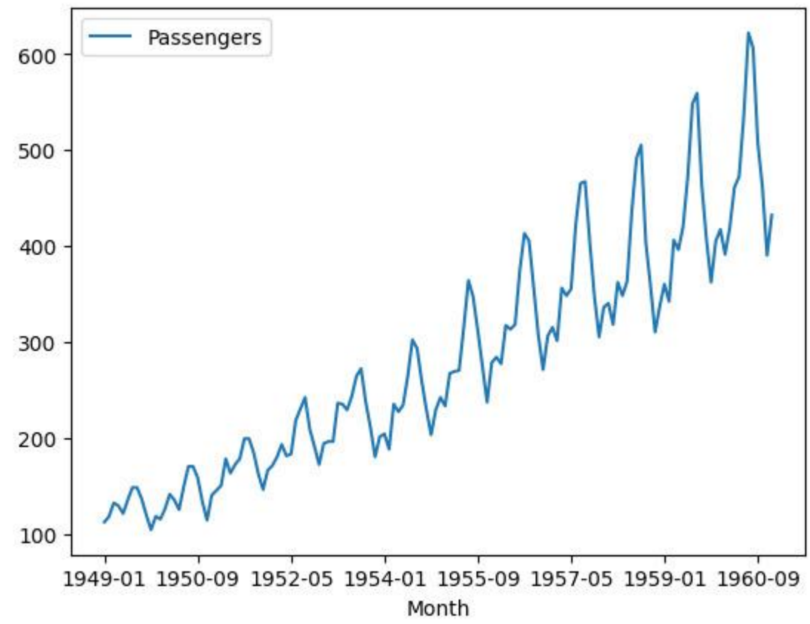
ALGORITHM

Step 1: Import the required libraries: pandas, numpy, matplotlib, and statsmodels.
Step 2: Load the dataset using the pandas library and check its structure.
Step 3: Plot the time series data to visualize its pattern.
Step 4: Check the trend of the time series using the decompose function of statsmodels.
Step 5: Identify the seasonality of the time series using the decompose function of statsmodels.
Step 6: Check the stationarity of the time series using the ADF test from the statsmodels library.
Step 7: If the time series is not stationary, perform the differencing operation to make it stationary.
Step 8: Remove white noise using any suitable method such as moving average or exponential smoothing.
Step 9: Plot the final time series data after removing the noise.

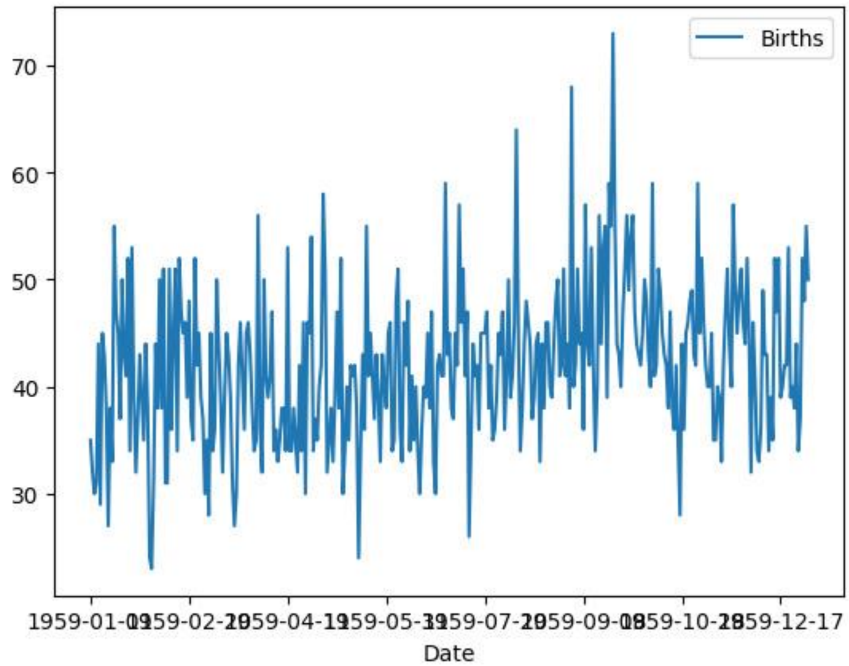
PROGRAM

```
from pandas import read_csv
from matplotlib import pyplot
series = read_csv('passenger.csv', header=0, index_col=0)
series.plot()
pyplot.show()
```

OUTPUT



Passengers
600
500
400
300
200
100
1949-01
1950-09
1952-05
1954-01
1955-09
1957-05
1959-01
1960-09
Month



Births
70
60
50
40
30
Date

DATASET

	A	B	C
1	Date	Births	
2	#########	35	
3	#########	32	
4	#########	30	
5	#########	31	
6	#########	44	
7	#########	29	
8	#########	45	
9	#########	43	
10	#########	38	
11	#########	27	
12	#########	38	
13	#########	33	
14	#########	55	
15	#########	47	
16	#########	45	
17	#########	37	
18	#########	50	
19	#########	43	

	A	B	C
1	Month	Passengers	
2	1949-01	112	
3	1949-02	118	
4	1949-03	132	
5	1949-04	129	
6	1949-05	121	
7	1949-06	135	
8	1949-07	148	
9	1949-08	148	
10	1949-09	136	
11	1949-10	119	
12	1949-11	104	
13	1949-12	118	
14	1950-01	115	
15	1950-02	126	
16	1950-03	141	
17	1950-04	135	
18	1950-05	125	
19	1950-06	149	

RESULT

The given python program is executed successfully