Assignment 1

Tittle: Find the correlation matrix.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

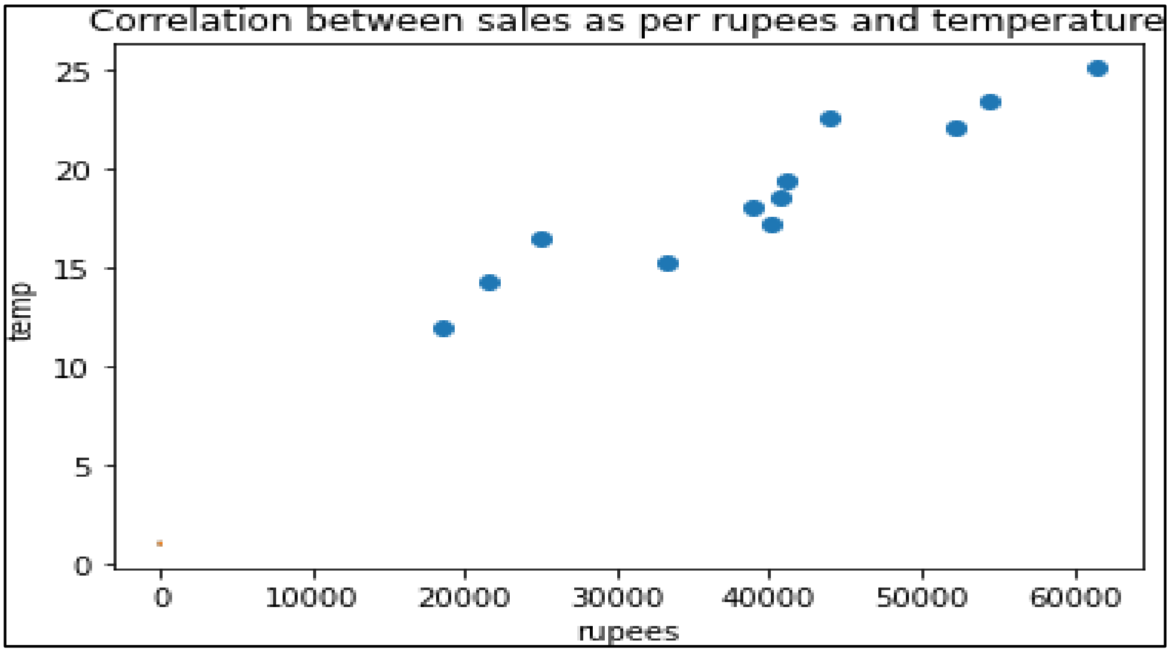
Remark:

Program:

```
#Correlation Matrix
import numpy as np

import matplotlib.pyplot as plt


x = [21545, 25000, 18500, 33255, 40633, 52200, 41200,

  61400, 54400, 39000, 44000, 40200],


y = [14.2, 16.4, 11.9, 15.2, 18.5, 22.1,

  19.4, 25.1, 23.4, 18.1, 22.6, 17.2]


matrix = np.corrcoef(x,y)

print(matrix)


plt.scatter(x,y)
```

Output:

```
[[1.         0.94821432]
 [0.94821432 1.        ]]
```



Assignment 2

Tittle: Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

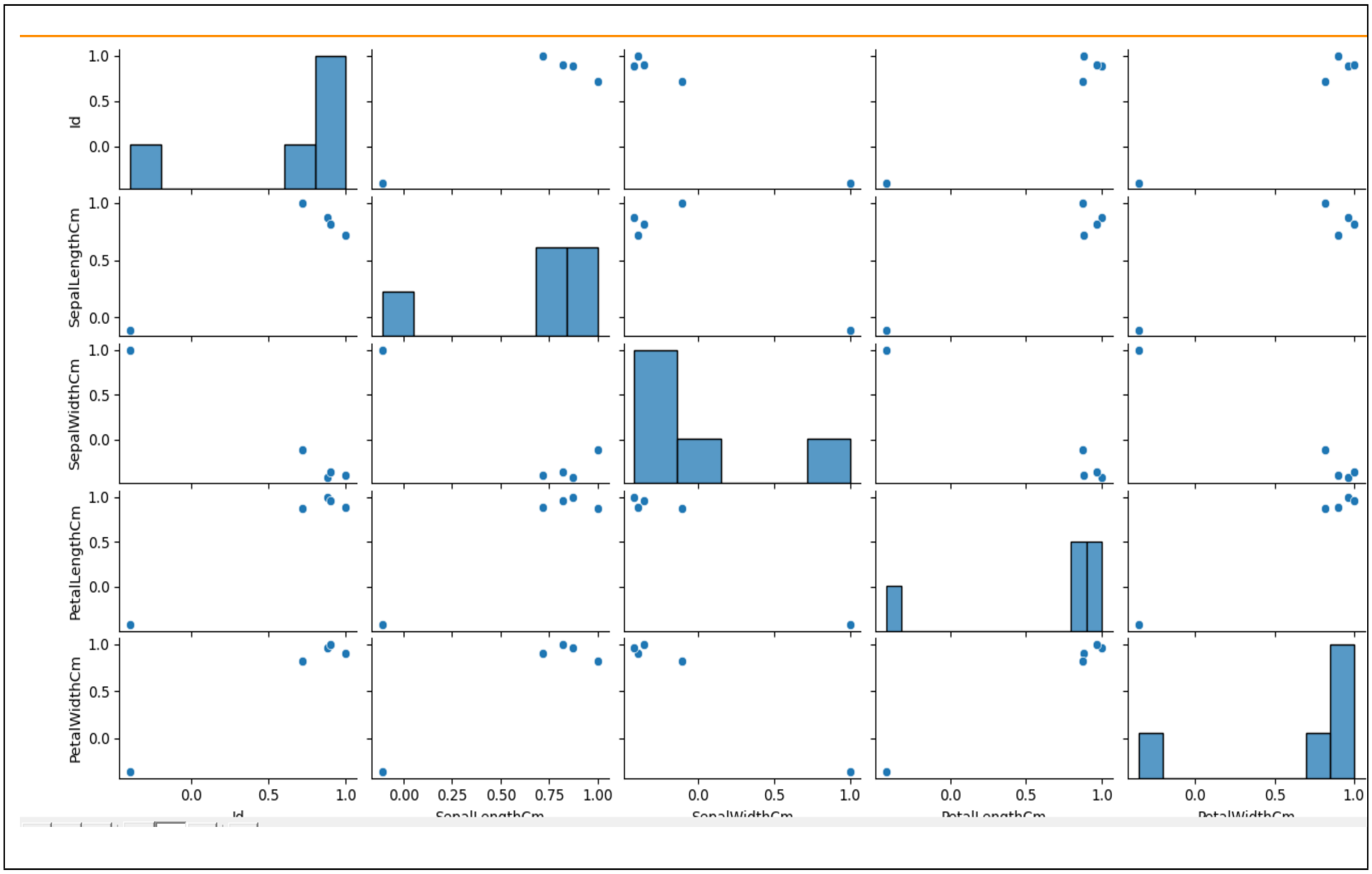
Remark:

Program:

```
#Iris Dataset

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df=pd.read_csv("Iris.csv")
rel = df.corr()
print(rel)
sns.pairplot(rel)
plt.show()
```

Output:

```
PS C:\Python\Scripts> & C:/Python/python.exe "c:/Python/Scripts/correlation in iris dataset.py"
                     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Id             1.000000       0.716676     -0.397729       0.882747      0.899759
SepalLengthCm  0.716676       1.000000     -0.109369       0.871754      0.817954
SepalWidthCm  -0.397729      -0.109369      1.000000      -0.420516     -0.356544
PetalLengthCm  0.882747       0.871754     -0.420516       1.000000      0.962757
PetalWidthCm   0.899759       0.817954     -0.356544       0.962757      1.000000
```


Id
SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00

Assignment 3

Title: Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

Program:

```
#Anova

import pandas as pd
from seaborn import load_dataset
import statsmodels.formula.api as sm
import statsmodels.stats.multicomp as multi

iris = load_dataset("iris")

my_subset = iris[iris["species"].isin(['setosa', 'virginica'])]
subset_model = sm.ols(formula='sepal_length ~ C(species)', data=my_subset)
print(subset_model.fit().summary())

my_subset.groupby("species").mean()
print(my_subset.groupby("species").std())

multi_comp = multi.MultiComparison(iris['sepal_length'], iris['species'])
print(multi_comp.tukeyhsd().summary())
```

Output:

```
PS C:\Python\Scripts> & C:/Python/python.exe c:/Python/Scripts/ANOVA.py
                            OLS Regression Results
==============================================================================
Dep. Variable:           sepal_length   R-squared:                       0.707
Model:                            OLS   Adj. R-squared:                  0.704
Method:                 Least Squares   F-statistic:                     236.7
Date:                Thu, 03 Mar 2022   Prob (F-statistic):           6.89e-28
Time:                        17:59:13   Log-Likelihood:                -74.349
No. Observations:                 100   AIC:                             152.7
Df Residuals:                      98   BIC:                             157.9
Df Model:                           1
Covariance Type:            nonrobust
============================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Intercept                   5.0060      0.073     68.854      0.000       4.862       5.150
C(species)[T.virginica]     1.5820      0.103     15.386      0.000       1.378       1.786
==============================================================================
Omnibus:                        2.651   Durbin-Watson:                   2.191
Prob(Omnibus):                  0.266   Jarque-Bera (JB):                2.318
Skew:                           0.127   Prob(JB):                        0.314
Kurtosis:                       3.701   Cond. No.                         2.62
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
           sepal_length  sepal_width  petal_length  petal_width
species
setosa          0.35249     0.379064      0.173664     0.105386
virginica       0.63588     0.322497      0.551895     0.274650
   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=========================================================
  group1     group2   meandiff p-adj lower  upper  reject
---------------------------------------------------------
    setosa versicolor     0.93   0.0 0.6862 1.1738   True
    setosa  virginica    1.582   0.0 1.3382 1.8258   True
versicolor  virginica    0.652   0.0 0.4082 0.8958   True
---------------------------------------------------------
```

Assignment 4

Title: Apply linear regression Model techniques to predict the data on any dataset.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

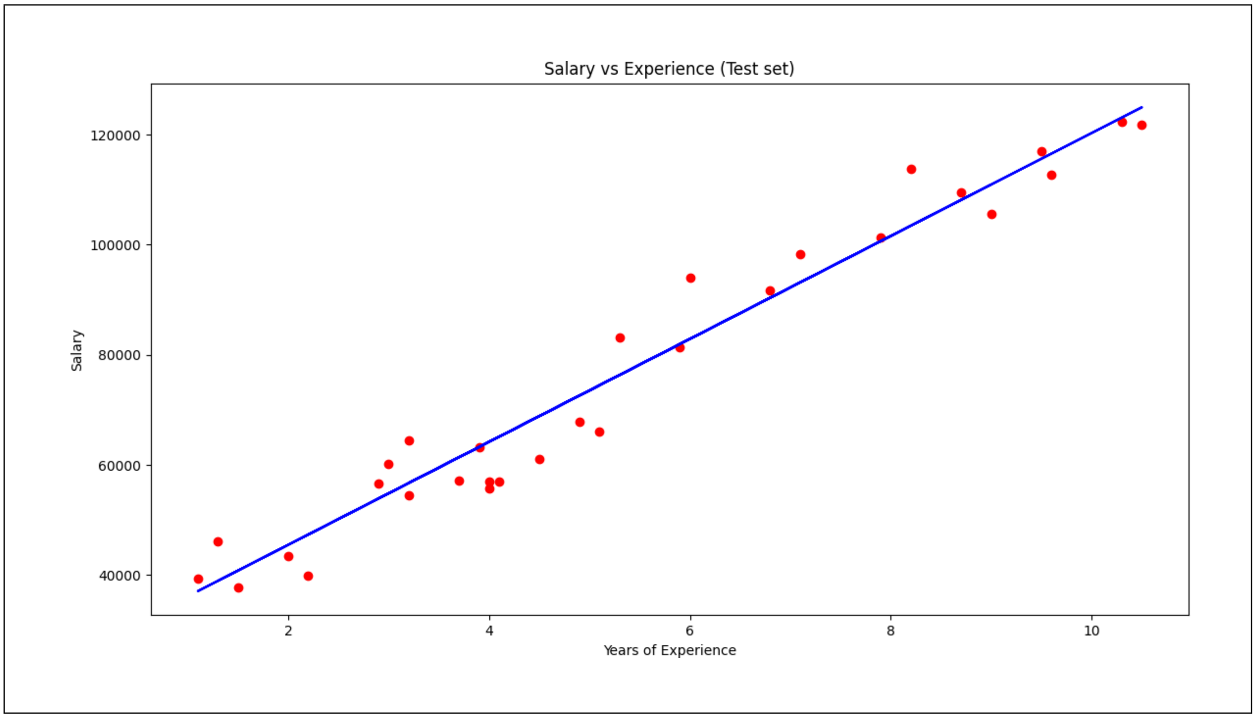
Date:

Remark:

Program:

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

dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

from sklearn.linear_model import LinearRegression regressor =
LinearRegression() regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

plt.scatter(X_train,y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')

plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Output



Salary vs Experience (Test set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Years of Experience

Assignment 5

Title: Apply logical regression Model techniques to predict the data on any dataset.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

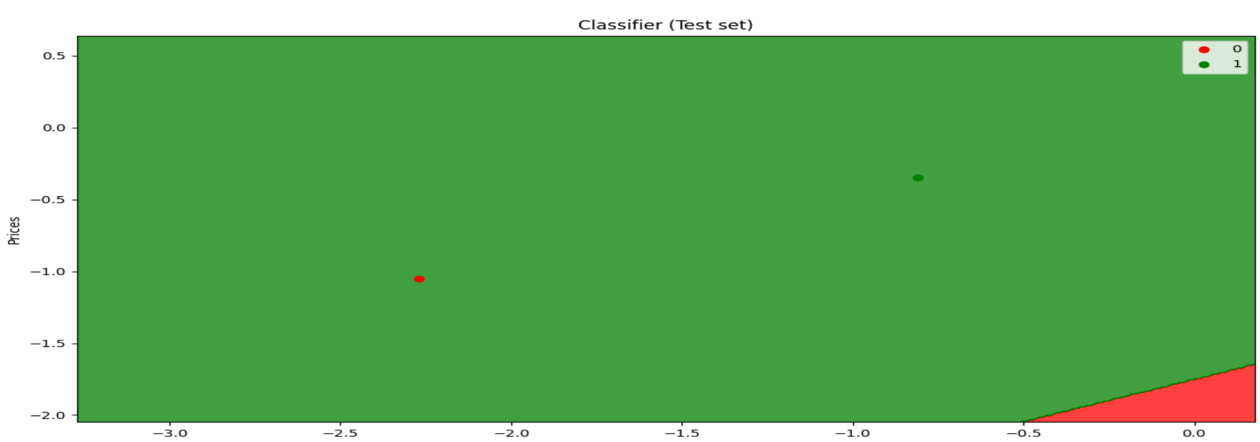
Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
 from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap

dataset = pd.read_csv('Room dataset 2.csv') # input
x = dataset.iloc[:, [0, 3]].values #output
y = dataset.iloc[:, 4].values
xtrain, xtest, ytrain, ytest = train_test_split(
    x,y,test_size = 0.25, random_state = 0)
sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain) xtest =
sc_x.transform(xtest)
print (xtrain[0:10, :])
classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)
y_pred = classifier.predict(xtest)
cm = confusion_matrix(ytest, y_pred)
```

```
print ("Confusion Matrix : \n", cm)
print ("Accuracy : ", accuracy_score(ytest, y_pred))
X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() -
            1, stop = X_set[:, 0].max() + 1, step = 0.01),
         np.arange(start = X_set[:, 1].min() - 1,
            stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(
     np.array([X1.ravel(), X2.ravel()]).T).reshape(
     X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
       c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Classifier (Test set)')
plt.xlabel('Area')
plt.ylabel('Prices')
plt.legend() plt.show()
```

Output:

```
PS C:\Python\Scripts> & C:/Python/python.exe "c:/Python/Scripts/logical regression.py"
[[-1.29777137 -0.87481777]
 [ 0.16222142 -0.52489066]
 [ 1.13554995  1.39970842]]
Confusion Matrix :
 [[0 1]
 [0 1]]
Accuracy :  0.5
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



Assignment 6

Title: Clustering algorithms for unsupervised classification**.**

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

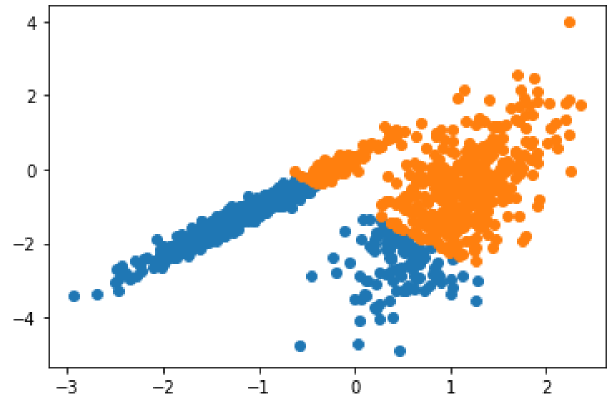
Date:

Remark:

Program:

```
# k-means clustering

from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from matplotlib import pyplot #
define dataset
X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
n_clusters_pe r_class=1, random_state=4)
# define the model
model = KMeans(n_clusters=2) #
fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster for
cluster in clusters:
  # get row indexes for samples with this cluster
  row_ix = where(yhat == cluster)
  # create scatter of these samples
pyplot.scatter(X[row_ix, 0], X[row_ix, 1]) #
show the plot
pyplot.show()
```

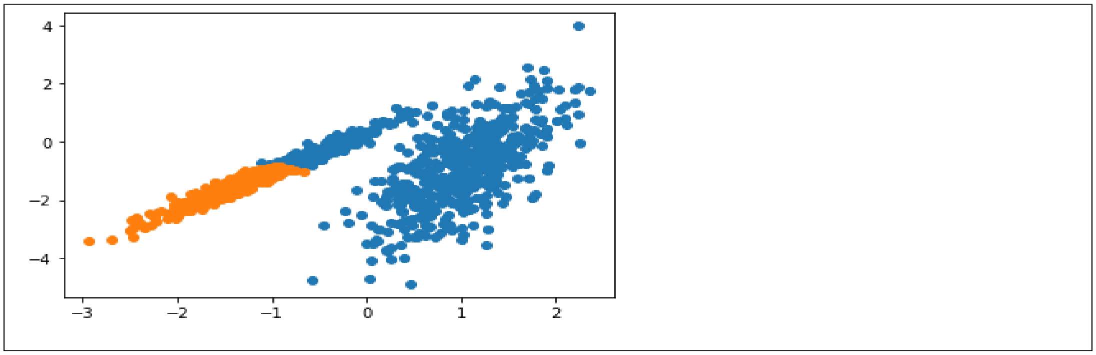
Output:



Program 2:

```
# agglomerative clustering 6.2

from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import AgglomerativeClustering
from matplotlib import pyplot
# define dataset
X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
n_clusters_pe r_class=1, random_state=4)
# define the model
model = AgglomerativeClustering(n_clusters=2)
# fit model and predict clusters
yhat = model.fit_predict(X) #
retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
  # get row indexes for samples with this cluster
  row_ix = where(yhat == cluster)
  # create scatter of these samples
pyplot.scatter(X[row_ix, 0], X[row_ix, 1]) #
show the plot
pyplot.show()
```

Output 2:



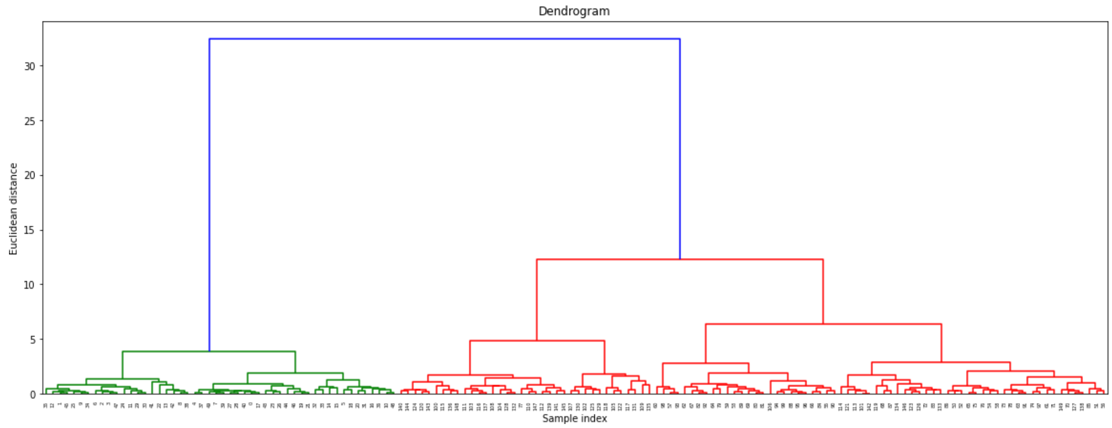
Program 3:

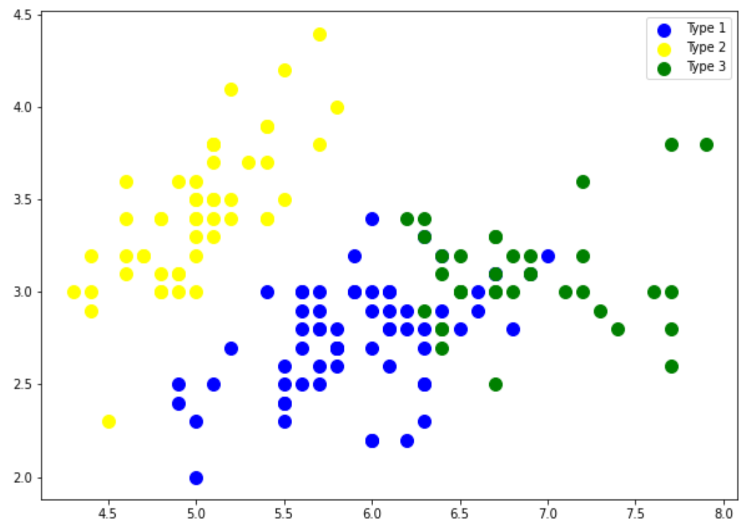
```
#Assignment 6.3

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt from
sklearn import datasets  import
scipy.cluster.hierarchy as sc import
matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

# Import iris data
iris = datasets.load_iris()

iris_data = pd.DataFrame(iris.data)
iris_data.columns = iris.feature_names
iris_data['flower_type'] = iris.target
iris_data.head()

iris_X = iris_data.iloc[:, [0, 1, 2,3]].values
iris_Y = iris_data.iloc[:,4].values

"""plt.figure(figsize=(10,  7))
plt.scatter(iris_X[iris_Y == 0, 0], iris_X[iris_Y == 0, 1], s=100, c='blue', label='Type 1')
plt.scatter(iris_X[iris_Y == 1, 0], iris_X[iris_Y == 1, 1], s=100, c='yellow', label='Type 2')
plt.scatter(iris_X[iris_Y == 2, 0], iris_X[iris_Y == 2, 1], s=100, c='green', label='Type 3')
plt.legend()
plt.show()"""
```

```
# Plot dendrogram
plt.figure(figsize=(20, 7))
plt.title("Dendrograms")

# Create dendrogram
sc.dendrogram(sc.linkage(iris_X, method='ward'))

plt.title('Dendrogram') plt.xlabel('Sample
index') plt.ylabel('Euclidean distance')

cluster = AgglomerativeClustering(
   n_clusters=3, affinity='euclidean', linkage='ward')

cluster.fit(iris_X) labels =
cluster.labels_ print(labels)

plt.figure(figsize=(10, 7))
plt.scatter(iris_X[labels == 0, 0], iris_X[labels == 0, 1], s = 100, c = 'blue', label = 'Type 1')
plt.scatter(iris_X[labels == 1, 0], iris_X[labels == 1, 1], s = 100, c = 'yellow', label = 'Type 2')
plt.scatter(iris_X[labels == 2, 0], iris_X[labels == 2, 1], s = 100, c = 'green', label = 'Type 3')
plt.legend()
plt.show()
```

Output:

```
PS C:\Python\Scripts> & C:/Python/python.exe "c:/Python/Scripts/Clustering Divisive.py"
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```


Dendrogram
Euclidean distance
Sample index
0
5
10
15
20
25
30


Type 1
Type 2
Type 3
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

Assignment 7

Title: Association algorithms for supervised classification on any dataset.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

Program:

```
import numpy as np import

pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules


# Changing the working location to the location of the file cd

C:\Users\Dev\Desktop\Kaggle\Apriori Algorithm


# Loading the Data

data = pd.read_excel('Online_Retail.xlsx')

data.head()

# Exploring the columns of the data data.columns

# Exploring the different regions of transactions

data.Country.unique()

# Stripping extra spaces in the description data['Description']

= data['Description'].str.strip()

# Dropping the rows without any invoice number
```

```
data.dropna(axis = 0, subset =['InvoiceNo'], inplace = True)
data['InvoiceNo'] = data['InvoiceNo'].astype('str')

# Dropping all transactions which were done on credit
data = data[~data['InvoiceNo'].str.contains('C')]
# Transactions done in France
basket_France = (data[data['Country'] =="France"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))

# Transactions done in the United Kingdom
basket_UK = (data[data['Country'] =="United Kingdom"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))

# Transactions done in Portugal
basket_Por = (data[data['Country'] =="Portugal"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))

basket_Sweden = (data[data['Country'] =="Sweden"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
```

```
                    sum().unstack().reset_index().fillna(0)
                    .set_index('InvoiceNo'))
# Defining the hot encoding function to make
the data suitable # for the concerned libraries
def hot_encode(x):
          if(x<= 0):
                    return 0
          if(x>= 1):
                    return 1


# Encoding the datasets
basket_encoded =
basket_France.applymap(hot_encode)
basket_France = basket_encoded
basket_encoded =
basket_UK.applymap(hot_encode)
basket_UK = basket_encoded
basket_encoded =
basket_Por.applymap(hot_encode)
basket_Por = basket_encoded
basket_encoded =
basket_Sweden.applymap(hot_encode)
basket_Sweden = basket_encoded
# Building the model
frq_items = apriori(basket_France, min_support = 0.05, use_colnames = True)
```

```
# Collecting the inferred rules in a dataframe

rules = association_rules(frq_items, metric ="lift", min_threshold = 1) rules

= rules.sort_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

frq_items = apriori(basket_UK, min_support = 0.01, use_colnames = True)

rules = association_rules(frq_items, metric ="lift", min_threshold = 1) rules =

rules.sort_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

frq_items = apriori(basket_Por, min_support = 0.05, use_colnames = True)

rules = association_rules(frq_items, metric ="lift", min_threshold = 1) rules =

rules.sort_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

frq_items = apriori(basket_Sweden, min_support = 0.05, use_colnames = True) rules =

association_rules(frq_items, metric ="lift", min_threshold = 1)

rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())
```

OUTPUT:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Japan', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Israel', 'Finland', 'Bahrain', 'Greece', 'Hong Kong', 'Singapore',
       'Lebanon', 'United Arab Emirates', 'Saudi Arabia',
       'Czech Republic', 'Canada', 'Unspecified', 'Brazil', 'USA',
       'European Community', 'Malta', 'RSA'], dtype=object)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
44	(JUMBO BAG WOODLAND ANIMALS)	(POSTAGE)	0.076531	0.765306	0.076531	1.000	1.306667	0.017961	inf
258	(PLASTERS IN TIN CIRCUS PARADE, RED TOADSTOOL ...	(POSTAGE)	0.051020	0.765306	0.051020	1.000	1.306667	0.011974	inf
270	(PLASTERS IN TIN WOODLAND ANIMALS, RED TOADSTO...	(POSTAGE)	0.053571	0.765306	0.053571	1.000	1.306667	0.012573	inf
301	(SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...	(SET/6 RED SPOTTY PAPER PLATES)	0.102041	0.127551	0.099490	0.975	7.644000	0.086474	34.897959
302	(SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...	(SET/6 RED SPOTTY PAPER CUPS)	0.102041	0.137755	0.099490	0.975	7.077778	0.085433	34.489796

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
116	(BEADED CRYSTAL HEART PINK ON STICK)	(DOTCOM POSTAGE)	0.011036	0.037928	0.010768	0.975728	25.725872	0.010349	39.637371
2019	(SUKI SHOULDER BAG, JAM MAKING SET PRINTED)	(DOTCOM POSTAGE)	0.011625	0.037928	0.011196	0.963134	25.393807	0.010755	26.096206
2296	(HERB MARKER THYME, HERB MARKER MINT)	(HERB MARKER ROSEMARY)	0.010714	0.012375	0.010232	0.955000	77.173095	0.010099	21.947227
2302	(HERB MARKER PARSLEY, HERB MARKER ROSEMARY)	(HERB MARKER THYME)	0.011089	0.012321	0.010553	0.951691	77.240055	0.010417	20.444951
2300	(HERB MARKER THYME, HERB MARKER PARSLEY)	(HERB MARKER ROSEMARY)	0.011089	0.012375	0.010553	0.951691	76.905682	0.010416	20.443842

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1170	(SET 12 COLOUR PENCILS DOLLY GIRL)	(SET 12 COLOUR PENCILS SPACEBOY)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf
1171	(SET 12 COLOUR PENCILS SPACEBOY)	(SET 12 COLOUR PENCILS DOLLY GIRL)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf
1172	(SET 12 COLOUR PENCILS DOLLY GIRL)	(SET OF 4 KNICK KNACK TINS LONDON)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf
1173	(SET OF 4 KNICK KNACK TINS LONDON)	(SET 12 COLOUR PENCILS DOLLY GIRL)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf
1174	(SET 12 COLOUR PENCILS DOLLY GIRL)	(SET OF 4 KNICK KNACK TINS POPPIES)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(12 PENCILS SMALL TUBE SKULL)	(PACK OF 72 SKULL CAKE CASES)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
1	(PACK OF 72 SKULL CAKE CASES)	(12 PENCILS SMALL TUBE SKULL)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
4	(36 DOILIES DOLLY GIRL)	(ASSORTED BOTTLE TOP MAGNETS)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
5	(ASSORTED BOTTLE TOP MAGNETS)	(36 DOILIES DOLLY GIRL)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
180	(CHILDRENS CUTLERY DOLLY GIRL)	(CHILDRENS CUTLERY CIRCUS PARADE)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf

Assignment 8

Title: Developing and implementing Decision Tree model on the dataset.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

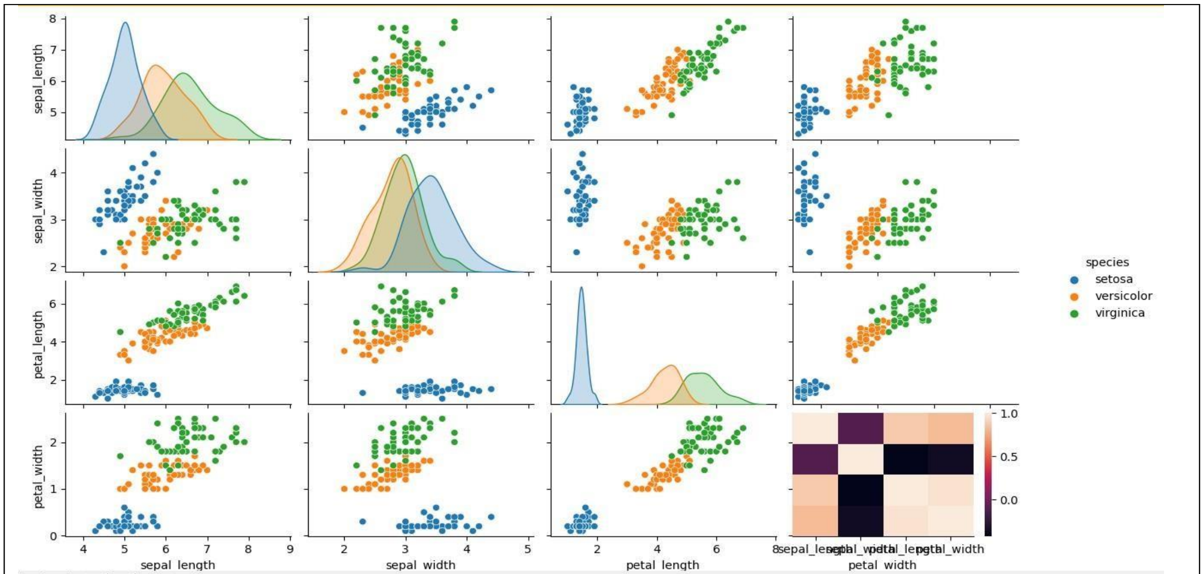
Program:

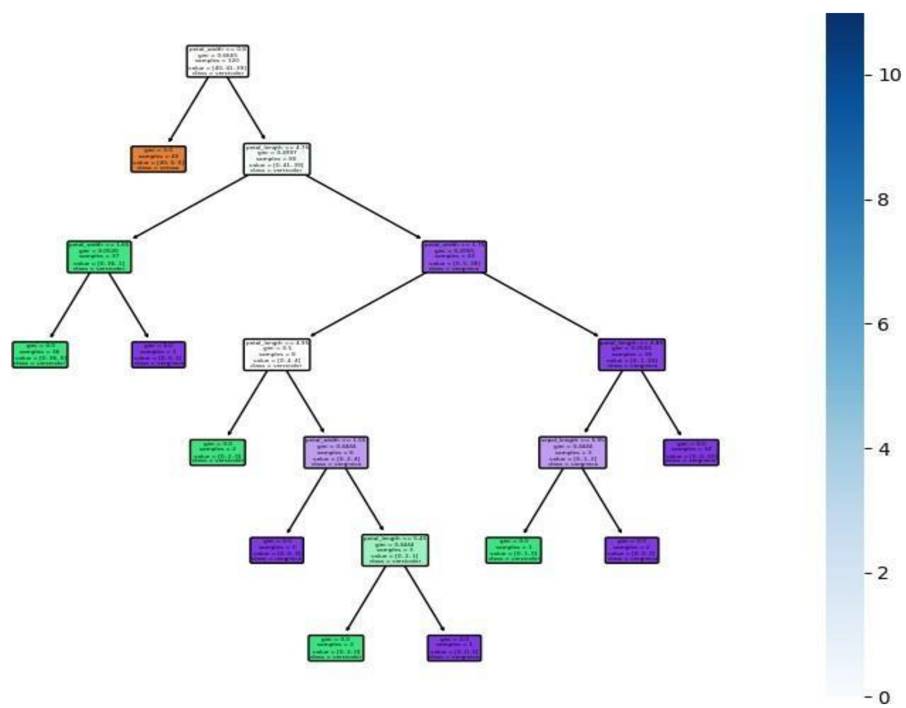
```
import pandas as pd import
numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder#for train test splitting
from sklearn.model_selection import train_test_split#for decision tree
object from sklearn.tree import DecisionTreeClassifier#for checking
testing results
from sklearn.metrics import classification_report, confusion_matrix#for visualizing tree
from sklearn.tree import plot_tree
df = sns.load_dataset('iris')
df.head()
df.info() df.shape
df.isnull().any()
sns.pairplot(data=df, hue = 'species')
sns.heatmap(df.corr())
target = df['species'] df1
= df.copy()
df1 = df1.drop('species', axis =1)
X = df1
print(target)
le = LabelEncoder()
target = le.fit_transform(target)
print(target)
y = target
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size = 0.2, random_state = 42)
print("Training split input- ", X_train.shape)
print("Testing split input- ", X_test.shape)
```

```python
dtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)
print('Decision Tree Classifier Created')
y_pred = dtree.predict(X_test)
print("Classification report - \n", classification_report(y_test,y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,square = True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted  label')
all_sample_title = 'Accuracy Score: {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
dec_tree = plot_tree(decision_tree=dtree, feature_names = df1.columns,
            class_names =["setosa", "vercicolor", "verginica"] , filled = True , precision = 4, rounded =
            True)

plt.show()
```

Output:

```
PS C:\Python\Scripts> & C:/Python/python.exe "c:/Python/Scripts/decision tree.py"
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
         ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Training split input-  (120, 4)
Testing split input-  (30, 4)
Decision Tree Classifier Created
Classification report -
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```


sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica



Assignment 9

Title: Bayesian classification on any dataset.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

Program:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

# Load the iris dataset
iris = load_iris()

# Store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# Split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)

# Create a Gaussian Naive Bayes model
gnb = GaussianNB()

# Fit the model on the training data
gnb.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = gnb.predict(X_test)

# Comparing actual response values (y_test) with predicted response values (y_pred)
print("Gaussian Naive Bayes model accuracy (in %):", metrics.accuracy_score(y_test, y_pred) * 100)
```

Output:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print("Gaussian Naive Bayes model accuracy (in %):", metrics.accuracy_score(y_test, y_pred) * 100)
```

```
Gaussian Naive Bayes model accuracy (in %): 95.0
```

Assignment 10

Title: SVM classification on any dataset.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

Program:

```
 #Mount the drive
from google.colab import drive
import accuracy_score, confusion_matrix, classification_report
import train_test_split
import pandas as pd
import numpy as np
drive.mount("/content/drive", force_remount=True)
df = pd.read_csv("/content/Iris.csv")

x = df.iloc[:, 1:5].values
y = df.iloc[:, 5].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=2, random_state=100)
df.head(5)
#Spliting the dataset for training & testing purpose
#Training the Model using fit() function
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
# Predict species (Setosa, Versicolor, or Virginica) for a new iris flower
y_pred=dt.predict(x_test)

sepal_length = input("Enter the sepal length: ")
sepal_width = input("Enter the sepal width: ")
petal_length = input("Enter the petal length: ")
petal_width = input("Enter the petal width: ")
y_pred1 = dt.predict([[sepal_length, sepal_width, petal_length, petal_width]])
print("The flower belongs to:", y_pred1)
# Evaluating the model
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

Output:

```
Accuracy Score :  0.9666666666666667

Classification Report :

        Iris-setosa        1.00        1.00      1.00      11
  Iris-versicolor          1.00        0.83      0.91       6
  Iris-virginica         0.93        1.00      0.96      13
```

```
accuracy                       0.97    30
macro avg       0.98    0.94   0.96    30
weighted avg    0.97    0.97   0.97    30

 Confusion Matrix
  [ 11   0   0]
  [ 0   5   1]
  [ 0 1   3 ]
```

Assignment 11

Title: Text mining algorithms on unstructured dataset.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

#Cleaning data

```
import pandas as pd

df = pd.read_excel("D: \ KR&AI \ Lab \ DataSet\Tweets.xlsx") # Check

the column names df.columns

# Removing neutral Reviews

review_df = review_df[review_df['airline_sentiment'] != 'neutral']

print(review_df.shape)

review_df.head(5)

# convert the categorical values to numeric using the factorize() method

sentiment_label = review_df.airline_sentiment.factorize()

# retrieve all the text data from the dataset.  tweet = review_df.text.values #

Tokenize all the words in the text

from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num_words=5000)  tokenizer.fit_on_texts(tweet)

encoded_docs = tokenizer.texts_to_sequences(tweet)

from tensorflow.keras.preprocessing.sequence import pad_sequences

padded_sequence = pad_sequences(encoded_docs, maxlen=200)

# Sentimental analysis using RNN
```

Program:

```
# Building the text classifier, using RNN LSTM model. from

tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM,Dense, Dropout, SpatialDropout1D from

tensorflow.keras.layers import Embedding

embedding_vector_length = 32

model = Sequential()

model.add(Embedding(vocab_size, embedding_vector_length, input_length=200))

model.add(SpatialDropout1D(0.25))

model.add(LSTM(50, dropout=0.5, recurrent_dropout=0.5))

model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

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

print(model.summary())

# Train the sentiment analysis model for 5 epochs on the whole dataset with a batch size of 32 and a
validation split of 20%.
history = model.fit(padded_sequence,sentiment_label[0],validation_split=0.2, epochs=5, batch_size=32)
```

Output:

```
Epoch 1/5
289/289 [==============================] - 471s 2s/step - loss: 0.4916 - accuracy: 0.7980 - val_loss: 0.2133 - val_accuracy: 0
9164
Epoch 2/5
289/289 [==============================] - 457s 2s/step - loss: 0.2282 - accuracy: 0.9118 - val_loss: 0.1624 - val_accuracy: 0
9428
Epoch 3/5
289/289 [==============================] - 423s 1s/step - loss: 0.1755 - accuracy: 0.9340 - val_loss: 0.1667 - val_accuracy: 0
9446
Epoch 4/5
289/289 [==============================] - 420s 1s/step - loss: 0.1292 - accuracy: 0.9519 - val_loss: 0.1678 - val_accuracy: 0
9402
Epoch 5/5
289/289 [==============================] - 420s 1s/step - loss: 0.1117 - accuracy: 0.9600 - val_loss: 0.1818 - val_accuracy: 0
9433
```

#Creating the RNN LSTM Learning model

```
# Sentimental analysis using RNN

# Testing the sentiment analysis model on new data

# Define a function that takes a text as input and outputs its prediction label. def

predict_sentiment(text):

    tw = tokenizer.texts_to_sequences([text]) tw =

    pad_sequences(tw,maxlen=200)

    prediction = int(model.predict(tw).round().item()) print("Predicted

    label: ", sentiment_label[1][prediction])

test_sentence1 = "I enjoyed my journey on this flight."

predict_sentiment(test_sentence1)

test_sentence2 = "This is the worst flight experience of my life!" predict_sentiment(test_sentence2)
```

Output:

```
test_sentence1 = "I enjoyed my journey on this flight."
predict_sentiment(test_sentence1)

test_sentence2 = "This is the worst flight experience of my life!"
predict_sentiment(test_sentence2)

Predicted label:  positive
Predicted label:  negative
```

Assignment 12

Title: Plot the cluster data using python visualization.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

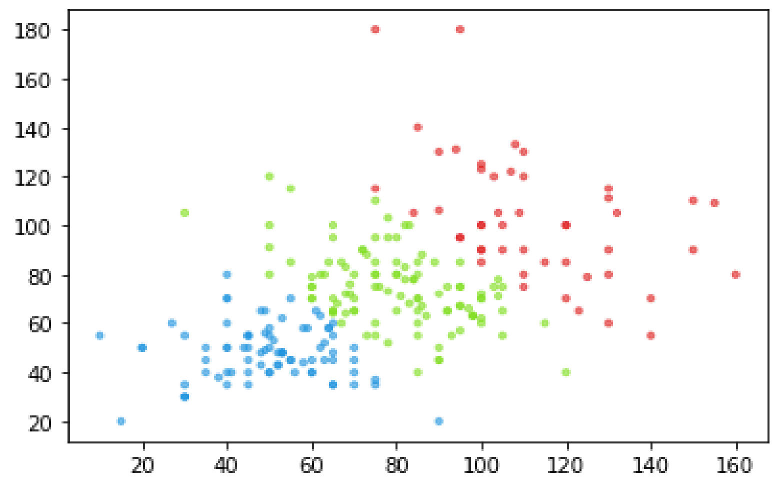
Remark:

Program:

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

df = pd.read_csv('Pokemon.csv') # prepare
data
types = df['Type 1'].isin(['Grass', 'Fire', 'Water'])
drop_cols = ['Type 1', 'Type 2', 'Generation', 'Legendary', '#'] df =
df[types].drop(columns = drop_cols)
print(df.head())

import numpy as np # k means
kmeans = KMeans(n_clusters=3, random_state=0) df['cluster'] =
kmeans.fit_predict(df[['Attack', 'Defense']]) # get centroids
centroids = kmeans.cluster_centers_ cen_x =
[i[0] for i in centroids] cen_y = [i[1] for i in
centroids]
## add to df
df['cen_x'] = df.cluster.map({0:cen_x[0], 1:cen_x[1], 2:cen_x[2]})
df['cen_y'] = df.cluster.map({0:cen_y[0], 1:cen_y[1], 2:cen_y[2]}) #
define and map colors
colors = ['#DF2020', '#81DF20', '#2095DF']
df['c'] = df.cluster.map({0:colors[0], 1:colors[1], 2:colors[2]})

plt.scatter(df.Attack, df.Defense, c=df.c, alpha = 0.6, s=10)
plt.show()
```

Output:

```
PS C:\Python\Scripts> & C:/Python/python.exe "c:/Python/Scripts/visualization of cluster.py"
                     Name  Total  HP  Attack  Defense  Sp. Atk  Sp. Def  Speed
0               Bulbasaur    318  45      49       49       65       65     45
1                 Ivysaur    405  60      62       63       80       80     60
2                Venusaur    525  80      82       83      100      100     80
3   VenusaurMega Venusaur    625  80     100      123      122      120     80
4              Charmander    309  39      52       43       60       50     65
```



Assignment 13

Title: Creating & Visualizing Neural Network for the given data. (Use python).

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

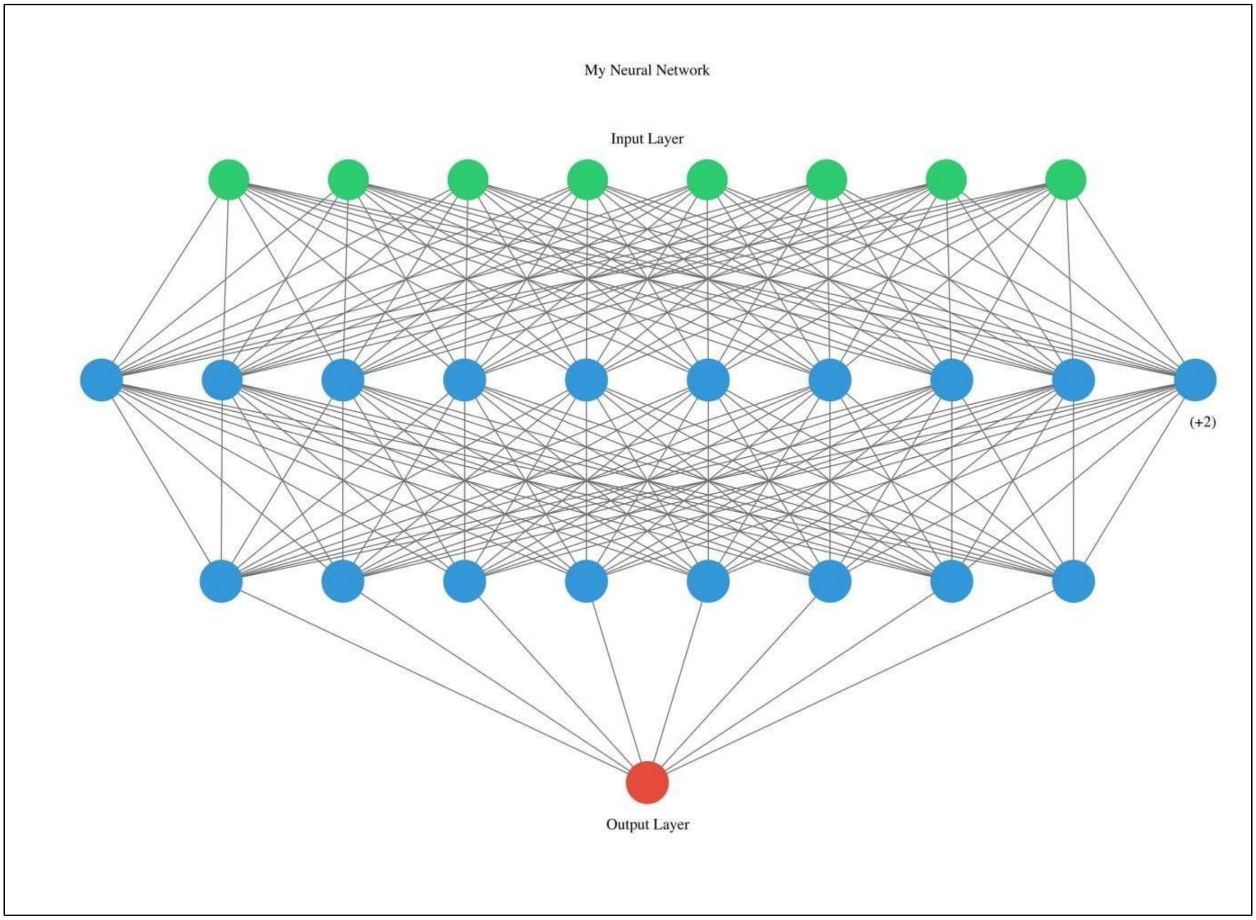
Date:

Remark:

Program:

```
# Create your first MLP in
Keras from keras.models
import Sequentialfrom
keras.layers import Dense
import numpy

# fix random seed for reproducibility
numpy.random.seed(7)

# load pima indians dataset

dataset = numpy.loadtxt("pima-indians- diabetes.csv",delimiter=",")
# split into input (X) and output (Y) variables
X=dataset[:,0:8]
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
# evaluate the model
```

Output:



Assignment 14

Title: Recognize optical character using ANN.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

Program:

```
# Importing the OCR library
import pytesseract

# Specifying the path
pytesseract.pytesseract.tesseract_cmd = r'C:/Program Files/Tesseract-OCR/tesseract.exe'

# Reading the image
image = cv2.imread('1.png')

# Extraction of text from image
text = pytesseract.image_to_string(image)

#formating the data
# Create the voice_text variable to store the data. voice_text = ""
# Pre-processing the data for i in text.split():

voice_text += i + ' '
voice_text = voice_text[:-1] voice_text

from gtts import gTTS
from playsound import playsound tts = gTTS(voice_text) tts.save("test.mp3")
playsound("test.mp3")
```

Output :

```
Attitude

is a little thing

that makes
a

Big Difference



'Attitude is a little thing that makes a Big Difference'
```

Assignment 15

Title: Write a program to implement CNN.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

Program:

```
#importing the required libraries
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
#loading data (X_train,y_train) , (X_test,y_test)=mnist.load_data()
#reshaping data
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], X_train.shape[2], 1))
X_test = X_test.reshape((X_test.shape[0],X_test.shape[1],X_test.shape[2],1))
#checking the shape after reshaping
print(X_train.shape)
print(X_test.shape)
#normalizing the pixel values
X_train=X_train/255

X_test=X_test/255
```

```
#defining model
model=Sequential()
#adding convolution layer
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
#adding pooling layer
model.add(MaxPool2D(2,2))
#adding fully connected layer
model.add(Flatten())
model.add(Dense(100,activation='relu'))
#adding output layer
model.add(Dense(10,activation='softmax'))
#compiling the model
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['ac
curacy')
#fitting the model
model.fit(X_train,y_train,epochs=10)
```

Output:

```
Epoch 1/10
1875/1875 [==============================] - 27s 14ms/step - loss: 0.8464 - accuracy: 0.7492
Epoch 2/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.3448 - accuracy: 0.8985
Epoch 3/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.2882 - accuracy: 0.9149
Epoch 4/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.2433 - accuracy: 0.9281
Epoch 5/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.2081 - accuracy: 0.9383
Epoch 6/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1841 - accuracy: 0.9442
Epoch 7/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1670 - accuracy: 0.9502
Epoch 8/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.1532 - accuracy: 0.9546
Epoch 9/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1426 - accuracy: 0.9578
Epoch 10/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1329 - accuracy: 0.9600
```

Assignment 16

Title: Write a program to implement RNN.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

Program:

```
import numpy as np
import matplotlib.pyplot as plt
class ReccurentNN:
def init (self, char_to_idx, idx_to_char, vocab, h_size=75,seq_len=20, clip_value=5,
epochs=50, learning_rate=1e-2):
self.n_h = h_size
self.seq_len = seq_len # number of characters in each batch/time steps
self.clip_value = clip_value # maximum allowed value for the gradients self.epochs =
epochs self.learning_rate = learning_rate self.char_to_idx = char_to_idx #
dictionary that maps characters to an index self.idx_to_char = idx_to_char # dictionary
that maps indices to characters self.vocab = vocab # number of unique characters in the
training text
# smoothing out loss as batch SGD is noisy
self.smooth_loss = -np.log(1.0 / self.vocab) * self.seq_len
# initialize parameters self.params = {}
self.params["W_xh"] = np.random.randn(self.vocab, self.n_h) * 0.01
self.params["W_hh"] = np.identity(self.n_h) * 0.01
self.params["b_h"] = np.zeros((1, self.n_h))
self.params["W_hy"] = np.random.randn(self.n_h, self.vocab) * 0.01
self.params["b_y"] = np.zeros((1, self.vocab))
self.h0 = np.zeros((1, self.n_h)) # value of the hidden state at time step t = -1
```

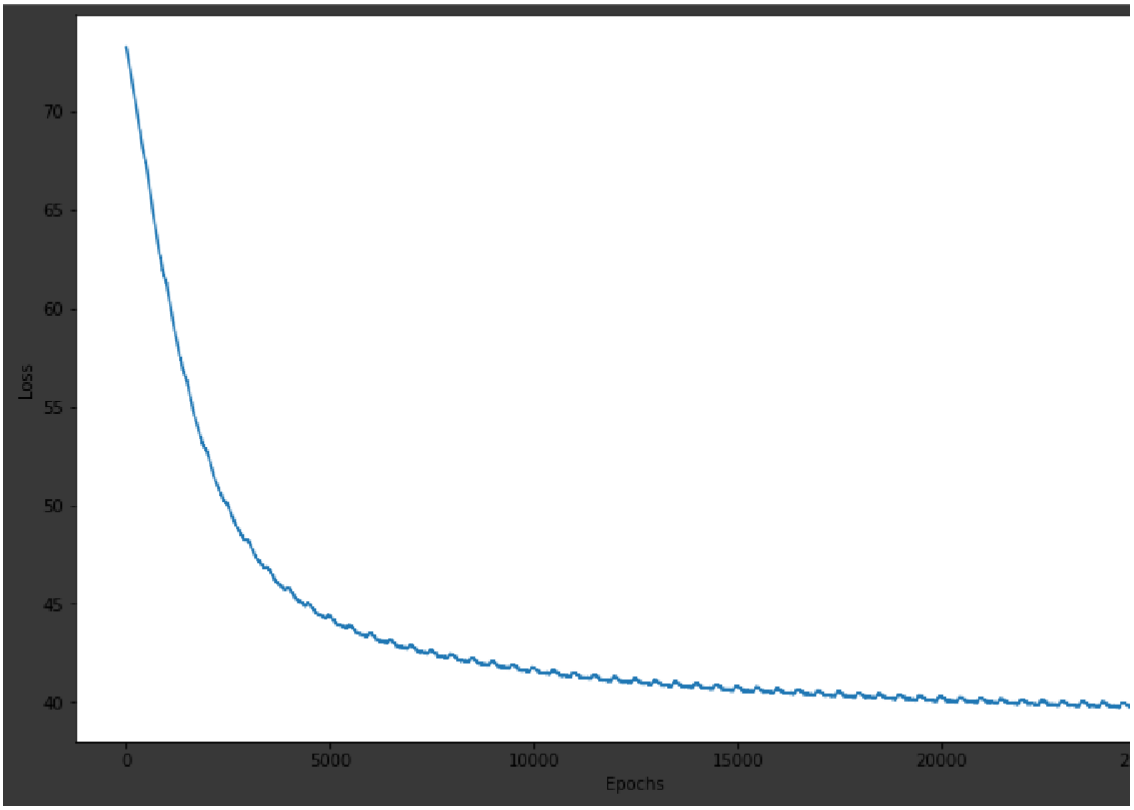
```
# initialize gradients and memory parameters for Adagrad
self.grads = {}
self.m_params = {}
for key in self.params:
self.grads["d" + key] = np.zeros_like(self.params[key])
self.m_params["m" + key] = np.zeros_like(self.params[key])
def _encode_text(self, X):
X_encoded = []
for char in X:
X_encoded.append(self.char_to_idx[char])
return X_encoded
def _prepare_batches(self, X, index):
X_batch_encoded = X[index: index + self.seq_len]
y_batch_encoded = X[index + 1: index + self.seq_len + 1]
X_batch = []
y_batch = []
for i in X_batch_encoded:
one_hot_char = np.zeros((1, self.vocab))
one_hot_char[0][i] = 1
X_batch.append(one_hot_char)
for j in y_batch_encoded:
one_hot_char = np.zeros((1, self.vocab))
one_hot_char[0][j] = 1
y_batch.append(one_hot_char)
return X_batch, y_batch
def _softmax(self, x):
# max value is substracted for numerical stability
# # https://stats.stackexchange.com/a/338293
e_x = np.exp(x - np.max(x))
return e_x / np.sum(e_x)
def _forward_pass(self, X):
h = {} # stores hidden states
h[-1] = self.h0 # set initial hidden state at t=-1
y_pred = {} # stores softmax output probabilities
# iterate over each character in the input sequence
for t in range(self.seq_len):
h[t] = np.tanh(np.dot(X[t], self.params["W_xh"]) + np.dot(h[t - 1],
self.params["W_hh"]) + self.params["b_h"])
```

```
y_pred[t] = self._softmax(np.dot(h[t], self.params["W_hy"]) + self.params["b_y"])
self.ho = h[t]
return y_pred, h
def _backward_pass(self, X, y, y_pred, h):
dh_next = np.zeros_like(h[0])
for t in reversed(range(self.seq_len)):
dy = np.copy(y_pred[t])
dy[0][np.argmax(y[t])] -= 1 # predicted y - actual y
self.grads["dW_hy"] += np.dot(h[t].T, dy)
self.grads["db_y"] += dy
dhidden = (1 - h[t] ** 2) * (np.dot(dy, self.params["W_hy"].T) + dh_next)
dh_next = np.dot(dhidden, self.params["W_hh"].T)
self.grads["dW_hh"] += np.dot(h[t - 1].T, dhidden)
self.grads["dW_xh"] += np.dot(X[t].T, dhidden)
self.grads["db_h"] += dhidden
# clip gradients to mitigate exploding gradients
for grad, key in enumerate(self.grads):
np.clip(self.grads[key], -self.clip_value,self.clip_value, out=self.grads[key])
return
def _update(self):
for key in self.params:
self.m_params["m" + key] += self.grads["d" + key] * self.grads["d" + key]
self.params[key] -= self.grads["d" + key] * self.learning_rate /
(np.sqrt(self.m_params["m" + key]) +1e-8)
def test(self, test_size, start_index):

res = ""
x = np.zeros((1, self.vocab))
x[0][start_index] = 1
for i in range(test_size):
# forward propagation
h = np.tanh(np.dot(x, self.params["W_xh"]) + np.dot(self.h0, self.params["W_hh"]) +
self.params["b_h"])
y_pred = self._softmax(np.dot(h, self.params["W_hy"]) + self.params["b_y"])
# get a random index from the probability distribution of y
index = np.random.choice(range(self.vocab), p=y_pred.ravel())
# set x-one_hot_vector for the next character
x = np.zeros((1, self.vocab))
x[0][index] = 1
```

```
# find the char with the index and concat to the output string
char = self.idx_to_char[index]
res += char
return res
def train(self, X):
J = []
num_batches = len(X) // self.seq_len
X_trimmed = X[:num_batches * self.seq_len]
# trim end of the input text so that we have full sequences
X_encoded = self._encode_text(X_trimmed)
# transform words to indices to enable processing
for i in range(self.epochs):
for j in range(0, len(X_encoded) - self.seq_len, self.seq_len):
X_batch, y_batch = self._prepare_batches(X_encoded, j)
y_pred, h = self._forward_pass(X_batch)
loss = 0
for t in range(self.seq_len):
loss += -np.log(y_pred[t][0, np.argmax(y_batch[t])])
self.smooth_loss = self.smooth_loss * 0.999 + loss * 0.001
J.append(self.smooth_loss)
self._backward_pass(X_batch, y_batch, y_pred, h)
self._update()
print('Epoch:', i + 1, "\tLoss:", loss, "")
return J, self.params
with open('Harry-Potter.txt') as f:
text = f.read().lower()
# use only a part of the text to make the process faster
text = text[:20000]
text = [char for char in text if char not in ["(", ")", "\"", "'", ".", "?", "!", ",", "-"]]
text = [char for char in text if char not in ["(", ")", "\"", "'"]]
chars = set(text)
vocab = len(chars)
print(f"Length of training text {len(text)}")
print(f"Size of vocabulary {vocab}")
# creating the encoding decoding dictionaries
char_to_idx = {w: i for i, w in enumerate(chars)}
idx_to_char = {i: w for i, w in enumerate(chars)}
parameter_dict = {
'char_to_idx': char_to_idx,
'idx_to_char': idx_to_char,
'vocab': vocab,
'h_size': 75,
'seq_len': 20,
# keep small to avoid diminishing/exploding gradients
'clip_value': 5,
'epochs': 50,
'learning_rate': 1e-2,}
 model = ReccurentNN(**parameter_dict)
loss, params = model.train(text)
plt.figure(figsize=(12, 8))
plt.plot([i for i in range(len(loss))], loss)
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.show()
print(model.test(50,10))
```

Output:

```
Epoch: 1 Loss: 5 6.938160313575075
Epoch: 2 Loss: 49.479841032771944
Epoch: 3 Loss: 44.287300754487774
Epoch: 4 Loss: 42.75894603770088
Epoch: 5 Loss: 40.962449282519785
Epoch: 6 Loss: 41.06907316142755
Epoch: 7 Loss: 39.77795494997328
Epoch: 8 Loss: 41.059521063295485
Epoch: 9 Loss: 39.848893648177594
Epoch:10 Loss: 40.42097045126549
Epoch:11 Loss: 39.183043247471126
Epoch:12 Loss: 40.09713939411275
Epoch:13 Loss: 38.786694845855145
Epoch:14 Loss: 39.41259563289025
Epoch:15 Loss: 38.87094988626352
Epoch:16 Loss: 38.80896936130275
Epoch:17 Loss: 38.65301294936609
Epoch:18 Loss: 38.2922486206415
Epoch:19 Loss: 38.120326247610286
Epoch:20 Loss: 37.94743442371039
Epoch:21 Loss: 37.781826419304245
Epoch:22 Loss: 38.02242197941186
Epoch:23 Loss: 37.34639374983505
Epoch:24 Loss: 37.383830387022115
Epoch:25 Loss: 36.863261576664286
Epoch:26 Loss: 36.81717706027801
Epoch:27 Loss: 35.98781618662626
Epoch: 28 Loss: 34.883143187020806
Epoch: 29 Loss: 35.74233839750379
Epoch: 30 Loss: 34.17457373354039
Epoch: 31 Loss: 34.3659838303625
Epoch: 32 Loss: 34.6155982440106
Epoch: 33 Loss: 33.428021716569035
Epoch: 34 Loss: 33.06226727751935
Epoch: 35 Loss: 33.23334401686566
Epoch: 36 Loss: 32.9818416477839
Epoch: 37 Loss: 33.155764725505655
Epoch: 38 Loss: 32.937205806520474
Epoch: 39 Loss: 32.93063638107538
Epoch: 40 Loss: 32.943368437981256
Epoch: 41 Loss: 32.92520056534523
Epoch: 42 Loss: 32.96074563399301
Epoch: 43 Loss: 32.974579784369666
Epoch: 44 Loss: 32.86483014312194
Epoch: 45 Loss: 33.10532379921245
Epoch: 46 Loss: 32.89950584889016
Epoch: 47 Loss: 33.11303116056217
Epoch: 48 Loss: 32.731237824441756
Epoch: 49 Loss: 32.742918023080314
Epoch: 50 Loss: 32.421869906086144
```


Loss
70
65
60
55
50
45
40
0
5000
10000
15000
20000
Epochs

Assignment 17

Title: Write a program to implement GAN.

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

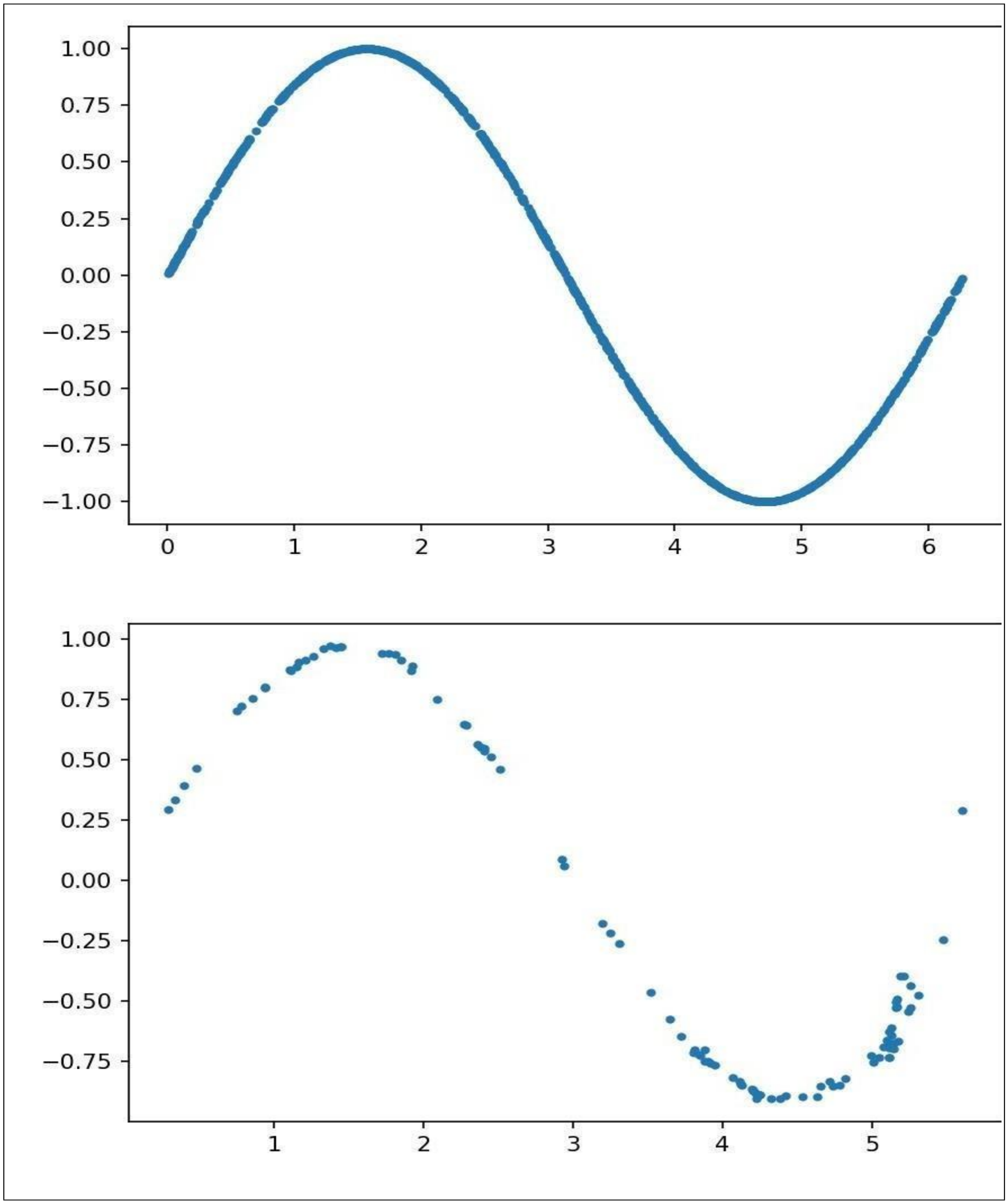
Remark:

Program:

```
# Create a PyTorch data loader batch_size = 32
train_loader = torch.utils.data.DataLoader(
train_set, batch_size=batch_size, shuffle=True )
#Implementing the Discriminator, in PyTorch, the neural network models are represented by
classes that inherit from nn.Module
class Discriminator(nn.Module):
def init (self):
super(). init ()
self.model = nn.Sequential(nn.Linear(2, 256),
nn.ReLU(),nn.Dropout(0.3),nn.Linear(256,128),nn.ReLU(),nn.Dropout(0.3),
                        nn.Linear(128, 64),nn.ReLU(),nn.Dropout(0.3),nn.Linear(64,1),nn.Sigmoid(),)
def forward(self, x):
output = self.model(x)
return output
#instantiate a Discriminator object
discriminator = Discriminator()
#Implementing the Generator, create a Generator class that inherits from nn.Module
class Generator(nn.Module):
def init (self):
super(). init ()
self.model = nn.Sequential(nn.Linear(2, 16),nn.ReLU(),nn.Linear(16,
32),nn.ReLU(),nn.Linear(32, 2),)
def forward(self, x):
output = self.model(x)
return output
generator = Generator()
#set up parameters to use during training lr = 0.001 num_epochs = 300
loss_function = nn.BCELoss()

#Create the optimizers using torch.optim
optimizer_discriminator = torch.optim.Adam(discriminator.parameters(), lr=lr)
optimizer_generator = torch.optim.Adam(generator.parameters(), lr=lr)
```

```
# implement a training loop
for epoch in range(num_epochs):
for n, (real_samples, _) in enumerate(train_loader):
# Data for training the discriminator
real_samples_labels = torch.ones((batch_size, 1))
latent_space_samples = torch.randn((batch_size, 2))
generated_samples = generator(latent_space_samples)
generated_samples_labels = torch.zeros((batch_size, 1))
all_samples = torch.cat((real_samples, generated_samples))
all_samples_labels = torch.cat((real_samples_labels, generated_samples_labels) )
# Training the discriminator discriminator.zero_grad()
output_discriminator = discriminator(all_samples)
loss_discriminator = loss_function(
output_discriminator, all_samples_labels)
loss_discriminator.backward()
optimizer_discriminator.step()
# Data for training the generator
latent_space_samples = torch.randn((batch_size, 2))
# Training the generator
generator.zero_grad()
generated_samples = generator(latent_space_samples)

output_discriminator_generated = discriminator(generated_samples)
loss_generator = loss_function(output_discriminator_generated, real_samples_labels)
loss_generator.backward()
optimizer_generator.step()


# Show loss
if epoch % 10 == 0 and n == batch_size - 1:
print(f"Epoch: {epoch} Loss D.: {loss_discriminator}")
print(f"Epoch: {epoch} Loss G.: {loss_generator}")
# Checking the Samples Generated by the GAN
latent_space_samples = torch.randn(100, 2)
generated_samples = generator(latent_space_samples)
generated_samples = generated_samples.detach()
plt.plot(generated_samples[:, 0], generated_samples[:, 1], ".")
```

Output:


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
1
2
3
4
5

Assignment 18

Tittle: Web scraping experiments (by using tools).

Name: Pravin Santosh Adhav

Class: MCA II

Roll No: MC232501

Date:

Remark:

Program:

```
#Here Import all the packages which are required for extracting the information from particular
URL
import pandas as pd
import requests
import csv
from bs4 import BeautifulSoup
"""Here Request package is used for getting the request of information from URL"""
url = 'https://islamqa.info/en/answers/1/interruption-of-wudhu'
page= requests.get(url)
page
page.content
"""Here with the help of BeautifulSoup package we have to convert the text in HTML
format"""
soup= BeautifulSoup(page.content,'html.parser')
soup
answer=soup.findAll(attrs={'class':'content'})
answer
answer[0].text

answer[0].text.replace('\n'," ")
summary= soup.find(attrs={'class':'title is-4 is-size-5-touch'}).text.replace('\n'," ")
summary

questionNo= int(soup.find(attrs={'class':'subtitle has-text-weight-bold has-title-case cursor-
pointer tooltip'}).text.replace('\n'," "))
questionNo
source= soup.find(attrs={'class':'subtitle is-6 has-text-weight-bold is-
capitalized'}).text.replace('\n',"").replace('source:',"")
source
```

```
"""*Pandas Library used:*"""
data =[[url,answer,summary,questionNo,source]]
data
df = pd.DataFrame(data,columns=['url','answer','summary','questionNo','source'])
df
"""Here data is fetched and create one pagedata.csv file """
for i in range(1,10):
URL = 'https://islamqa.info/en/answers/'+str(i)
page = requests.get(URL)
if(page.status_code==200):
print('Data Fetched Successfully',i)
soup=soup= BeautifulSoup(page.content,'html.parser')
answer=soup.findAll(attrs={'class':'content'})
A=answer[0].text.replace('\n'," ")
S=soup.find(attrs={'class':'title is-4 is-size-5-touch'}).text.replace('\n'," ")
QN=int(soup.find(attrs={'class':'subtitle has-text-weight-bold has-title-case cursor-pointer
tooltip'}).text.replace('\n'," "))
S1=soup.find(attrs={'class':'subtitle is-6 has-text-weight-bold is-
capitalized'}).text.replace('\n',"").replace('source:',"")
data.insert(QN,[URL,QN,A,S,S1])
else:
print('URL NOT FOUND',i)
df = pd.DataFrame(data,columns=['url','answer','summary','questionNo','source'])
df.to_csv('pagedata.csv')
```

Output:

```
Data Fetched Successfully 1
Data Fetched Successfully 2
Data Fetched Successfully 3
Data Fetched Successfully 4
Data Fetched Successfully 5
Data Fetched Successfully 6
Data Fetched Successfully 7
Data Fetched Successfully 8
Data Fetched Successfully 9
```