Laboratory File

MCA-II Sem-III

Course and Code

Knowledge Representation and Artificial Intelligence-ML, DL (IT31L)

Course Coordinators

Prof. Kirti Samrit

Prof. Dharmendra Singh

For Academic Year 2022-2023

Department of Master of Computer Application

ZEAL EDUCATION SOCIETY'S
ZEAL INSTITUTE OF BUSINESS ADMINISTRATION, COMPUTER APPLICATION AND RESEARCH (ZIBACAR)
NARHE | PUNE | INDIA
PUN CODE: IMMP013170 DTE CODE: 6152 AISHE CODE: C-41828

CERTIFICATE

This is to certify that Mr. / Miss **Akash Kallappa Fulari** of Class MCA (SEM-III), Roll No. **MC21017**, Exam Seat No. _____, has completed all the Term Work / Practical work in the subject "**Knowledge Representation and Artificial Intelligence - ML, DL (IT31L)"** satisfactorily in the Department **of MCA** as prescribed by **Savitribai Phule Pune University** in the academic year 2022- 2023.

Date :

Course Coordinator **Director**

Examiner1 Sign **Examiner2 Sign**

INDEX

Course and Code: - KRAI - ML,DL (IT31L) Laboratory File

Sr. No.	Practical	Date	Page No.	Remark	Sign of Faculty
1	Find the correlation matrix.				
2	Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.				
3	Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.				
4	Apply linear regression Model techniques to predict the data on any dataset.				
5	Apply logical regression Model techniques to predict the data on any dataset.				
6	Clustering algorithms for unsupervised classification.				
7	Association algorithms for supervised classification on any dataset				
8	Developing and implementing Decision Tree model on the dataset				
9	Bayesian classification on any dataset.				
10	SVM classification on any dataset				
11	Text Mining algorithms on unstructured dataset				
12	Plot the cluster data using python visualizations.				
13	Creating & Visualizing Neural Network for the given data. (Use python)				
14	Recognize optical character using ANN.				
15	Write a program to implement CNN				
16	Write a program to implement RNN				
17	Write a program to implement GAN				
18	Web scraping experiments (by using tools)				

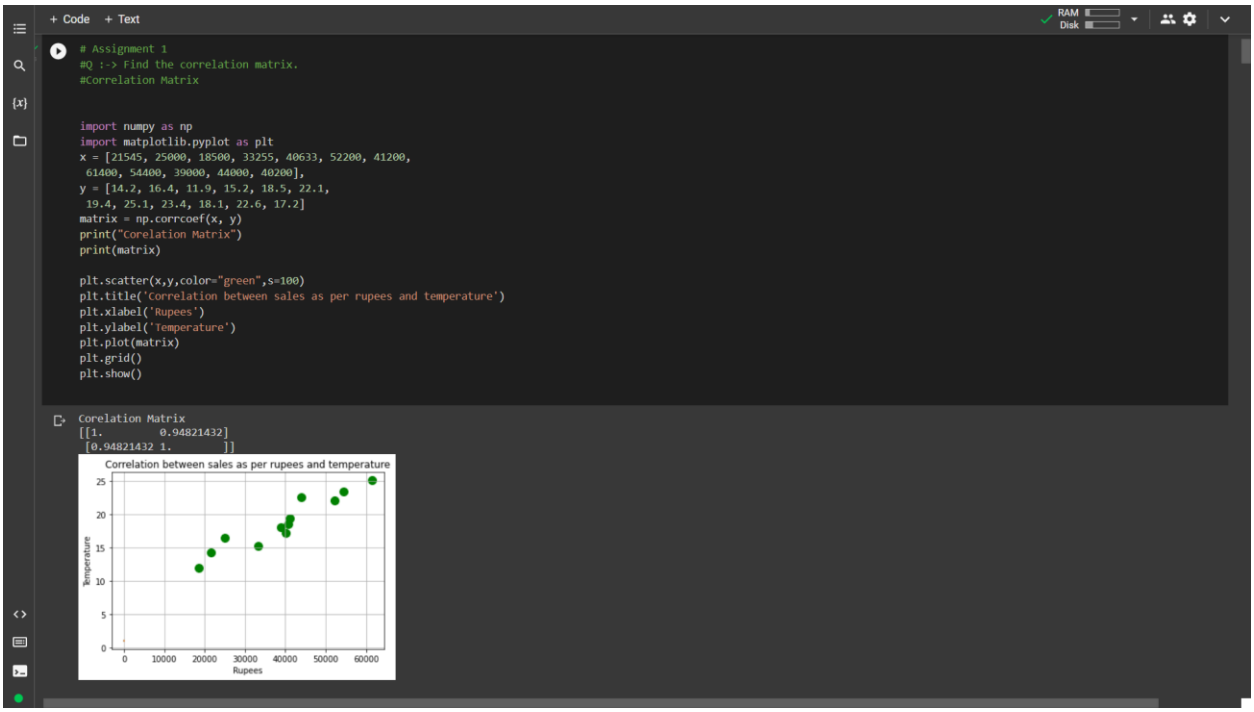
Assignment: 01

TITLE : Find the correlation matrix.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 1
#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("Corelation Matrix")
print(matrix)

plt.scatter(x,y,color="green",s=100)
plt.title('Correlation between sales as per rupees and temperature')
plt.xlabel('Rupees')
plt.ylabel('Temperature')
plt.plot(matrix)
plt.grid()
plt.show()
```

Output :


+ Code
+ Text
RAM
Disk
Assignment 1
#Q :-> Find the correlation matrix.
#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("Corelation Matrix")
print(matrix)
plt.scatter(x,y,color="green",s=100)
plt.title('Correlation between sales as per rupees and temperature')
plt.xlabel('Rupees')
plt.ylabel('Temperature')
plt.plot(matrix)
plt.grid()
plt.show()
Corelation Matrix
[[1. 0.94821432]
[0.94821432 1.]]
Correlation between sales as per rupees and temperature
Temperature
Rupees
0
5
10
15
20
25
0
10000
20000
30000
40000
50000
60000

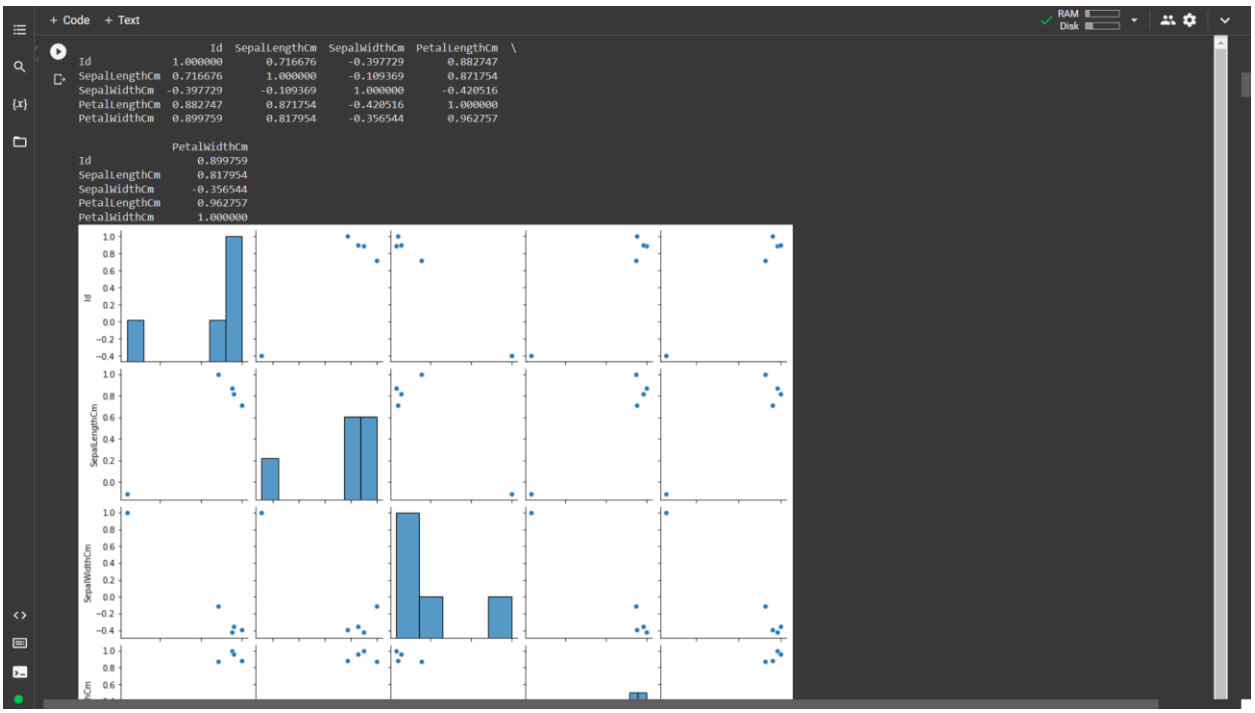
Assignment: 02

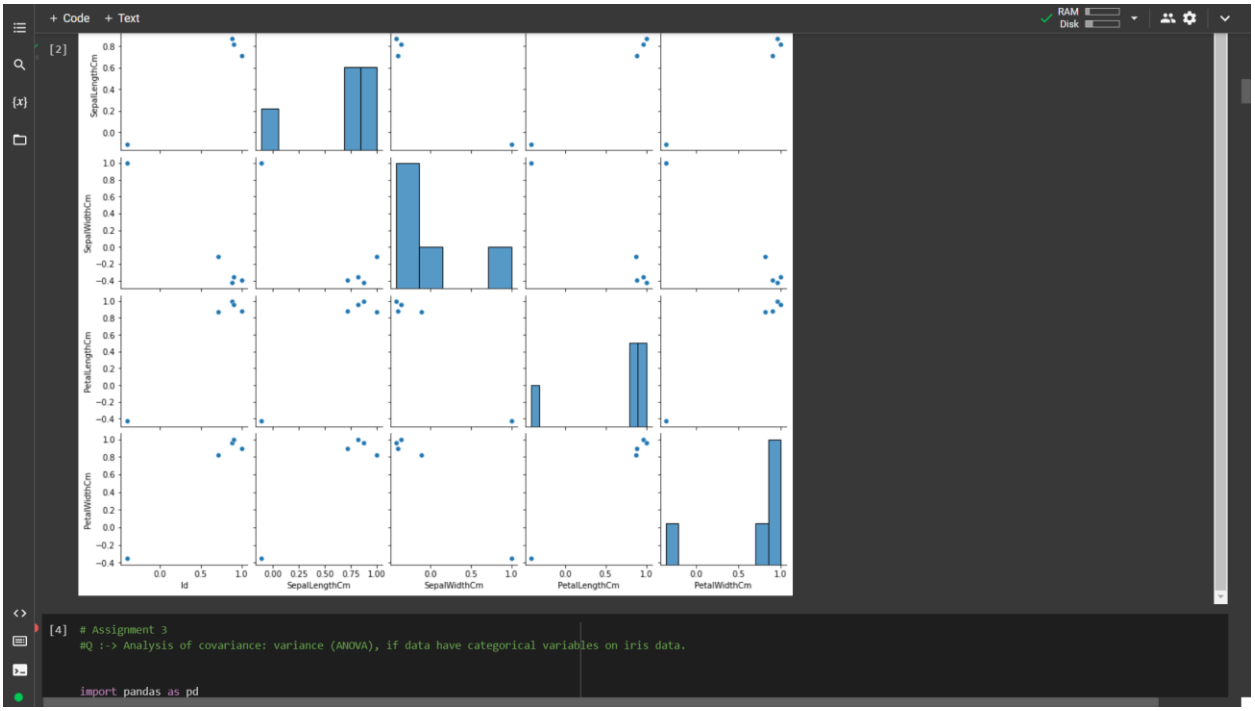
TITLE : Plot the correlation plot on the dataset and visualize giving an overview of relationships among data on iris data.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 2
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv("/content/Iris.csv")
rel = df.corr()
print(rel)
sns.pairplot(rel)
plt.show()
```

Output :


+ Code
+ Text
RAM
Disk
Id SepalLengthCm SepalWidthCm PetalLengthCm \
Id 1.000000 0.716676 -0.397729 0.882747
SepalLengthCm 0.716676 1.000000 -0.109369 0.871754
SepalWidthCm -0.397729 -0.109369 1.000000 -0.420516
PetalLengthCm 0.882747 0.871754 -0.420516 1.000000
PetalWidthCm 0.899759 0.817954 -0.356544 0.962757
PetalWidthCm
Id 0.899759
SepalLengthCm 0.817954
SepalWidthCm -0.356544
PetalLengthCm 0.962757
PetalWidthCm 1.000000
Id
SepalLengthCm
SepalWidthCm


+ Code
+ Text
RAM
Disk
[2]
SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm
Id
SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm
[4] # Assignment 3
#Q :-> Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.
import pandas as pd

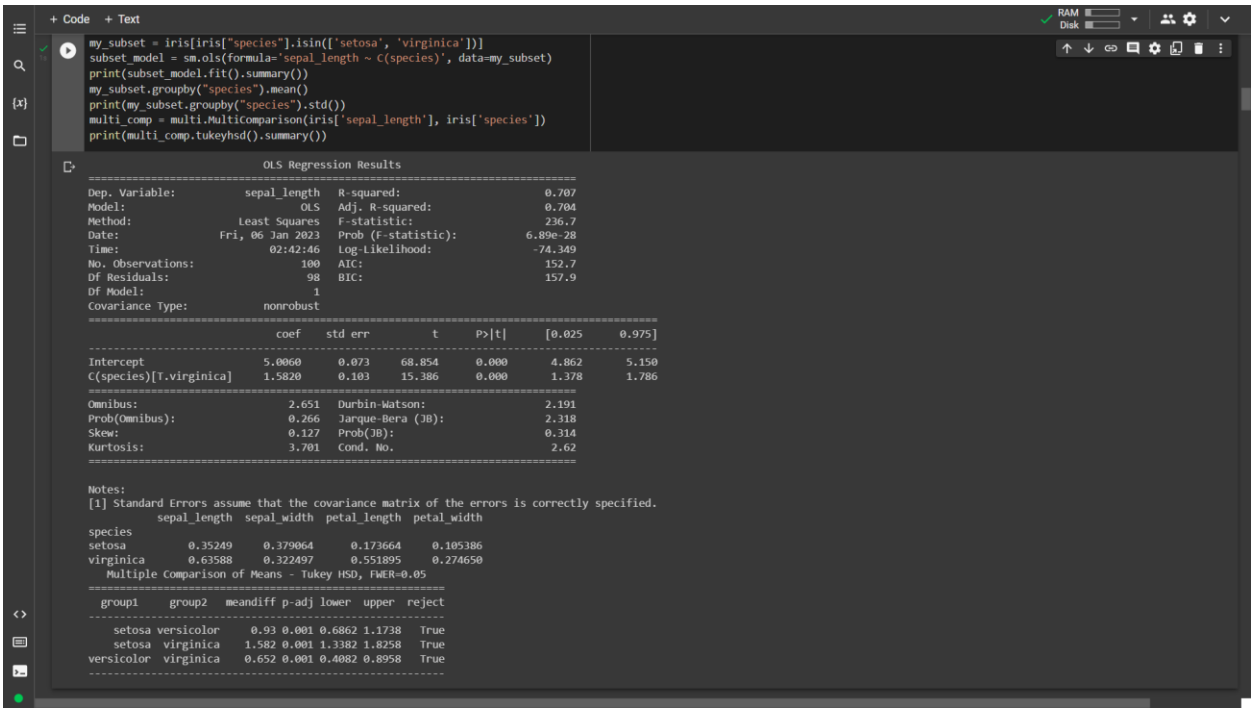
Assignment: 03

TITLE : Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 3
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 :



```
+ Code + Text                                                                    RAM Disk

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())

                            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:                Fri, 06 Jan 2023   Prob (F-statistic):          6.89e-28
Time:                        02:42:46   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 0.001 0.6862 1.1738   True
    setosa  virginica    1.582 0.001 1.3382 1.8258   True
versicolor  virginica    0.652 0.001 0.4082 0.8958   True
-----------------------------------------------------
```

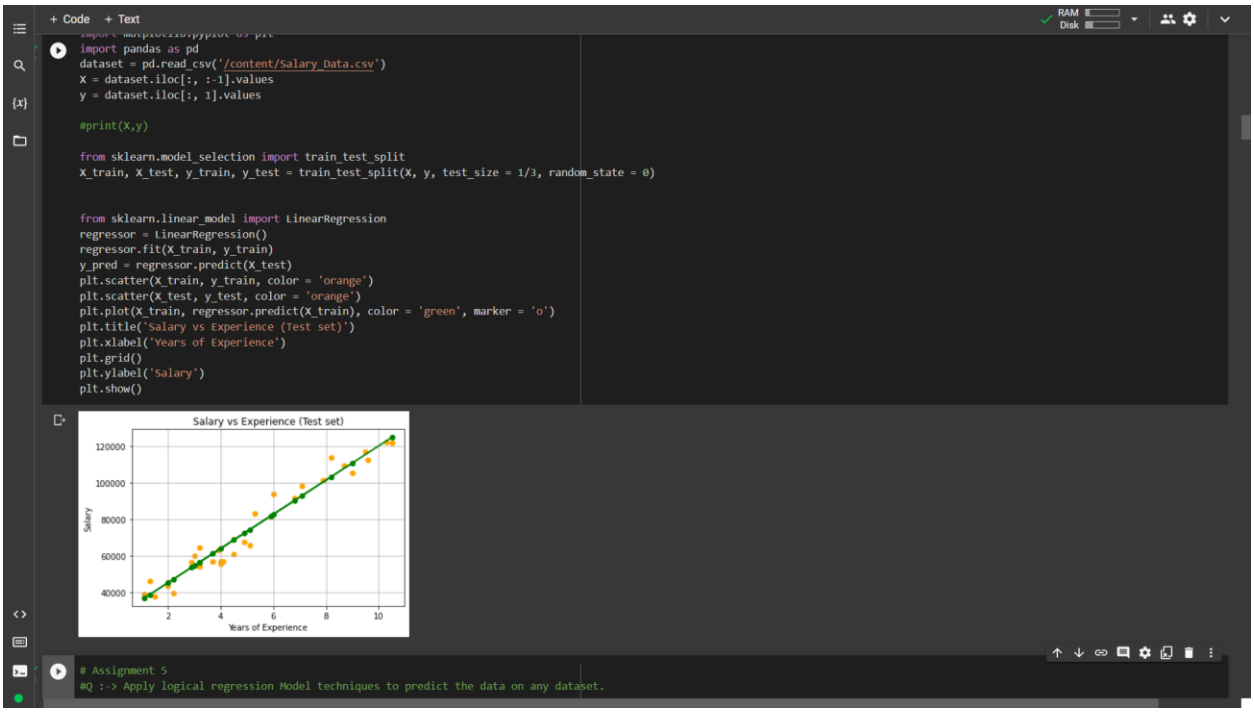
Assignment: 04

TITLE : Apply linear regression Model techniques to predict the data on any dataset.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 4
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('/content/Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
#print(X,y)
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 = 'orange')
plt.scatter(X_test, y_test, color = 'orange')
plt.plot(X_train, regressor.predict(X_train), color = 'green', marker = 'o')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.grid()
plt.ylabel('Salary')
plt.show()
```

Output :


+ Code
+ Text
RAM
Disk
import pandas as pd
dataset = pd.read_csv('/content/Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
#print(X,y)
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 = 'orange')
plt.scatter(X_test, y_test, color = 'orange')
plt.plot(X_train, regressor.predict(X_train), color = 'green', marker = 'o')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.grid()
plt.ylabel('Salary')
plt.show()
Salary vs Experience (Test set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Years of Experience
Assignment 5
#Q :-> Apply logical regression Model techniques to predict the data on any dataset.

Assignment: 05

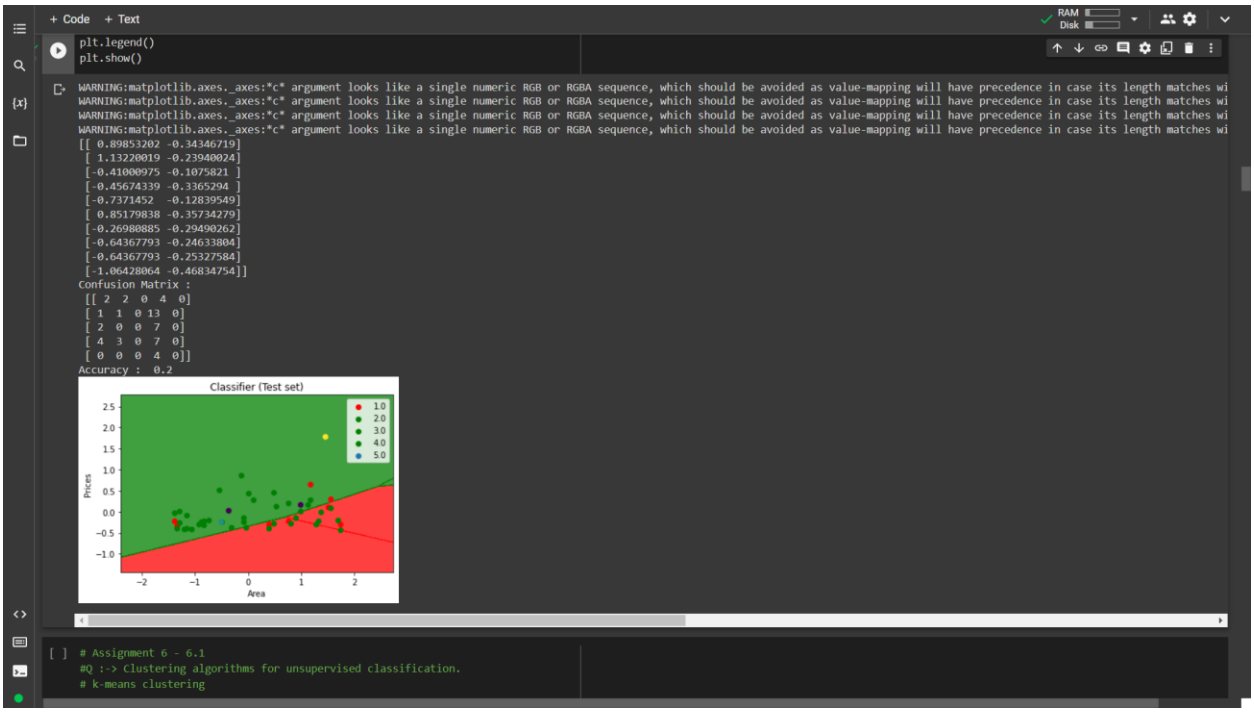
TITLE : Apply logical regression Model techniques to predict the data on any dataset.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 5
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('/content/ChurnData.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 :


+ Code + Text
RAM
Disk
plt.legend()
plt.show()
WARNING:matplotlib.axes._axes:*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 wi
WARNING:matplotlib.axes._axes:*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 wi
WARNING:matplotlib.axes._axes:*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 wi
WARNING:matplotlib.axes._axes:*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 wi
[[0.89853202 -0.34346719]
[1.13220019 -0.23940024]
[-0.41000975 -0.1075821]
[-0.45674339 -0.3365294]
[-0.7371452 -0.12839549]
[0.85179838 -0.35734279]
[-0.26980885 -0.29490262]
[-0.64367793 -0.24633804]
[-0.64367793 -0.25327584]
[-1.06428064 -0.46834754]]
Confusion Matrix :
[[2 2 0 4 0]
[1 1 0 13 0]
[2 0 0 7 0]
[4 3 0 7 0]
[0 0 0 4 0]]
Accuracy : 0.2
Classifier (Test set)
1.0
2.0
3.0
4.0
5.0
Prices
Area
[ ] # Assignment 6 - 6.1
#Q :-> Clustering algorithms for unsupervised classification.
k-means clustering

Assignment: 06

TITLE : Clustering algorithms for unsupervised classification.
Class : MCA -II
Roll No. : MC21017
Date :

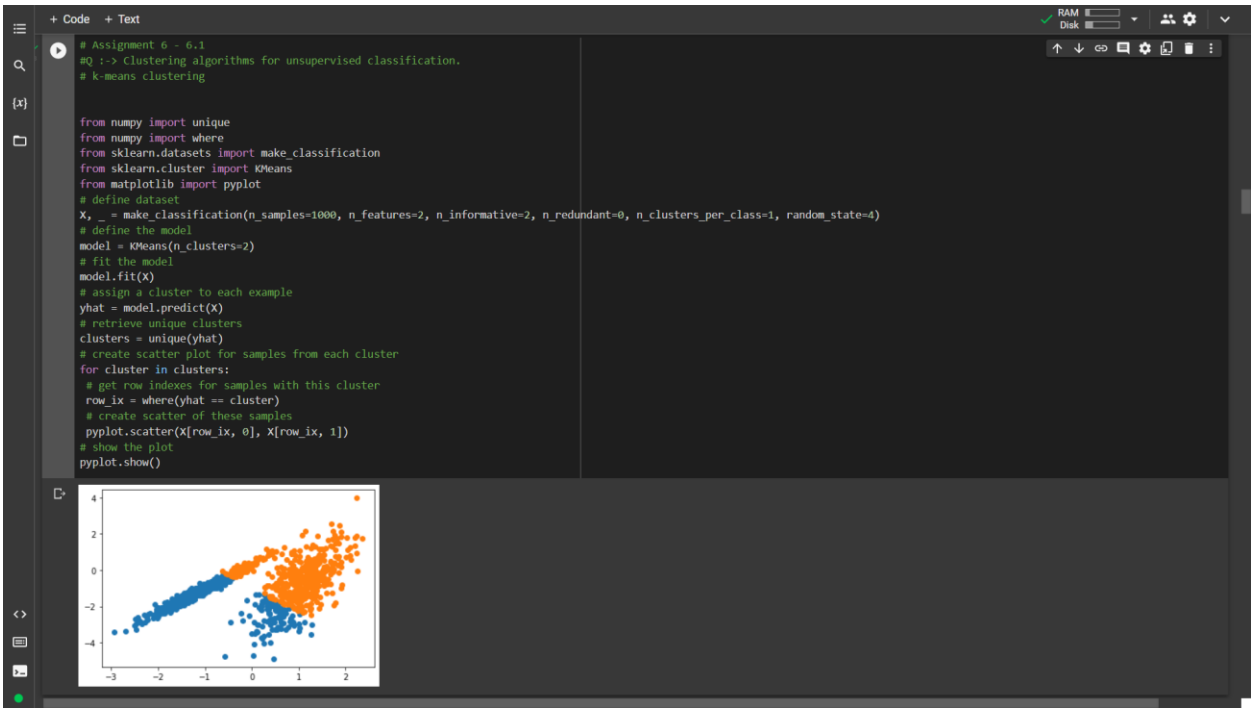
Program :

```
# Assignment 6
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_per_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()
```

```
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_per_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()



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

# 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()
```

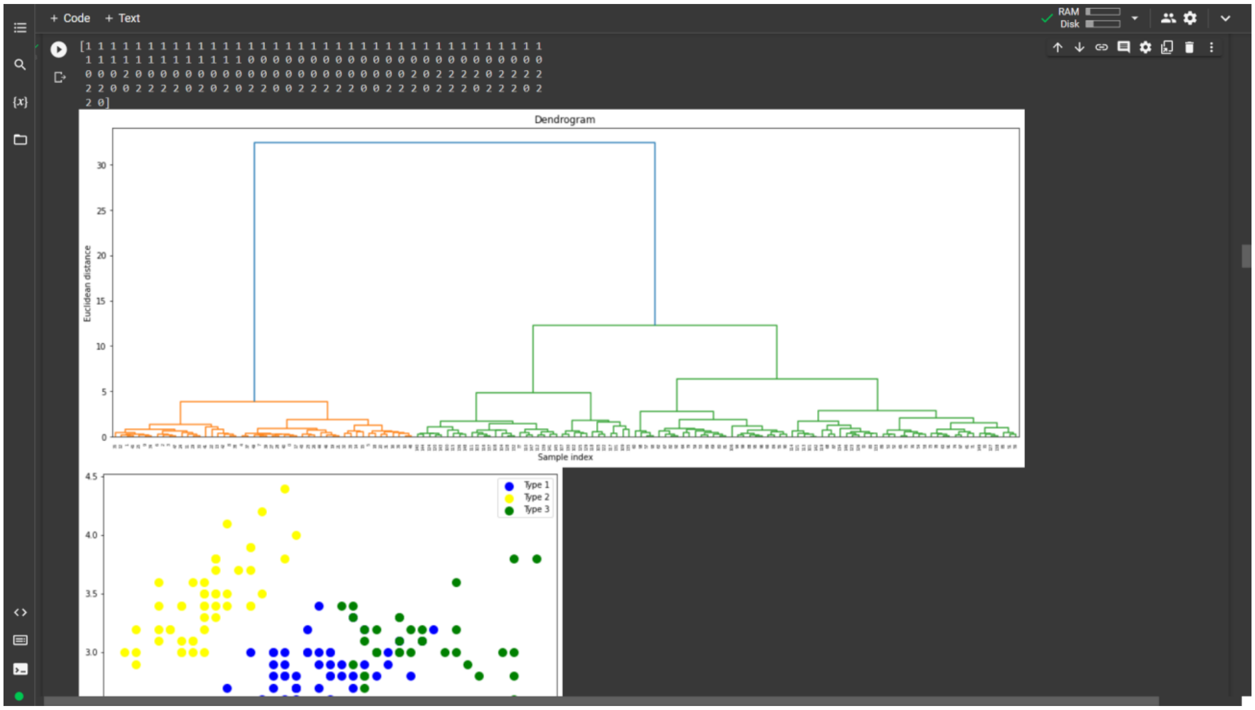
Output :

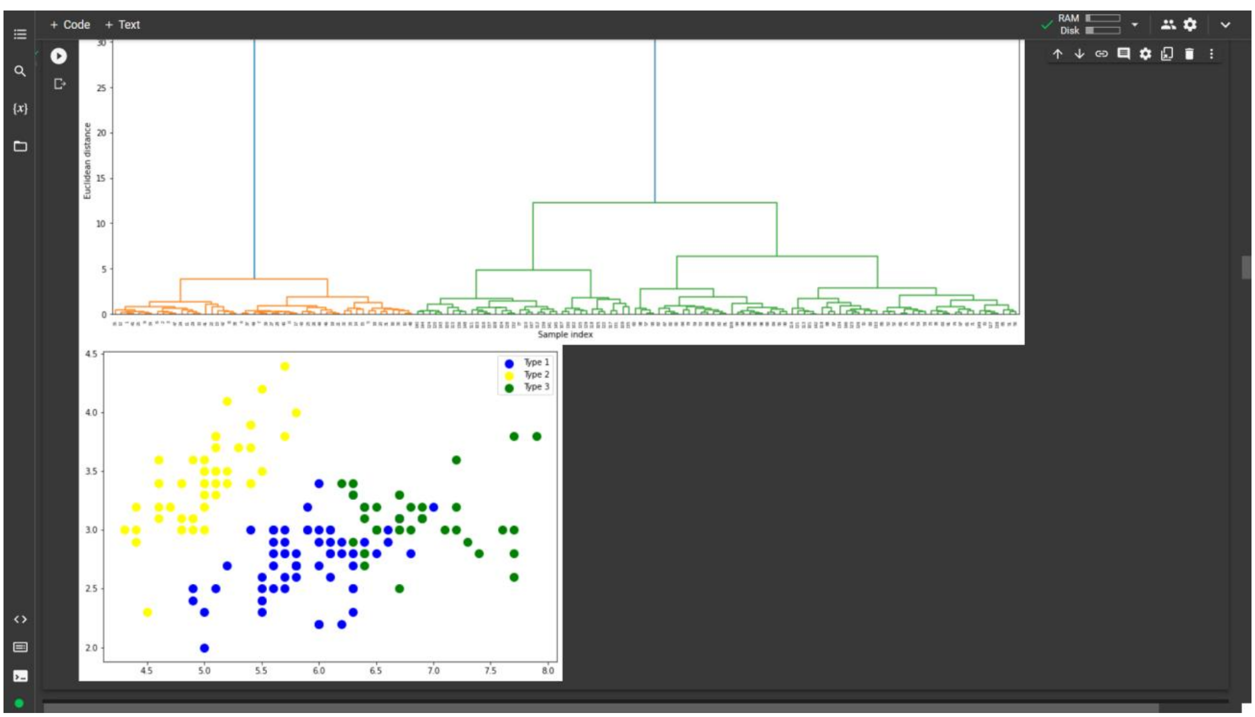


```
# Assignment 6 - 6.1
#Q :-> Clustering algorithms for unsupervised classification.
# k-means clustering


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_per_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()
```

```
# Assignment 6 - 6.2
# 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_per_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()
```

```
[ ] # Assignment 6 - 6.3
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
```


+ Code + Text
RAM
Disk
[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
Type 1
Type 2
Type 3


+ Code + Text
RAM
Disk
Euclidean distance
Sample index
Type 1
Type 2
Type 3

Assignment: 07

TITLE : Association algorithms for supervised classification on any dataset.
Class : MCA -II
Roll No. : MC21017
Date :

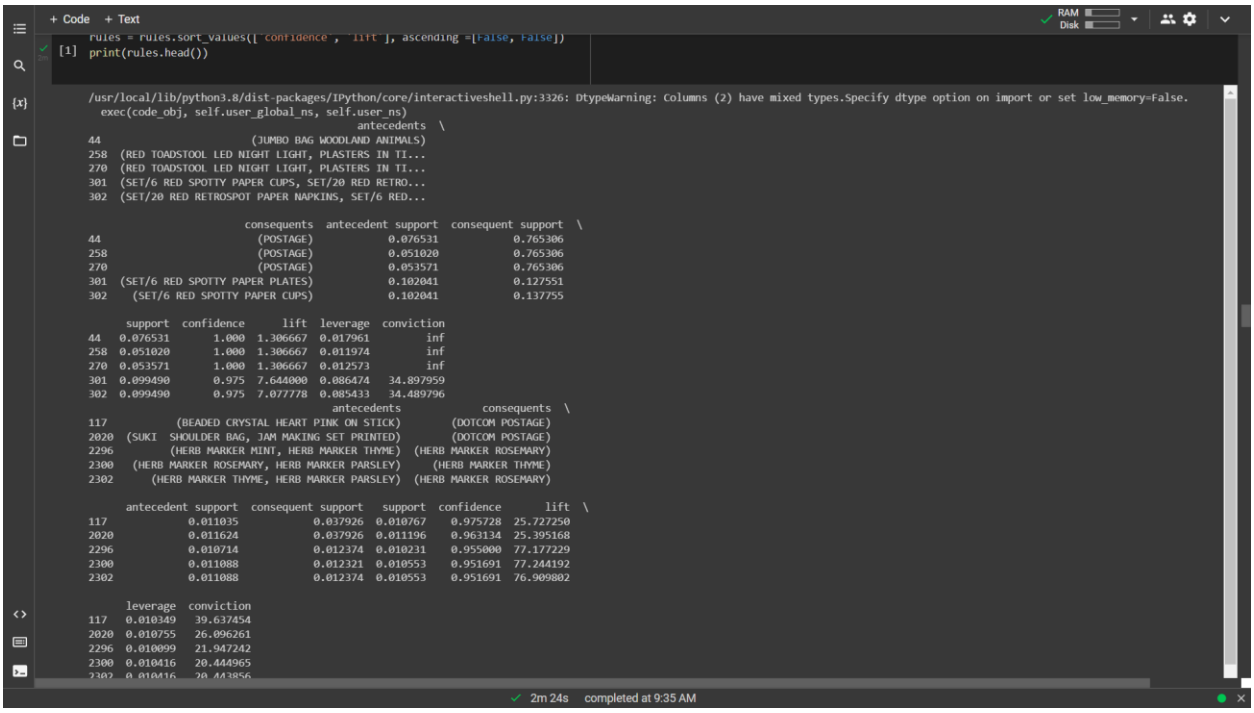
Program :

```
# Assignment 7
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
data = pd.read_csv('/content/Online Retail.csv')
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']
```

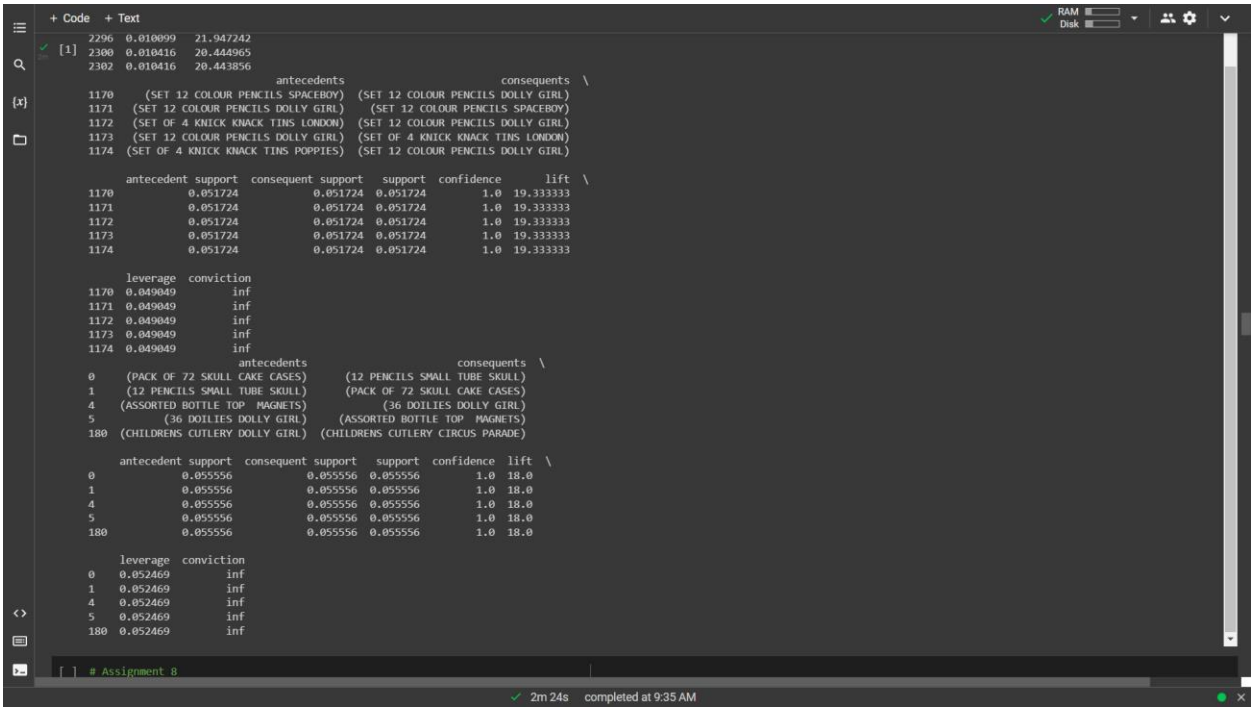
```
.sum().unstack().reset_index().fillna(0)
.set_index('InvoiceNo'))
# Transactions done in Portugal
basket_Por = (data[data['Country'] =="Portugal"]
.groupby(['InvoiceNo', 'Description'])['Quantity']
.sum().unstack().reset_index().fillna(0)
.set_index('InvoiceNo'))
basket_Sweden = (data[data['Country']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



```
+ Code + Text                                                                                    RAM Disk

[1] rules = rules.sort_values(["confidence", "lift"], ascending =[False, False])
    print(rules.head())

/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326: DtypeWarning: Columns (2) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
                                              antecedents  \
44                          (JUMBO BAG WOODLAND ANIMALS)
258  (RED TOADSTOOL LED NIGHT LIGHT, PLASTERS IN TI...
270  (RED TOADSTOOL LED NIGHT LIGHT, PLASTERS IN TI...
301  (SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...
302  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...

                        consequents  antecedent support  consequent support  \
44                        (POSTAGE)            0.076531            0.765306
258                       (POSTAGE)            0.051020            0.765306
270                       (POSTAGE)            0.053571            0.765306
301  (SET/6 RED SPOTTY PAPER PLATES)            0.102041            0.127551
302    (SET/6 RED SPOTTY PAPER CUPS)            0.102041            0.137755

      support  confidence      lift  leverage  conviction
44   0.076531       1.000  1.306667  0.017961         inf
258  0.051020       1.000  1.306667  0.011974         inf
270  0.053571       1.000  1.306667  0.012573         inf
301  0.099490       0.975  7.644000  0.086474   34.897959
302  0.099490       0.975  7.077778  0.085433   34.489796
                                        antecedents            consequents  \
117          (BEADED CRYSTAL HEART PINK ON STICK)        (DOTCOM POSTAGE)
2020  (SUKI  SHOULDER BAG, JAM MAKING SET PRINTED)        (DOTCOM POSTAGE)
2296        (HERB MARKER MINT, HERB MARKER THYME)  (HERB MARKER ROSEMARY)
2300   (HERB MARKER ROSEMARY, HERB MARKER PARSLEY)     (HERB MARKER THYME)
2302      (HERB MARKER THYME, HERB MARKER PARSLEY)  (HERB MARKER ROSEMARY)

      antecedent support  consequent support   support  confidence       lift  \
117             0.011035            0.037926  0.010767    0.975728  25.727250
2020            0.011624            0.037926  0.011196    0.963134  25.395168
2296            0.010714            0.012374  0.010231    0.955000  77.177229
2300            0.011088            0.012321  0.010553    0.951691  77.244192
2302            0.011088            0.012374  0.010553    0.951691  76.909802

      leverage  conviction
117   0.010349   39.637454
2020  0.010755   26.096261
2296  0.010099   21.947242
2300  0.010416   20.444965
2302  0.010416   20.443856

                                                                                  2m 24s  completed at 9:35 AM
```



```
+ Code + Text                                                                                    RAM Disk

      2296  0.010099   21.947242
[1]   2300  0.010416   20.444965
      2302  0.010416   20.443856
                                    antecedents                       consequents  \
1170     (SET 12 COLOUR PENCILS SPACEBOY)  (SET 12 COLOUR PENCILS DOLLY GIRL)
1171   (SET 12 COLOUR PENCILS DOLLY GIRL)    (SET 12 COLOUR PENCILS SPACEBOY)
1172   (SET OF 4 KNICK KNACK TINS LONDON)  (SET 12 COLOUR PENCILS DOLLY GIRL)
1173   (SET 12 COLOUR PENCILS DOLLY GIRL)  (SET OF 4 KNICK KNACK TINS LONDON)
1174  (SET OF 4 KNICK KNACK TINS POPPIES)  (SET 12 COLOUR PENCILS DOLLY GIRL)

      antecedent support  consequent support   support  confidence       lift  \
1170            0.051724            0.051724  0.051724         1.0  19.333333
1171            0.051724            0.051724  0.051724         1.0  19.333333
1172            0.051724            0.051724  0.051724         1.0  19.333333
1173            0.051724            0.051724  0.051724         1.0  19.333333
1174            0.051724            0.051724  0.051724         1.0  19.333333

      leverage  conviction
1170  0.049049         inf
1171  0.049049         inf
1172  0.049049         inf
1173  0.049049         inf
1174  0.049049         inf
                          antecedents                       consequents  \
0     (PACK OF 72 SKULL CAKE CASES)      (12 PENCILS SMALL TUBE SKULL)
1     (12 PENCILS SMALL TUBE SKULL)      (PACK OF 72 SKULL CAKE CASES)
4   (ASSORTED BOTTLE TOP  MAGNETS)           (36 DOILIES DOLLY GIRL)
5          (36 DOILIES DOLLY GIRL)     (ASSORTED BOTTLE TOP  MAGNETS)
180  (CHILDRENS CUTLERY DOLLY GIRL)  (CHILDRENS CUTLERY CIRCUS PARADE)

     antecedent support  consequent support   support  confidence  lift  \
0              0.055556            0.055556  0.055556         1.0  18.0
1              0.055556            0.055556  0.055556         1.0  18.0
4              0.055556            0.055556  0.055556         1.0  18.0
5              0.055556            0.055556  0.055556         1.0  18.0
180            0.055556            0.055556  0.055556         1.0  18.0

     leverage  conviction
0    0.052469         inf
1    0.052469         inf
4    0.052469         inf
5    0.052469         inf
180  0.052469         inf

[ ] # Assignment 8

                                                                                  2m 24s  completed at 9:35 AM
```

```python
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()
```

```
<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
```

✓ 10s completed at 9:36 AM

Assignment: 08

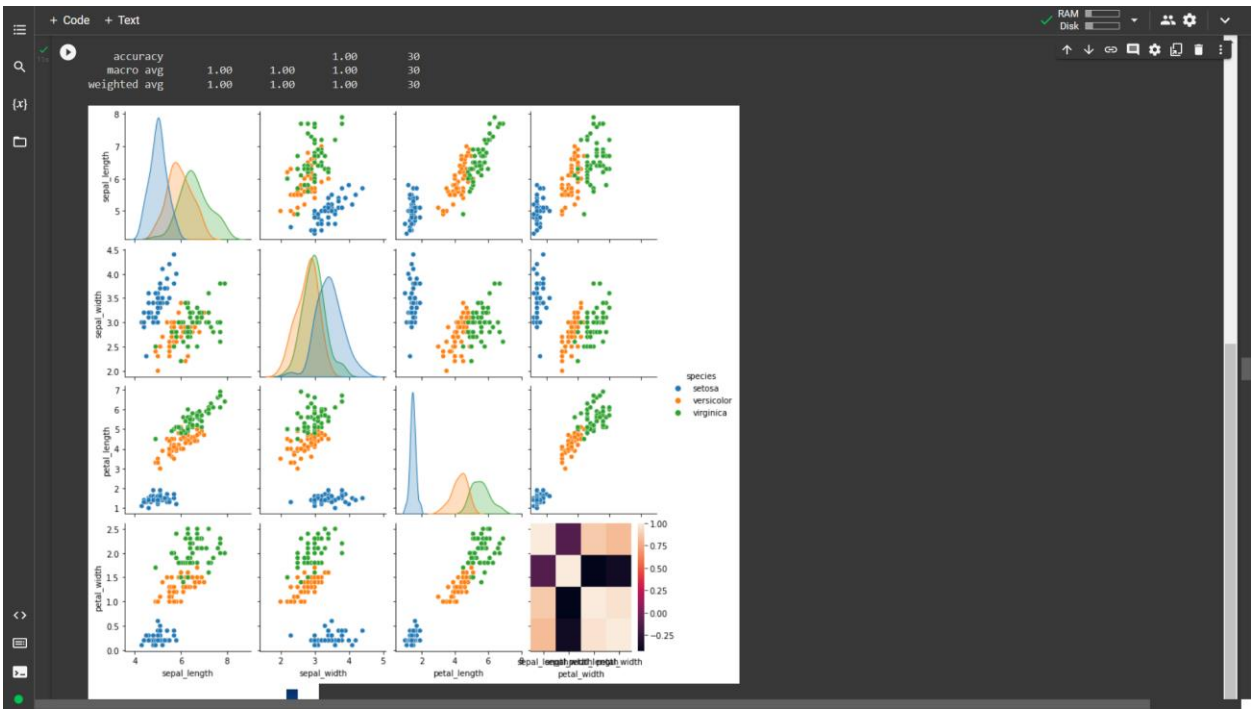
TITLE : Developing and implementing Decision Tree model on the dataset.
Class : MCA -II
Roll No. : MC21017
Date :

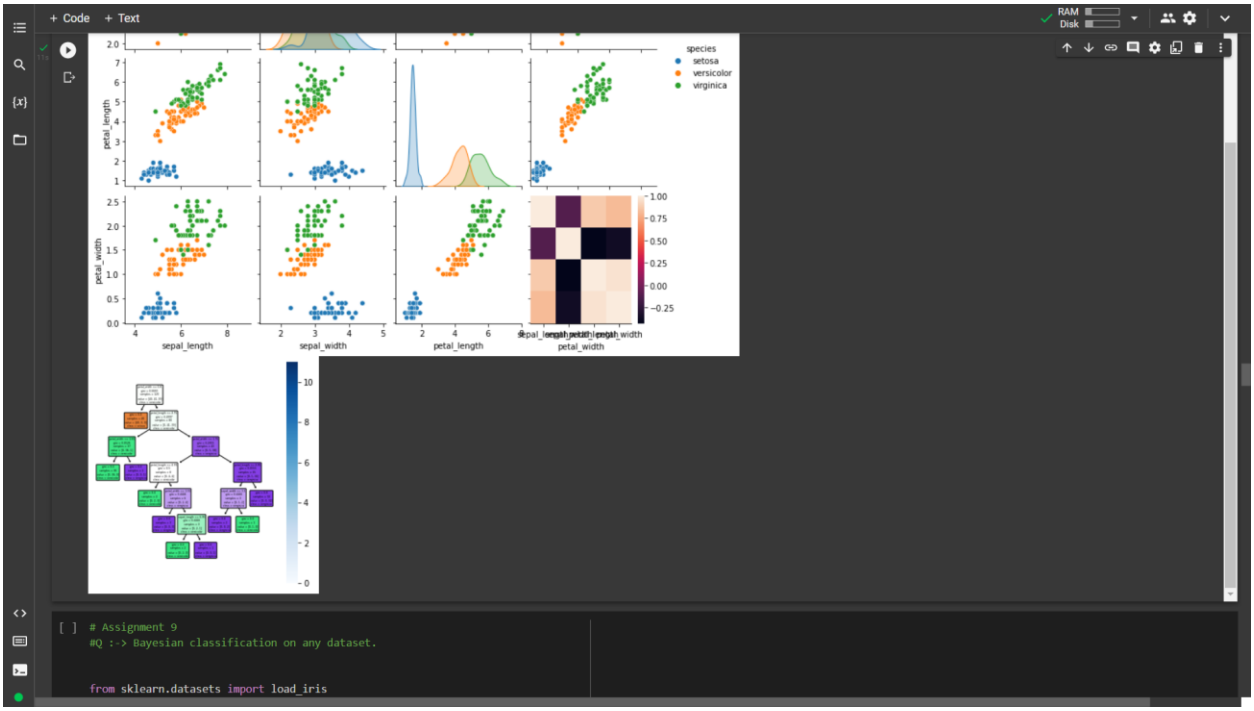
Program :

```
#Assignment 8
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)
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 :


+ Code + Text
RAM
Disk
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


+ Code + Text
RAM
Disk
species
setosa
versicolor
virginica
sepal_length
sepal_width
petal_length
petal_width
[ ] # Assignment 9
#Q :-> Bayesian classification on any dataset.
from sklearn.datasets import load_iris

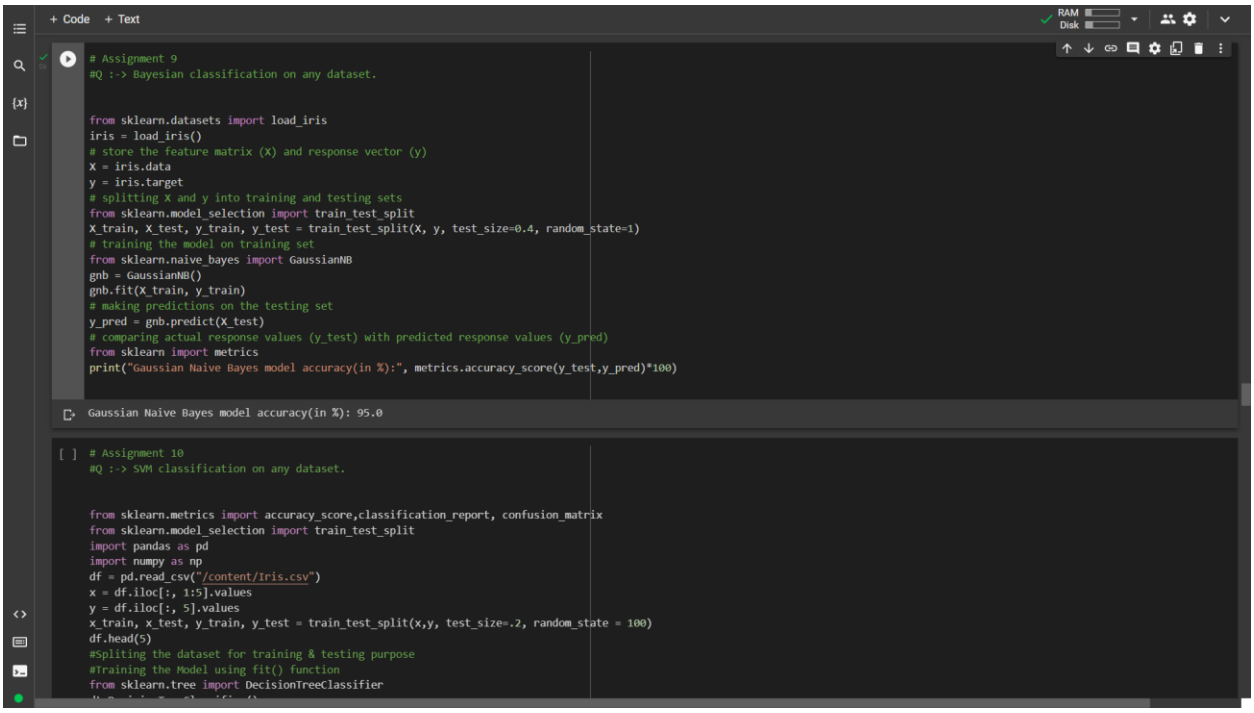
Assignment: 09

TITLE : Bayesian classification on any dataset.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 9


from sklearn.datasets import load_iris
iris = load_iris()
# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target
# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# making predictions on the testing set
y_pred = gnb.predict(X_test)
# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):",
metrics.accuracy_score(y_test,y_pred)*100)
```

Output :



```
+ Code + Text                                                    RAM Disk

# Assignment 9
#Q :-> Bayesian classification on any dataset.


from sklearn.datasets import load_iris
iris = load_iris()
# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target
# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# making predictions on the testing set
y_pred = gnb.predict(X_test)
# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
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
    #Q :-> SVM classification on any dataset.


    from sklearn.metrics import accuracy_score,classification_report, confusion_matrix
    from sklearn.model_selection import train_test_split
    import pandas as pd
    import numpy as np
    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
```

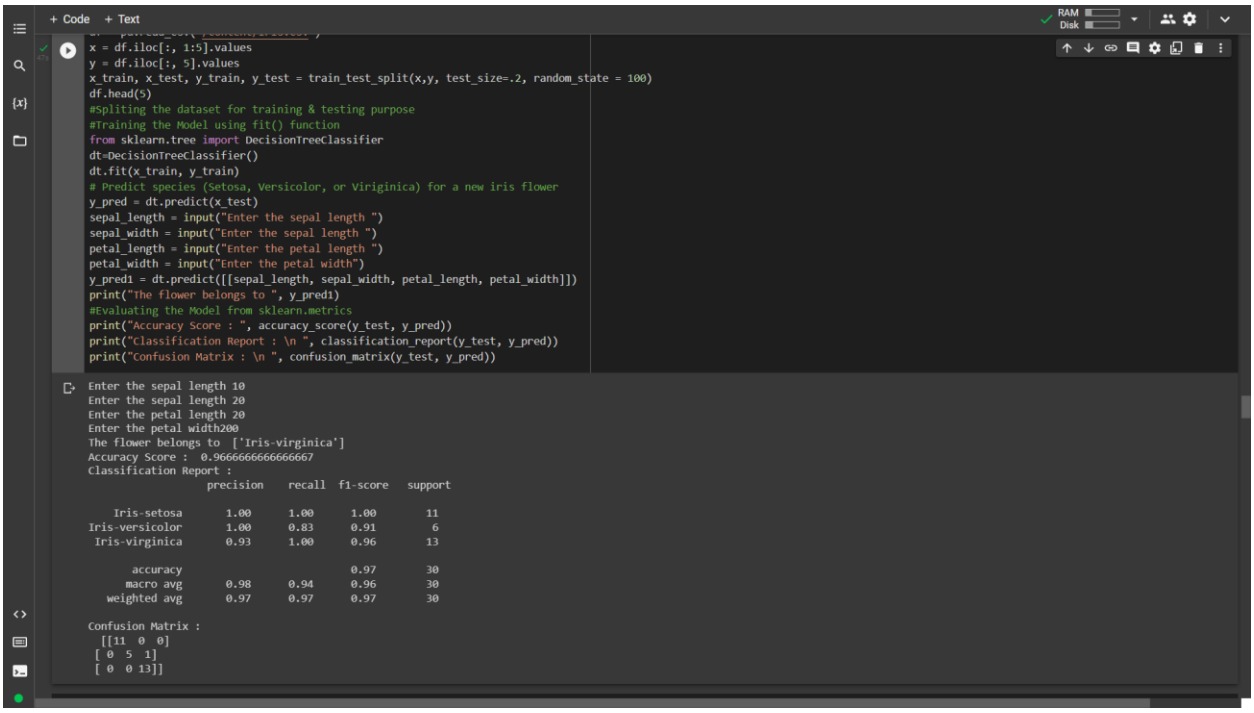
Assignment: 10

TITLE : SVM classification on any dataset.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 10
from sklearn.metrics import accuracy_score,classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
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 Viriginica) for a new iris flower
y_pred = dt.predict(x_test)
sepal_length = input("Enter the sepal length ")
sepal_width = input("Enter the sepal length ")
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 from sklearn.metrics
print("Accuracy Score : ", accuracy_score(y_test, y_pred))
```

Output :



```
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 Viriginica) for a new iris flower
y_pred = dt.predict(x_test)
sepal_length = input("Enter the sepal length ")
sepal_width = input("Enter the sepal length ")
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 from sklearn.metrics
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))
```

```
Enter the sepal length 10
Enter the sepal length 20
Enter the petal length 20
Enter the petal width200
The flower belongs to  ['Iris-virginica']
Accuracy Score :  0.9666666666666667
Classification Report :
                  precision    recall  f1-score   support

     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  0 13]]
```

Assignment: 11

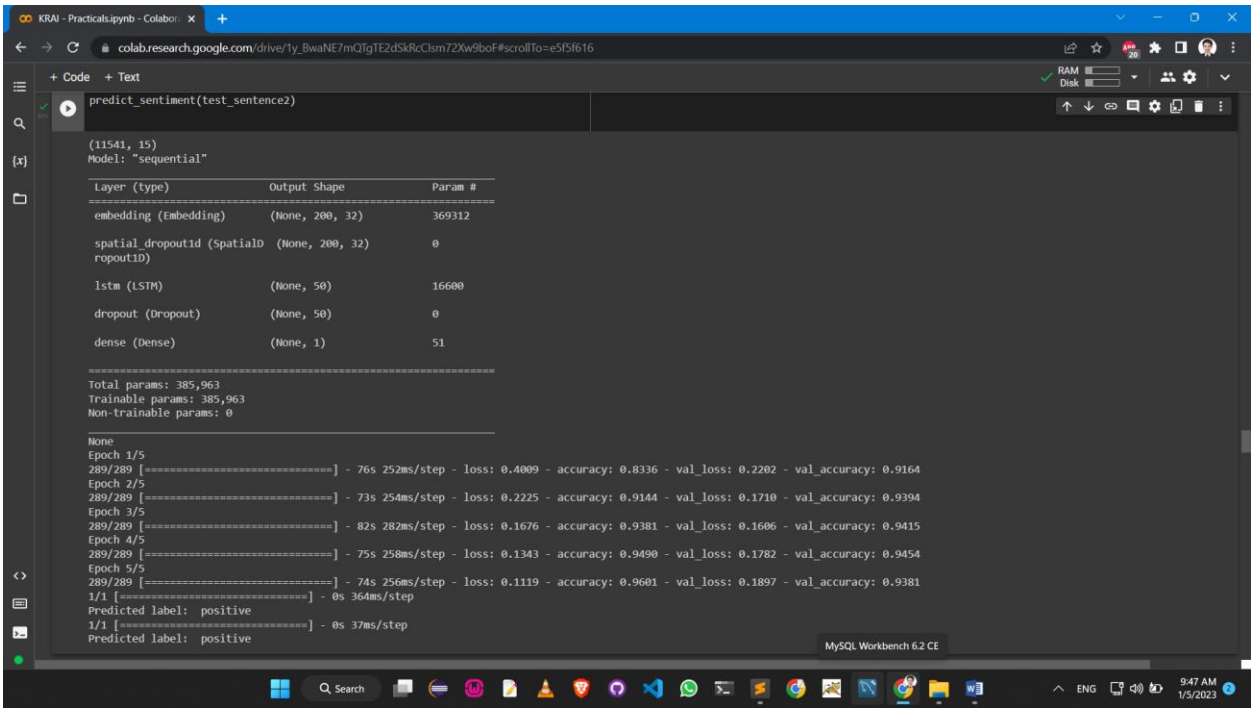
TITLE : Text Mining algorithms on an unstructured dataset.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 11
import pandas as pd
review_df = pd.read_csv("/content/tweets.csv")
# 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
# 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
```

```
vocab_size = len(list(encoded_docs))
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)

# 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):
    tokenizer = Tokenizer(num_words = 1000)
    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 :


KRAI - Practicals.ipynb - Colabor
colab.research.google.com/drive/1y_BwaNE7mQTgTE2dSkRcCIsm72XW9boF#scrollTo=e5f5f616
+ Code + Text
RAM
Disk
predict_sentiment(test_sentence2)
(11541, 15)
Model: "sequential"
Layer (type) Output Shape Param #
embedding (Embedding) (None, 200, 32) 369312
spatial_dropout1d (SpatialD ropout1D) (None, 200, 32) 0
lstm (LSTM) (None, 50) 16600
dropout (Dropout) (None, 50) 0
dense (Dense) (None, 1) 51
Total params: 385,963
Trainable params: 385,963
Non-trainable params: 0
None
Epoch 1/5
289/289 [==========] - 76s 252ms/step - loss: 0.4009 - accuracy: 0.8336 - val_loss: 0.2202 - val_accuracy: 0.9164
Epoch 2/5
289/289 [==========] - 73s 254ms/step - loss: 0.2225 - accuracy: 0.9144 - val_loss: 0.1710 - val_accuracy: 0.9394
Epoch 3/5
289/289 [==========] - 82s 282ms/step - loss: 0.1676 - accuracy: 0.9381 - val_loss: 0.1606 - val_accuracy: 0.9415
Epoch 4/5
289/289 [==========] - 75s 258ms/step - loss: 0.1343 - accuracy: 0.9490 - val_loss: 0.1782 - val_accuracy: 0.9454
Epoch 5/5
289/289 [==========] - 74s 256ms/step - loss: 0.1119 - accuracy: 0.9601 - val_loss: 0.1897 - val_accuracy: 0.9381
1/1 [==========] - 0s 364ms/step
Predicted label: positive
1/1 [==========] - 0s 37ms/step
Predicted label: positive
MySQL Workbench 6.2 CE
Search
ENG
9:47 AM
1/5/2023

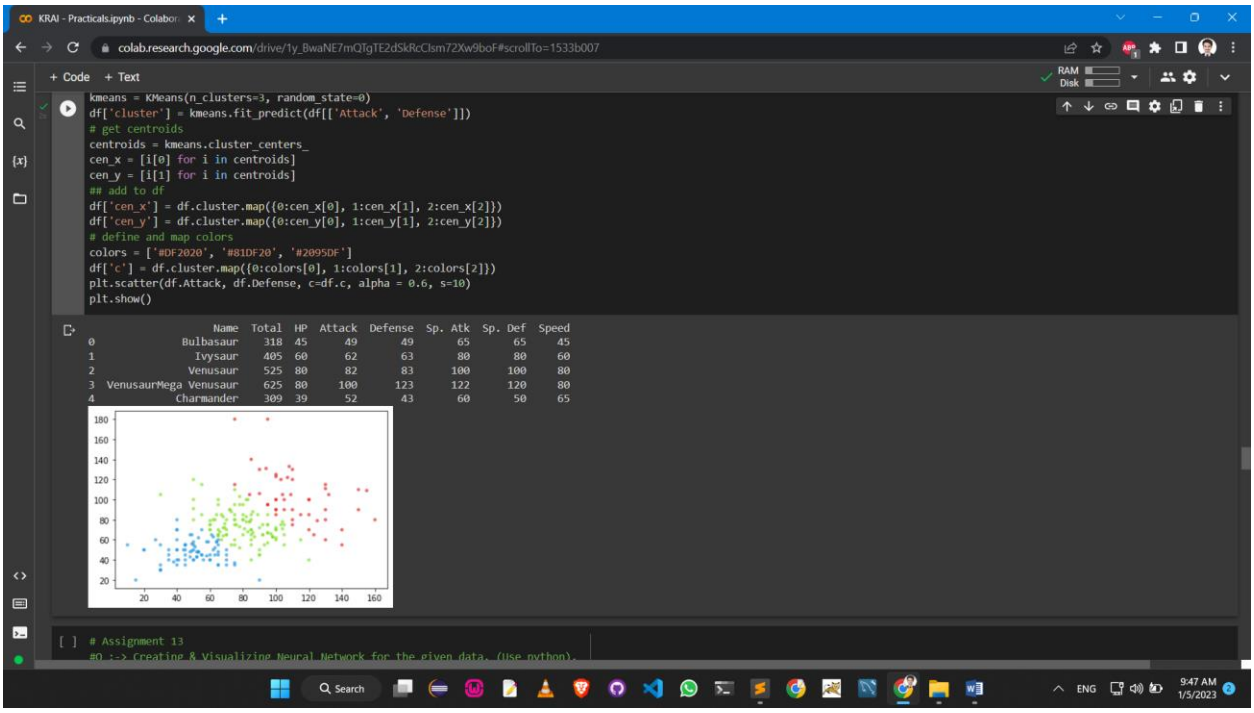
Assignment: 12

TITLE : Plot the cluster data using python visualizations.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 12
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]})
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 :


KRAI - Practicals.ipynb - Colabor
colab.research.google.com/drive/1y_BwaNE7mQTgTE2dSkRcClsm72Xw9boF#scrollTo=1533b007
+ Code + Text
RAM
Disk
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()
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
180
160
140
120
100
80
60
40
20
20 40 60 80 100 120 140 160
[ ] # Assignment 13
#Q :-> Creating & Visualizing Neural Network for the given data. (Use python).
Search
ENG
9:47 AM
1/5/2023

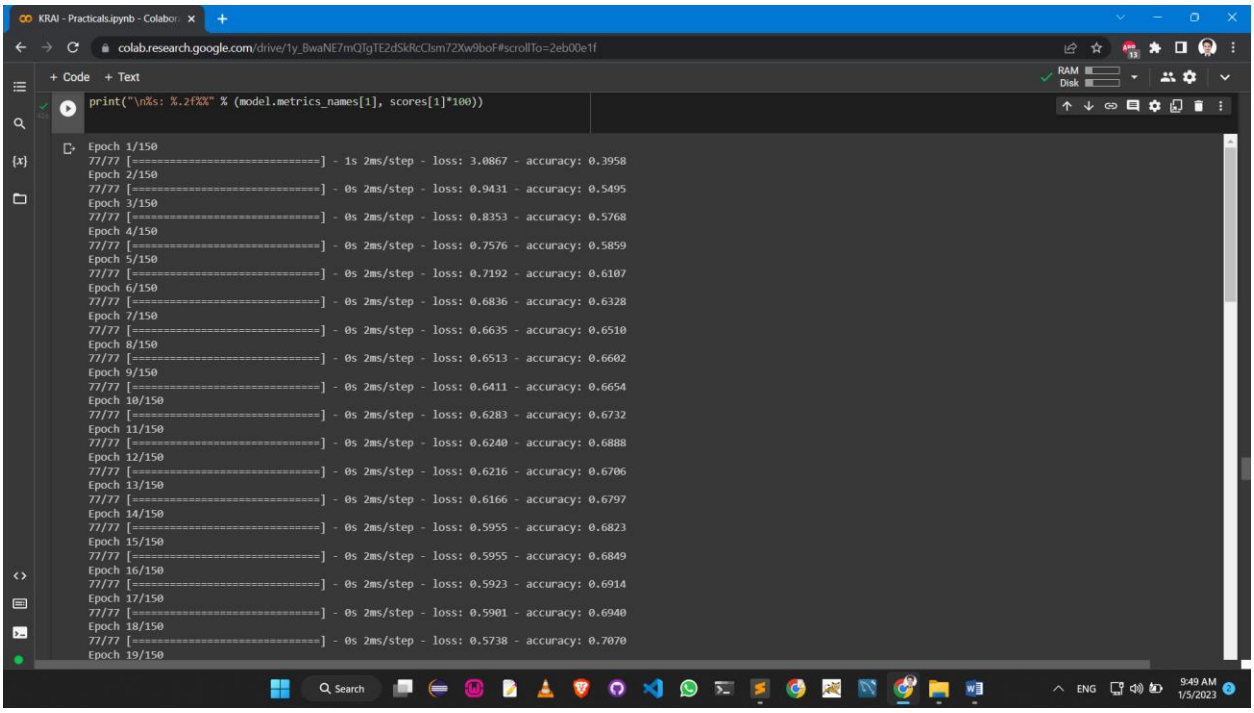
Assignment: 13

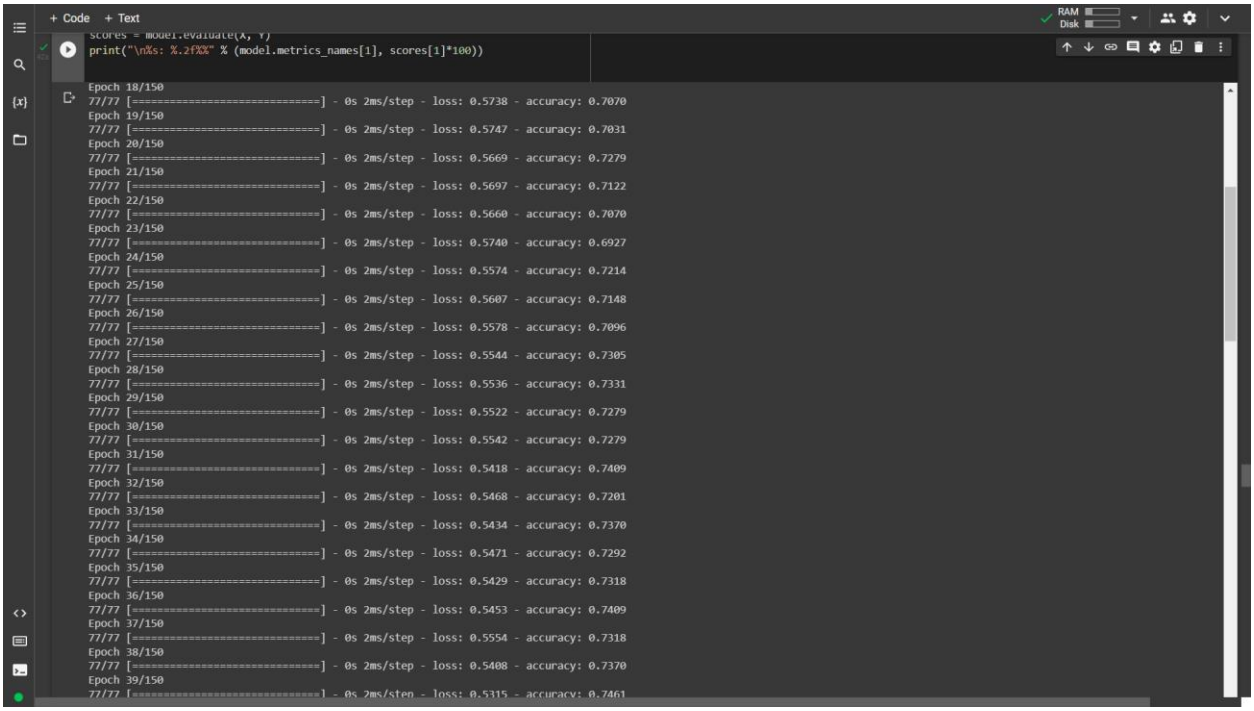
TITLE : Creating & Visualizing Neural Network for the given data. (Use python).
Class : MCA -II
Roll No. : MC21017
Date :

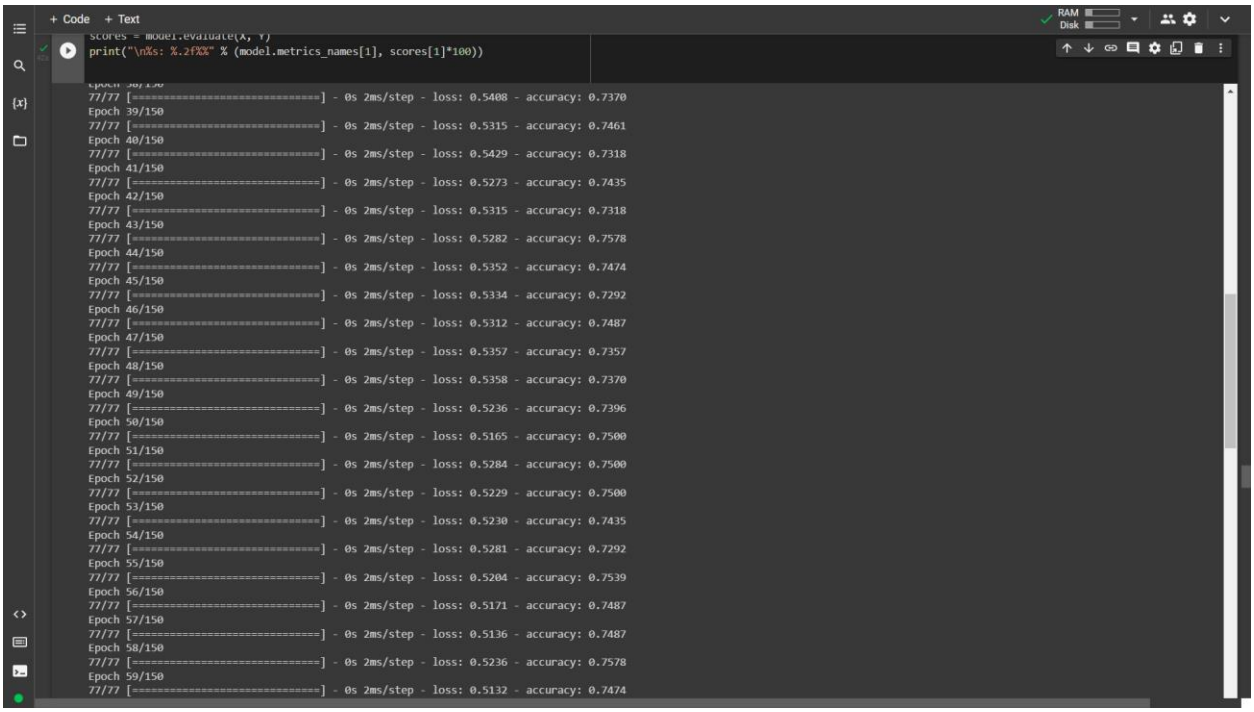
Program :

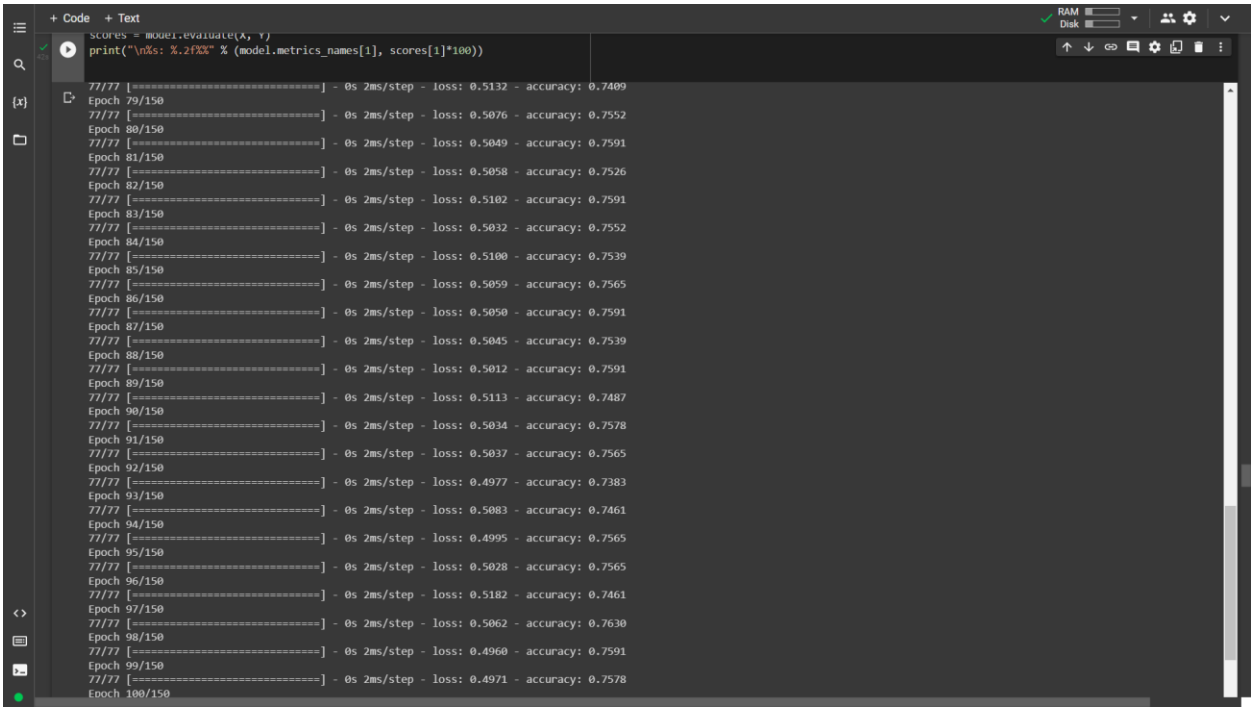
```
# Assignment 13
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
numpy.random.seed(7)
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

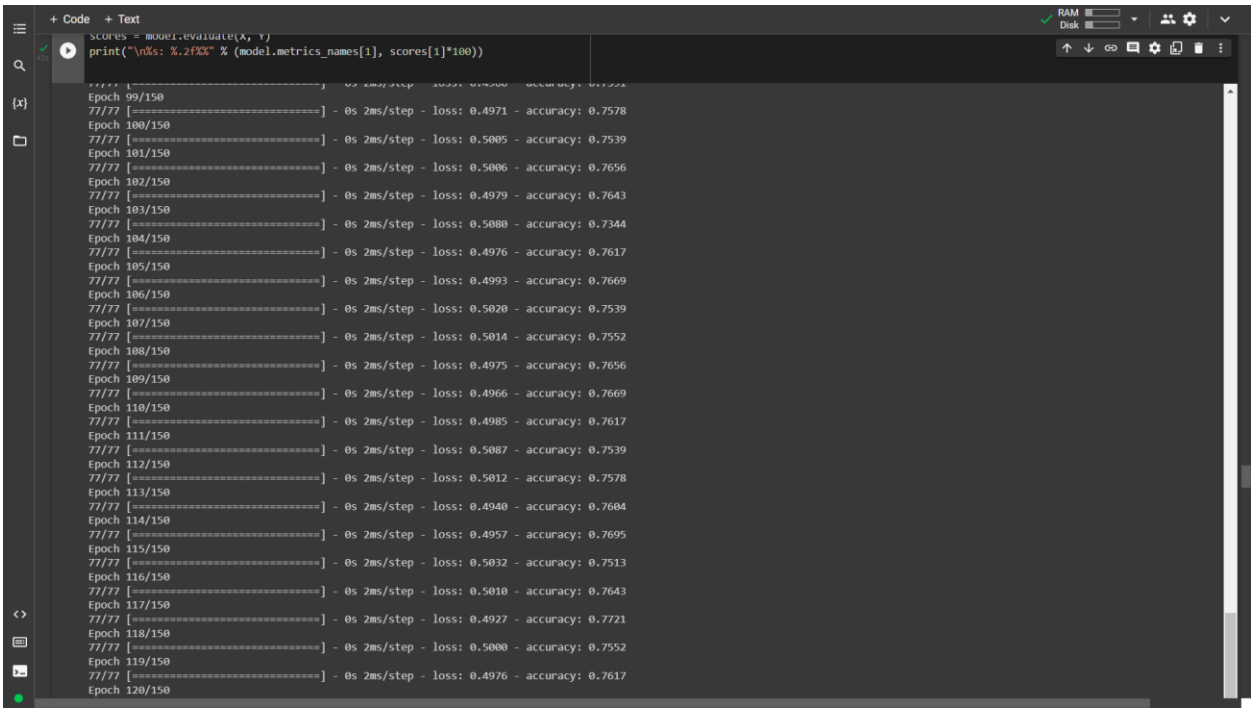
Output :

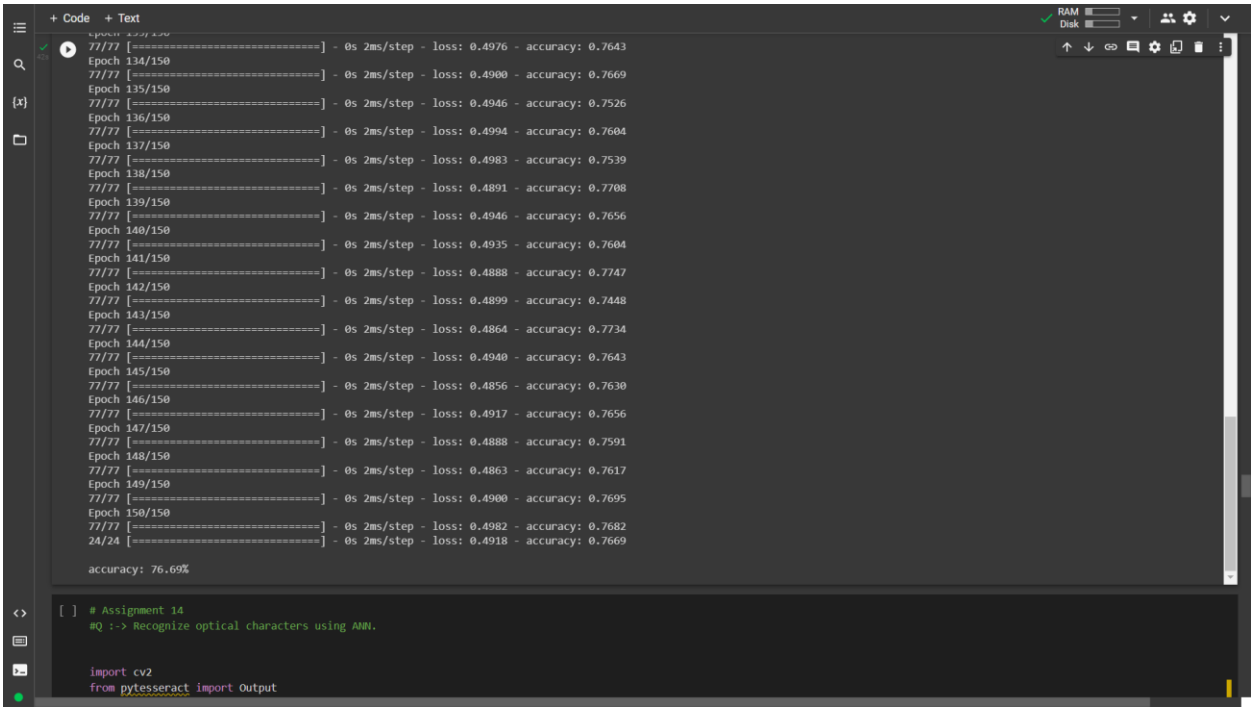

KRAI - Practicals.ipynb - Colabor
colab.research.google.com/drive/1y_BwaNE7mQTgTE2dSkRcCIsm72Xw9boF#scrollTo=2eb00e1f
+ Code + Text
RAM
Disk
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
Epoch 1/150
77/77 [==============================] - 1s 2ms/step - loss: 3.0867 - accuracy: 0.3958
Epoch 2/150
77/77 [==============================] - 0s 2ms/step - loss: 0.9431 - accuracy: 0.5495
Epoch 3/150
77/77 [==============================] - 0s 2ms/step - loss: 0.8353 - accuracy: 0.5768
Epoch 4/150
77/77 [==============================] - 0s 2ms/step - loss: 0.7576 - accuracy: 0.5859
Epoch 5/150
77/77 [==============================] - 0s 2ms/step - loss: 0.7192 - accuracy: 0.6107
Epoch 6/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6836 - accuracy: 0.6328
Epoch 7/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6635 - accuracy: 0.6510
Epoch 8/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6513 - accuracy: 0.6602
Epoch 9/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6411 - accuracy: 0.6654
Epoch 10/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6283 - accuracy: 0.6732
Epoch 11/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6240 - accuracy: 0.6888
Epoch 12/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6216 - accuracy: 0.6706
Epoch 13/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6166 - accuracy: 0.6797
Epoch 14/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5955 - accuracy: 0.6823
Epoch 15/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5955 - accuracy: 0.6849
Epoch 16/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5923 - accuracy: 0.6914
Epoch 17/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5901 - accuracy: 0.6940
Epoch 18/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5738 - accuracy: 0.7070
Epoch 19/150
Search
ENG
9:49 AM
1/5/2023


+ Code + Text
RAM
Disk
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
Epoch 18/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5738 - accuracy: 0.7070
Epoch 19/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5747 - accuracy: 0.7031
Epoch 20/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5669 - accuracy: 0.7279
Epoch 21/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5697 - accuracy: 0.7122
Epoch 22/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5660 - accuracy: 0.7070
Epoch 23/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5740 - accuracy: 0.6927
Epoch 24/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5574 - accuracy: 0.7214
Epoch 25/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5607 - accuracy: 0.7148
Epoch 26/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5578 - accuracy: 0.7096
Epoch 27/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5544 - accuracy: 0.7305
Epoch 28/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5536 - accuracy: 0.7331
Epoch 29/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5522 - accuracy: 0.7279
Epoch 30/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5542 - accuracy: 0.7279
Epoch 31/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5418 - accuracy: 0.7409
Epoch 32/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5468 - accuracy: 0.7201
Epoch 33/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5434 - accuracy: 0.7370
Epoch 34/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5471 - accuracy: 0.7292
Epoch 35/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5429 - accuracy: 0.7318
Epoch 36/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5453 - accuracy: 0.7409
Epoch 37/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5554 - accuracy: 0.7318
Epoch 38/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5408 - accuracy: 0.7370
Epoch 39/150


+ Code + Text
RAM
Disk
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
77/77 [==========] - 0s 2ms/step - loss: 0.5408 - accuracy: 0.7370
Epoch 39/150
77/77 [==========] - 0s 2ms/step - loss: 0.5315 - accuracy: 0.7461
Epoch 40/150
77/77 [==========] - 0s 2ms/step - loss: 0.5429 - accuracy: 0.7318
Epoch 41/150
77/77 [==========] - 0s 2ms/step - loss: 0.5273 - accuracy: 0.7435
Epoch 42/150
77/77 [==========] - 0s 2ms/step - loss: 0.5315 - accuracy: 0.7318
Epoch 43/150
77/77 [==========] - 0s 2ms/step - loss: 0.5282 - accuracy: 0.7578
Epoch 44/150
77/77 [==========] - 0s 2ms/step - loss: 0.5352 - accuracy: 0.7474
Epoch 45/150
77/77 [==========] - 0s 2ms/step - loss: 0.5334 - accuracy: 0.7292
Epoch 46/150
77/77 [==========] - 0s 2ms/step - loss: 0.5312 - accuracy: 0.7487
Epoch 47/150
77/77 [==========] - 0s 2ms/step - loss: 0.5357 - accuracy: 0.7357
Epoch 48/150
77/77 [==========] - 0s 2ms/step - loss: 0.5358 - accuracy: 0.7370
Epoch 49/150
77/77 [==========] - 0s 2ms/step - loss: 0.5236 - accuracy: 0.7396
Epoch 50/150
77/77 [==========] - 0s 2ms/step - loss: 0.5165 - accuracy: 0.7500
Epoch 51/150
77/77 [==========] - 0s 2ms/step - loss: 0.5284 - accuracy: 0.7500
Epoch 52/150
77/77 [==========] - 0s 2ms/step - loss: 0.5229 - accuracy: 0.7500
Epoch 53/150
77/77 [==========] - 0s 2ms/step - loss: 0.5230 - accuracy: 0.7435
Epoch 54/150
77/77 [==========] - 0s 2ms/step - loss: 0.5281 - accuracy: 0.7292
Epoch 55/150
77/77 [==========] - 0s 2ms/step - loss: 0.5204 - accuracy: 0.7539
Epoch 56/150
77/77 [==========] - 0s 2ms/step - loss: 0.5171 - accuracy: 0.7487
Epoch 57/150
77/77 [==========] - 0s 2ms/step - loss: 0.5136 - accuracy: 0.7487
Epoch 58/150
77/77 [==========] - 0s 2ms/step - loss: 0.5236 - accuracy: 0.7578
Epoch 59/150
77/77 [==========] - 0s 2ms/step - loss: 0.5132 - accuracy: 0.7474


+ Code + Text
RAM
Disk
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
77/77 [==========] - 0s 2ms/step - loss: 0.5132 - accuracy: 0.7409
Epoch 79/150
77/77 [==========] - 0s 2ms/step - loss: 0.5076 - accuracy: 0.7552
Epoch 80/150
77/77 [==========] - 0s 2ms/step - loss: 0.5049 - accuracy: 0.7591
Epoch 81/150
77/77 [==========] - 0s 2ms/step - loss: 0.5058 - accuracy: 0.7526
Epoch 82/150
77/77 [==========] - 0s 2ms/step - loss: 0.5102 - accuracy: 0.7591
Epoch 83/150
77/77 [==========] - 0s 2ms/step - loss: 0.5032 - accuracy: 0.7552
Epoch 84/150
77/77 [==========] - 0s 2ms/step - loss: 0.5100 - accuracy: 0.7539
Epoch 85/150
77/77 [==========] - 0s 2ms/step - loss: 0.5059 - accuracy: 0.7565
Epoch 86/150
77/77 [==========] - 0s 2ms/step - loss: 0.5050 - accuracy: 0.7591
Epoch 87/150
77/77 [==========] - 0s 2ms/step - loss: 0.5045 - accuracy: 0.7539
Epoch 88/150
77/77 [==========] - 0s 2ms/step - loss: 0.5012 - accuracy: 0.7591
Epoch 89/150
77/77 [==========] - 0s 2ms/step - loss: 0.5113 - accuracy: 0.7487
Epoch 90/150
77/77 [==========] - 0s 2ms/step - loss: 0.5034 - accuracy: 0.7578
Epoch 91/150
77/77 [==========] - 0s 2ms/step - loss: 0.5037 - accuracy: 0.7565
Epoch 92/150
77/77 [==========] - 0s 2ms/step - loss: 0.4977 - accuracy: 0.7383
Epoch 93/150
77/77 [==========] - 0s 2ms/step - loss: 0.5083 - accuracy: 0.7461
Epoch 94/150
77/77 [==========] - 0s 2ms/step - loss: 0.4995 - accuracy: 0.7565
Epoch 95/150
77/77 [==========] - 0s 2ms/step - loss: 0.5028 - accuracy: 0.7565
Epoch 96/150
77/77 [==========] - 0s 2ms/step - loss: 0.5182 - accuracy: 0.7461
Epoch 97/150
77/77 [==========] - 0s 2ms/step - loss: 0.5062 - accuracy: 0.7630
Epoch 98/150
77/77 [==========] - 0s 2ms/step - loss: 0.4960 - accuracy: 0.7591
Epoch 99/150
77/77 [==========] - 0s 2ms/step - loss: 0.4971 - accuracy: 0.7578
Epoch 100/150


+ Code + Text
RAM
Disk
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
Epoch 99/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4971 - accuracy: 0.7578
Epoch 100/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5005 - accuracy: 0.7539
Epoch 101/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5006 - accuracy: 0.7656
Epoch 102/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4979 - accuracy: 0.7643
Epoch 103/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5080 - accuracy: 0.7344
Epoch 104/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4976 - accuracy: 0.7617
Epoch 105/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4993 - accuracy: 0.7669
Epoch 106/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5020 - accuracy: 0.7539
Epoch 107/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5014 - accuracy: 0.7552
Epoch 108/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4975 - accuracy: 0.7656
Epoch 109/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4966 - accuracy: 0.7669
Epoch 110/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4985 - accuracy: 0.7617
Epoch 111/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5087 - accuracy: 0.7539
Epoch 112/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5012 - accuracy: 0.7578
Epoch 113/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4940 - accuracy: 0.7604
Epoch 114/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4957 - accuracy: 0.7695
Epoch 115/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5032 - accuracy: 0.7513
Epoch 116/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5010 - accuracy: 0.7643
Epoch 117/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4927 - accuracy: 0.7721
Epoch 118/150
77/77 [==============================] - 0s 2ms/step - loss: 0.5000 - accuracy: 0.7552
Epoch 119/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4976 - accuracy: 0.7617
Epoch 120/150


+ Code + Text
RAM
Disk
77/77 [==============================] - 0s 2ms/step - loss: 0.4976 - accuracy: 0.7643
Epoch 134/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4900 - accuracy: 0.7669
Epoch 135/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4946 - accuracy: 0.7526
Epoch 136/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4994 - accuracy: 0.7604
Epoch 137/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4983 - accuracy: 0.7539
Epoch 138/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4891 - accuracy: 0.7708
Epoch 139/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4946 - accuracy: 0.7656
Epoch 140/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4935 - accuracy: 0.7604
Epoch 141/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4888 - accuracy: 0.7747
Epoch 142/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4899 - accuracy: 0.7448
Epoch 143/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4864 - accuracy: 0.7734
Epoch 144/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4940 - accuracy: 0.7643
Epoch 145/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4856 - accuracy: 0.7630
Epoch 146/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4917 - accuracy: 0.7656
Epoch 147/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4888 - accuracy: 0.7591
Epoch 148/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4863 - accuracy: 0.7617
Epoch 149/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4900 - accuracy: 0.7695
Epoch 150/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4982 - accuracy: 0.7682
24/24 [==============================] - 0s 2ms/step - loss: 0.4918 - accuracy: 0.7669
accuracy: 76.69%
[ ] # Assignment 14
#Q :-> Recognize optical characters using ANN.
import cv2
from pytesseract import Output

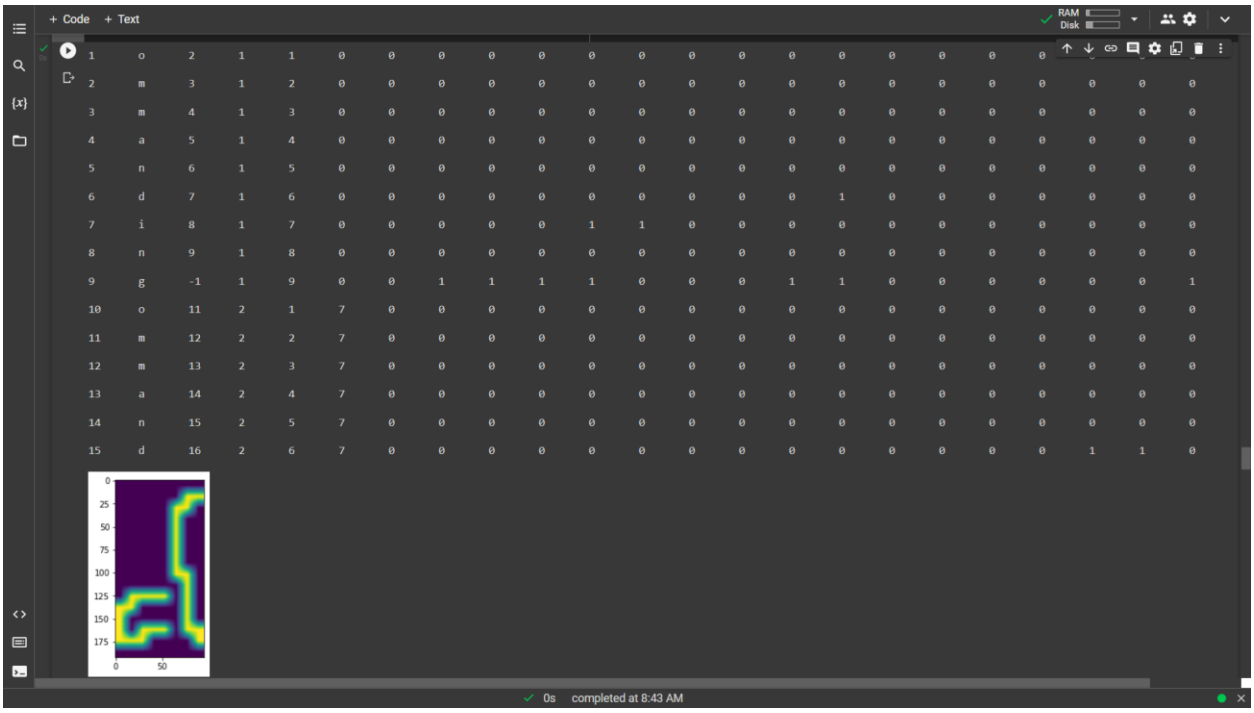
Assignment: 14

TITLE : Recognize optical characters using ANN.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 14
import os
import sys
import cv2
import numpy as np
import matplotlib.pyplot as plt

input_f = '/content/letter.data'
img_resize_factor = 12
start, end = 6, -1
height, width = 16, 8
with open(input_f, 'r') as f:
    for line in f.readlines():
        data = np.array([255*float(x) for x in line.split('\t')[start:end]])
        img = np.reshape(data, (height, width))
        img_scaled = cv2.resize(img, None, fx=img_resize_factor, fy=img_resize_factor)
        print(line)
        #cv2.imshow('Img', img_scaled)
        plt.imshow(img_scaled)

        c = cv2.waitKey()
        if c == 27:
            break
```

Output :



Assignment: 15

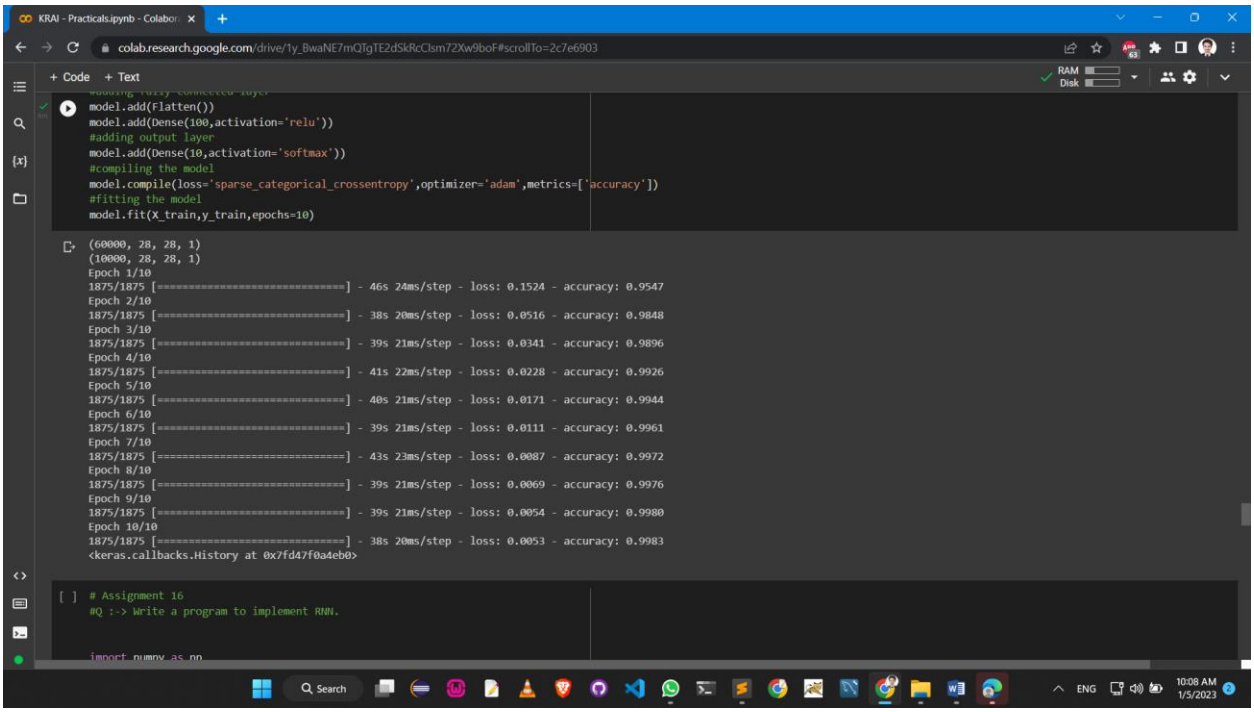
TITLE : Write a program to implement CNN.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 15
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=['accuracy'])
#fitting the model
model.fit(X_train,y_train,epochs=10)
```

Output :



Assignment: 16

TITLE : Write a program to implement RNN.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 16
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
```

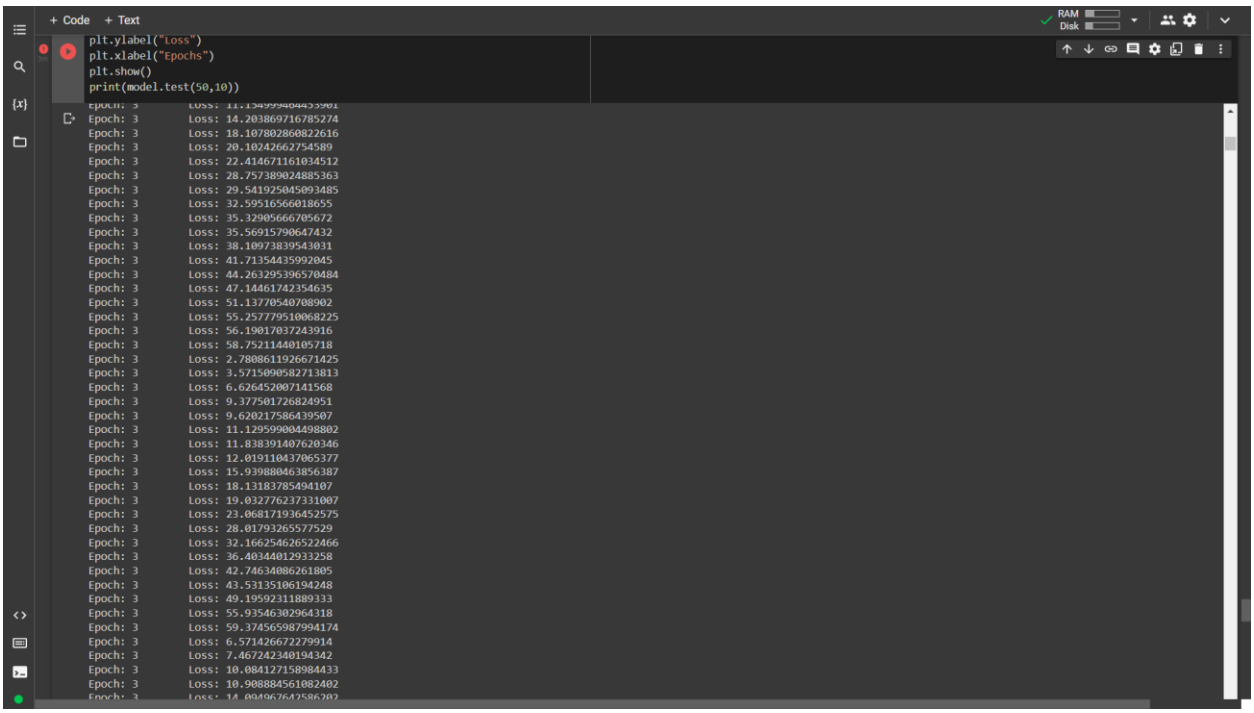
```
    def _softmax(self, x):
        # max value is substracted for numerical stability
        # # https://stats.stackexchange.com/a/338293
        e_x = np.exp(x - np.max(x))
        return e_x / np.sum(e_x)

    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('/content/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)
print(dir(model))
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 :


+ Code
+ Text
RAM
Disk
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.show()
print(model.test(50,10))
Epoch: 3 Loss: 14.203869716785274
Epoch: 3 Loss: 18.107802860822616
Epoch: 3 Loss: 20.10242662754589
Epoch: 3 Loss: 22.414671161034512
Epoch: 3 Loss: 28.757389024885363
Epoch: 3 Loss: 29.541925045093485
Epoch: 3 Loss: 32.59516566018655
Epoch: 3 Loss: 35.32905666705672
Epoch: 3 Loss: 35.56915790647432
Epoch: 3 Loss: 38.10973839543031
Epoch: 3 Loss: 41.71354435992045
Epoch: 3 Loss: 44.263295396570484
Epoch: 3 Loss: 47.14461742354635
Epoch: 3 Loss: 51.13770540708902
Epoch: 3 Loss: 55.257779510068225
Epoch: 3 Loss: 56.19017037243916
Epoch: 3 Loss: 58.75211440105718
Epoch: 3 Loss: 2.7808611926671425
Epoch: 3 Loss: 3.5715090582713813
Epoch: 3 Loss: 6.626452007141568
Epoch: 3 Loss: 9.377501726824951
Epoch: 3 Loss: 9.620217586439507
Epoch: 3 Loss: 11.129599004498802
Epoch: 3 Loss: 11.838391407620346
Epoch: 3 Loss: 12.019110437065377
Epoch: 3 Loss: 15.939880463856387
Epoch: 3 Loss: 18.13183785494107
Epoch: 3 Loss: 19.032776237331007
Epoch: 3 Loss: 23.068171936452575
Epoch: 3 Loss: 28.01793265577529
Epoch: 3 Loss: 32.166254626522466
Epoch: 3 Loss: 36.40344012933258
Epoch: 3 Loss: 42.74634086261805
Epoch: 3 Loss: 43.53135106194248
Epoch: 3 Loss: 49.19592311889333
Epoch: 3 Loss: 55.93546302964318
Epoch: 3 Loss: 59.374565987994174
Epoch: 3 Loss: 6.571426672279914
Epoch: 3 Loss: 7.46724234019434 2
Epoch: 3 Loss: 10.084127158984433
Epoch: 3 Loss: 10.908884561082402

Assignment: 17

TITLE : Write a program to implement GAN.
Class : MCA -II
Roll No. : MC21017
Date :

Program :

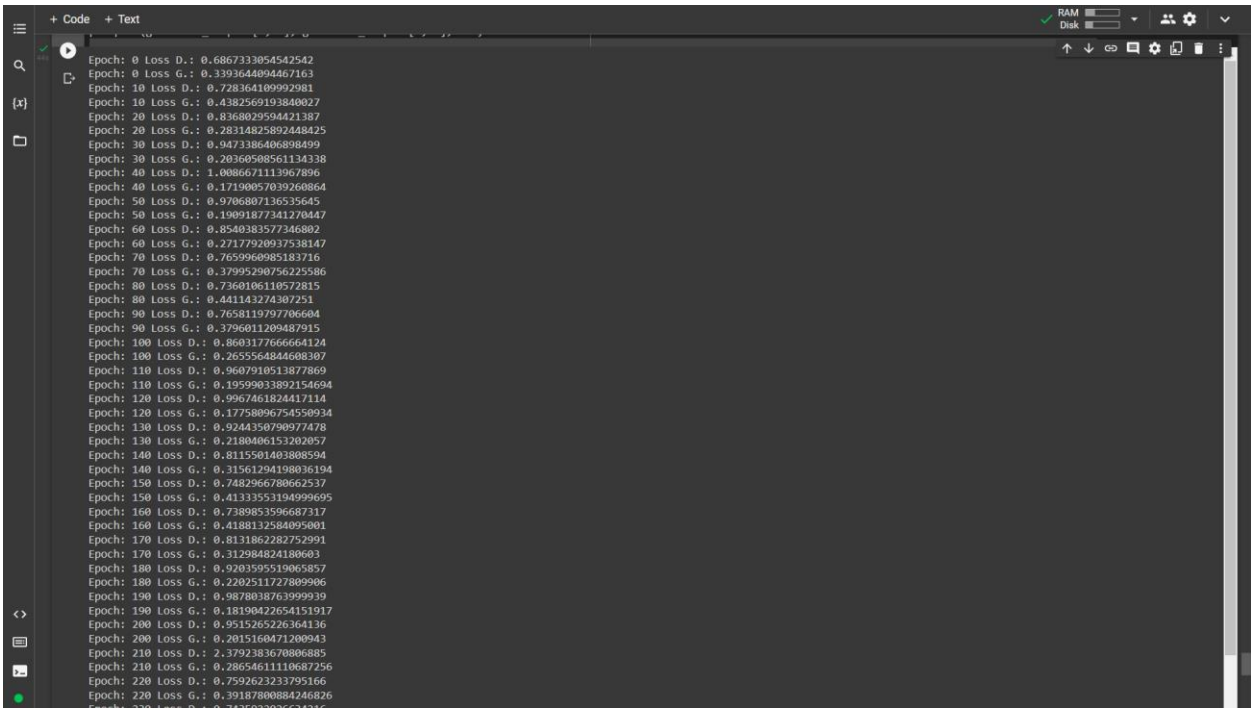
```
# Assignment 17
import torch
from torch import nn

import math
import matplotlib.pyplot as plt

torch.manual_seed(111)
train_data_length = 1024
train_data = torch.zeros((train_data_length, 2))
train_data[:, 0] = 2 * math.pi * torch.rand(train_data_length)
train_data[:, 1] = torch.sin(train_data[:, 0])
train_labels = torch.zeros(train_data_length)
train_set = [(train_data[i], train_labels[i]) for i in range(train_data_length)]
plt.plot(train_data[:, 0], train_data[:, 1], ".")

# 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):
```

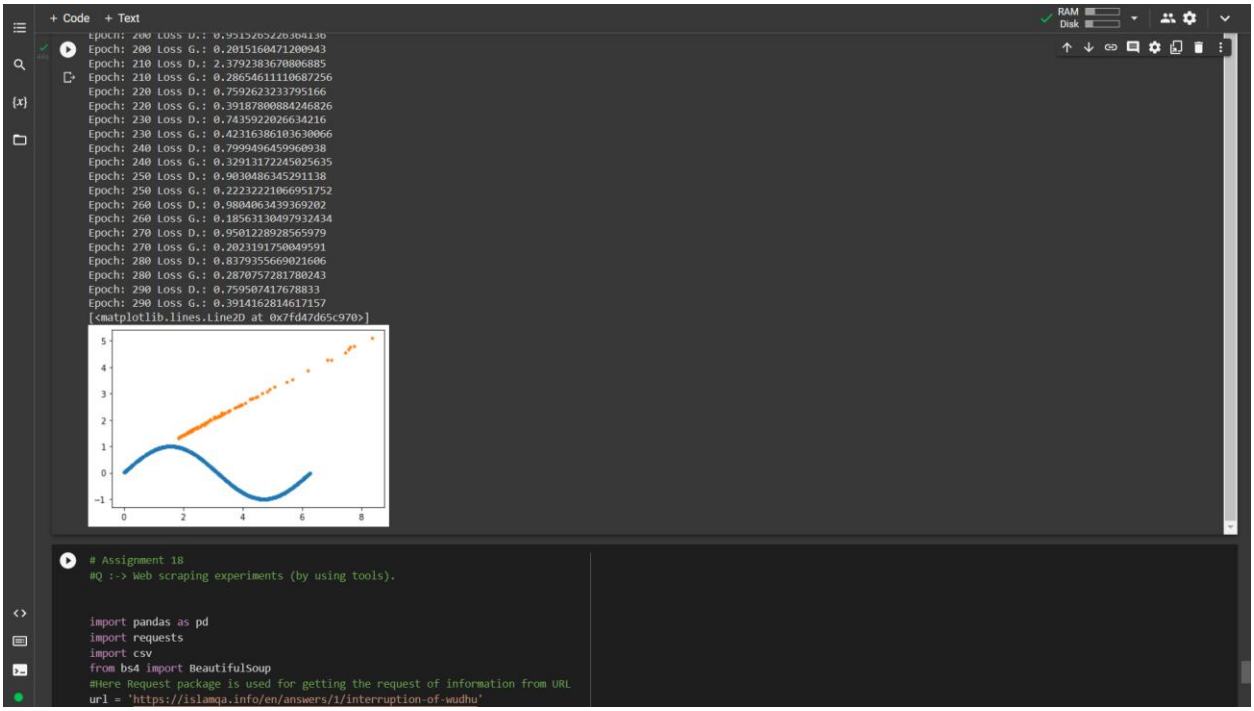
```
        super().__init__()
        self.model = nn.Sequential(nn.Linear(2, 256),
nn.ReLU(),nn.Dropout(0.3),nn.Linear(256,128),nn.ReLU(),nn.Dropout(0.3),nn.Linear(12
8, 64),nn.ReLU(),nn.Dropout(0.3),nn.Linear(64,1),nn.Sigmoid(),)
    def forward(self, x):
        output = self.model(x)
        return 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 :


+ Code + Text
RAM
Disk
Epoch: 0 Loss D.: 0.6867333054542542
Epoch: 0 Loss G.: 0.3393644094467163
Epoch: 10 Loss D.: 0.728364109992981
Epoch: 10 Loss G.: 0.4382569193840027
Epoch: 20 Loss D.: 0.8368029594421387
Epoch: 20 Loss G.: 0.28314825892448425
Epoch: 30 Loss D.: 0.9473386406898499
Epoch: 30 Loss G.: 0.20360508561134338
Epoch: 40 Loss D.: 1.0086671113967896
Epoch: 40 Loss G.: 0.17190057039260864
Epoch: 50 Loss D.: 0.9706807136535645
Epoch: 50 Loss G.: 0.19091877341270447
Epoch: 60 Loss D.: 0.8540383577346802
Epoch: 60 Loss G.: 0.27177920937538147
Epoch: 70 Loss D.: 0.7659960985183716
Epoch: 70 Loss G.: 0.37995290756225586
Epoch: 80 Loss D.: 0.7360106110572815
Epoch: 80 Loss G.: 0.441143274307251
Epoch: 90 Loss D.: 0.7658119797706604
Epoch: 90 Loss G.: 0.3796011209487915
Epoch: 100 Loss D.: 0.8603177666664124
Epoch: 100 Loss G.: 0.2655564844608307
Epoch: 110 Loss D.: 0.9607910513877869
Epoch: 110 Loss G.: 0.19599033892154694
Epoch: 120 Loss D.: 0.9967461824417114
Epoch: 120 Loss G.: 0.17758096754550934
Epoch: 130 Loss D.: 0.9244350790977478
Epoch: 130 Loss G.: 0.2180406153202057
Epoch: 140 Loss D.: 0.8115501403808594
Epoch: 140 Loss G.: 0.3156129419803619
Epoch: 150 Loss D.: 0.748296678066253
Epoch: 150 Loss G.: 0.41333553194999695
Epoch: 160 Loss D.: 0.7389853596687317
Epoch: 160 Loss G.: 0.4188132584095001
Epoch: 170 Loss D.: 0.8131862282752991
Epoch: 170 Loss G.: 0.312984824180603
Epoch: 180 Loss D.: 0.9203595519065857
Epoch: 180 Loss G.: 0.2202511727809906
Epoch: 190 Loss D.: 0.9878038763999939
Epoch: 190 Loss G.: 0.18190422654151917
Epoch: 200 Loss D.: 0.9515265226364136
Epoch: 200 Loss G.: 0.2015160471200943
Epoch: 210 Loss D.: 2.3792383670806885
Epoch: 210 Loss G.: 0.2865461111068725 6
Epoch: 220 Loss D.: 0.7592623233795166
Epoch: 220 Loss G.: 0.39187800884246826


+ Code + Text
RAM
Disk
Epoch: 200 Loss G.: 0.2015160471200943
Epoch: 210 Loss D.: 2.3792383670806885
Epoch: 210 Loss G.: 0.2865461111068725 6
Epoch: 220 Loss D.: 0.7592623233795166
Epoch: 220 Loss G.: 0.39187800884246826
Epoch: 230 Loss D.: 0.7435922026634216
Epoch: 230 Loss G.: 0.42316386103630066
Epoch: 240 Loss D.: 0.7999496459960938
Epoch: 240 Loss G.: 0.32913172245025635
Epoch: 250 Loss D.: 0.9030486345291138
Epoch: 250 Loss G.: 0.22232221066951752
Epoch: 260 Loss D.: 0.9804063439369202
Epoch: 260 Loss G.: 0.18563130497932434
Epoch: 270 Loss D.: 0.9501228928565979
Epoch: 270 Loss G.: 0.2023191750049591
Epoch: 280 Loss D.: 0.8379355669021606
Epoch: 280 Loss G.: 0.287075728178024 3
Epoch: 290 Loss D.: 0.759507417678833
Epoch: 290 Loss G.: 0.3914162814617157
[<matplotlib.lines.Line2D at 0x7fd47d65c970>]
Assignment 18
#Q :-> Web scraping experiments (by using tools).
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'

Assignment: 18

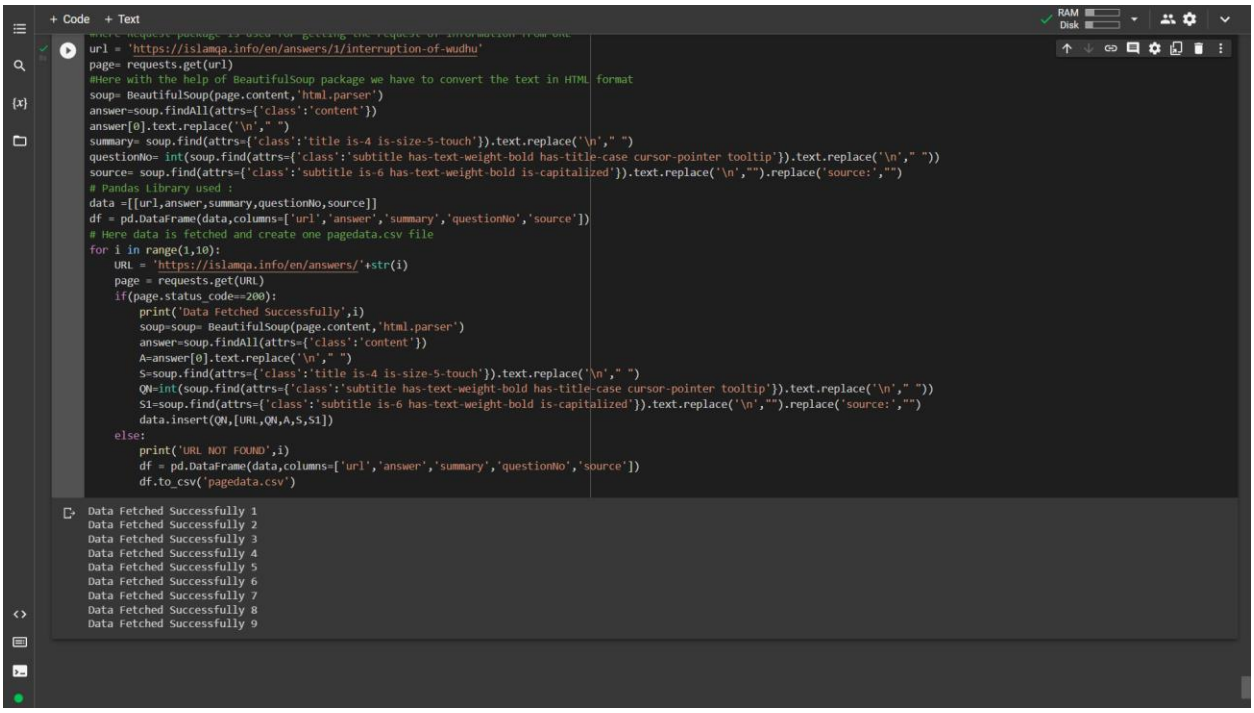
TITLE : Web scraping experiments (by using tools).
Class : MCA -II
Roll No. : MC21017
Date :

Program :

```
# Assignment 18
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)
#Here with the help of BeautifulSoup package we have to convert the text in HTML
format
soup= BeautifulSoup(page.content,'html.parser')
answer=soup.findAll(attrs={'class':'content'})
answer[0].text.replace('\n'," ")
summary= soup.find(attrs={'class':'title is-4 is-size-5-touch'}).text.replace('\n'," ")
questionNo= int(soup.find(attrs={'class':'subtitle has-text-weight-bold has-title-case
cursor-pointer tooltip'}).text.replace('\n'," "))
source= soup.find(attrs={'class':'subtitle is-6 has-text-weight-bold is-
capitalized'}).text.replace('\n',"").replace('source:',"")
# Pandas Library used :
data =[[url,answer,summary,questionNo,source]]
df = pd.DataFrame(data,columns=['url','answer','summary','questionNo','source'])
# 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 :



```
+ Code + Text
RAM
Disk
url = 'https://islamqa.info/en/answers/1/interruption-of-wudhu'
page= requests.get(url)
#Here with the help of BeautifulSoup package we have to convert the text in HTML format
soup= BeautifulSoup(page.content,'html.parser')
answer=soup.findAll(attrs={'class':'content'})
answer[0].text.replace('\n'," ")
summary= soup.find(attrs={'class':'title is-4 is-size-5-touch'}).text.replace('\n'," ")
questionNo= int(soup.find(attrs={'class':'subtitle has-text-weight-bold has-title-case cursor-pointer tooltip'}).text.replace('\n'," "))
source= soup.find(attrs={'class':'subtitle is-6 has-text-weight-bold is-capitalized'}).text.replace('\n',"").replace('source:',"")
# Pandas Library used :
data =[[url,answer,summary,questionNo,source]]
df = pd.DataFrame(data,columns=['url','answer','summary','questionNo','source'])
# 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')

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
```