

PRACTICAL JOURNAL

ON

IT31L: Practical (ML/DL)

SUBMITTED BY

Aditya Sanjeev Kulkarni

Roll No: 901

SAVITRIBAI PHULE PUNE UNIVERSITY (SPPU)
IN PARTIAL FULFILLMENT OF DEGREE FOR MASTER OF
COMPUTER APPLICATION



DR. D. Y. PATIL UNITECH SOCIETY'S
DR. D. Y. PATIL INSTITUTE OF MANAGEMENT
AND RESEARCH, PIMPRI, PUNE-18

Academic Year 2021-22



Dr. D. Y. Patil Unitech Society's

**Dr. D.Y. PATIL INSTITUTE OF MANAGEMENT & RESEARCH,
Sant Tukaram Nagar, Pimpri, Pune-411018, Maharashtra,
India.(Approved by All India Council for Technical Education &
Recognized by the Savitribai Phule Pune University)**

Date:

CERTIFICATE

This is to certify that **Mr. Aditya Sanjeev Kulkarni** has successfully completed the practical on IT31L practical (ML/DL) as a partial fulfilment of their *Master of Computer Application (Sem-III)* under the curriculum of **Savitribai Phule Pune University (SPPU), Pune** for the academic year 2021-22.

Mr. Amit Shrivastava
Subject Incharge

Ms. Swati Narkhede
Course Coordinator

Dr. Shikha Dubey
MCA, HOD

Table Of Contents

1. Find the correlation matrix.....	4
2. Plot the correlation plot on dataset and visualize giving an overview of Relationships among data on iris data.....	5
3. Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data	10
4. Apply linear regression Model techniques to predict the data on any dataset	11
5. Apply logical regression Model techniques to predict the data on any dataset.....	16
6. Clustering algorithms for unsupervised classification.....	19
7. Association algorithms for supervised classification on any dataset	22
8. Developing and implementing Decision Tree model on the dataset	24
9. Bayesian classification on any dataset.....	26
10. SVM classification on any dataset.....	28
11. Text Mining algorithms on unstructured dataset	30
12. Plot the cluster data using python visualizations.	32
13. Creating & Visualizing Neural Network for the given data. (Use python) ..	37
14. Recognize optical character using ANN.....	39
15. Write a program to implement CNN	41
16. Write a program to implement RNN	43
17. Write a program to implement GAN	46
18. Web scraping experiments (by using tools).....	51

1. Find the correlation matrix.

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

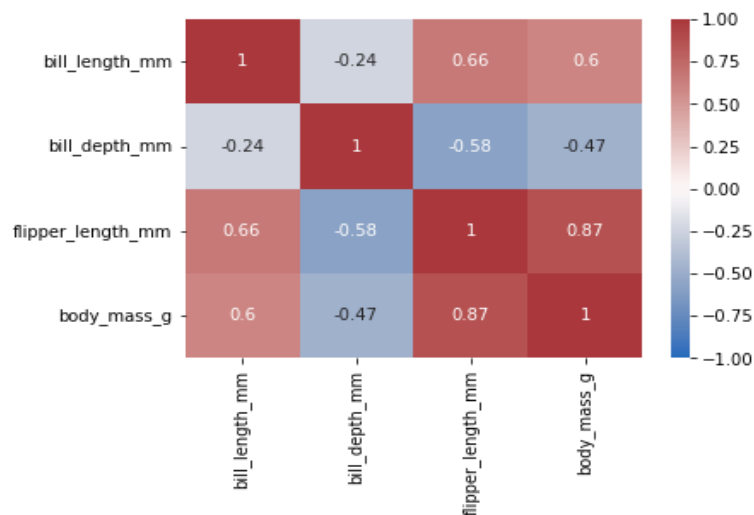
```
# load the dataset
df=sns.load_dataset('penguins')
```

```
matrix=df.corr().round(2)
matrix
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
bill_length_mm	1.00	-0.24	0.66	0.60
bill_depth_mm	-0.24	1.00	-0.58	-0.47
flipper_length_mm	0.66	-0.58	1.00	0.87
body_mass_g	0.60	-0.47	0.87	1.00

```
sns.heatmap(matrix,annot=True,cmap='vlag',vmax=1,vmin=-1)
```

```
<AxesSubplot:>
```



```
plt.show()
```

```
plt.savefig('heatmap.png')
```

<Figure size 432x288 with 0 Axes>

2. Plot the correlation plot on dataset and visualize giving an overview of Relationships among data on iris data.

```
import pandas as pd
```

```
from sklearn import datasets
```

```
iris=datasets.load_iris()
```

```
iris
```

```
df=pd.DataFrame(data=iris.data,columns=iris.feature_names)
```

```
df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	
0		5.1	3.5	1.4	0.2
1		4.9	3.0	1.4	0.2
2		4.7	3.2	1.3	0.2

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)		
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	
...
145	6.7	3.0	5.2	2.3	
146	6.3	2.5	5.0	1.9	
147	6.5	3.0	5.2	2.0	
148	6.2	3.4	5.4	2.3	
149	5.9	3.0	5.1	1.8	

150 rows × 4 columns

```
df['target']=iris.target
```

```
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
r=df["sepal length (cm)"].corr(df["petal length (cm)"])
```

```
r
```

```
0.8717537758865831
```

```
corr=df.corr()
```

```
corr
```

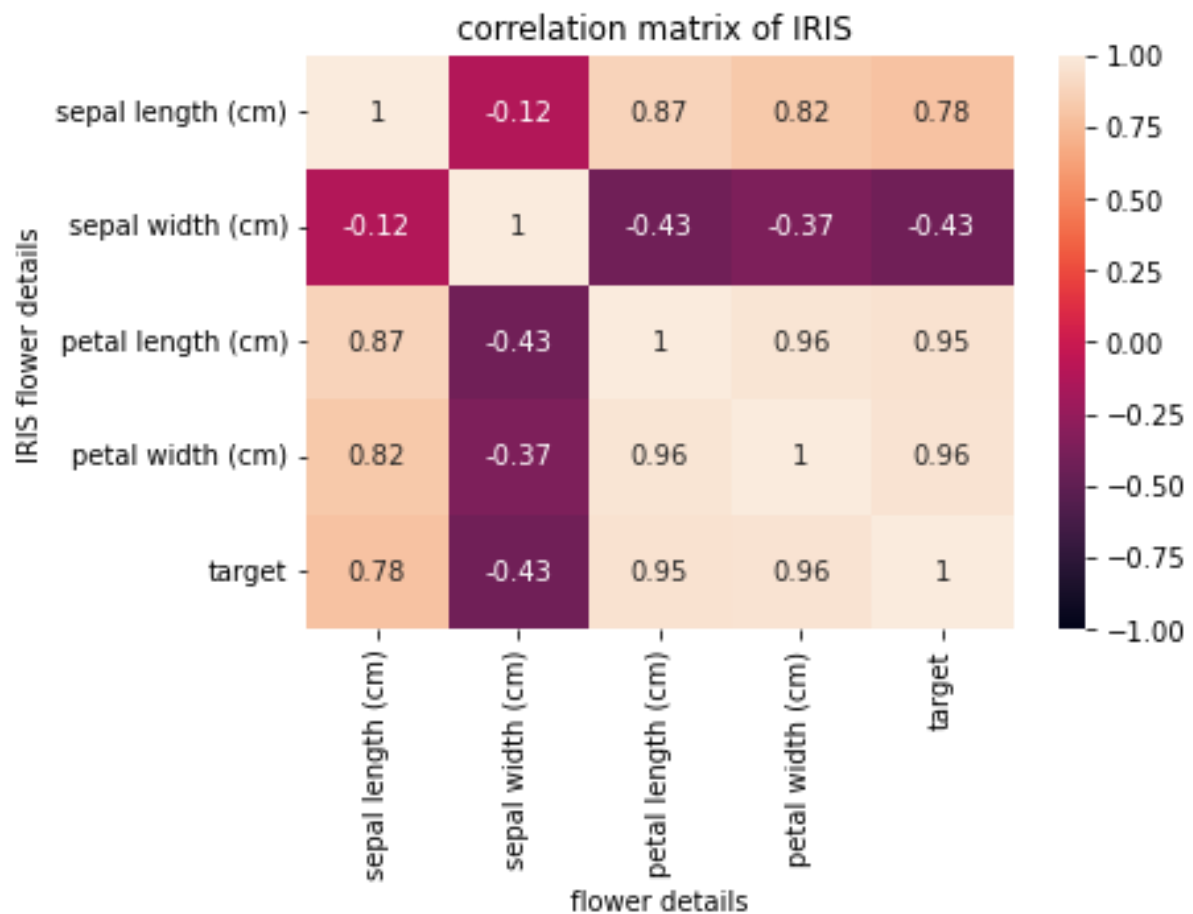
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
--	-------------------	------------------	-------------------	------------------	--------

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941	0.782561
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126	0.426658
petal length (cm)	0.871754	-0.428440	1.000000	0.962865	0.949035
petal width (cm)	0.817941	-0.366126	0.962865	1.000000	0.956547
target	0.782561	-0.426658	0.949035	0.956547	1.000000

```

import seaborn as sns
import matplotlib.pyplot as plt
hm=sns.heatmap(df.corr(),annot=True,vmax=1,vmin=-1)
hm.set(xlabel="flower details",ylabel="IRIS flower details",title="correlation
matrix of IRIS ")
plt.show()
plt.savefig("plotting correlation.jpg")

```

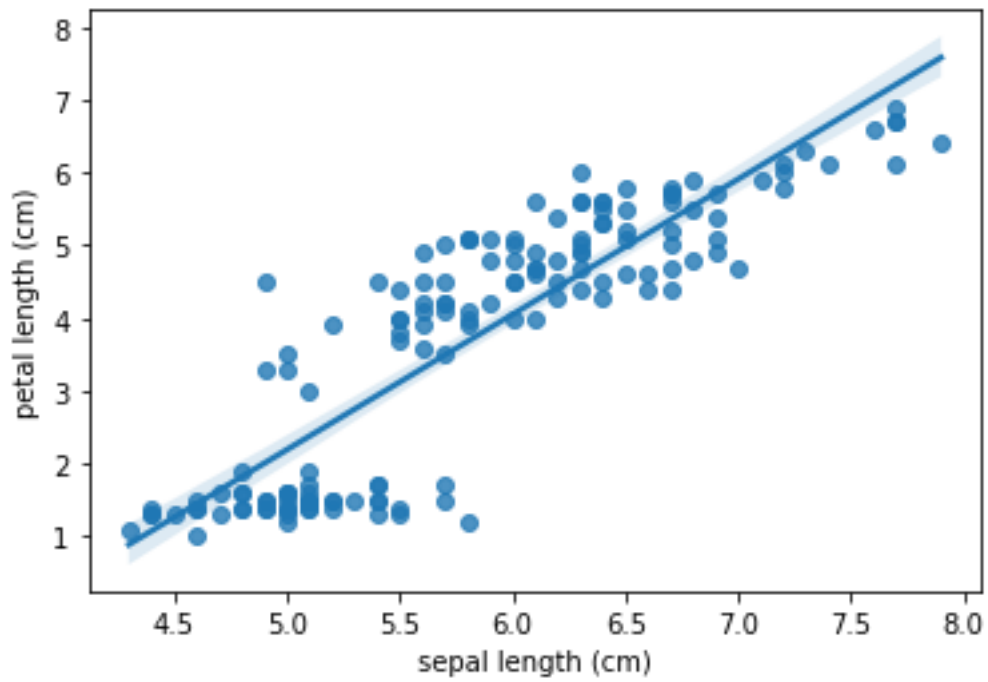


<Figure size 432x288 with 0 Axes>

#use the function regplotto make a scatterplot

```
sns.regplot(x=df["sepal length (cm)"],y=df["petal length (cm)"])
```

```
<AxesSubplot:xlabel='sepal length (cm)', ylabel='petal length (cm)'>
```

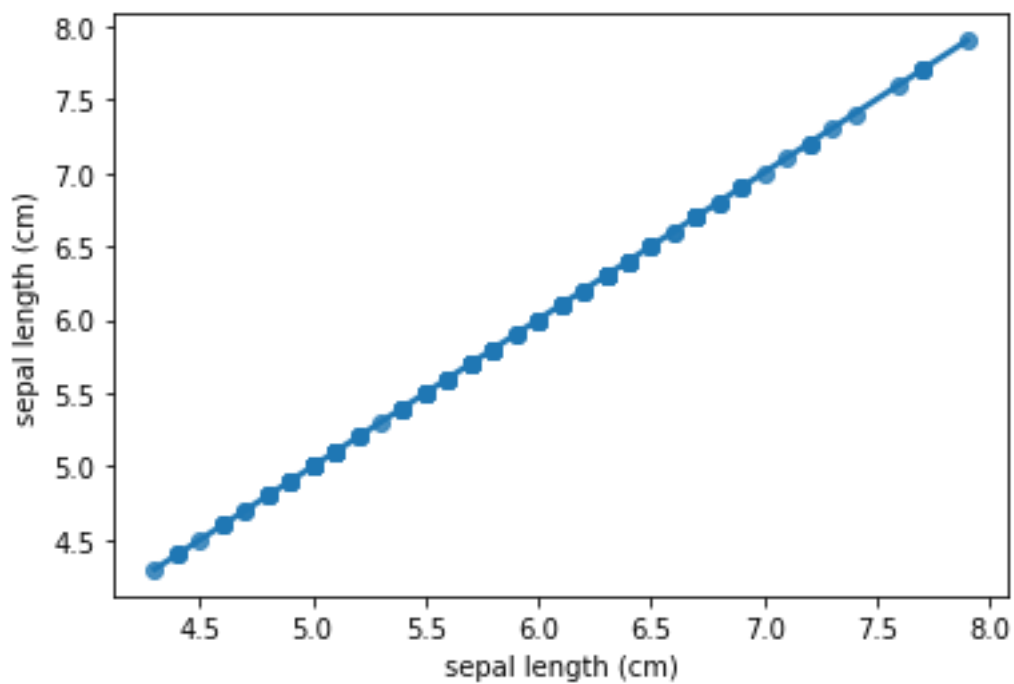
```
r=df["sepal length (cm)"].corr(df["sepal length (cm)"])
```

```
r
```

```
1.0
```

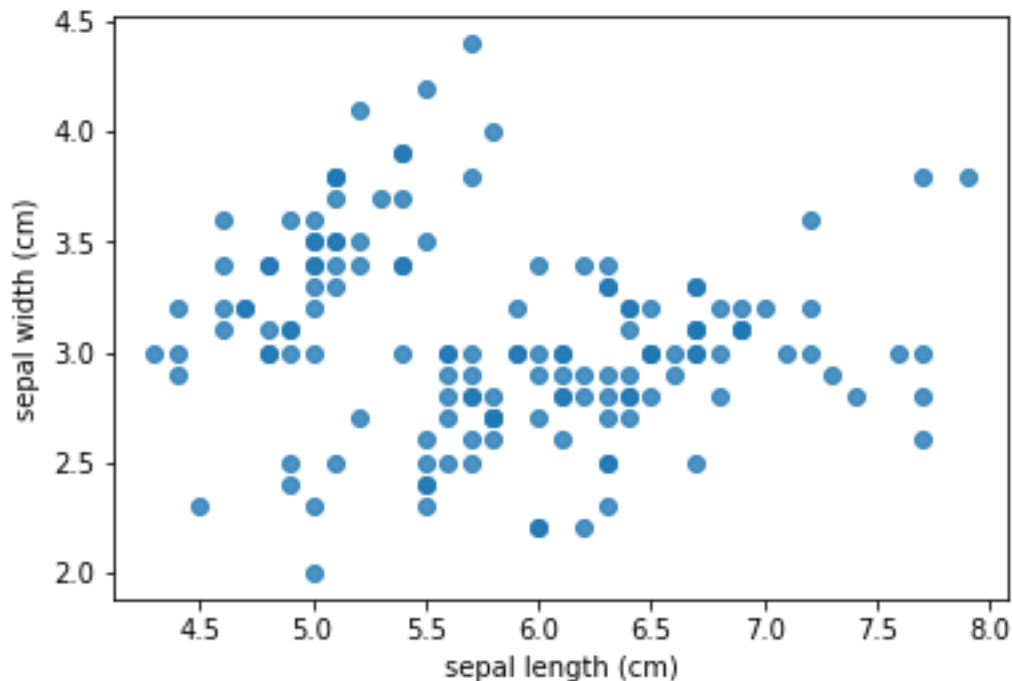
```
sns.regplot(x=df["sepal length (cm)"],y=df["sepal length (cm)"])
```

```
<AxesSubplot:xlabel='sepal length (cm)', ylabel='sepal length (cm)'>
```



```
sns.regplot(x=df["sepal length (cm)"],y=df["sepal width (cm)"], fit_reg=False)
```

```
<AxesSubplot:xlabel='sepal length (cm)', ylabel='sepal width (cm)'\>
```



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

```
import pandas as pd
```

```
df = pd.read_csv("./data.txt",sep='\t')
```

```
df.head()
```

```
df[['jobcat_name','prevexp']].groupby('jobcat_name').mean()
```

```
mgr = df[df.jobcat_name=='Manager']['prevexp']
```

```
cle = df[df.jobcat_name=='Clerical']['prevexp']
```

```
cust = df[df.jobcat_name=='Custodial']['prevexp']
```

```
from scipy import stats
```

```
f_statistic, p_value = stats.f_oneway(mgr, cle, cust)
```

```

print("F_Statistic: {0}, P-Value: {1}".format(f_statistic,p_value))

from statsmodels.formula.api import ols

model_name = ols('prevexp ~ C(jobcat_name)', data=df).fit()

model_name.summary()

```

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

```

import pandas as pd
from sklearn.datasets import load_iris
iris=load_iris()
df=pd.DataFrame(data=iris.data,columns=iris.feature_names)
df["target"]=iris.target
df.head()

```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```

df=pd.read_csv('Iris.csv')
df

```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

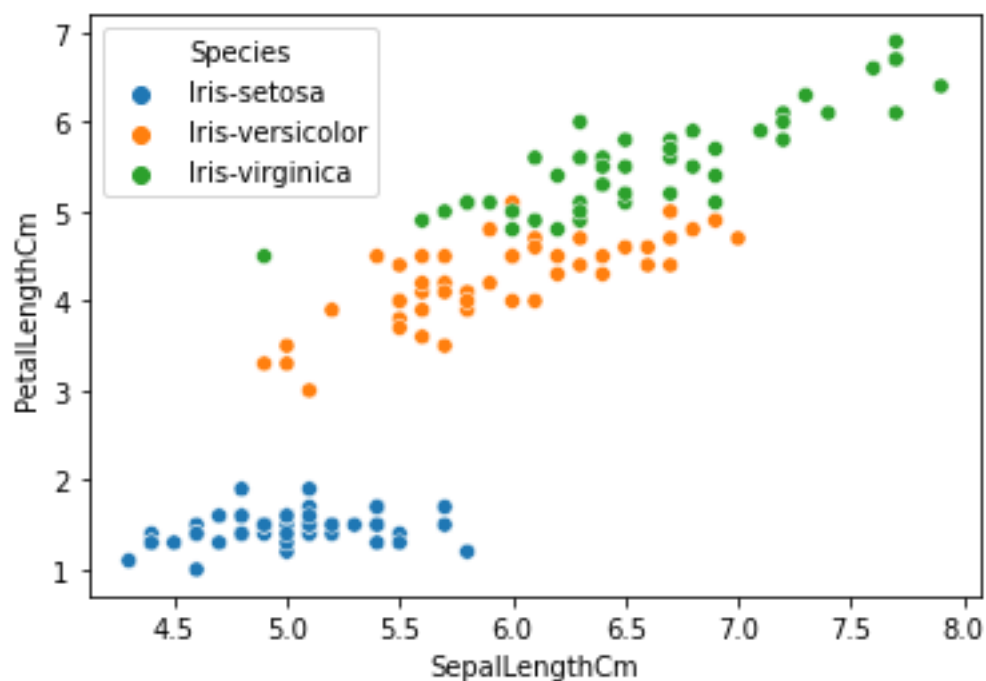
150 rows x 6 columns

```
from matplotlib import pyplot as plt
```

```
import seaborn as sns
```

```
sns.scatterplot(data=df,x='SepalLengthCm',y='PetalLengthCm',hue='Species')
```

```
<AxesSubplot:xlabel='SepalLengthCm', ylabel='PetalLengthCm'>
```



```
y=df[['SepalWidthCm']]
```

```
y
```

SepalWidthCm	
0	3.5
1	3.0
2	3.2
3	3.1
4	3.6
...	...
145	3.0
146	2.5
147	3.0
148	3.4
149	3.0

150 rows x 1 columns

```
x=df[['SepalLengthCm']]
```

```
x
```

SepalLengthCm	
0	5.1
1	4.9
2	4.7
3	4.6
4	5.0
...	...
145	6.7
146	6.3
147	6.5
148	6.2
149	5.9

150 rows x 1 columns

```

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
x_train

```

SepalLengthCm

64	5.6
30	4.8
141	6.9
59	5.2
74	6.4
...	...
31	5.4
46	5.1
115	6.4
104	6.5
40	5.0

105 rows x 1 columns

```
x_test.head()
```

SepalLengthCm

47	4.6
68	6.2
2	4.7
18	5.7
16	5.4

```
y_test.head()
```

SepalWidthCm

47	3.2
68	2.2
2	3.2

SepalWidthCm

18	3.8
16	3.9

```
y_train.head()
```

SepalWidthCm

64	2.9
30	3.1
141	3.1
59	2.7
74	2.9

```
from sklearn.linear_model import LinearRegression
```

```
LR=LinearRegression()
```

```
LR.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_pred=LR.predict(x_test)
```

```
y_pred[0:5]
```

```
array([[3.0786979 ],  
       [3.02951804],  
       [3.07562416],  
       [3.04488674],  
       [3.05410797]])
```

```
y_test.head()
```

SepalWidthCm

47	3.2
68	2.2
2	3.2
18	3.8
16	3.9

```
from sklearn.metrics import mean_squared_error
```

```
mean_squared_error(y_test,y_pred)
```

```
0.2327396540269164
```

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

```
import pandas as pd
```

```
df=pd.read_csv("diabetes.csv")
```

```
df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows x 9 columns


```
feature_cols=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin',
',BMI','DiabetesPedigreeFunction','Age']
```

```
x=df[feature_cols]
```

```
y=df.Outcome
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)
```

```
x_test
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
581	6	109	60	27	0	25.0	0.206	27
323	13	152	90	33	29	26.8	0.731	43
333	12	106	80	0	0	23.6	0.137	44
513	2	91	62	0	0	27.3	0.525	22
123	5	132	80	0	0	26.8	0.186	69
...
330	8	118	72	19	0	23.1	1.476	46
609	1	111	62	13	182	24.0	0.138	23
263	3	142	80	15	0	32.4	0.200	63
626	0	125	68	0	0	24.7	0.206	21
728	2	175	88	0	0	22.9	0.326	22

```
192 rows x 8 columns
```

```
from sklearn.linear_model import LogisticRegression
```

```
logreg=LogisticRegression()
```

```
logreg.fit(x_train,y_train)
```

```
logreg
```

```
y_pred=logreg.predict(x_test)
```

y_pred

```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0])
```

df.head()

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
from sklearn import metrics
```

```
cnf_matrix=metrics.confusion_matrix(y_test,y_pred)
```

cnf_matrix

```
array([[116, 11],
       [ 31, 34]], dtype=int64)
```

```
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
```

Accuracy: 0.78125

```
print("Precision:",metrics.precision_score(y_test,y_pred))
```

```
Precision: 0.7555555555555555
```

```
print("Recall:",metrics.recall_score(y_test,y_pred))
```

Recall: 0.5230769230769231

```
y_pred_proba=logreg.predict_proba(x_test)[::,1]
```

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

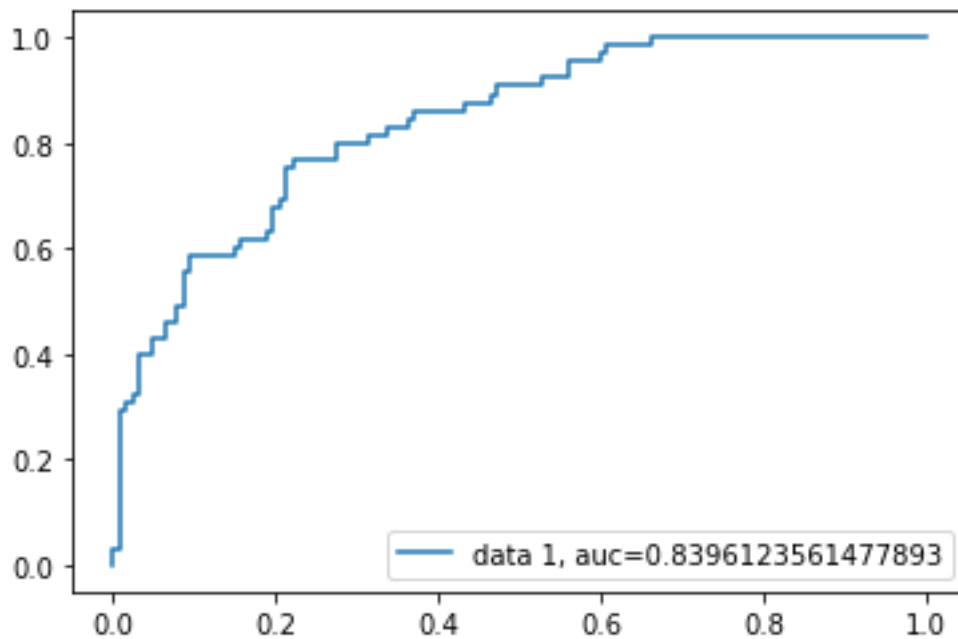
```
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
```

```
auc = metrics.roc_auc_score(y_test, y_pred_proba)
```

```
plt.plot(fpr, tpr, label="data 1, auc="+str(auc))
```

```
plt.legend(loc=4)
```

```
plt.show()
```



6. Clustering algorithms for unsupervised classification.

```
import pandas as pd
```

```
df = pd.read_csv('/content/Mall_Customers.csv')
```

```
list(df.columns)
```

```
x = df.iloc[:,3:]
```

```

df.describe()

from sklearn.cluster import KMeans

km = KMeans(n_clusters=12, random_state=0)

labels = km.fit_predict(x)

km.inertia_

sse = []

for k in range(1,41):

    km = KMeans(n_clusters=k, random_state=0)

    labels = km.fit_predict(x)

    sse.append(km.inertia_)

from sklearn.metrics import silhouette_score

silh = []

for k in range(2,16):

    km = KMeans(n_clusters=k, random_state=0)

    labels = km.fit_predict(x)

    score = silhouette_score(x, labels)

    silh.append(score)

km = KMeans(n_clusters=5, random_state=0)

labels = km.fit_predict(x)

km.labels_

km.cluster_centers_

df[labels==2] # Boolean filter

one = df[labels==1]

one.to_csv('one.csv')

print('Cluster-0:', len(df[labels==0]))

print('Cluster-1:', len(df[labels==1]))

print('Cluster-2:', len(df[labels==2]))

print('Cluster-3:', len(df[labels==3]))

print('Cluster-4:', len(df[labels==4]))

new = [[45, 76]]

```

```
km.predict(new)[0]
```

```
new = [[25, 36]]
```

```
km.predict(new)[0]
```

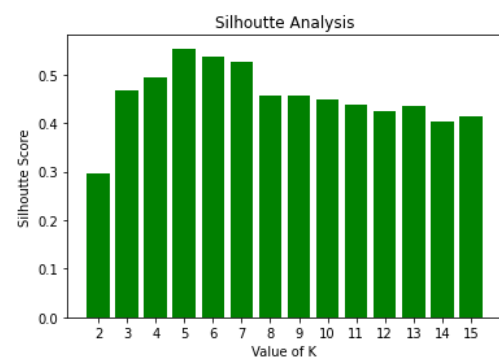
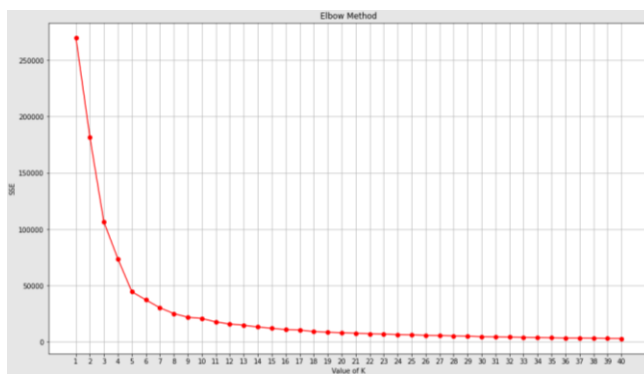
```
new = [[85, 76]]
```

```
km.predict(new)[0]
```

```
new = [[45, 47]]
```

```
km.predict(new)[0]
```

Output:



```
✓ [33] # Export the cluster  
0s one.to_csv('one.csv')
```

```
✓ [34] print('Cluster-0:', len(df[labels==0]))  
0s      print('Cluster-1:', len(df[labels==1]))  
      print('Cluster-2:', len(df[labels==2]))  
      print('Cluster-3:', len(df[labels==3]))  
      print('Cluster-4:', len(df[labels==4]))
```

```
Cluster-0: 35  
Cluster-1: 81  
Cluster-2: 39  
Cluster-3: 22  
Cluster-4: 23
```

7. Association algorithms for supervised classification on any dataset

```
dataset = [['Apple', 'Beer', 'Rice', 'Chicken'],
           ['Apple', 'Beer', 'Rice'],
           ['Apple', 'Beer'],
           ['Apple', 'Pear'],
           ['Milk', 'Beer', 'Rice', 'Chicken'],
           ['Milk', 'Beer', 'Rice'],
           ['Milk', 'Beer'],
           ['Apple', 'Pear']]

# Import the transaction encoder
from mlxtend.preprocessing import TransactionEncoder

# Create the object
trans = TransactionEncoder()

# Apply the operation
df_t = trans.fit_transform(dataset)

trans.columns_

import pandas as pd

# Create a structured dataframe
df = pd.DataFrame(df_t, columns=trans.columns_)

# Support count
sum(df['Rice']) / len(df)

# Generate frequent itemsets
from mlxtend.frequent_patterns import apriori
freq_itemset = apriori(df, min_support=0.25, use_colnames=True)
freq_itemset


# Generate strong association rules
from mlxtend.frequent_patterns import association_rules
rules = association_rules(freq_itemset,
                          metric='confidence',
```

```

min_threshold=0.5)
rules
rules = rules[['antecedents','consequents','support','confidence']]
rules['antecedent_len'] = rules['antecedents'].apply(lambda x: len(x))
nrules = rules[(rules['antecedent_len'] == 1) &
               (rules['support'] > 0.30)]
nrules
# Prediction / Suggestion / Recommendation
nrules[nrules['antecedents'] == {'Apple'}]['consequents'][1]
rules.sort_values(by='confidence', ascending=False)
# Export the rules
rules.to_csv('rules.csv', index=False)

```

Output:

✓ 0s  rules.sort_values(by='confidence', ascending=False)

	antecedents	consequents	support	confidence	antecedent_len
14	(Apple, Rice)	(Beer)	0.250	1.000000	2
2	(Pear)	(Apple)	0.250	1.000000	1
24	(Rice, Milk)	(Beer)	0.250	1.000000	2
4	(Chicken)	(Beer)	0.250	1.000000	1
6	(Milk)	(Beer)	0.375	1.000000	1
20	(Chicken)	(Beer, Rice)	0.250	1.000000	1
8	(Rice)	(Beer)	0.500	1.000000	1
9	(Chicken)	(Rice)	0.250	1.000000	1
18	(Chicken, Rice)	(Beer)	0.250	1.000000	2
17	(Chicken, Beer)	(Rice)	0.250	1.000000	2
13	(Apple, Beer)	(Rice)	0.250	0.666667	2
23	(Beer, Milk)	(Rice)	0.250	0.666667	2
26	(Milk)	(Beer, Rice)	0.250	0.666667	1

8. Developing and implementing Decision Tree model on the dataset

```
import pandas as pd

# Data import
df = pd.read_csv('/content/sample_data/Social_Network_Ads.csv')
df.shape

# input
x = df[['Age', 'EstimatedSalary']]

# output
y = df['Purchased']

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, random_state=0, test_size=0.25)

# Import the class
from sklearn.ensemble import RandomForestClassifier

# Create the object
classifier = RandomForestClassifier(random_state=0, n_estimators=10)

# n_estimators -> number of trees in the forest

# Train the algorithm with data
classifier.fit(x_train, y_train)

# Predictions
y_pred = classifier.predict(x_test)

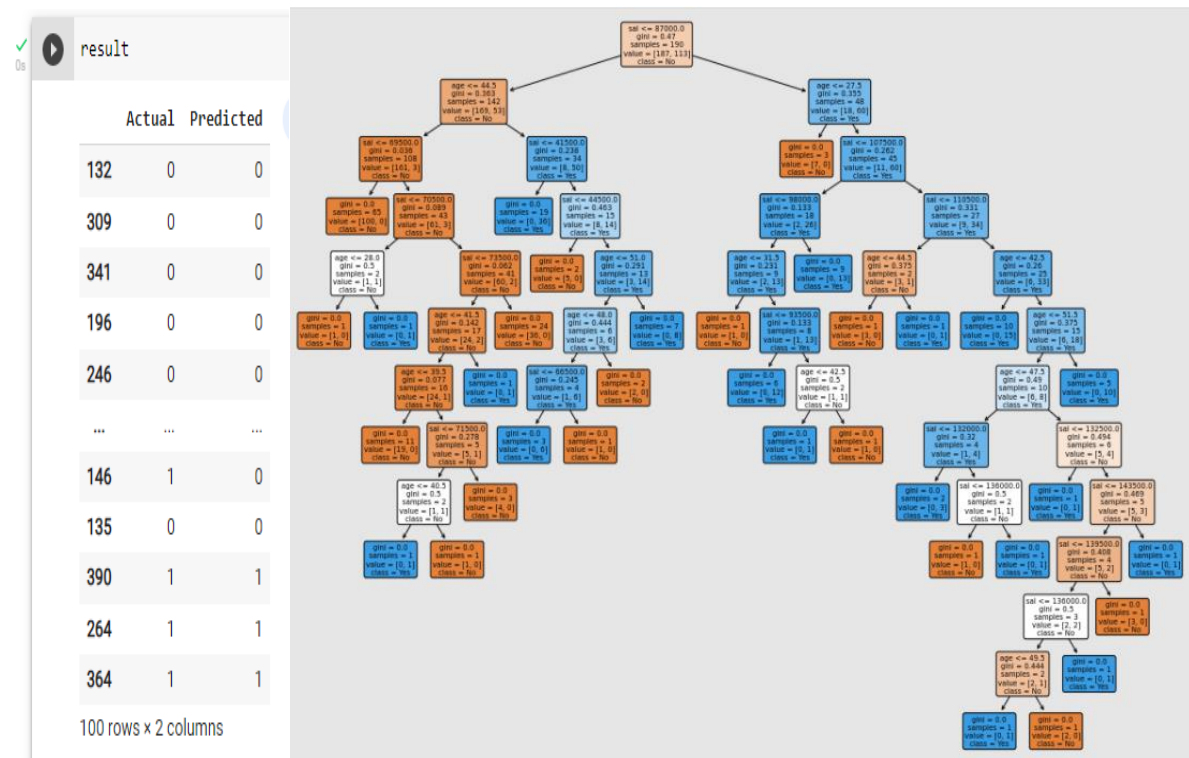
# Combine the data
result = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred
})

Result

from sklearn.tree import plot_tree
```



```
class_names=['No','Yes'], filled=True, rounded=True);
```



9. Bayesian classification on any dataset.

```
# Import packages
import pandas as pd
import seaborn as sns

# Data import
df = pd.read_csv('/content/sample_data/iris.csv')

# Let's describe
df.describe()

# Check the clusters
sns.pairplot(df, hue='species')

# input data
x = df.drop('species', axis = 1)

# output data
y = df['species']

sns.countplot(x = y)
y.value_counts()

# Cross validation -> hold out method
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, random_state=0, train_size=0.75)

# Import the class
from sklearn.naive_bayes import GaussianNB

# Create the object
classifier = GaussianNB()

# Train the algorithm with dataset
classifier.fit(x_train, y_train)

# Predictions
y_pred = classifier.predict(x_test)

# Import all functions
```

```

from sklearn.metrics import plot_confusion_matrix, accuracy_score
from sklearn.metrics import classification_report

# Plot the confusion matrix
plot_confusion_matrix(classifier, x_test, y_test)

# Accuracy
accuracy_score(y_test, y_pred)

# Classification report
print(classification_report(y_test, y_pred))

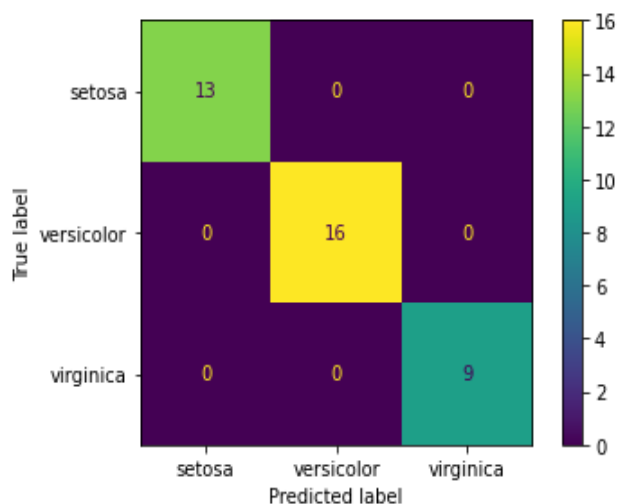
# Print the probabilities
classifier.predict_proba(x_test)

new1 = [[5.1,3.7,1.5,0.4]]
new2 = [[6.8,2.8,4.8,1.4]]
new3 = [[7.7,2.6,6.9,2.3]]

# Predictions
classifier.predict(new1)[0]
classifier.predict(new2)[0]
classifier.predict(new3)[0]

```

Output:



10. SVM classification on any dataset

SVM

1) classification approach, it can easily handle multiple continuous and categorical variables

2) SVM constructs a Hyperplane in multidimensional space to separate different classes.

3) SVM generates an optimal hyperplane in an iterative manner which is used to minimize an error

4) the basic idea of SVM is to find a Max Marginal Hyperplane (MMH) to divide the dataset into classes

Support vectors

are the data points which are closest to the hyperplane. These points will define the separating line better margins

```
from sklearn import datasets
```

```
cancer=datasets.load_breast_cancer()
```

```
print("features:",cancer.feature_names)
```

```
features: ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
'mean smoothness' 'mean compactness' 'mean concavity'
'mean concave points' 'mean symmetry' 'mean fractal dimension'
'radius error' 'texture error' 'perimeter error' 'area error'
'smoothness error' 'compactness error' 'concavity error'
'concave points error' 'symmetry error' 'fractal dimension error'
'worst radius' 'worst texture' 'worst perimeter' 'worst area'
'worst smoothness' 'worst compactness' 'worst concavity'
'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

```
print("Labels:",cancer.target_names)
```

```
Labels: ['malignant' 'benign']
```

```
cancer.data.shape
```

```
(569, 30)
```

splitting data

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(cancer.data,cancer.target,test_size=.3)

from sklearn import svm
clf=svm.SVC(kernel='linear')
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)

from sklearn import metrics

print("accuracy:",metrics.accuracy_score(y_test,y_pred))

accuracy: 0.9590643274853801

print("precision:",metrics.precision_score(y_test,y_pred))

print("recall:",metrics.recall_score(y_test,y_pred))

precision: 0.9478260869565217
recall: 0.990909090909091
```

11. Text Mining algorithms on unstructured dataset.

```
import pandas as pd
df = pd.read_csv('/content/sample_data/SMSSpamCollection', sep='\t',
names = ['class','body_text'])
import string
string.punctuation
# Function to count the punctuation symbols
def count_punct(text):
    count = sum([1 for x in text if x in string.punctuation])
    return(round(count/(len(text)-text.count(' '))*100,2))
s = 'Hello, friends! How are you? Welcome to Pune.!!!'
count_punct(s)
# Add feature of punctuation percentages
df['punct%'] = df['body_text'].apply(lambda x: count_punct(x))
# Add the column body length to it
df['body_len'] = df['body_text'].apply(lambda x: len(x) - x.count(" "))
from nltk.corpus import stopwords
s_words = stopwords.words('english')
s_words;
from nltk.stem import PorterStemmer
ps = PorterStemmer()
# analyzer function
def clean_text(text):
    data = [x for x in text if x not in string.punctuation]
    data = "".join(data)
    data = [ps.stem(x) for x in data.split() if x not in s_words]
    return data
clean_text(s)
# Seperate the input and output
```

```

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

# Import tfidf vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(analyzer=clean_text)
X_trans = tfidf.fit_transform(X['body_text'])
X_vect = pd.concat([X[['body_len', 'punct%']],
                    .reset_index(drop=True),
                    pd.DataFrame(X_trans.toarray()), axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X_vect, y, stratify=y, random_state=0)

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_test, y_pred)

```

Output:

df

	class	body_text	punct%	body_len
0	ham	Go until Jurong point, crazy. Available only ...	9.78	92
1	ham	Ok lar... Joking wif u oni...	25.00	24
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	4.69	128
3	ham	U dun say so early hor... U c already then say...	15.38	39
4	ham	Nah I don't think he goes to usf, he lives aro...	4.08	49
...
5567	spam	This is the 2nd time we have tried 2 contact u...	6.11	131
5568	ham	Will u b going to esplanade fr home?	3.45	29
5569	ham	Pity, * was in mood for that. So...any other s...	14.58	48
5570	ham	The guy did some bitching but I acted like i'd...	1.00	100
5571	ham	Rofl. Its true to its name	4.76	21

5572 rows x 4 columns

```
[ ] print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
ham	0.96	1.00	0.98	1206
spam	1.00	0.75	0.86	187
accuracy			0.97	1393
macro avg	0.98	0.87	0.92	1393
weighted avg	0.97	0.97	0.96	1393

12. Plot the cluster data using python visualizations.

```
# Import packages
import pandas as pd

# Import the dataset
df = pd.read_csv('/content/sample_data/Mall_Customers.csv')

# Input data
x = df.iloc[:,3:]

x

# Summerize
df.describe()

# import seaborn package
import seaborn as sns

sns.kdeplot(df['Age'])
sns.kdeplot(df['Annual Income (k$)'])
sns.kdeplot(df['Spending Score (1-100)'])
sns.boxplot(df['Age'])
sns.boxplot(df['Annual Income (k$)'])
sns.boxplot(df['Spending Score (1-100)'])

# Import the class
from sklearn.cluster import KMeans

# Create the object
km = KMeans(n_clusters=12, random_state=0)

# Train the algorithm
labels = km.fit_predict(x)

# Sum of squared errors
km.inertia_

# elbow method
sse = []
for k in range(1,41):
```



```

km = KMeans(n_clusters=k, random_state=0)
labels = km.fit_predict(x)
sse.append(km.inertia_)
import matplotlib.pyplot as plt
plt.figure(figsize=(16,9))
plt.title('Elbow Method')
plt.xlabel('Value of K')
plt.ylabel('SSE')
plt.grid()
plt.xticks(range(1,41))
plt.plot(range(1,41), sse, marker='o', color='r')
# Silhoutte method
from sklearn.metrics import silhouette_score
silh = []
for k in range(2,16):
    km = KMeans(n_clusters=k, random_state=0)
    labels = km.fit_predict(x)
    score = silhouette_score(x, labels)
    silh.append(score)
# plot the silhoutte scores
plt.title('Silhoutte Analysis')
plt.xlabel('Value of K')
plt.ylabel('Silhoutte Score')
plt.xticks(range(2,16))
plt.bar(range(2,16), silh, color='g')
# Create the object
km = KMeans(n_clusters=5, random_state=0)
# Train the algorithm
labels = km.fit_predict(x)
# Cluster labels

```

```
km.labels_  
# SSE  
km.inertia_  
# Extract the clusters  
df[labels==2] # Boolean filtering  
one = df[labels==1]  
# Export the cluster  
one.to_csv('one.csv')  
print('Cluster-0:', len(df[labels==0]))  
print('Cluster-1:', len(df[labels==1]))  
print('Cluster-2:', len(df[labels==2]))  
print('Cluster-3:', len(df[labels==3]))  
print('Cluster-4:', len(df[labels==4]))  
# Prediction  
new = [[45, 76]]  
km.predict(new)[0]  
# Prediction  
new = [[25, 36]]  
km.predict(new)[0]  
# Prediction  
new = [[85, 76]]  
km.predict(new)[0]  
# Prediction  
new = [[45, 47]]  
km.predict(new)[0]  
# Visualization of clusters  
plt.title('Unclustered data')  
plt.xlabel('Annual Income')  
plt.ylabel('Spending Score')  
plt.grid()
```

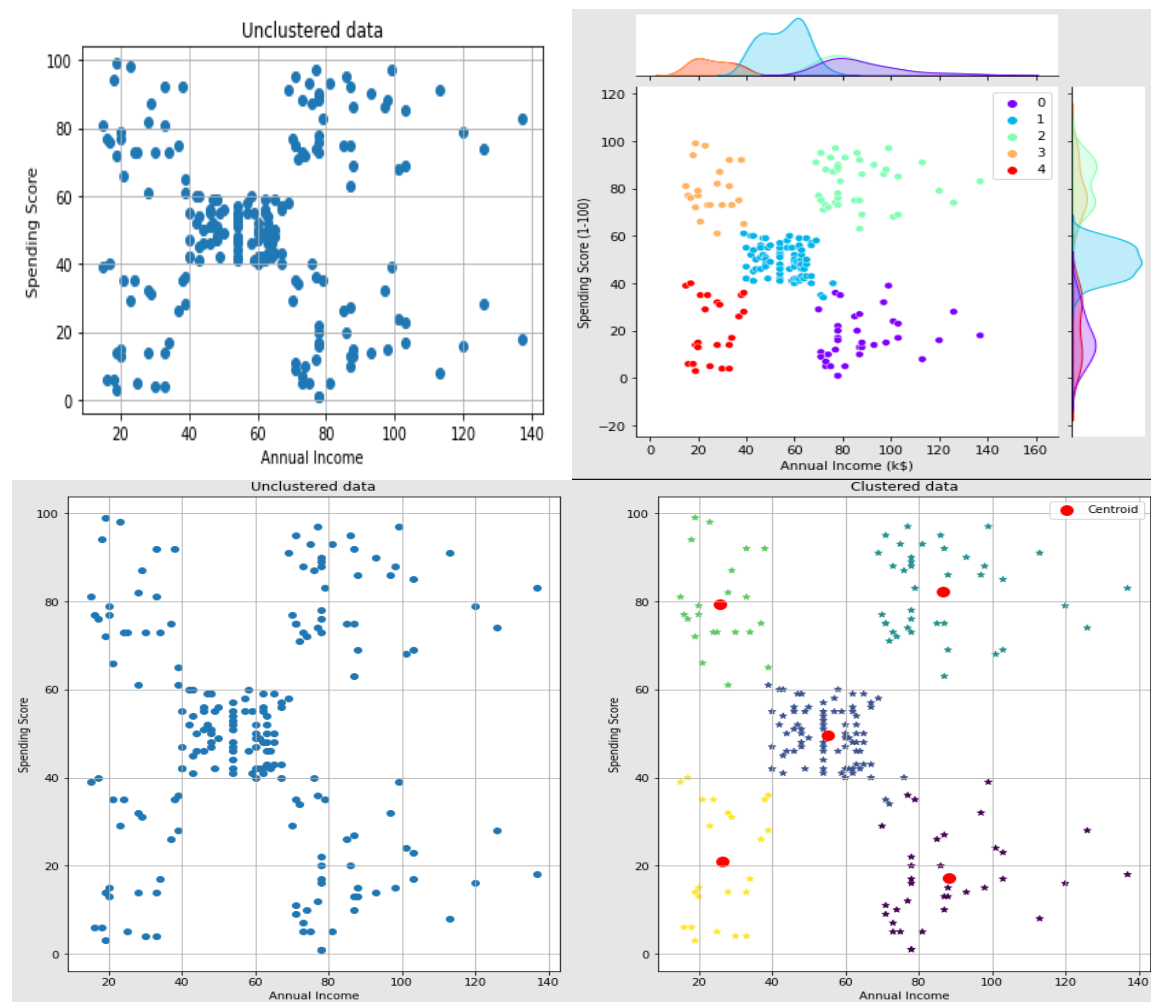
```

plt.scatter(x['Annual Income (k$)'], x['Spending Score (1-100)'])
# Save the centroids
cent = km.cluster_centers_
# Visualization of clusters
plt.title('Clustered data')
plt.xlabel('Annual Income')
plt.grid()
plt.scatter(x['Annual Income (k$)'], x['Spending Score (1-100)'],
            c = labels, marker='*')
plt.scatter(cent[:,0], cent[:,1], s=100, marker='o', color='r')
# Combined plot
plt.figure(figsize=(16,9))
plt.subplot(1,2,1)
plt.title('Unclustered data')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.grid()
plt.scatter(x['Annual Income (k$)'], x['Spending Score (1-100)'])
plt.subplot(1,2,2)
plt.title('Clustered data')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.grid()
plt.scatter(x['Annual Income (k$)'], x['Spending Score (1-100)'],
            c = labels, marker='*')
plt.scatter(cent[:,0], cent[:,1], s=100, marker='o', color='r',
            label = 'Centroid')
plt.legend()
plt.savefig('Clusters.png')
import seaborn as sns

```

```
# Visualization using joint plot
p = sns.jointplot(x=x['Annual Income (k$)'],
y=x['Spending Score (1-100)'],
hue = labels,palette='rainbow', )
# sns.jointplot(x=cent[:,0], y=cent[:,1])
p.savefig('seaborn_clusters.png')
```

Output:



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

```
from google.colab import drive
drive.mount('/content/drive')
from keras.layers import Dense
from keras.models import Sequential
import numpy as np
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
# load dataset
dataset = np.loadtxt('/content/sample_data/pima-new (1).csv', delimiter=',')
dataset
# input data
X = dataset[:, :8]
# output data
Y = dataset[:, 8]
Y
# create the model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu')) # Input layer
model.add(Dense(8, activation='relu')) # Hiddel layer
model.add(Dense(1, activation='sigmoid')) # Output layer
# compile model
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# train the model
model.fit(X, Y, epochs=200, batch_size=10)
```

```
# Evaluate the model

scores = model.evaluate(X, Y)

scores

new = [[7,475,82,69,120,22.2,0.645,57]]

model.predict(new)

from keras.utils.vis_utils import plot_model

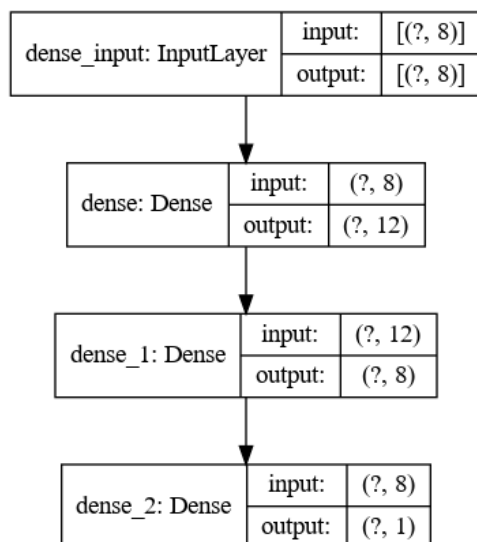
plot_model(model, show_shapes=True, show_layer_names=True,
           to_file='neural_network.png')
```

Output:

```
✓ 42s # train the model
model.fit(X, Y, epochs=200, batch_size=10)

Epoch 172/200
77/77 [=====] - 0s 2ms/step - loss: 0.5381 - accuracy: 0.7344
Epoch 173/200
77/77 [=====] - 0s 2ms/step - loss: 0.5259 - accuracy: 0.7539
Epoch 174/200
77/77 [=====] - 0s 2ms/step - loss: 0.5361 - accuracy: 0.7240
Epoch 175/200
77/77 [=====] - 0s 2ms/step - loss: 0.5390 - accuracy: 0.7318
Epoch 176/200
77/77 [=====] - 0s 2ms/step - loss: 0.5213 - accuracy: 0.7539
Epoch 177/200
77/77 [=====] - 0s 2ms/step - loss: 0.5225 - accuracy: 0.7357
Epoch 178/200
77/77 [=====] - 0s 2ms/step - loss: 0.5297 - accuracy: 0.7435
Epoch 179/200
77/77 [=====] - 0s 2ms/step - loss: 0.5264 - accuracy: 0.7370
Epoch 180/200
77/77 [=====] - 0s 2ms/step - loss: 0.5247 - accuracy: 0.7422
Epoch 181/200
77/77 [=====] - 0s 2ms/step - loss: 0.5345 - accuracy: 0.7318
Epoch 182/200
77/77 [=====] - 0s 2ms/step - loss: 0.5303 - accuracy: 0.7487
Epoch 183/200
77/77 [=====] - 0s 2ms/step - loss: 0.5304 - accuracy: 0.7383
Epoch 184/200

[ ] plot_model(model, show_shapes=True, show_layer_names=True,
               to_file='neural_network.png')
```



14. Recognize optical character using ANN.

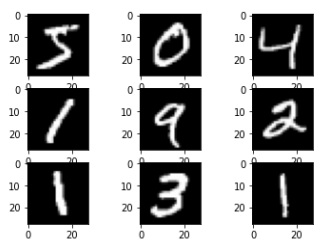
```
from keras.datasets import mnist
import matplotlib.pyplot as plt
(X_train, y_train), (X_test, y_test) = mnist.load_data()
plt.subplot(3,3,1)
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
plt.subplot(3,3,2)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
plt.subplot(3,3,3)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
plt.subplot(3,3,4)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
plt.subplot(3,3,5)
plt.imshow(X_train[4], cmap=plt.get_cmap('gray'))
plt.subplot(3,3,6)
plt.imshow(X_train[5], cmap=plt.get_cmap('gray'))
plt.subplot(3,3,7)
plt.imshow(X_train[6], cmap=plt.get_cmap('gray'))
plt.subplot(3,3,8)
plt.imshow(X_train[7], cmap=plt.get_cmap('gray'))
plt.subplot(3,3,9)
plt.imshow(X_train[8], cmap=plt.get_cmap('gray'))
from keras.layers import Dense
from keras.models import Sequential
import numpy as np
num_pixels = X_train[0].shape[0] * X_train[0].shape[1]
# Reshape
X_train = X_train.reshape(X_train.shape[0], num_pixels)
X_test = X_test.reshape(X_test.shape[0], num_pixels)
```

```

import pandas as pd
pd.DataFrame(X_train).describe()
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
set(y_train)
from keras.utils import np_utils
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
y_train.shape
# Create the model
model = Sequential()
model.add(Dense(784, input_dim= 784, activation='relu'))
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
# Train the algorithm
model.fit(X_train, y_train, validation_data=(X_test, y_test),
        epochs=10, batch_size=200)
scores = model.evaluate(X_train, y_train)
scores

```

Output:



```

# Train the algorithm
model.fit(X_train, y_train, validation_data=(X_test, y_test),
        epochs=10, batch_size=200)
Epoch 1/10
300/300 [=====] - 6s 17ms/step - loss: 0.2786 - accuracy: 0.9213 - val_loss: 0.1401 - val_accuracy
Epoch 2/10
300/300 [=====] - 5s 16ms/step - loss: 0.1111 - accuracy: 0.9676 - val_loss: 0.1001 - val_accuracy
Epoch 3/10
300/300 [=====] - 5s 16ms/step - loss: 0.0712 - accuracy: 0.9797 - val_loss: 0.0773 - val_accuracy
Epoch 4/10
300/300 [=====] - 5s 16ms/step - loss: 0.0506 - accuracy: 0.9853 - val_loss: 0.0674 - val_accuracy
Epoch 5/10
300/300 [=====] - 5s 16ms/step - loss: 0.0374 - accuracy: 0.9891 - val_loss: 0.0635 - val_accuracy
Epoch 6/10
300/300 [=====] - 5s 16ms/step - loss: 0.0266 - accuracy: 0.9932 - val_loss: 0.0598 - val_accuracy
Epoch 7/10
261/300 [=====>....] - ETA: 0s - loss: 0.0193 - accuracy: 0.9952

```


15. Write a program to implement CNN

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPool2D
from keras.layers import Flatten
# Create the object of model
classifier = Sequential()
# Add first convolution layer
# Parameters - filters, kernel size, input shape, activation
classifier.add(Conv2D(32,(3,3), input_shape = (64, 64, 3),
    activation = 'relu'))
# Add first max pooling layer
classifier.add(MaxPool2D(pool_size = (2,2)))
# Add second convolution layer
classifier.add(Conv2D(32, (3,3), activation = 'relu'))
# Add max pooling layer
classifier.add(MaxPool2D(pool_size = (2,2)))
# Convert the 2D data to 1D format
classifier.add(Flatten())
# Add the output layer
classifier.add(Dense(units=1, activation='sigmoid'))
# Compile the model
classifier.compile(optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'])
```

```

# Image augmentation
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1/255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True)
test_datagen = ImageDataGenerator(rescale = 1./255)

# Import the train images
train = train_datagen.flow_from_directory('/content/sample_data',
    target_size=(64, 64),
    batch_size=32,
    class_mode='binary')
test = test_datagen.flow_from_directory('/content/sample_data',
    target_size=(64, 64),
    batch_size=32,
    class_mode='binary')

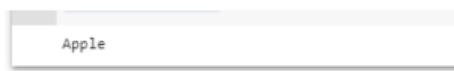
# Train the algorithm
classifier.fit(train, epochs=10, validation_data=test,
    validation_steps=10)
train.class_indices

# Prediction
import numpy as np
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
test_image = load_img('/content/sample_data/sample1.jpg', target_size=(64,
64))
test_image = img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
#test_image.shape

```

```
result = classifier.predict(test_image)
if result[0][0] == 1:
    print('Orange')
else:
    print('Apple')
```

Output:



16. Write a program to implement RNN

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
# Data import
df = pd.read_csv('/content/sample_data/Google_Stock_Price_Train.csv')
# first 5 entries
df.head()
df.describe()
df.info()
training_set = df.iloc[:,[1,2]].values
# Visualize the trend
plt.plot(training_set)
```

```
# Feature scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
training_set_scaled = scaler.fit_transform(training_set)

# The scaled data
training_set_scaled

# plot the scaled data
plt.plot(training_set_scaled)

X_train = []
y_train = []
for i in range(60, 1258):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

# Import the classes
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

# Create the model
regressor = Sequential()

# add LSTM layer
regressor.add(LSTM(units = 50, return_sequences = True,
    input_shape = (X_train.shape[1], 1)))
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 50))
```

```
regressor.add(Dropout(0.2))
# Output layer
regressor.add(Dense(1))
# Compile the model
regressor.compile(optimizer='adam', loss='mean_squared_error')
# Train the algorithm
regressor.fit(X_train, y_train, epochs=100, batch_size = 32)
testing_set =
pd.read_csv('/content/sample_data/Google_Stock_Price_Test.csv')
testing_set.shape
testing_set
real_stock_price = testing_set.iloc[:,[1,2]].values
real_stock_price
dataset_total = pd.concat((df['Open'],
testing_set['Open']), axis = 0)
dataset_total
inputs = dataset_total[len(dataset_total) -
len(testing_set) - 60:].values
inputs.shape
inputs = inputs.reshape(-1,2)
inputs.shape
# Perform the scaling
inputs = scaler.transform(inputs)
Inputs
```

Output:

✓ Os	inputs	✓ Os	[29] testing_set																																																																																																									
	array([[0.9299055 , 0.93086447], [0.92750577, 0.9439371], [0.93876032, 0.9337778], [0.93483518, 0.93112593], [0.94636878, 0.96556296], [0.97510976, 0.9595122], [0.97808617, 1.], [0.98076494, 0.97071731], [0.98450406, 0.96038994], [0.9371419, 0.9281379], [0.90804747, 0.87670644], [0.92153434, 0.93784899], [0.93165414, 0.95235961], [0.88812412, 0.88593198], [0.87032145, 0.88518498], [0.90743359, 0.91538275], [0.89941588, 0.91773582], [0.9089404, 0.90210469], [0.89456061, 0.91568155], [0.9132934, 0.88936822], [0.86589404, 0.88987245], [0.90335962, 0.89601658], [0.91777662, 0.93149943], [0.94114145, 0.95745793], [0.96413424, 0.9638822], [0.96971501, 0.95058547],																																																																																																											
			<table><tr><th></th><th>Date</th><th>Open</th><th>High</th><th>Low</th><th>Close</th><th>Volume</th></tr><tr><td>0</td><td>1/3/2017</td><td>778.81</td><td>789.63</td><td>775.80</td><td>786.14</td><td>1,657,300</td></tr><tr><td>1</td><td>1/4/2017</td><td>788.36</td><td>791.34</td><td>783.16</td><td>786.90</td><td>1,073,000</td></tr><tr><td>2</td><td>1/5/2017</td><td>786.08</td><td>794.48</td><td>785.02</td><td>794.02</td><td>1,335,200</td></tr><tr><td>3</td><td>1/6/2017</td><td>795.26</td><td>807.90</td><td>792.20</td><td>806.15</td><td>1,640,200</td></tr><tr><td>4</td><td>1/9/2017</td><td>806.40</td><td>809.97</td><td>802.83</td><td>806.65</td><td>1,272,400</td></tr><tr><td>5</td><td>1/10/2017</td><td>807.86</td><td>809.13</td><td>803.51</td><td>804.79</td><td>1,176,800</td></tr><tr><td>6</td><td>1/11/2017</td><td>805.00</td><td>808.15</td><td>801.37</td><td>807.91</td><td>1,065,900</td></tr><tr><td>7</td><td>1/12/2017</td><td>807.14</td><td>807.39</td><td>799.17</td><td>806.36</td><td>1,353,100</td></tr><tr><td>8</td><td>1/13/2017</td><td>807.48</td><td>811.22</td><td>806.69</td><td>807.88</td><td>1,099,200</td></tr><tr><td>9</td><td>1/17/2017</td><td>807.08</td><td>807.14</td><td>800.37</td><td>804.61</td><td>1,362,100</td></tr><tr><td>10</td><td>1/18/2017</td><td>805.81</td><td>806.21</td><td>800.99</td><td>806.07</td><td>1,294,400</td></tr><tr><td>11</td><td>1/19/2017</td><td>805.12</td><td>809.48</td><td>801.80</td><td>802.17</td><td>919,300</td></tr><tr><td>12</td><td>1/20/2017</td><td>806.91</td><td>806.91</td><td>801.69</td><td>805.02</td><td>1,670,000</td></tr><tr><td>13</td><td>1/23/2017</td><td>807.25</td><td>820.87</td><td>803.74</td><td>819.31</td><td>1,963,600</td></tr></table>		Date	Open	High	Low	Close	Volume	0	1/3/2017	778.81	789.63	775.80	786.14	1,657,300	1	1/4/2017	788.36	791.34	783.16	786.90	1,073,000	2	1/5/2017	786.08	794.48	785.02	794.02	1,335,200	3	1/6/2017	795.26	807.90	792.20	806.15	1,640,200	4	1/9/2017	806.40	809.97	802.83	806.65	1,272,400	5	1/10/2017	807.86	809.13	803.51	804.79	1,176,800	6	1/11/2017	805.00	808.15	801.37	807.91	1,065,900	7	1/12/2017	807.14	807.39	799.17	806.36	1,353,100	8	1/13/2017	807.48	811.22	806.69	807.88	1,099,200	9	1/17/2017	807.08	807.14	800.37	804.61	1,362,100	10	1/18/2017	805.81	806.21	800.99	806.07	1,294,400	11	1/19/2017	805.12	809.48	801.80	802.17	919,300	12	1/20/2017	806.91	806.91	801.69	805.02	1,670,000	13	1/23/2017	807.25	820.87	803.74	819.31	1,963,600
	Date	Open	High	Low	Close	Volume																																																																																																						
0	1/3/2017	778.81	789.63	775.80	786.14	1,657,300																																																																																																						
1	1/4/2017	788.36	791.34	783.16	786.90	1,073,000																																																																																																						
2	1/5/2017	786.08	794.48	785.02	794.02	1,335,200																																																																																																						
3	1/6/2017	795.26	807.90	792.20	806.15	1,640,200																																																																																																						
4	1/9/2017	806.40	809.97	802.83	806.65	1,272,400																																																																																																						
5	1/10/2017	807.86	809.13	803.51	804.79	1,176,800																																																																																																						
6	1/11/2017	805.00	808.15	801.37	807.91	1,065,900																																																																																																						
7	1/12/2017	807.14	807.39	799.17	806.36	1,353,100																																																																																																						
8	1/13/2017	807.48	811.22	806.69	807.88	1,099,200																																																																																																						
9	1/17/2017	807.08	807.14	800.37	804.61	1,362,100																																																																																																						
10	1/18/2017	805.81	806.21	800.99	806.07	1,294,400																																																																																																						
11	1/19/2017	805.12	809.48	801.80	802.17	919,300																																																																																																						
12	1/20/2017	806.91	806.91	801.69	805.02	1,670,000																																																																																																						
13	1/23/2017	807.25	820.87	803.74	819.31	1,963,600																																																																																																						

17. Write a program to implement GAN

```
from __future__ import print_function, division
from keras.datasets import mnist
from keras.layers import Input, Dense, Reshape, Flatten, Dropout
from keras.layers import BatchNormalization, Activation, ZeroPadding2D
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
```

```

import matplotlib.pyplot as plt
import sys
import numpy as np
class GAN():
    def __init__(self):
        self.img_rows = 28
        self.img_cols = 28
        self.channels = 1
        self.img_shape = (self.img_rows, self.img_cols, self.channels)
        self.latent_dim = 100
        optimizer = Adam(0.0002, 0.5)
        # Build and compile the discriminator
        self.discriminator = self.build_discriminator()
        self.discriminator.compile(loss='binary_crossentropy',
            optimizer=optimizer,
            metrics=['accuracy'])
        # Build the generator
        self.generator = self.build_generator()
        # The generator takes noise as input and generates imgs
        z = Input(shape=(self.latent_dim,))
        img = self.generator(z)
        # For the combined model we will only train the generator
        self.discriminator.trainable = False
        # The discriminator takes generated images as input and determines
        validity
        validity = self.discriminator(img)
        # The combined model (stacked generator and discriminator)
        # Trains the generator to fool the discriminator
        self.combined = Model(z, validity)
        self.combined.compile(loss='binary_crossentropy',

```

```

optimizer=optimizer)

def build_generator(self):
model = Sequential()
model.add(Dense(256, input_dim=self.latent_dim))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(512))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(1024))
model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8))
model.add(Dense(np.prod(self.img_shape), activation='tanh'))
model.add(Reshape(self.img_shape))
model.summary()
noise = Input(shape=(self.latent_dim,))
img = model(noise)
return Model(noise, img)

def build_discriminator(self):
model = Sequential()
model.add(Flatten(input_shape=self.img_shape))
model.add(Dense(512))
model.add(LeakyReLU(alpha=0.2))
model.add(Dense(256))
model.add(LeakyReLU(alpha=0.2))
model.add(Dense(1, activation='sigmoid'))
model.summary()
img = Input(shape=self.img_shape)
validity = model(img)
return Model(img, validity)

```



```

def train(self, epochs, batch_size=128, sample_interval=50):
# Load the dataset
(X_train, _), (_, _) = mnist.load_data()

# Rescale -1 to 1
X_train = X_train / 127.5 - 1.

X_train = np.expand_dims(X_train, axis=3)

# Adversarial ground truths
valid = np.ones((batch_size, 1))
fake = np.zeros((batch_size, 1))

for epoch in range(epochs):
# -----
# Train Discriminator
# -----

# Select a random batch of images
idx = np.random.randint(0, X_train.shape[0], batch_size)
imgs = X_train[idx]

noise = np.random.normal(0, 1, (batch_size, self.latent_dim))

# Generate a batch of new images
gen_imgs = self.generator.predict(noise)

# Train the discriminator
d_loss_real = self.discriminator.train_on_batch(imgs, valid)
d_loss_fake = self.discriminator.train_on_batch(gen_imgs, fake)
d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

# -----
# Train Generator
# -----

noise = np.random.normal(0, 1, (batch_size, self.latent_dim))

# Train the generator (to have the discriminator label samples as valid)
g_loss = self.combined.train_on_batch(noise, valid)

# Plot the progress

```

```

print ("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" % (epoch, d_loss[0],
100*d_loss[1], g_loss))

# If at save interval => save generated image samples
if epoch % sample_interval == 0:
    self.sample_images(epoch)
    def sample_images(self, epoch):
        r, c = 5, 5
        noise = np.random.normal(0, 1, (r * c, self.latent_dim))
        gen_imgs = self.generator.predict(noise)
        # Rescale images 0 - 1
        gen_imgs = 0.5 * gen_imgs + 0.5
        fig, axs = plt.subplots(r, c)
        cnt = 0
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(gen_imgs[cnt, :, :, 0], cmap='gray')
                axs[i,j].axis('off')
            cnt += 1
        fig.savefig("/content/sample_data/d.jpg" % epoch)
        plt.close()
    gan = GAN()
    gan.train(epochs=200, batch_size=32, sample_interval=200)

```

Output:

```
✓ 19s ▶ gan = GAN()
gan.train(epochs=200, batch_size=32, sample_interval=200)
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_7 (Dense)	(None, 512)	401920
leaky_re_lu_5 (LeakyReLU)	(None, 512)	0
dense_8 (Dense)	(None, 256)	131328
leaky_re_lu_6 (LeakyReLU)	(None, 256)	0
dense_9 (Dense)	(None, 1)	257

=====
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0

Model: "sequential_3"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

18. Web scraping experiments (by using tools)

```
import urllib
import urllib.request
# create the reponse object
```

```

response =
urllib.request.urlopen('https://en.wikipedia.org/wiki/Rajgad_Fort')
response
html = response.read()
print(html)
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'html')
data = soup.get_text(strip=True)
data
images = soup.find_all('img')
images[3]
images[3]['title']
images[3]['src']
soup.title
soup.title.string
text = [x for x in data.split()]
import nltk
frq = nltk.FreqDist(text)
frq.plot(20, cumulative=False)
from nltk.corpus import stopwords
swords = stopwords.words('english')
clean_tokens = []
for x in text:
    if x.lower() not in swords:
        clean_tokens.append(x.lower())
clean_tokens
frq = nltk.FreqDist(clean_tokens)
frq.plot(20, cumulative=False)
from nltk.stem import PorterStemmer
ps = PorterStemmer()

```

```
clean_tokens = [ps.stem(x) for x in clean_tokens]
```

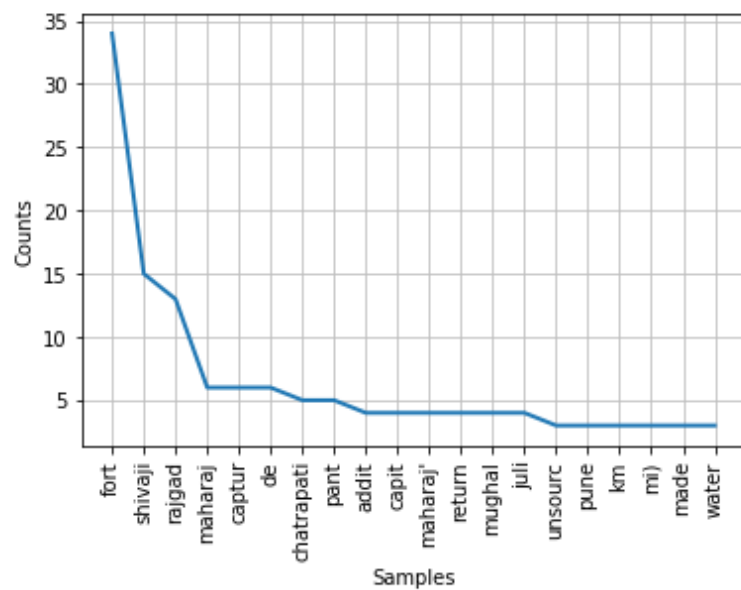
```
frq = nltk.FreqDist(clean_tokens)
```

```
frq.plot(20, cumulative=False)
```

OUTPUT:

```
[ ] clean_tokens
```

```
['rajgad',  
'fort',  
'_',  
'wikipediarajgad',  
'fortfrom',  
'wikipedia',  
'free',  
'encyclopediajump',  
'navigationjump',  
'searchthis',  
'articleneeds',  
'additional',  
'citations',  
'forverification.please',  
'helpimprove',  
'articlebyadding',  
'citations',  
'reliable',  
'sources.',  
'unsourced',  
'material',  
'may',  
'challenged',  
'removed.find',  
'sources:"rajgad',
```



<AxesSubplot:xlabel='Samples', ylabel='Counts'>