Practical 1

1.1. Numpy **Date: / /2022**

Aim:

1. **Creating blank array, with predefined data, with pattern specific data**

Code:

```
import numpy as np
#initalize 16 elements in a 1-D array
a = np.arange(16)
a
#type of array
a.dtype
```

Output:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])

dtype('int64')
```

2. **Slicing and Updating elements**

Code:

```
#slicing
#Basic slicing

c = a[1:4]
c

#Reverse slicing
d= a[::-1]
d

#updating elements

g = g*10 - 10
g
```

Output:

```
array([1, 2, 3])
```

```
array([15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0])
```

```
array([[-10,   0,  10,  20],
       [ 30,  40,  50,  60],
       [ 70,  80,  90, 100],
       [110, 120, 130, 140]])
```

3. Shape manipulations

Code:

```
# Shape manipulation
# 1D to 2D
a = np.array([0,5,10,15,20,25,30,35,40,45,50,55])
# A = a.reshape(3,4)
print(a.reshape(3,4))
# 2D to 1D
# A = A.ravel()
print(a.ravel())
```

Output:

```
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]
[ 0  5 10 15 20 25 30 35 40 45 50 55]
```

4. Looping over arrays

Code:

```
#Looping
#Print all elements

for x in a:
  print(x, end = ' ')

#Loop, print only even elements

for x in a:
  if (x%2==0):
    print(x, end = ' ')
```

Output:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

0 2 4 6 8 10 12 14
```

5. Reading files in numpy

Code:

```
#file
#after uploading file
with open("numpyread.txt", "r") as f:
  a = f.read()
  b = np.array(list(a.replace(" ", "")), dtype=int)
  print(b)
```

Output:

```
[1 2 4 5 6 7 8]
```

6. Use numpy vs list for matrix multiplication of 1000 X 1000 array and evaluate computing

performance.

Code:

```
#System Module
import sys

#Declaring 2 lists of 1000 elements
print("Declaring 2 lists of 1000 elements")
list1 = range(1000)
list2 = range(1000,2000)
print(list1)
print(list2)
print("Size of each element of list1 in bytes: ", sys.getsizeof(list1))
print("Size of whole list1 in bytes: ", sys.getsizeof(list1)*len(list1))
print("Size of each element of list2 in bytes: ", sys.getsizeof(list2))
print("Size of whole list2 in bytes: ", sys.getsizeof(list1)*len(list2))

#Declaring 2 arrays of 1000 elements
print("\n\nDeclaring 2 arrays of 1000 elements")
arr1 = np.arange(1000)
arr2 = np.arange(1000,2000)
print(arr1)
print(arr2)
print("Size of each element of the Numpy Array1 in bytes: ", arr1.itemsize)
print("Size of the whole Numpy array in bytes: ", arr1.size*arr1.itemsize)
print("Size of each element of the Numpy Array2 in bytes: ", arr2.itemsize)
print("Size of the whole Numpy Array2 in bytes: ", arr2.size*arr2.itemsize)


import time

#Capturing time before multiplication of Python Lists
initialTime1 = time.time()

list3 = [(a*b) for a,b in zip(list1,list2)]

#Calculating execution time
print("Time taken by 2 Lists to perform multiplication: ", (time.time() - initialTime1), "seconds")

#Capturing time before multiplication of Numpy Arrays
initialTime2 = time.time()

arr3 = arr1*arr2

print("Time taken by 2 Arrays to perform multiplication: ", (time.time() - initialTime2), "seconds")
```

Output:

```
Time taken by 2 Lists to perform multiplication:  0.0003693103790283203 seconds
Time taken by 2 Arrays to perform multiplication:  0.00024008750915527344 seconds
```

1.2. Pandas

Aim:

1. **Creating data frame**

Code:

```
import pandas as pd
data = [10,20,30,40,50,60]
df = pd.DataFrame(data, columns=['Numbers'])
df
```

Output:

	Numbers
0	10
1	20
2	30
3	40
4	50
5	60

2. **Reading files**

Code:

```
data = pd.read_csv('/content/iris.data.csv')
data.head()
```

Output:

	5.1	3.5	1.4	0.2	Iris-setosa
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

3. Slicing manipulations

Code:

```
student = pd.DataFrame({'Name': ['A', 'B', 'C','D', 'E', 'F','G', 'H'],
                        'Score': [65, 70, 75, 80, 85, 90, 95, 100]})
print(student.loc[0:5, 'Name'])
print(student.iloc[0:5, 0:2])
```

Output:

```
0    A
1    B
2    C
3    D
4    E
5    F
Name: Name, dtype: object
```

```
  Name  Score
0    A     65
1    B     70
2    C     75
3    D     80
4    E     85
```

4. Exporting data files

Code:

```
# First: create your Data Frames
student = pd.DataFrame({'Name': ['Maaz', 'Krish', 'Riya','Kunal', 'Kartik',
 'Rohan','Frenny', 'Sahil'],
                        'Score': [96, 69, 70, 88, 79, 64, 62, 57]})
student

# Second: exporting/saving our DataFrame 'student' into CSV file
student_data_csv = student.to_csv('Student_Score.csv', index=True)

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

Output:

	Unnamed: 0	Name	Score
0	0	A	65
1	1	B	70
2	2	C	75
3	3	D	80
4	4	E	85
5	5	F	90
6	6	G	95
7	7	H	100

5. Columns and row manipulations with loops

Code:

```
#IterTuples
for i in country.itertuples():
print(i)
```

Output:

```
Pandas(Index=0, Country='Russia', Rank=121)
Pandas(Index=1, Country='Colombia', Rank=40)
Pandas(Index=2, Country='Chile', Rank=100)
Pandas(Index=3, Country='Equador', Rank=130)
Pandas(Index=4, Country='Nigeria', Rank=11)
```

6. Use pandas for masking data and reading if in Boolean format.

Code:

```
df = pd.DataFrame({"A": [1,None,3,4,5],
            "B": [7,4,1,2,8],
            "C": [9,6,3,2,1],
            "D": [8,7,4,None,3]})
df
```

Output:

	A	B	C	D
0	1.0	7	9	8.0
1	NaN	4	6	7.0
2	3.0	1	3	4.0
3	4.0	2	2	NaN
4	5.0	8	1	3.0

1.3. Matplotlib

Aim:

1. Importing matplotlib

Code:

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

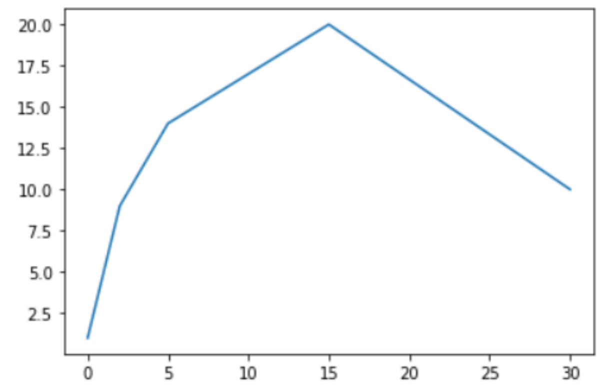
2. Importing matplotlib

Code:

```
#simple line chart
xpoints = np.array([0,2,5, 15,30])
ypoints = np.array([1,9,14, 20, 10])
plt.plot(xpoints,ypoints)
plt.show
```

Output:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



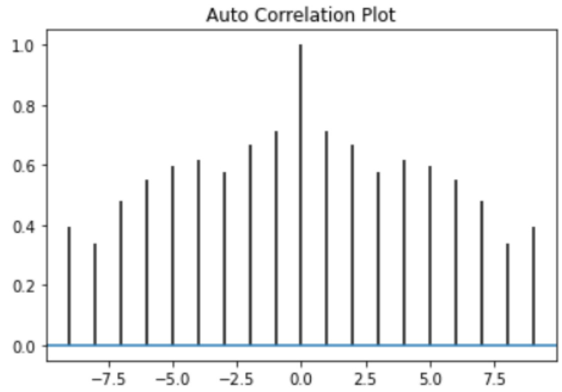
3. Correlation chart

Code:

```
#correlation chart
data = np.array([12.0, 24.0, 7., 20.0,
            7.0, 22.0, 18.0,22.0,
            6.0, 7.0, 20.0, 13.0,
            8.0, 5.0, 8,10.0,15.0,25.0])

plt.title("Auto Correlation Plot")
plt.acorr(data, maxlags = 9)
```

Output:

```
(array([-9, -8, -7, -6, -5, -4, -3, -2, -1,  0,  1,  2,  3,  4,  5,  6,  7,
         8,  9]),
 array([0.39512538, 0.33630185, 0.481603  , 0.55050387, 0.59760956,
        0.61823295, 0.57815796, 0.66674479, 0.71361612, 1.        ,
        0.71361612, 0.66674479, 0.57815796, 0.61823295, 0.59760956,
        0.55050387, 0.481603  , 0.33630185, 0.39512538]),
 <matplotlib.collections.LineCollection at 0x7f1179f300a0>,
 <matplotlib.lines.Line2D at 0x7f1179fa37f0>)
```



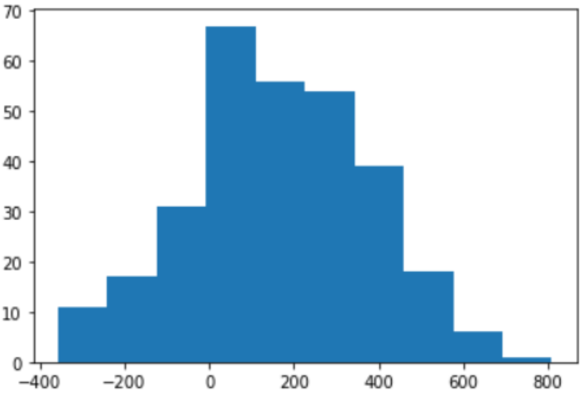
4. Histogram

Code:

```
#histogram
x = np.random.normal(150,200,300)
plt.hist(x)
```

Output:

```
(array([11., 17., 31., 67., 56., 54., 39., 18.,  6.,  1.]),
 array([-355.87379496, -239.45484555, -123.03589613,   -6.61694672,
         109.8020027 ,  226.22095212,  342.63990153,  459.05885095,
         575.47780036,  691.89674978,  808.31569919]),
 <a list of 10 Patch objects>)
```

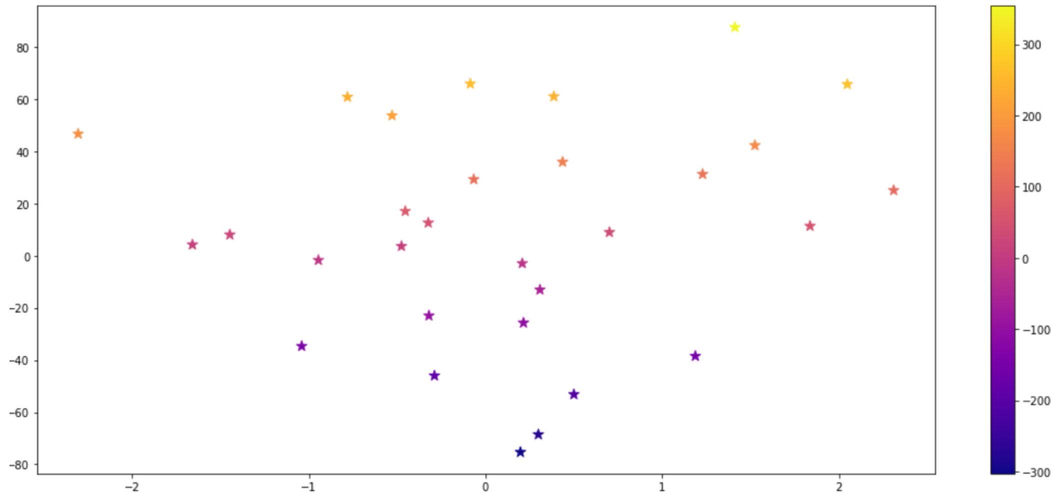


5. Plotting of Multivariate data

Code:

```
#Multivariate data
plt.rcParams['figure.figsize'] = [15, 6.5]
plt.rcParams['figure.autolayout'] = True

def func(x, y):
  return 3 * x + 4 * y - 2 + np.random.randn(30)

x, y = np.random.randn(2, 30)
y *= 50
z = func(x, y)

fig, ax = plt.subplots()
s = ax.scatter(x,y, c=z, s=100, marker ='*', cmap = 'plasma')

fig.colorbar(s)

plt.show()
```

Output:

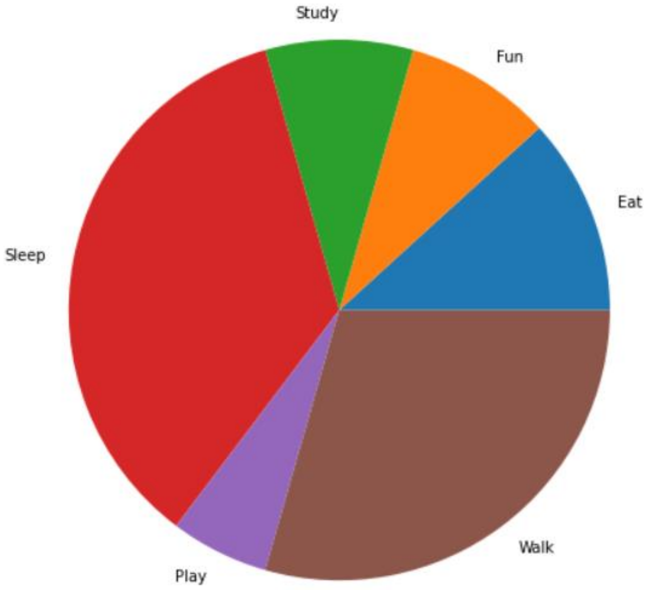


6. Plot Pi Chart

Code:

```
#pi chart
y = np.array([20,15,15,60,10, 50])
mylabels = ["Eat", "Fun", "Study", "Sleep", "Play", "Walk"]

plt.pie(y, labels = mylabels)
```

Output:



Conclusion/Summary:

Student Signature & Date	Marks:	Evaluator Signature & Date

Practical 2

2. Linear Regression	**Date: / /2022**

Aim:

Select the Dataset of your choice and respond to following questions.

- **Why do you want to apply regression on selected dataset? Discuss the full story behind the dataset.**
- **How many total observations in data?**
- **How many independent variables?**
- **Which is dependent variable?**
- **Which are most useful variable in estimation? Prove using correlation.**
- **Implement linear regression using OLS method.**
- **Implement linear regression using Gradient Descent from scratch.**
- **Implement linear regression using sklearn API.**
- **Quantify goodness of your model and discuss steps taken for improvement (RMSE, SSE, R2Score).**
- **Discuss comparison of different methods.**

Solution:

1) Why do you want to apply regression on selected dataset? Discuss the full story behind the dataset?

Answer: Consider you own an ice cream business and you would like to create a model that could predict the daily revenue in dollars based on the outside air temperature (degC). So to make this kind of prediction where we want have an input parameter aka outside temperate(DegC) and revenue that can be generated as our output it is best choice to use a linear regression model to extrapolate the results and cater them to our needs.

Independent variable X: Outside Air Temperature

Dependant variable Y: Overall daily revenue generated in dollars

2) How many total observations in data?

Answer: There are total 500 observations in the data set

Code:

```
data = pd.read_csv("IceCreamData.csv")

data
```

Output:

```
data = pd.read_csv("IceCreamData.csv")
data
```

	Temperature	Revenue
0	24.566884	534.799028
1	26.005191	625.190122
2	27.790554	660.632289
3	20.595335	487.706960
4	11.503498	316.240194
...	...	...
495	22.274899	524.746364
496	32.893092	755.818399
497	12.588157	306.090719
498	22.362402	566.217304
499	28.957736	655.660388

500 rows × 2 columns

3) How many independent variables?

Answer: There is one independent variable as visible which is Independent variable X: Outside Air Temperature

4) Which are most useful variable in estimation? Prove using correlation.

Answer: The most useful variable from our dataset is Temperature.

5) Implement linear regression using sklearn API.

Answer: In sklearn library in Python linear regression is implemented using OLS method.

Code:

#Split 80% for training and 20% for testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.2, random_state=0)

#Create Linear Regressor and fit data

```
reg = LinearRegression(fit_intercept = True)

reg.fit(X_train.values,y_train.values)


#Obtaining best-fit Line

print('Linear coefficient is=' , reg.coef_)

print('Intercept is=' , reg.intercept_)


#prediction

y_predict = reg.predict(X_test)

print(y_predict)
```

Output:

```
[ ] #Split 80% for training and 20% for testing

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.2, random_state=0)
```

```
[ ] #Create Linear Regressor and fit data

    reg = LinearRegression(fit_intercept = True)
    reg.fit(X_train.values,y_train.values)
```

```
LinearRegression()
```

```
[ ] #Obtaining best-fit Line

    print('Linear coefficient is=' , reg.coef_)
    print('Intercept is=' , reg.intercept_)
```

```
Linear coefficient is= [[21.5133908]]
Intercept is= [43.73357869]
```

Prediction

```
y_predict = reg.predict(X_test)
print(y_predict)
```

```
[623.82532723]
[667.48717467]
[468.72433832]
[546.82733151]
[443.41191785]
[622.95162777]
[377.64639971]
[367.0607334 ]
[945.67057977]
[893.79551974]
[694.45445099]
[546.05047608]
[420.58523672]
[391.08500303]
[597.0141581 ]
[283.23582775]
[655.50055011]
[380.98796154]
[412.31810124]
[371.05055651]
[510.23910289]
[479.70270426]
[456.68206658]
[640.1157508 ]
[281.65224383]
[314.1894674 ]
[470.01363777]
[559.72453055]
[539.75091165]
[307.72368191]
[508.65180339]
[571.43237276]
```

6) Quantify goodness of your model and discuss steps taken for improvement (RMSE, SSE, R2Score)

Answer:

Code:

pred_values = reg.predict(X_test.values)

mae = metrics.mean_absolute_error(y_test, pred_values)

rmse = np.sqrt(mse)

r2 = metrics.r2_score(y_test, pred_values)

print('Results')

print("MAE:",mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R-Squared:", r2)

Output:

```
pred_values = reg.predict(X_test.values)

mae = metrics.mean_absolute_error(y_test, pred_values)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test, pred_values)

print('Results')
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results
MAE: 18.303213530102884
MSE: 528.2150684519337
RMSE: 22.982929936192505
R-Squared: 0.9837324255882577
```

Conclusion/Summary:		
Student Signature & Date	**Marks:**	**Evaluator Signature & Date**

Practical 3

3. Logistic Regression	**Date: / /2023**

Aim:

Select Dataset of your choice and respond to following questions.

- Why you want to apply classification on selected dataset? Discuss full story behind dataset.

- How many total observations in data?

- How many independent variables?

- Which is dependent variable?

- Which are most useful variable in classification? Prove using correlation.

- Imlement Logistic regression using sklearn

Solution:

1) Why do you want to apply regression on selected dataset? Discuss the full story behind the Dataset.

Answer: Consider you create a model that could predict that the Person has been placed or not. We have input parameters like ssc_p, hsc_p, degree_p, hsc_s, specialization for building the model.

2) How many total observations in data?

Answer: There are total 215 observations in the data set

3) How many independent variables?

Answer: There are 12 independent variables

4) Which is dependent variable? Answer:

Status is dependent variable.

Which is dependent variable? Answer: Status is dependent variable.

5) Implement logistic regression using sklearn.

```
[3] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
```

```
[4] from google.colab import files
    uploaded = files.upload()
```

```
Choose Files Placement_...ll_class.csv
• Placement_data_full_class.csv(text/csv) - 18185 bytes, last modified: 2/27/2023 - 100% done
Saving Placement_data_full_class.csv to Placement_data_full_class.csv
```

```
[5] df = pd.read_csv('Placement_data_full_class.csv')
    df
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	NaN
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

```
[8] df = df.drop('sl_no', axis=1)
    df = df.drop('salary', axis=1)
```

```
df["gender"] = df["gender"].astype('category')
df["ssc_b"] = df["ssc_b"].astype('category')
df["hsc_b"] = df["hsc_b"].astype('category')
df["degree_t"] = df["degree_t"].astype('category')
df["workex"] = df["workex"].astype('category')
df["specialisation"] = df["specialisation"].astype('category')
df["status"] = df["status"].astype('category')
df["hsc_s"] = df["hsc_s"].astype('category')
df.dtypes
```

```
gender            category
ssc_p              float64
ssc_b             category
hsc_p              float64
hsc_b             category
hsc_s             category
degree_p           float64
degree_t          category
workex            category
etest_p            float64
specialisation    category
mba_p              float64
status            category
dtype: object
```

```
df["gender"] = df["gender"].cat.codes
df["ssc_b"] = df["ssc_b"].cat.codes
df["hsc_b"] = df["hsc_b"].cat.codes
df["degree_t"] = df["degree_t"].cat.codes
df["workex"] = df["workex"].cat.codes
df["specialisation"] = df["specialisation"].cat.codes
df["status"] = df["status"].cat.codes
df["hsc_s"] = df["hsc_s"].cat.codes

df
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status
0	1	67.00	1	91.00	1	1	58.00	2	0	55.0	1	58.80	1
1	1	79.33	0	78.33	1	2	77.48	2	1	86.5	0	66.28	1
2	1	65.00	0	68.00	0	0	64.00	0	0	75.0	0	57.80	1
3	1	56.00	0	52.00	0	2	52.00	2	0	66.0	1	59.43	0
4	1	85.80	0	73.60	0	1	73.30	0	0	96.8	0	55.50	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	1	80.60	1	82.00	1	1	77.60	0	0	91.0	0	74.49	1
211	1	58.00	1	60.00	1	2	72.00	2	0	74.0	0	53.62	1
212	1	67.00	1	67.00	1	1	73.00	0	1	59.0	0	69.72	1

```
X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values

Y
```

```
array([1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0], dtype=int8)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

df.head()
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status
0	1	67.00	1	91.00	1	1	58.00	2	0	55.0	1	58.80	1
1	1	79.33	0	78.33	1	2	77.48	2	1	86.5	0	66.28	1
2	1	65.00	0	68.00	0	0	64.00	0	0	75.0	0	57.80	1
3	1	56.00	0	52.00	0	2	52.00	2	0	66.0	1	59.43	0
4	1	85.80	0	73.60	0	1	73.30	0	0	96.8	0	55.50	1

```
[18] from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state=0, solver='lbfgs', max_iter=1000).fit(X_train, Y_train)

model.score(X_test, Y_test)

0.8604651162790697
```

```
model.predict([[0, 87, 0, 95, 0, 2, 78, 2, 0, 0, 1, 0]])
```

```
array([1], dtype=int8)
```

```
[20] Y_pred = model.predict(X_test)

     Y_pred
```

```
array([1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1],
      dtype=int8)
```

```
[22] from sklearn.metrics import confusion_matrix, accuracy_score

     print(confusion_matrix(Y_test, Y_pred))
     print(accuracy_score(Y_test, Y_pred))
```

```
[[ 9  3]
 [ 3 28]]
0.8604651162790697
```

Conclusion/Summary:

Student Signature & Date	Marks:	Evaluator Signature & Date

Practical 4

4.KNN	**Date: / /2023**

Aim:

Multi Class Classification (KNN)
Select Dataset of your choice and respond to following questions.
- Why you want to apply classification on selected dataset? Discuss full story behind dataset.
- How many total observations in data?
- How many independent variables?
- Which is dependent variable?
- Which is the most useful variable in classification? Prove using correlation.
- Implement KNN using sklearn api.
- Implement code to find best value of k by splitting data in train and test.
- Quantify goodness of your model and discuss steps taken for improvement.
- Can we use KNN for regression also? Why / Why not?
- Discuss drawbacks of algorithms such as KNN.

Code:

1) Why you want to apply classification on selected dataset? Discuss full story behind dataset.
Answer: Suppose you own a wine making & exporting business, then depending on the attributes you want to classify the wine in class 0/1/2. Number of instances are 178. Class distribution is - class_0 (59), class_1 (71), class_2 (48).
It has attributes - 13 numeric, predictive attributes and the class. 13 numeric attributes as follows:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

2) How many total observations in data?
Answer: There are total 178 observations in the data set

3) How many independent variables?
Answer: There are 13 independent variable which are all numeric attributes.

4) Which is dependent variable?
Answer: There are 2 dependent variables which are predictive attributes & class.

5) Which are most useful variable in estimation? Prove using correlation.
Answer: The most useful variable from our dataset are numeric attributes.

6) Implement KNN using sklearn api.

Code: Knowing our data

```
import pandas as pd
from sklearn.datasets import load_wine
wine = load_wine()
```

```
wine.feature_names
```

```
['alcohol',
 'malic_acid',
 'ash',
 'alcalinity_of_ash',
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline']
```

```
[ ] wine.target_names
```

```
array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

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

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

```
[ ] df['target'] = wine.target
df.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

```
df[df.target==1].head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	prolin
59	12.37	0.94	1.36	10.6	88.0	1.98	0.57	0.28	0.42	1.95	1.05	1.82	520.
60	12.33	1.10	2.28	16.0	101.0	2.05	1.09	0.63	0.41	3.27	1.25	1.67	680.
61	12.64	1.36	2.02	16.8	100.0	2.02	1.41	0.53	0.62	5.75	0.98	1.59	450.
62	13.67	1.25	1.92	18.0	94.0	2.10	1.79	0.32	0.73	3.80	1.23	2.46	630.
63	12.37	1.13	2.16	19.0	87.0	3.50	3.10	0.19	1.87	4.45	1.22	2.87	420.

[ ] df[df.target==2].head()

```
[ ] df[df.target==2].head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proli
130	12.86	1.35	2.32	18.0	122.0	1.51	1.25	0.21	0.94	4.10	0.76	1.29	63(
131	12.88	2.99	2.40	20.0	104.0	1.30	1.22	0.24	0.83	5.40	0.74	1.42	53(
132	12.81	2.31	2.40	24.0	98.0	1.15	1.09	0.27	0.83	5.70	0.66	1.36	56(
133	12.70	3.55	2.36	21.5	106.0	1.70	1.20	0.17	0.84	5.00	0.78	1.29	60(
134	12.51	1.24	2.25	17.5	85.0	2.00	0.58	0.60	1.25	5.45	0.75	1.51	65(

```
[ ] df['wine_name'] =df.target.apply(lambda x: wine.target_names[x])
    df.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

```
df[123:133]
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proli
123	13.05	5.80	2.13	21.5	86.0	2.62	2.65	0.30	2.01	2.60	0.73	3.10	380
124	11.87	4.31	2.39	21.0	82.0	2.86	3.03	0.21	2.91	2.80	0.75	3.64	380
125	12.07	2.16	2.17	21.0	85.0	2.60	2.65	0.37	1.35	2.76	0.86	3.28	378
126	12.43	1.53	2.29	21.5	86.0	2.74	3.15	0.39	1.77	3.94	0.69	2.84	352
127	11.79	2.13	2.78	28.5	92.0	2.13	2.24	0.58	1.76	3.00	0.97	2.44	466
128	12.37	1.63	2.30	24.5	88.0	2.22	2.45	0.40	1.90	2.12	0.89	2.78	342
129	12.04	4.30	2.38	22.0	80.0	2.10	1.75	0.42	1.35	2.60	0.79	2.57	580
130	12.86	1.35	2.32	18.0	122.0	1.51	1.25	0.21	0.94	4.10	0.76	1.29	630
131	12.88	2.99	2.40	20.0	104.0	1.30	1.22	0.24	0.83	5.40	0.74	1.42	530
132	12.81	2.31	2.40	24.0	98.0	1.15	1.09	0.27	0.83	5.70	0.66	1.36	560

\+ Code + Text

```
[ ] df[:60]
    df[60:131]
    df[131:]
```

7) Implement code to find best value of k by splitting data in train and test

Code: Training and testing

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] X = df.drop(['target','target==0','wine_name'], axis='columns')
    y = df.target
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

```
[ ] len(X_train)

142
```

```
[ ] len(X_test)

36
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=20)
```

```
[ ] knn.fit(X_train, y_train)

KNeighborsClassifier(n_neighbors=20)
```

```
[ ] knn.predict([[14.57,2.55,2.89,20.3,102.0,2.87,4.66,0.34,2.50,8.12,1.77,4.02,1500.02]])

/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(
array([0])
```

8) Quantify goodness of your model and discuss steps taken for improvement.

Code: Score and accuracy

```
knn.score(X_test, y_test)
```

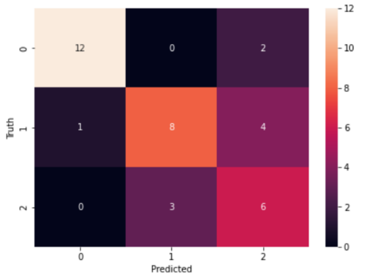
```
0.7222222222222222
```

```
from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[12,  0,  2],
       [ 1,  8,  4],
       [ 0,  3,  6]])
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(7,5))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(42.0, 0.5, 'Truth')
```




```
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

           0       0.92      0.86      0.89        14
           1       0.73      0.62      0.67        13
           2       0.50      0.67      0.57         9

    accuracy                           0.72        36
   macro avg       0.72      0.71      0.71        36
weighted avg       0.75      0.72      0.73        36
```

9) Can we use KNN for regression also? Why / Why not?
Answer: Yes, KNN can also be used for regression in the same way we do it for classification as KNN works best for numeric values.

10) Discuss drawbacks of algorithms such as KNN.
Answer: Disadvantages of KNN

1. Does not work well with large dataset: In large datasets, the cost of calculating the distance between the new point and each existing point is huge which degrades the performance of the algorithm.

2. Does not work well with high dimensions: The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.

3. Need feature scaling: We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.

4. KNN is sensitive to noise in the dataset. We need to manually impute missing values and remove outliers.

Conclusion/Summary:		
Student Signature & Date	**Marks:**	**Evaluator Signature & Date**

Practical 5

Date: / /2023

Aim: Find a dataset with number of samples smaller than number of features. Apply principle component analysis to select K best features.
Use Support Vector Machines/Naïve Bayes to train predictive model. Compare model accuracy and time required for training with full dataset and with selected K features. (use Sci-kit-learn library)

Solution:

"Wine Quality" dataset from the UCI Machine Learning Repository is being used. This dataset contains 1599 instances and 11 attributes describing various properties of different wines. The goal is to predict the quality of the wine on a scale of 0 to 10.

```
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import time

data = load_wine()
X = data.data
y = data.target


K = 5
pca = PCA(n_components=K)
X_pca = pca.fit_transform(X)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2,
random_state=42)



svm = SVC()
start_time = time.time()
svm.fit(X_train, y_train)
train_time = time.time() - start_time

y_pred = svm.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy with full dataset:", acc)
print("Training time with full dataset:", train_time)
```

```
Accuracy with full dataset: 0.8055555555555556
Training time with full dataset: 0.016330480575561523
```

```
svm_pca = SVC()
start_time = time.time()
svm_pca.fit(X_train_pca, y_train_pca)
train_time_pca = time.time() - start_time

y_pred_pca = svm_pca.predict(X_test_pca)
acc_pca = accuracy_score(y_test_pca, y_pred_pca)
print("Accuracy with PCA-selected features:", acc_pca)
print("Training time with PCA-selected features:", train_time_pca)
```

```
Accuracy with PCA-selected features: 0.7777777777777778
Training time with PCA-selected features: 0.002498626708984375
```

```
nb = GaussianNB()
start_time = time.time()
nb.fit(X_train, y_train)
train_time = time.time() - start_time


y_pred = nb.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy with full dataset:", acc)
print("Training time with full dataset:", train_time)
```

```
Accuracy with full dataset: 1.0
Training time with full dataset: 0.012969017028808594
```

```
nb_pca = GaussianNB()
start_time = time.time()
nb_pca.fit(X_train_pca, y_train_pca)
train_time_pca = time.time() - start_time

y_pred_pca = nb_pca.predict(X_test_pca)
acc_pca = accuracy_score(y_test_pca, y_pred_pca)
print("Accuracy with PCA-selected features:", acc_pca)
print("Training time with PCA-selected features:", train_time_pca)
```

```
Accuracy with PCA-selected features: 1.0
Training time with PCA-selected features: 0.0018205642700195312
```

Conclusion/Summary:		
Student Signature & Date	**Marks:**	**Evaluator Signature & Date**

Practical 6

6. Decision Tree	Date: / /2023
Aim: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.	

Solution:

Importing Necessary Libraries.

```
import numpy as np
import math
import csv
```

To Read Data from CSV File.

```
def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)
```

Creating a node class which can be used to create a tree-like structure where each node represents an attribute or decision point.

```
class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute
```

Splitting the data recursively based on different features to construct the decision tree.

```
def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict
```

Selecting the best feature to split the data at each node of the decision tree, feature with lowest entropy.

```
def entropy(S):
    items = np.unique(S)

    if items.size == 1:
        return 0

    counts = np.zeros((items.shape[0], 1))
    sums = 0

    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums
```

Selecting the best feature to split the data at each node of the decision tree, feature with highest gain ratio.

```
def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy / iv
```

Creating a function which recursively creates a decision tree by dividing the data into sub tables based on the highest gain ratio.

```
def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    gains = np.zeros((data.shape[1] - 1, 1))

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)

    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)

    items, dict = subtables(data, split, delete=True)

    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node
```

Defining two functions that can be used to print a decision tree in a readable format.

```
def empty(size):
    s = ""
    for x in range(size):
        s += "    "
    return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
    print(empty(level), node.attribute)
    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)
```

This code reads in a dataset using the read_data function and creates a decision tree using the create_node function. It then prints the decision tree using the print_tree function.

```
metadata, traindata = read_data("/content/play_tennis.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

```
 day
    D1
        b'No'
    D10
        b'Yes'
    D11
        b'Yes'
    D12
        b'Yes'
    D13
        b'Yes'
    D14
        b'No'
    D2
        b'No'
    D3
        b'Yes'
    D4
        b'Yes'
    D5
        b'Yes'
    D6
        b'No'
    D7
        b'Yes'
    D8
        b'No'
    D9
        b'Yes'
```

Conclusion/Summary:		
Student Signature & Date	Marks:	Evaluator Signature & Date

Practical 7

7. Principle Component Analysis	Date: / /2023
Aim: Practical Implementation of Principle Component Analysis(PCA).	

We are using twitter US airlines for text classification using RNN and LSTM according to sentiments of text

Solution:

IMPORTING DATASET

```
[1] from sklearn.datasets import load_digits
    import pandas as pd

    dataset = load_digits()
    dataset.keys()

    dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

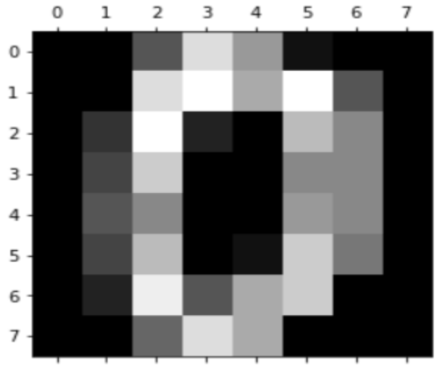
```
[2] dataset.data.shape

    (1797, 64)
```

```
[3] dataset.data[0]

    array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
           15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
           12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
            0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
           10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

DATA VISUALIZATION

```
from matplotlib import pyplot as plt
%matplotlib inline
plt.gray()
plt.matshow(dataset.data[0].reshape(8,8))
```

```
<matplotlib.image.AxesImage at 0x7ff9f83d8340>
<Figure size 432x288 with 0 Axes>
```




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

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_7	pixel_7_0	pixel_7_1	pixel_7_2	pixel_7_3	pi
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	

5 rows × 64 columns

```
[7] df.describe()
```

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_7	pixel_7_0	pixel_7_1
count	1797.0	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	...	1797.000000	1797.000000	1797.000000	1797.000000
mean	0.0	0.303840	5.204786	11.835838	11.848080	5.781859	1.362270	0.129661	0.005565	1.993879	...	3.725097	0.206455	0.000556	0.279354
std	0.0	0.907192	4.754826	4.248842	4.287388	5.666418	3.325775	1.037383	0.094222	3.196160	...	4.919406	0.984401	0.023590	0.934302
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
25%	0.0	0.000000	1.000000	10.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
50%	0.0	0.000000	4.000000	13.000000	13.000000	4.000000	0.000000	0.000000	0.000000	0.000000	...	1.000000	0.000000	0.000000	0.000000
75%	0.0	0.000000	9.000000	15.000000	15.000000	11.000000	0.000000	0.000000	0.000000	3.000000	...	7.000000	0.000000	0.000000	0.000000
max	0.0	8.000000	16.000000	16.000000	16.000000	16.000000	16.000000	15.000000	2.000000	16.000000	...	16.000000	13.000000	1.000000	9.000000

SCALING THE DATA AND THEN SPLITTING IT USING TRAIN TEST SPLIT FUNCTION

```python
X = df
y = dataset.target
```

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

```
array([[ 0.        , -0.33501649, -0.04308102, ..., -1.14664746,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  0.54856067,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  1.56568555,
         1.6951369 , -0.19600752],
       ...,
       [ 0.        , -0.33501649, -0.88456568, ..., -0.12952258,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -0.67419451, ...,  0.8876023 ,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649,  1.00877481, ...,  0.8876023 ,
        -0.26113572, -0.19600752]])
```

```python
[11] from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=30)
```

USING LOGISTIC REGRESSION FOR DIGITS CLASSIFICATION

```python
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

```
0.9722222222222222
```

Use PCA to reduce dimensions

```python
In [88]: X
```

Out[88]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_7	pixel_7_0	pixel_7_1	pixel_7_2	pixel_7_3	pi
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	4.0	0.0	0.0	0.0	2.0	14.0	
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	6.0	16.0	
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	9.0	
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	5.0	12.0	
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	8.0	0.0	0.0	1.0	8.0	12.0	

1797 rows × 64 columns

Use components such that 95% of variance is retained

In [90]:
```
from sklearn.decomposition import PCA

pca = PCA(0.95)
X_pca = pca.fit_transform(X)
X_pca.shape
```

Out[90]: (1797, 29)

In [91]:
```
pca.explained_variance_ratio_
```

Out[91]:
```
array([0.14890594, 0.13618771, 0.11794594, 0.08409979, 0.05782415,
       0.0491691 , 0.04315987, 0.03661373, 0.03353248, 0.03078806,
       0.02372341, 0.02272697, 0.01821863, 0.01773855, 0.01467101,
       0.01409716, 0.01318589, 0.01248138, 0.01017718, 0.00905617,
       0.00889538, 0.00797123, 0.00767493, 0.00722904, 0.00695889,
       0.00596081, 0.00575615, 0.00515158, 0.0048954 ])
```

In [92]:
```
pca.n_components_
```

Out[92]: 29

PCA created 29 components out of 64 original columns

In [94]:
```
X_pca
```

Out[94]:
```
array([[ -1.25946645,  21.27488348,  -9.46305462, ...,   3.67072108,
         -0.9436689 ,  -1.13250195],
       [  7.9576113 , -20.76869896,   4.43950604, ...,   2.18261819,
         -0.51022719,   2.31354911],
       [  6.99192297,  -9.95598641,   2.95855808, ...,   4.22882114,
          2.1576573 ,   0.8379578 ],
       ...,
       [ 10.8012837 ,  -6.96025223,   5.59955453, ...,  -3.56866194,
          1.82444444,   3.53885886],
       [ -4.87210009,  12.42395362, -10.17086635, ...,   3.25330054,
          0.95484174,  -0.93895602],
       [ -0.34438963,   6.36554919,  10.77370849, ...,  -3.01636722,
          1.29752723,   2.58810313]])
```

In [47]:
```
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=30)
```

In [53]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000)
model.fit(X_train_pca, y_train)
model.score(X_test_pca, y_test)
```

Out[53]: 0.9694444444444444

Let's now select only two components

In [95]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
X_pca.shape
```

Out[95]:
```
(1797, 2)
```

In [96]:
```
X_pca
```

Out[96]:
```
array([[ -1.25946639,  21.27487891],
       [  7.95760922, -20.76869518],
       [  6.99192341,  -9.95598163],
       ...,
       [ 10.80128435,  -6.96025523],
       [ -4.87210315,  12.42395926],
       [ -0.34438701,   6.36554335]])
```

In [97]:
```
pca.explained_variance_ratio_
```

Out[97]:
```
array([0.14890594, 0.13618771])
```

You can see that both combined retains 0.14+0.13=0.27 or 27% of important feature information

In [98]:
```
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=30)

model = LogisticRegression(max_iter=1000)
model.fit(X_train_pca, y_train)
model.score(X_test_pca, y_test)
```

Out[98]:
```
0.6083333333333333
```

We get less accuancy (~60%) as using only 2 components did not retain much of the feature information. However in real life you will find many cases where using 2 or few PCA components can still give you a pretty good accuracy

Conclusion/Summary:		
Student Signature & Date	**Marks:**	**Evaluator Signature & Date**

Practical 8

8. CNN	Date: / /2023

Aim: Implement a Convolutional Neural Network (CNN) using Keras library.
a) Implement a Convolutional Neural Network (CNN) for a handwrittenCharacter Recognition. Use MNIST dataset to train the model. Generate test images by yourself.
b) Case Study to build a CNN model using python.
i) Build a dataset on home appliances (available at your home/ can takehelp from internet). Also use data augmentation technique to increase dataset.
ii) Preprocess the image to fit into the model
iii) Apply the CNN model and train over the preprocess data.
iv) Evaluate the model using confusion matrix.

First step will be to import Required libraries:

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

Read Dataset:

```
data = pd.read_csv('A_Z Handwritten Data.csv').astype('float32')
data.head(10)
```

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	...	0.639	0.640	0.641	0.642	0.643	0.644	0.645	0.646	0.647	0.648
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	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
2	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
3	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
4	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
5	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
6	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
7	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
8	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
9	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

10 rows × 785 columns

Splitting of Data:

```
X = data.drop('0',axis =
1)y = data['0']
```

Performing Split Using Sklearn:

from sklearn.model_selection import train_test_splitfrom
sklearn.utils import shuffle
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

Reshaping the Training and Testing data:

x_train = np.reshape(x_train.values, (x_train.shape[0], 28,28))x_test =
np.reshape(x_test.values, (x_test.shape[0], 28,28)) print("Shape of Training
data: ", x_train.shape)
print("Shape of Testing data: ", x_test.shape)

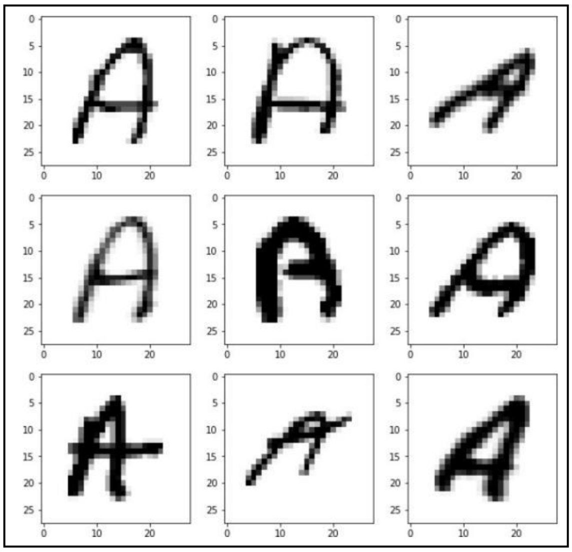
```
Shape of Training data:  (2681, 28, 28)
Shape of Testing data:  (671, 28, 28)
```

Shuffle the training data:

shuffle_data = shuffle(x_train)

Visualize our training data:

import cv2
fig, axes = plt.subplots(3,3, figsize = (10,10))axes =
axes.flatten()
for i in range(9):
  _, shu = cv2.threshold(shuffle_data[i], 30, 200, cv2.THRESH_BINARY)
axes[i].imshow(np.reshape(shuffle_data[i], (28,28)), cmap="Greys") plt.show()



Again Reshaping Data:

training of our model.
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) print("New shape of
training data: ", x_train.shape)
print("New shape of testing data: ", x_test.shape)

```
New shape of training data:  (2681, 28, 28, 1)
New shape of testing data:  (671, 28, 28, 1)
```

Model Creation:

import tensorflow
from tensorflow.keras.utils import to_categorical
y_training = to_categorical(y_train, num_classes = 26, dtype='int')y_testing =
to_categorical(y_test, num_classes = 26, dtype='int') from
tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropoutfrom
tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

model = Sequential()
model.add(Conv2D(64 , (3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPool2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D(2,2)) model.add(Flatten())
model.add(Dense(128,activation ="relu"))
model.add(Dense(256,activation ="relu"))
model.add(Dense(26,activation ="softmax"))
model.summary()

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 26, 26, 64)        640

 max_pooling2d_7 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 conv2d_8 (Conv2D)           (None, 11, 11, 64)        36928

 max_pooling2d_8 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_9 (Conv2D)           (None, 3, 3, 64)          36928

 max_pooling2d_9 (MaxPooling  (None, 1, 1, 64)         0
 2D)

 flatten_3 (Flatten)         (None, 64)                0

 dense_7 (Dense)             (None, 128)               8320

 dense_8 (Dense)             (None, 256)               33024

 dense_9 (Dense)             (None, 26)                6682

=================================================================
Total params: 122,522
Trainable params: 122,522
Non-trainable params: 0
_________________________________________________________________
```

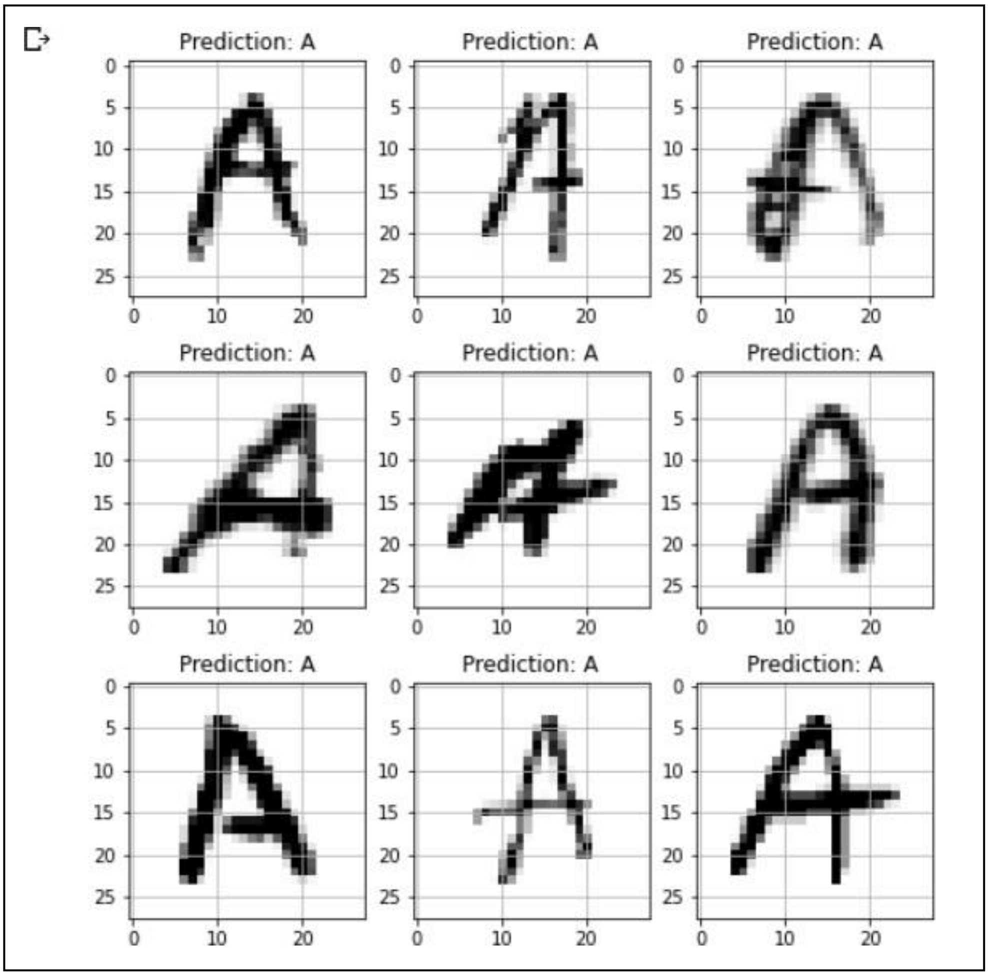
Compile and Fit our model:

```
model.compile(optimizer = Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_training, epochs=5,  validation_data = (x_test,y_testing))
```

```
Epoch 1/5
84/84 [==============================] - 7s 65ms/step - loss: nan - accuracy: 0.9884 - val_loss: nan - val_accuracy: 1.0000
Epoch 2/5
84/84 [==============================] - 4s 52ms/step - loss: nan - accuracy: 1.0000 - val_loss: nan - val_accuracy: 1.0000
Epoch 3/5
84/84 [==============================] - 6s 78ms/step - loss: nan - accuracy: 1.0000 - val_loss: nan - val_accuracy: 1.0000
Epoch 4/5
84/84 [==============================] - 5s 58ms/step - loss: nan - accuracy: 1.0000 - val_loss: nan - val_accuracy: 1.0000
Epoch 5/5
84/84 [==============================] - 5s 58ms/step - loss: nan - accuracy: 1.0000 - val_loss: nan - val_accuracy: 1.0000
```

Prediction Process:

```
fig, axes = plt.subplots(3,3, figsize=(8,9))axes =
axes.flatten()
for i,ax in enumerate(axes):
  image = np.reshape(x_test[i], (28,28))
  ax.imshow(image, cmap="Greys") pred =
  words[np.argmax(y_testing[i])]
  ax.set_title("Prediction: "+pred) ax.grid()
```



Conclusion/Summary:		
Student Signature & Date	**Marks:**	**Evaluator Signature & Date**

Practical 9

9. RNN for Text Classification **Date: / /2023**

Aim: Implement a RNN/LSTM to classify Text into categories according to the sentiment of the text.

We are using twitter US airlines for text classification using RNN and LSTM according to sentiments of text

Solution:

Importing necessary libraries like numpy, pandas, keras, matplotlib, sklearn and nltk

```
import numpy as np
import pandas as pd
import re
from keras.models import Model
from keras.layers import Dense, Input, Dropout, LSTM, Activation
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.initializers import glorot_uniform
from keras.utils import np_utils
from keras.callbacks import EarlyStopping
from nltk.corpus import stopwords
np.random.seed(1)
from sklearn.model_selection import train_test_split

#from emo_utils import *
import matplotlib.pyplot as plt

%matplotlib inline
```

Function to remove English stopwords from a pandas series

```
def remove_stopwords(input_text):
    '''
    Function to remove English stopwords from a Pandas Series.

    Parameters:
        input_text : text to clean
    Output:
        cleaned Pandas Series
    '''
    stopwords_list = stopwords.words('english')
    # Some words which might indicate a certain sentiment are kept via a whitelist
    whitelist = ["n't", "not", "no"]
    words = input_text.split()
    clean_words = [word for word in words if (word not in stopwords_list or word in whitelist) and
len(word) > 1]
```

```
    return " ".join(clean_words)

def remove_mentions(input_text):
    '''
    Function to remove mentions, preceded by @, in a Pandas Series

    Parameters:
        input_text : text to clean
    Output:
        cleaned Pandas Series
    '''
    return re.sub(r'@\w+', '', input_text)
```

Read Dataset

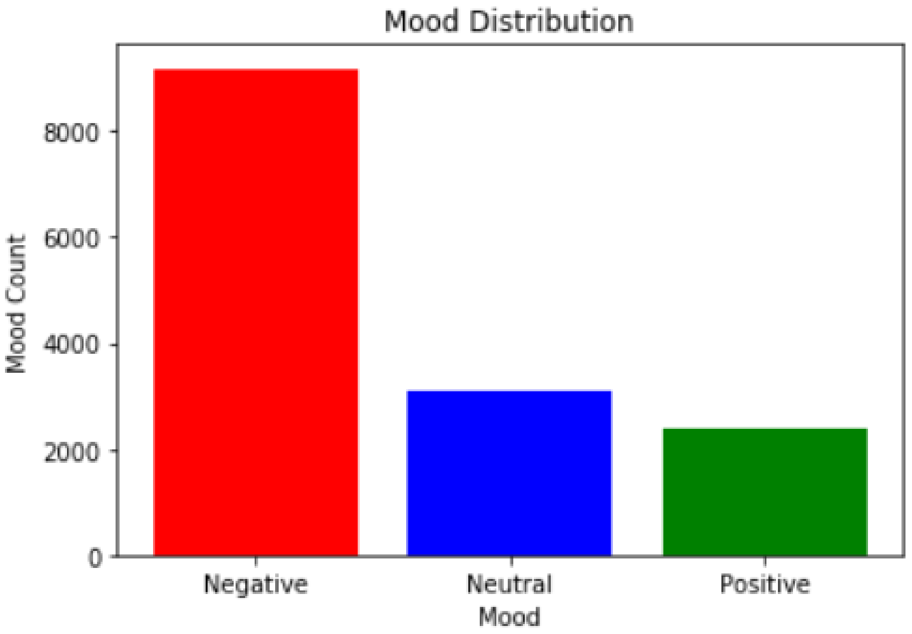
```
train_df = pd.read_csv("../input/twitter-airline-sentiment/Tweets.csv")
train_df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name	nega
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin	NaN
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino	NaN
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn	NaN
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino	NaN
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino	NaN

airline	airline_sentiment_gold	name	negativereason_gold	retweet_count	text	tweet_coord	tweet_created	tweet_location	user_timezone
Virgin America	NaN	cairdin	NaN	0	@VirginAmerica What @dhepburn said.	NaN	2015-02-24 11:35:52 -0800	NaN	Eastern Time (US & Canada)
Virgin America	NaN	jnardino	NaN	0	@VirginAmerica plus you've added commercials t...	NaN	2015-02-24 11:15:59 -0800	NaN	Pacific Time (US & Canada)
Virgin America	NaN	yvonnalynn	NaN	0	@VirginAmerica I didn't today... Must mean I n...	NaN	2015-02-24 11:15:48 -0800	Lets Play	Central Time (US & Canada)
Virgin America	NaN	jnardino	NaN	0	@VirginAmerica it's really aggressive to blast...	NaN	2015-02-24 11:15:36 -0800	NaN	Pacific Time (US & Canada)
Virgin America	NaN	jnardino	NaN	0	@VirginAmerica and it's a really big bad thing...	NaN	2015-02-24 11:14:45 -0800	NaN	Pacific Time (US & Canada)

Check labels and plott their sentiment graph

```
Mood = train_df['airline_sentiment'].value_counts()


index = [1,2,3]
plt.bar(index,Mood,color=['r','b','g'])
plt.xticks(index,['Negative','Neutral','Positive'])
plt.xlabel('Mood')
plt.ylabel('Mood Count')
plt.title('Mood Distribution')
```



Cleaning Data

```
train_df = train_df[['text', 'airline_sentiment']]
train_df.text = train_df.text.apply(remove_mentions)
train_df.loc[:,'sentiment'] = train_df.airline_sentiment.map({'negative':0,'neutral':1,'positive':2})
train_df = train_df.drop(['airline_sentiment'], axis=1)
train_df.head()
```

	text	sentiment
0	What said.	1
1	plus you've added commercials to the experien...	2
2	I didn't today... Must mean I need to take an...	1
3	it's really aggressive to blast obnoxious "en...	0
4	and it's a really big bad thing about it	0

Split Dataset

```
X_train, X_test, Y_train, Y_test = train_test_split(raw_docs_train, sentiment_train,
                                                    stratify=sentiment_train,
                                                    random_state=42,
                                                    test_size=0.1, shuffle=True)
print('# Train data samples:', X_train.shape)
print('# Test data samples:', X_test.shape)
assert X_train.shape[0] == Y_train.shape[0]
assert X_test.shape[0] == Y_test.shape[0]
```

```
# Train data samples: (13176,)
# Test data samples: (1464,)
```

Converting to hot encoding vector for softmax for neural network

```
num_labels = len(np.unique(sentiment_train))
Y_oh_train = np_utils.to_categorical(Y_train, num_labels)
Y_oh_test = np_utils.to_categorical(Y_test, num_labels)
print(Y_oh_train.shape)
```

```
(13176, 3)
```

Create Keras Embedding Layer

```
def pretrained_embedding_layer(word_to_vec_map, word_to_index):
    vocab_len = len(word_to_index) + 1
    emb_dim = word_to_vec_map["cucumber"].shape[0]     word vectors (= 50)

    emb_matrix = np.zeros((vocab_len,emb_dim))

    for word, index in word_to_index.items():
        emb_matrix[index, :] = word_to_vec_map[word]


    embedding_layer = Embedding(vocab_len, emb_dim, trainable = False)

    embedding_layer.build((None,))

    embedding_layer.set_weights([emb_matrix])

    return embedding_layer
```

LSTM Model

```
def ltsm_model(input_shape, word_to_vec_map, word_to_index):

    sentence_indices =  Input(shape=input_shape, dtype='int32')
    embedding_layer = pretrained_embedding_layer(word_to_vec_map, word_to_index)
    embeddings = embedding_layer(sentence_indices)

    X = LSTM(128, return_sequences=True)(embeddings)
    X = Dropout(0.5)(X)
    X = LSTM(128, return_sequences=False)(X)
    X = Dropout(0.5)(X)
    X = Dense(3, activation=None)(X)
    X = Activation('softmax')(X)

    model = Model(inputs=[sentence_indices], outputs=X)
     return model


model = ltsm_model((maxLen,), word_to_vec_map, word_to_index)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 26)                0
_________________________________________________________________
embedding_1 (Embedding)      (None, 26, 50)            20000050
_________________________________________________________________
lstm_1 (LSTM)                (None, 26, 128)           91648
_________________________________________________________________
dropout_1 (Dropout)          (None, 26, 128)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
_________________________________________________________________
activation_1 (Activation)    (None, 3)                 0
=================================================================
Total params: 20,223,669
Trainable params: 223,619
Non-trainable params: 20,000,050
```

Optimising Parameters using epochs and earlystopping

earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=3, verbose=0, mode='auto')

model.fit(X_train_indices, y=Y_oh_train, batch_size=512, epochs=20,
verbose=1, validation_data=(X_test_indices, Y_oh_test), callbacks=[earlystop])

```
Train on 13176 samples, validate on 1464 samples
Epoch 1/20
13176/13176 [==============================] - 4s 331us/step - loss: 0.8757 - acc: 0.6149 - val_loss: 0.8275 - val_acc: 0.627
7
Epoch 2/20
13176/13176 [==============================] - 3s 206us/step - loss: 0.7963 - acc: 0.6624 - val_loss: 0.7391 - val_acc: 0.694
0
Epoch 3/20
13176/13176 [==============================] - 3s 213us/step - loss: 0.7170 - acc: 0.7091 - val_loss: 0.6851 - val_acc: 0.722
0
Epoch 4/20
13176/13176 [==============================] - 3s 206us/step - loss: 0.6819 - acc: 0.7199 - val_loss: 0.6641 - val_acc: 0.728
8
Epoch 5/20
13176/13176 [==============================] - 3s 208us/step - loss: 0.6613 - acc: 0.7288 - val_loss: 0.6423 - val_acc: 0.734
3
Epoch 6/20
13176/13176 [==============================] - 3s 208us/step - loss: 0.6388 - acc: 0.7387 - val_loss: 0.6327 - val_acc: 0.742
5
Epoch 7/20
13176/13176 [==============================] - 3s 208us/step - loss: 0.6271 - acc: 0.7419 - val_loss: 0.6354 - val_acc: 0.743

Epoch 8/20
13176/13176 [==============================] - 3s 205us/step - loss: 0.6184 - acc: 0.7481 - val_loss: 0.6167 - val_acc: 0.749
3
Epoch 9/20
13176/13176 [==============================] - 3s 207us/step - loss: 0.5937 - acc: 0.7558 - val_loss: 0.6280 - val_acc: 0.748
6
Epoch 10/20
13176/13176 [==============================] - 3s 206us/step - loss: 0.5941 - acc: 0.7638 - val_loss: 0.6178 - val_acc: 0.751
4
Epoch 11/20
13176/13176 [==============================] - 3s 208us/step - loss: 0.5765 - acc: 0.7646 - val_loss: 0.6095 - val_acc: 0.750
0

Epoch 12/20
13176/13176 [==============================] - 3s 206us/step - loss: 0.5571 - acc: 0.7719 - val_loss: 0.6102 - val_acc: 0.755
5
Epoch 13/20
13176/13176 [==============================] - 3s 207us/step - loss: 0.5591 - acc: 0.7753 - val_loss: 0.5842 - val_acc: 0.759
6
Epoch 14/20
13176/13176 [==============================] - 3s 206us/step - loss: 0.5465 - acc: 0.7769 - val_loss: 0.5788 - val_acc: 0.765
7
Epoch 15/20
13176/13176 [==============================] - 3s 207us/step - loss: 0.5363 - acc: 0.7829 - val_loss: 0.6083 - val_acc: 0.755
5
Epoch 16/20
13176/13176 [==============================] - 3s 205us/step - loss: 0.5227 - acc: 0.7898 - val_loss: 0.6019 - val_acc: 0.763
0
Epoch 17/20
13176/13176 [==============================] - 3s 208us/step - loss: 0.5132 - acc: 0.7959 - val_loss: 0.5960 - val_acc: 0.755
5
```

Conclusion/Summary:		
Student Signature & Date	Marks:	Evaluator Signature & Date

Practical 10

10. K-means Clustering	Date: / /2023
Aim: Use K-Means Clustering and Hierarchical Clustering algorithm for following datasets.	

Solution:

Import Libraries

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
```

```
data = pd.read_csv('/bike-share.csv')
```

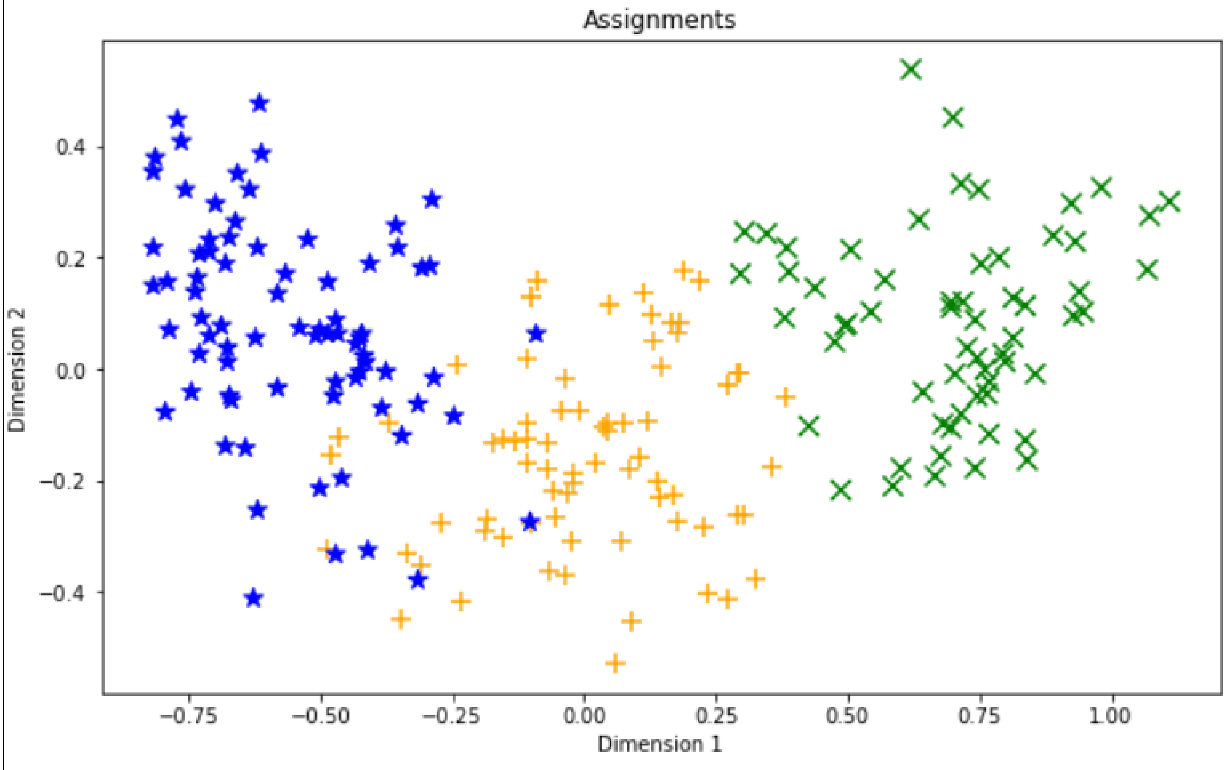
KMeans Clustering

```
model = KMeans(n_clusters = 3, init= 'k-means++', n_init=100, max_iter=1000)
km_clusters = model.fit_predict(features.values)
km_clusters
```

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2,
       2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 2, 2,
       2, 2, 2, 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, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1,
       2, 2, 2, 2, 1, 2, 2, 2, 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, 2, 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], dtype=int32)
```

Plotting

```
def plot_clusters(samples, clusters):
    col_dic = {0:'blue',1:'green',2:'orange'}
    mrk_dic = {0:'*',1:'x',2:'+'}
    colors = [col_dic[x] for x in clusters]
    markers = [mrk_dic[x] for x in clusters]
    plt.figure(figsize=(10,6))
    for sample in range(len(clusters)):
        plt.scatter(samples[sample][0], samples[sample][1], color = colors[sample], marker=marke
rs[sample], s=100)
    plt.xlabel('Dimension 1')
    plt.ylabel('Dimension 2')
    plt.title('Assignments')
    plt.show()

plot_clusters(features_2d, km_clusters)
```



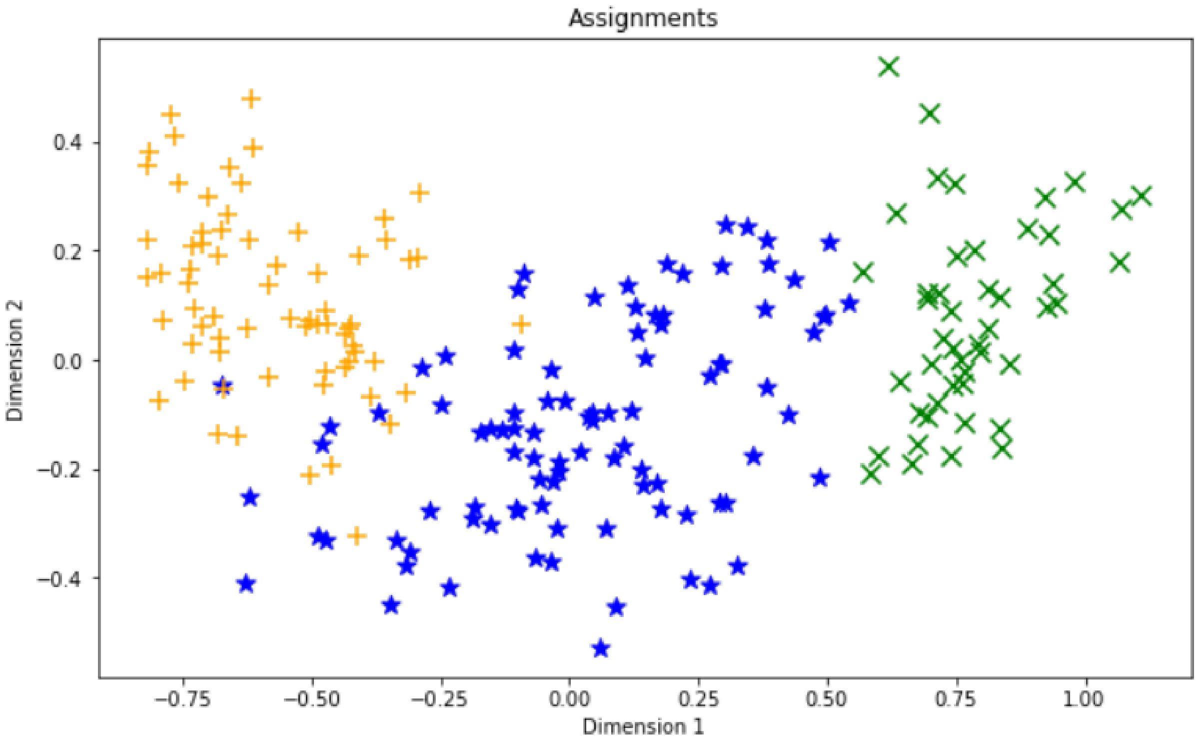
Hierarchical Clustering

```
agg_model = AgglomerativeClustering(n_clusters=3)
agg_clusters = agg_model.fit_predict(features.values)
agg_clusters
```

```
array([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, 2, 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, 0, 0,
       0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0,
       2, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2])
```

Plotting

```
plot_clusters(features_2d, agg_clusters)
```



Conclusion/Summary:		
Student Signature & Date	**Marks:**	**Evaluator Signature & Date**