

# South Point Institute of Technology And Management



## AI with Python Practical file (MCA201C)

**Submitted to-**

Mrs. Jyoti Sharma

**Submitted by-**

KUNAL

MCA 2<sup>nd</sup> year

23021541017

# Index

Sno.	Title	Pgno.	Signature
1.	Depth first search	3	
2.	Breadth first search	4	
3.	A*algorithm	5-7	
4.	Min-max algorithm of Game Theory	8-9	
5.	Write a Program to analyze data and display in the form of a bar graph for two departments of a company having employee id numbers on X-axis and their salaries on Y axis.	10	
6.	Write a program to analyze and draw a line graph to show the profits of a company in various years.	11	
7.	Customer segmentation project using K Means Clustering.	12-16	
8.	Music genre classification project.	17-20	
9.	Stock price prediction project using LSTM (Long short-term memory).	21-24	
10.	Fake news detection project.	25-31	

# 1. Depth-First Search

```
graph = {
    '5': ['3', '7'],
    '3': ['2', '4'],
    '7': ['8'],
    '2': [],
    '4': ['8'],
    '8': []
}

visited = set()

def dfs(visited, graph, node):
    if node not in visited:
        print(node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
    print("Depth-First Search:")
    dfs(visited, graph, '5')
```

## Output:

```
Depth-First Search:
```

```
5
```

```
3
```

```
2
```

```
4
```

```
8
```

```
7
```

```
> |
```

## 2. Breadth First Search

```
graph = {
    '5': ['3','7'],
    '3': ['2', '4'],
    '7': ['8'],
    '2': [],
    '4': ['8'],
    '8': []
}

visited = []
queue = []

def bfs(visited, graph, node): #function for BFS
    visited.append(node)
    queue.append(node)

    while queue:
        m = queue.pop(0)
        print (m, end = " ")
        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

### Output:

```
Following is the Breadth-First Search
>
5 3 7 2 4 8 > |
```

### 3. A\* Algorithm

```
tree = {'S': [['A', 1], ['B', 5], ['C', 8]],  
        'A': [['S', 1], ['D', 3], ['E', 7], ['G', 9]],  
        'B': [['S', 5], ['G', 4]],  
        'C': [['S', 8], ['G', 5]],  
        'D': [['A', 3]],  
        'E': [['A', 7]]}  
  
tree2 = {'S': [['A', 1], ['B', 2]],  
         'A': [['S', 1]],  
         'B': [['S', 2], ['C', 3], ['D', 4]],  
         'C': [['B', 2], ['E', 5], ['F', 6]],  
         'D': [['B', 4], ['G', 7]],  
         'E': [['C', 5]],  
         'F': [['C', 6]]}  
}  
  
heuristic = {'S': 8, 'A': 8, 'B': 4, 'C': 3, 'D': 5000, 'E': 5000, 'G': 0}  
heuristic2 = {'S': 0, 'A': 5000, 'B': 2, 'C': 3, 'D': 4, 'E': 5000, 'F': 5000, 'G': 0}  
  
cost = {'S': 0}  
  
def AStarSearch():  
    global tree, heuristic  
    closed = []  
    opened = [['S', 8]]  
  
    while True:
```

```

fn = [i[1] for i in opened]

chosen_index = fn.index(min(fn))

node = opened[chosen_index][0]

closed.append(opened[chosen_index])

del opened[chosen_index]

if closed[-1][0] == 'G':

    break

for item in tree[node]:

    if item[0] in [closed_item[0] for closed_item in closed]:

        continue

    cost.update({item[0]: cost[node] + item[1]})

    fn_node = cost[node] + heuristic[item[0]] + item[1]

    temp = [item[0], fn_node]

    opened.append(temp)

trace_node = 'G'

optimal_sequence = ['G']

for i in range(len(closed)-2, -1, -1):

    check_node = closed[i][0]

    if trace_node in [children[0] for children in tree[check_node]]:

        children_costs = [temp[1] for temp in tree[check_node]]

        children_nodes = [temp[0] for temp in tree[check_node]]


        if cost[check_node] + children_costs[children_nodes.index(trace_node)] == cost[trace_node]:


            optimal_sequence.append(check_node)

            trace_node = check_node

optimal_sequence.reverse()

return closed, optimal_sequence

if __name__ == '__main__':

    visited_nodes, optimal_nodes = AStarSearch()

    print('visited nodes: ' + str(visited_nodes))

    print('optimal nodes sequence: ' + str(optimal_nodes))

```

## Output:

```
visited nodes: [['S', 8], ['A', 9], ['B', 9], ['G', 9]]  
optimal nodes sequence: ['S', 'B', 'G']  
> |
```

## 4. Min-max algorithm of Game Theory

```
import math

def minimax (curDepth, nodeIndex,
            maxTurn, scores,
            targetDepth):

    if (curDepth == targetDepth):
        return scores[nodeIndex]

    if (maxTurn):
        return max(minimax(curDepth + 1, nodeIndex * 2,
                           False, scores, targetDepth),
                  minimax(curDepth + 1, nodeIndex * 2 + 1,
                           False, scores, targetDepth))

    else:
        return min(minimax(curDepth + 1, nodeIndex * 2,
                           True, scores, targetDepth),
                  minimax(curDepth + 1, nodeIndex * 2 + 1,
                           True, scores, targetDepth))

scores = [3, 5, 2, 9, 12, 5, 23, 23]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")
print(minimax(0, 0, True, scores, treeDepth))
```

Output:

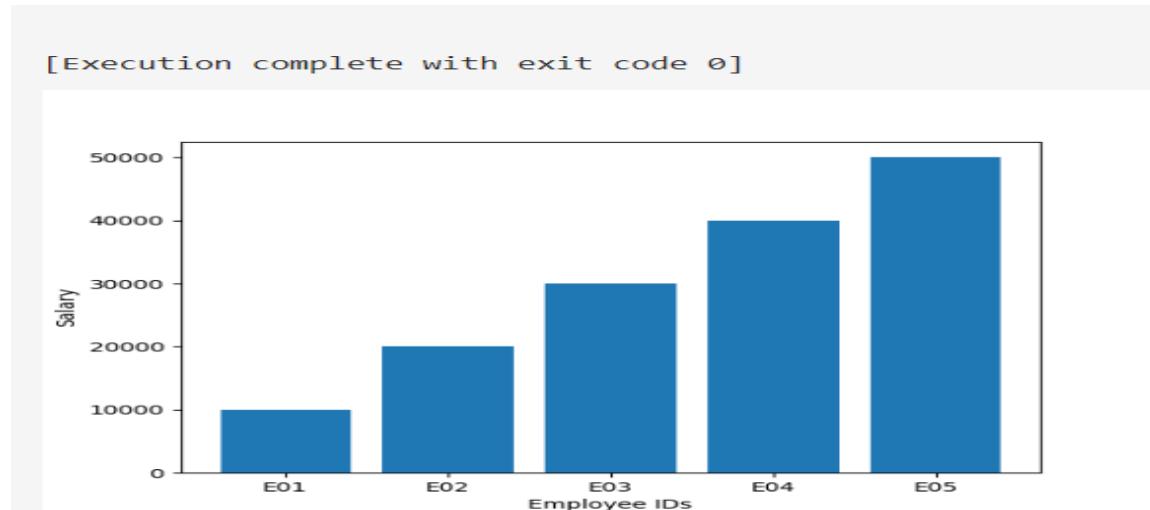
```
The optimal value is : 12
```

```
>
```

5. Write a Program to analyze data and display in the form of a bar graph for two departments of a company having employee id numbers on X-axis and their salaries on Y axis.

```
import matplotlib.pyplot as pl  
import numpy as np  
  
Empld=['E01','E02','E03','E04','E05']  
Sal = [10000,20000,30000,40000,50000]  
  
pl.xlabel("Employee IDs")  
pl.ylabel("Salary")  
pl.bar(Empld,Sal)  
pl.show()
```

Output:



6. Write a program to analyze and draw a line graph to show the profits of a company in various years.

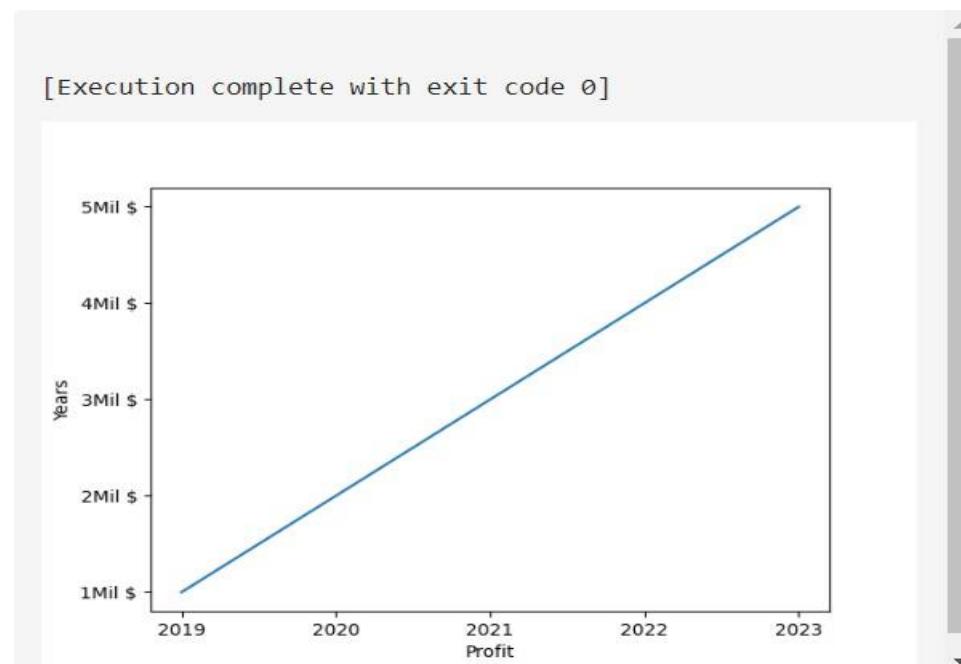
```
import matplotlib.pyplot as pl
import numpy as np

Yr =['2019','2020','2021','2022','2023']

Profit = ['1Mil $','2Mil $','3Mil $','4Mil $','5Mil $']

pl.xlabel("Profit")
pl.ylabel("Years")
pl.plot(Yr,Profit)
pl.show()
```

**Output:**



# 7. Customer segmentation project using K Means Clustering.

## . Imports:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

## . Data Collection & Analysis:

```
# loading the data from csv file to a Pandas DataFrame
customer_data = pd.read_csv('/content/Mall_Customers.csv')
```

```
# first 5 rows in the dataframe
customer_data.head()
```

	<b>CustomerID</b>	<b>Gender</b>	<b>Age</b>	<b>Annual Income (k\$)</b>	<b>Spending Score (1-100)</b>
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
# finding the number of rows and columns
customer_data.shape

(200, 5)

# getting some informations about the dataset
customer_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   CustomerID      200 non-null    int64  
 1   Gender          200 non-null    object  
 2   Age             200 non-null    int64  
 3   Annual Income (k$) 200 non-null    int64  
 4   Spending Score (1-100) 200 non-null    int64  
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
# checking for missing values
customer_data.isnull().sum()
```

```
CustomerID      0
Gender          0
Age             0
Annual Income (k$) 0
Spending Score (1-100) 0
dtype: int64
```

. Choosing the Annual Income Column & Spending Score column:

```
X=customer_data.iloc[:,[3,4]].values
```

```
print(X)
```

```
... [[ 15  39] [ 19  72] [ 24  73]
     [ 15  81] [ 19  14] [ 25   5]
     [ 16   6] [ 19  99] [ 25  73]
     [ 16  77] [ 20  15] [ 28  14]
     [ 17  40] [ 20  77] ...
     [ 17  76] [ 20  13] [126  28]
     [ 18   6] [ 20  79] [126  74]
     [ 18  94] [ 21  35] [137  18]
     [ 19   3] [ 21  66] [137  83]]
```

. Choosing the number of clusters:

. WCSS -> Within Clusters Sum of Squares

```
# finding wcss value for different number of clusters
```

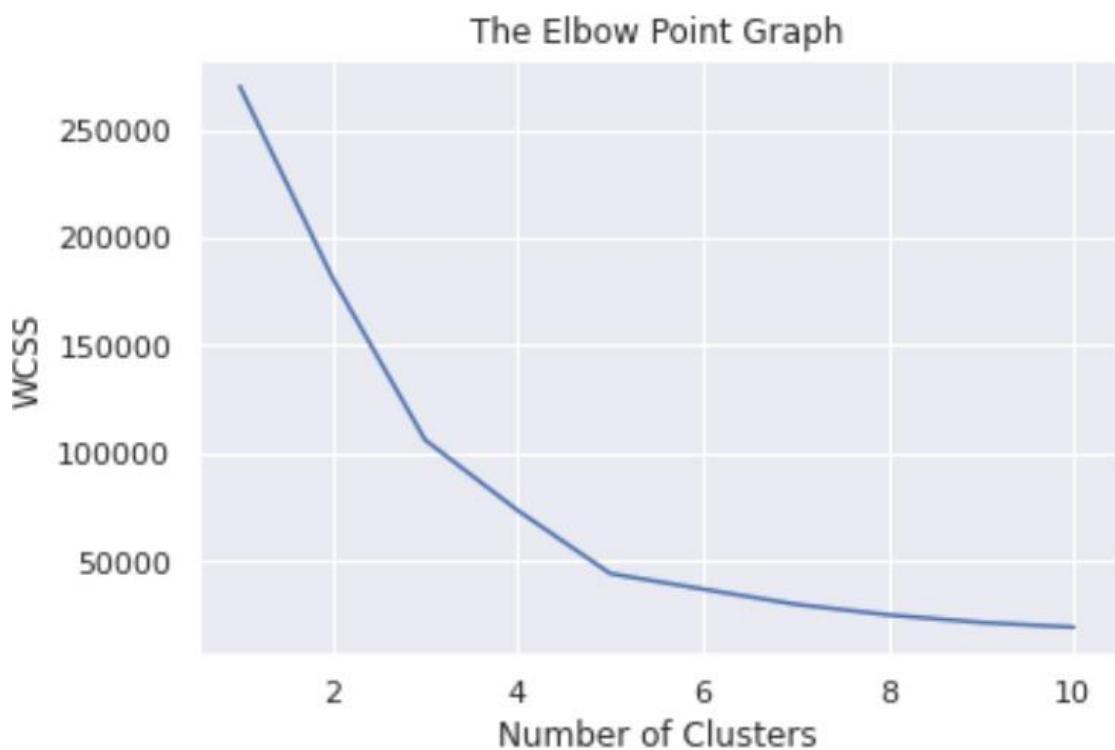
```
wcss = []
```

```
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
```

```
wcss.append(kmeans.inertia_)
```

```
# plot an elbow graph
```

```
sns.set()
plt.plot(range(1,11), wcss)
plt.title('The Elbow Point Graph')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



. Optimum Number of Clusters = 5

. Training the k-Means Clustering Model:

```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0)

# return a label for each data point based on their cluster
Y = kmeans.fit_predict(X)

print(Y)
```

```
[3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3
 1 3 1 3 1 3 0 3 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 4 2 0 2 4 2 4 2 0 2 4 2 4 2 4 2 4 2 0 2 4 2 4 2 4 2 4 2 4 2
 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4
 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2]
```

. 5 Clusters - 0, 1, 2, 3, 4

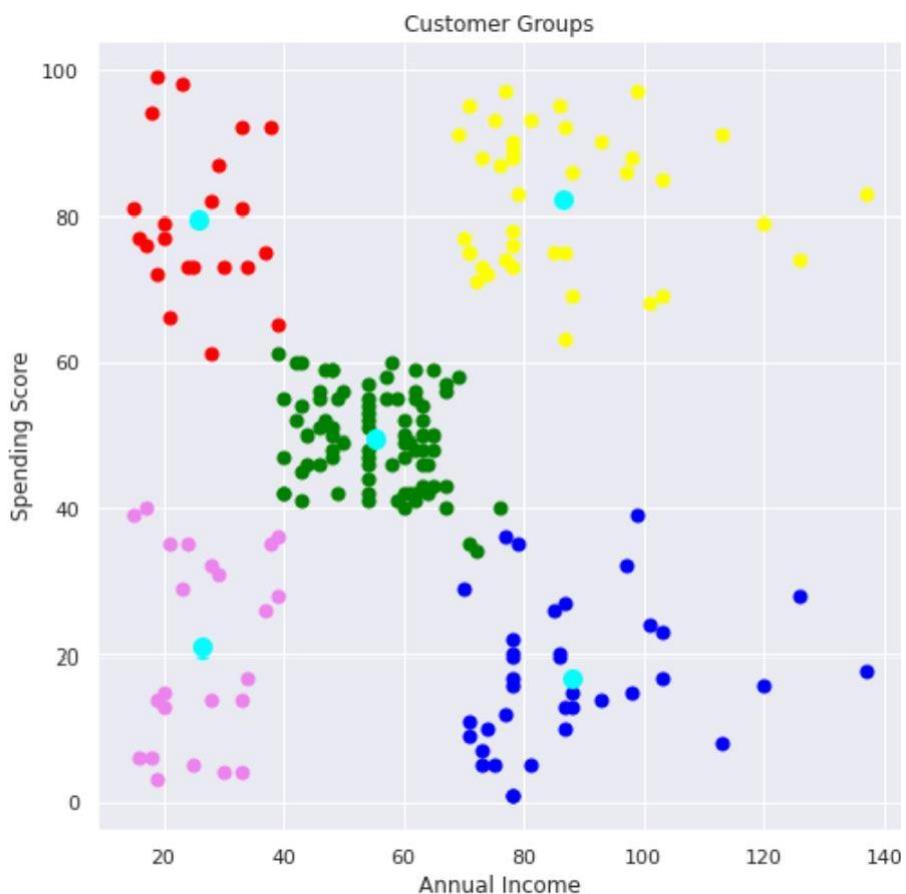
. Visualizing all the Clusters:

```
# plotting all the clusters and their Centroids
```

```
plt.figure(figsize=(8,8))
plt.scatter(X[Y==0,0], X[Y==0,1], s=50, c='green', label='Cluster 1')
plt.scatter(X[Y==1,0], X[Y==1,1], s=50, c='red', label='Cluster 2')
plt.scatter(X[Y==2,0], X[Y==2,1], s=50, c='yellow', label='Cluster 3')
plt.scatter(X[Y==3,0], X[Y==3,1], s=50, c='violet', label='Cluster 4')
plt.scatter(X[Y==4,0], X[Y==4,1], s=50, c='blue', label='Cluster 5')

# plot the centroids
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=100, c='cyan',
label='Centroids')

plt.title('Customer Groups')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.show()
```



## 8. Music genre classification project.

### . Imports:

```
from python_speech_features import mfcc
import scipy.io.wavfile as wav
import numpy as np

from tempfile import TemporaryFile
import os
import pickle
import random
import operator

import math
import numpy as np
```

### . Define a function to get the distance between feature vectors and find neighbors:

```
def getNeighbors(trainingSet, instance, k):
    distances = []
    for x in range (len(trainingSet)):
        dist = distance(trainingSet[x], instance, k )+ distance(instance, trainingSet[x], k)
        distances.append((trainingSet[x][2], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors
```

### . Identify the nearest neighbors:

```
def nearestClass(neighbors):
    classVote = {}

    for x in range(len(neighbors)):
        response = neighbors[x]
        if response in classVote:
            classVote[response]+=1
        else:
            classVote[response]=1

    sorter = sorted(classVote.items(), key = operator.itemgetter(1), reverse=True)
    return sorter[0][0]
```

**. Define a function for model evaluation:**

```
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range (len(testSet)):
        if testSet[x][-1]==predictions[x]:
            correct+=1
    return 1.0*correct/len(testSet)
```

**. Extract features from the dataset and dump these features into a binary .dat file “my.dat”:**

```
directory = "__path_to_dataset__"
f= open("my.dat" , 'wb')
i=0

for folder in os.listdir(directory):
    i+=1
    if i==11 :
        break
    for file in os.listdir(directory+folder):
        (rate,sig) = wav.read(directory+folder+"/"+file)
        mfcc_feat = mfcc(sig,rate ,winlen=0.020, appendEnergy = False)
        covariance = np.cov(np.matrix.transpose(mfcc_feat))
        mean_matrix = mfcc_feat.mean(0)
        feature = (mean_matrix , covariance , i)
        pickle.dump(feature , f)

f.close()
```

**. Train and test split on the dataset:**

```
dataset = []
def loadDataset(filename , split , trSet , teSet):
    with open("my.dat" , 'rb') as f:
        while True:
            try:
                dataset.append(pickle.load(f))
            except EOFError:
                f.close()
                break

    for x in range(len(dataset)):
        if random.random() <split :
            trSet.append(dataset[x])
        else:
            teSet.append(dataset[x])

trainingSet = []
testSet = []
loadDataset("my.dat" , 0.66, trainingSet, testSet)
```

. Make prediction using k Nearest Neighbours and get the accuracy on test data:

```
leng = len(testSet)
predictions = []
for x in range (leng):
    predictions.append(nearestClass(getNeighbors(trainingSet ,testSet[x] ,5)))

accuracy1 = getAccuracy(testSet , predictions)
print(accuracy1)
```

accuracy  
0.6943620178041543

. Test the classifier with new audio file

```
from python_speech_features import mfcc
import scipy.io.wavfile as wav
import numpy as np
from tempfile import TemporaryFile
import os
import pickle
import random
import operator

import math
import numpy as np
from collections import defaultdict

dataset = []
def loadDataset(filename):
    with open("my.dat" , 'rb') as f:
        while True:
            try:
                dataset.append(pickle.load(f))
            except EOFError:
                f.close()
                break

loadDataset("my.dat")

def distance(instance1 , instance2 , k ):
    distance =0
    mm1= instance1[0]
    cm1 = instance1[1]
    mm2 = instance2[0]
    cm2 = instance2[1]
    distance = np.trace(np.dot(np.linalg.inv(cm2), cm1))
    distance+=(np.dot(np.dot((mm2-mm1).transpose() , np.linalg.inv(cm2)) , mm2-mm1 ))
    distance+= np.log(np.linalg.det(cm2)) - np.log(np.linalg.det(cm1))
```

```

distance-= k
return distance

def getNeighbors(trainingSet , instance , k):
    distances =[]
    for x in range (len(trainingSet)):
        dist = distance(trainingSet[x], instance, k )+ distance(instance, trainingSet[x], k)
        distances.append((trainingSet[x][2], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

def nearestClass(neighbors):
    classVote ={}
    for x in range(len(neighbors)):
        response = neighbors[x]
        if response in classVote:
            classVote[response]+=1
        else:
            classVote[response]=1
    sorter = sorted(classVote.items(), key = operator.itemgetter(1), reverse=True)
    return sorter[0][0]

results=defaultdict(int)

i=1
for folder in os.listdir("./musics/wav_genres/"):
    results[i]=folder
    i+=1

(rate,sig)=wav.read("sample_test.wav")
mfcc_feat=mfcc(sig,rate,winlen=0.020,appendEnergy=False)
covariance = np.cov(np.matrix.transpose(mfcc_feat))
mean_matrix = mfcc_feat.mean(0)
feature=(mean_matrix,covariance,0)

pred=nearestClass(getNeighbors(dataset,feature , 5))

print(results[pred])

```

```

In [27]: (rate,sig)=wav.read("sample_test.wav")
mfcc_feat=mfcc(sig,rate,winlen=0.020,appendEnergy=False)
covariance = np.cov(np.matrix.transpose(mfcc_feat))
mean_matrix = mfcc_feat.mean(0)
feature=(mean_matrix,covariance,0)

In [28]: pred=nearestClass(getNeighbors(trainingSet , testSet[x] , 5))

In [29]: print(results[pred])
pop

```

## 9. Stock price prediction project using LSTM (Long short-term memory).

### . Imports:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

from matplotlib.pylab import rcParams
rcParams['figure.figsize']=20,10
from keras.models import Sequential
from keras.layers import LSTM,Dropout,Dense

from sklearn.preprocessing import MinMaxScaler
```

### . Read the dataset:

```
df=pd.read_csv("NSE-TATA.csv")
df.head()
```

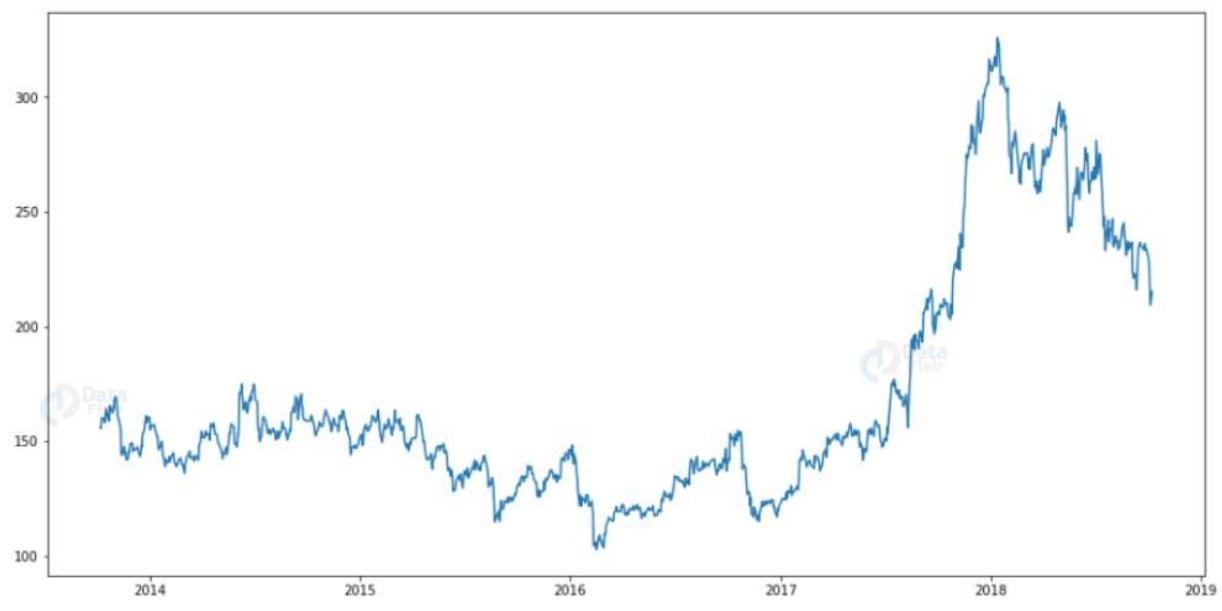
	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-10-08	208.00	222.25	206.85	216.00	215.15	4642146.0	10062.83
1	2018-10-05	217.00	218.60	205.90	210.25	209.20	3519515.0	7407.06
2	2018-10-04	223.50	227.80	216.15	217.25	218.20	1728786.0	3815.79
3	2018-10-03	230.00	237.50	225.75	226.45	227.60	1708590.0	3960.27
4	2018-10-01	234.55	234.60	221.05	230.30	230.90	1534749.0	3486.05

. Analyze the closing prices from dataframe:

```
df["Date"]=pd.to_datetime(df.Date,format="%Y-%m-%d")
df.index=df['Date']
```

```
plt.figure(figsize=(16,8))
plt.plot(df["Close"],label='Close Price history')
```

[<matplotlib.lines.Line2D at 0x7f1be3c225c0>]



. Sort the dataset on date time and filter “Date” and “Close” columns:

```
data=df.sort_index(ascending=True,axis=0)
new_dataset=pd.DataFrame(index=range(0,len(df)),columns=['Date','Close'])

for i in range(0,len(data)):
    new_dataset["Date"][i]=data['Date'][i]
    new_dataset["Close"][i]=data["Close"][i]
```

**. Normalize the new filtered dataset:**

```
scaler=MinMaxScaler(feature_range=(0,1))
final_dataset=new_dataset.values

train_data=final_dataset[0:987,:]
valid_data=final_dataset[987:,:]

new_dataset.index=new_dataset.Date
new_dataset.drop("Date",axis=1,inplace=True)
scaler=MinMaxScaler(feature_range=(0,1))
scaled_data=scaler.fit_transform(final_dataset)

x_train_data,y_train_data=[],[]

for i in range(60,len(train_data)):
    x_train_data.append(scaled_data[i-60:i,0])
    y_train_data.append(scaled_data[i,0])

x_train_data,y_train_data=np.array(x_train_data),np.array(y_train_data)

x_train_data=np.reshape(x_train_data,(x_train_data.shape[0],x_train_data.shape[1],1))
```

**. Build and train the LSTM model:**

```
lstm_model=Sequential()
lstm_model.add(LSTM(units=50,return_sequences=True,input_shape=(x_train_data.shape[1],1)))
lstm_model.add(LSTM(units=50))
lstm_model.add(Dense(1))

inputs_data=new_dataset[len(new_dataset)-len(valid_data)-60:].values
inputs_data=inputs_data.reshape(-1,1)
inputs_data=scaler.transform(inputs_data)

lstm_model.compile(loss='mean_squared_error',optimizer='adam')
lstm_model.fit(x_train_data,y_train_data,epochs=1,batch_size=1,verbose=2)
```

**. Take a sample of a dataset to make stock price predictions using the LSTM model:**

```
X_test=[]
for i in range(60,inputs_data.shape[0]):
    X_test.append(inputs_data[i-60:i,0])
X_test=np.array(X_test)

X_test=np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
predicted_closing_price=lstm_model.predict(X_test)
predicted_closing_price=scaler.inverse_transform(predicted_closing_price)
```

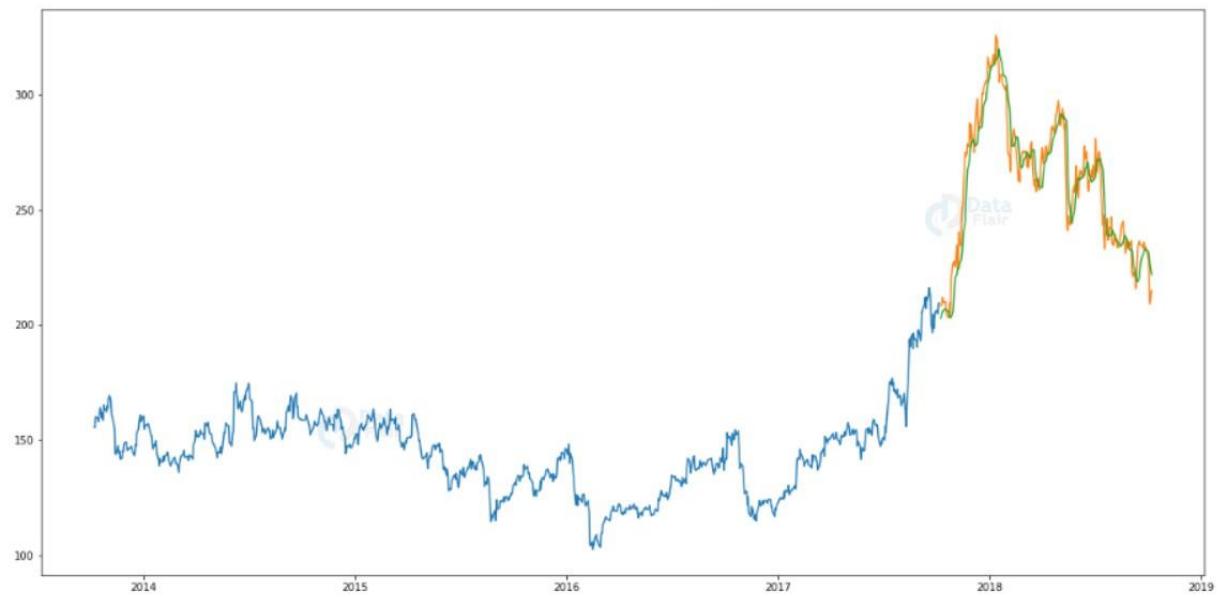
**. Save the LSTM model:**

```
lstm_model.save("saved_model.h5")
```

. Visualize the predicted stock costs with actual stock costs:

```
train_data=new_dataset[:987]
valid_data=new_dataset[987:]
valid_data['Predictions']=predicted_closing_price
plt.plot(train_data["Close"])
plt.plot(valid_data[['Close',"Predictions"]])
```

```
[<matplotlib.lines.Line2D at 0x7f1bb04b8b70>
 <matplotlib.lines.Line2D at 0x7f1bb04b8c88>]
```



# 10. Fake news detection project:

## . Imports:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import feature_extraction, linear_model, model_selection, preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

## . Read datasets:

```
fake = pd.read_csv("data/Fake.csv")
true = pd.read_csv("data/True.csv")
```

```
fake.shape
```

(23481, 4)

```
true.shape
```

(21417, 4)

. Data cleaning and preparation:

```
# Add flag to track fake and real  
fake['target'] = 'fake'  
true['target'] = 'true'  
  
# Concatenate dataframes  
data = pd.concat([fake, true]).reset_index(drop = True)  
data.shape
```

(44898, 5)

```
# Shuffle the data  
from sklearn.utils import shuffle  
data = shuffle(data)  
data = data.reset_index(drop=True)
```

```
# Check the data  
data.head()
```

	title	text	subject	date	target
0	EU Commission says all sides should stick to l...	BRUSSELS (Reuters) - The European Commission s...	worldnews	October 6, 2017	true
1	PRESIDENT TRUMP Looking at Executive Action on...	Remember during the effort to get Obamacare pa...	politics	Aug 1, 2017	fake
2	EU official says no sign Trump plans to ease R...	WASHINGTON (Reuters) - A senior European Union...	politicsNews	April 4, 2017	true
3	Subdued by Harvey, Congress reconvenes facing ...	WASHINGTON (Reuters) - Hurricane Harvey devast...	politicsNews	September 4, 2017	true
4	EVIL HILLARY SUPPORTERS Yell "F*ck Trump" ... Burn...	These people are sick and evil. They will stop...	politics	Nov 6, 2016	fake

```

# Removing the title (we will only use the text)
data.drop(["title"],axis=1,inplace=True)

# Convert to lowercase
data['text'] = data['text'].apply(lambda x: x.lower())

# Remove punctuation
import string

def punctuation_removal(text):
    all_list = [char for char in text if char not in string.punctuation]
    clean_str = ''.join(all_list)
    return clean_str

data['text'] = data['text'].apply(punctuation_removal)

# Removing stopwords
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')

data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))

data.head()

```

		text	subject	target
0	brussels reuters european commission said frid...		worldnews	true
1	remember effort get obamacare passed nancy pel...		politics	false
2	washington reuters senior european union offic...	politicsNews		true
3	washington reuters hurricane harvey devastated...	politicsNews		true
4	people sick evil stop nothing get way laws mea...		politics	false

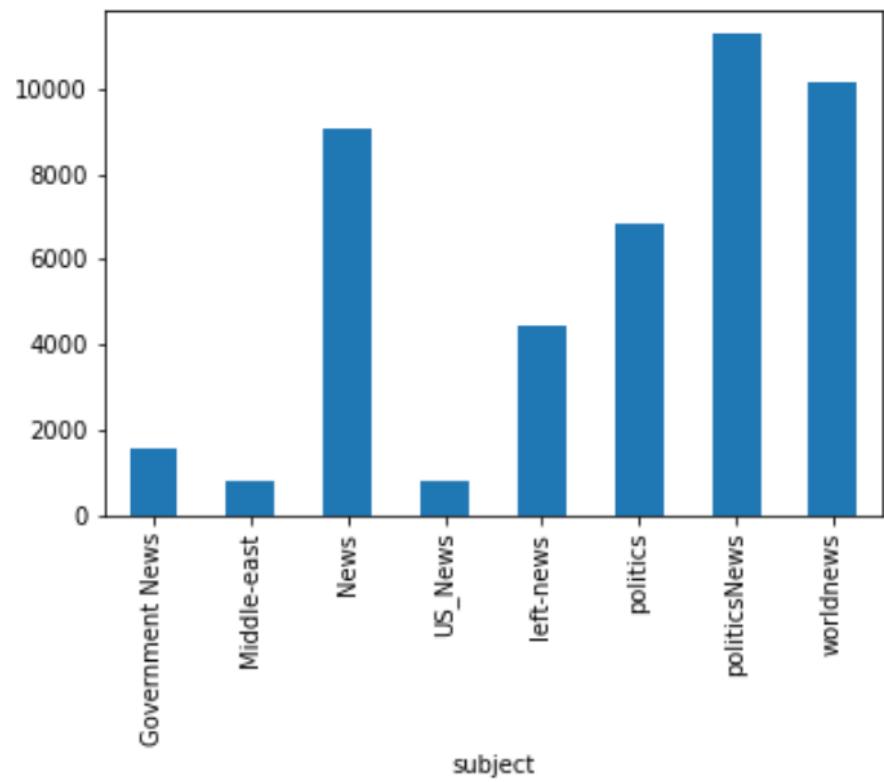
#### . Basic data exploration:

```

# How many articles per subject?
print(data.groupby(['subject'])['text'].count())
data.groupby(['subject'])['text'].count().plot(kind="bar")
plt.show()

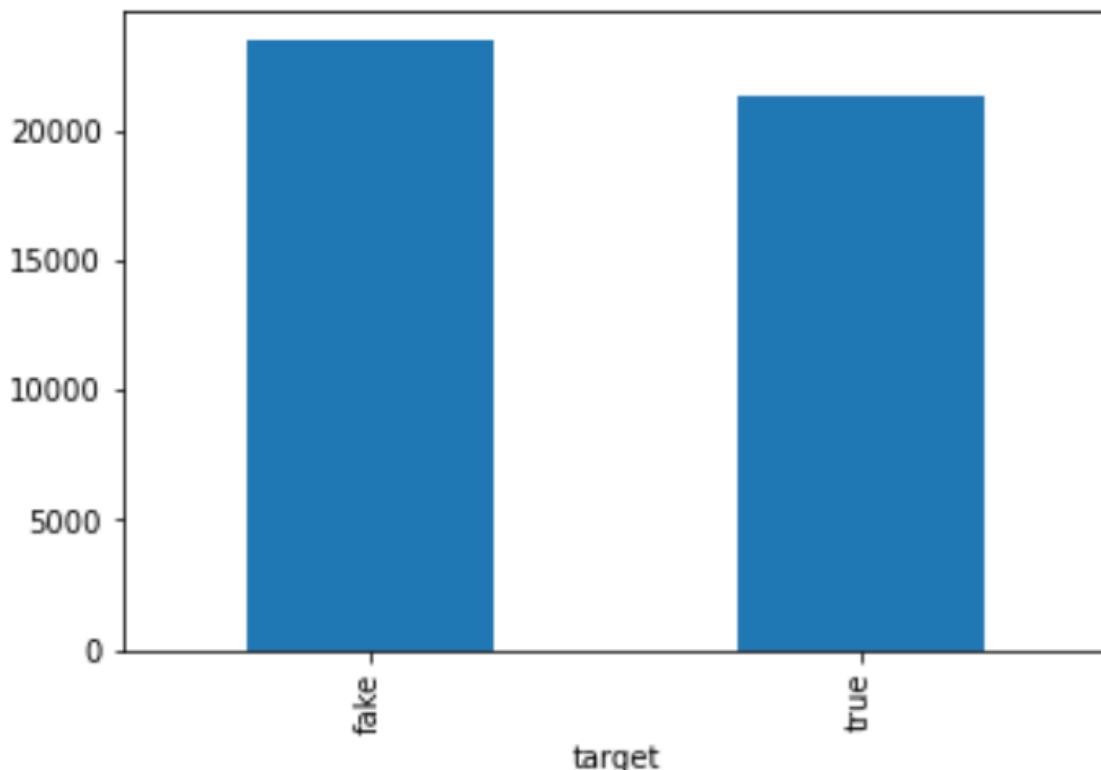
```

```
subject
Government News      1570
Middle-east          778
News                 9050
US_News              783
left-news            4459
politics             6841
politicsNews        11272
worldnews            10145
Name: text, dtype: int64
```



```
# How many fake and real articles?  
print(data.groupby(['target'])['text'].count())  
data.groupby(['target'])['text'].count().plot(kind="bar")  
plt.show()
```

```
target  
fake    23481  
true    21417  
Name: text, dtype: int64
```



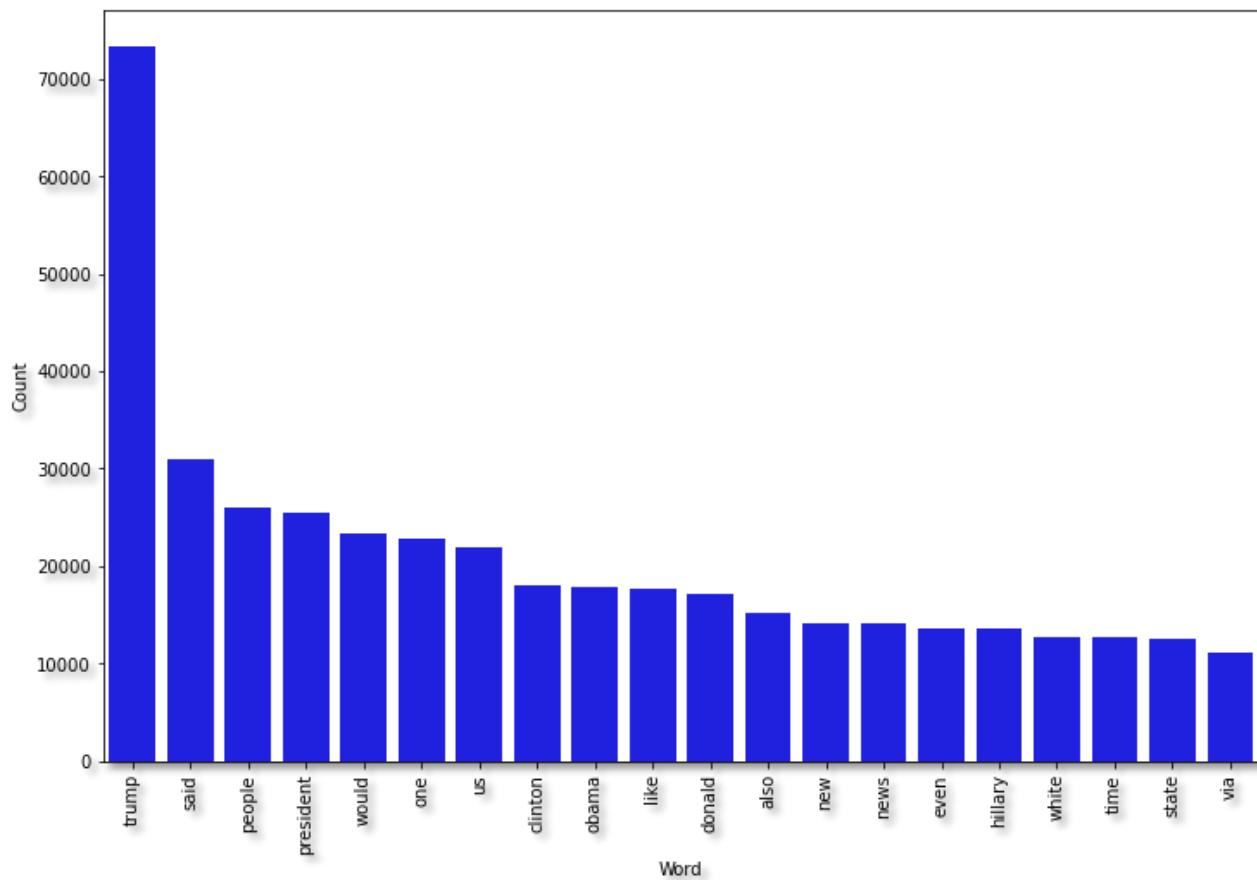
```
# Most frequent words counter  
from nltk import tokenize  
  
token_space = tokenize.WhitespaceTokenizer()  
  
def counter(text, column_text, quantity):  
    all_words = ' '.join([text for text in text[column_text]])  
    token_phrase = token_space.tokenize(all_words)
```

```

frequency = nltk.FreqDist(token_phrase)
df_frequency = pd.DataFrame({"Word": list(frequency.keys()),
                             "Frequency": list(frequency.values())})
df_frequency = df_frequency.nlargest(columns = "Frequency", n = quantity)
plt.figure(figsize=(12,8))
ax = sns.barplot(data = df_frequency, x = "Word", y = "Frequency", color = 'blue')
ax.set(ylabel = "Count")
plt.xticks(rotation='vertical')
plt.show()

# Most frequent words in fake news
counter(data[data["target"] == "fake"], "text", 20)

```



```

# Most frequent words in real news
counter(data[data["target"] == "true"], "text", 20)

```

