In [ ]:

```
# For sending GET requests from the API
import requests
# For saving access tokens and for file management when creating and adding to the data
import os
# For dealing with json responses we receive from the API
import json
# For displaying the data after
import pandas as pd
# For saving the response data in CSV format
import csv
# For parsing the dates received from twitter in readable formats
import datetime
import dateutil.parser
import unicodedata
#To add wait time between requests
import time
```

In [ ]:

```
#!pip3 uninstall twint
```

In [ ]:

```
#!pip3 install --user --upgrade git+https://github.com/twintproject/twint.git@origin/ma
```

In [ ]:

```
#political dataframe detailing candidates in senatorial races and who won them
#found here https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/PE
DOWNLOAD_DIR = "C:/Users/Mary Jane/Downloads/"

filename2 = "1976-2020-senate.csv"
#create a dataframe with the data from the CSV file

data = pd.read_csv(DOWNLOAD_DIR+filename2)
```

In [ ]:

```
data=data.loc[data['year'] == 2020]
data
```

In [ ]:

```
#groups by STATE and returns max votes recieved
winner=data.groupby(['state','state_fips'], as_index=True)['candidatevotes'].max()
data=data.merge(winner,left_on=['state','state_fips'],right_index=True)
data
```

In [ ]:

```
#if max votes recieved equals votes recieved return 1, if not then 0
import numpy as np
data['winner'] = np.where( data['candidatevotes_y'] == data['candidatevotes_x'] , 1, 0)
data
```

In [ ]:

```
data=data.loc[data['candidate'] != 'WRITE-IN']
```

In [ ]:

```
#34 senatorial seats up for election
```

```
data['winner'].sum()
```

In [ ]:

```
#create twitter search key for every candidate

data['Unique']=list(data['candidate'].str.lower()+","+" "+data['state'].astype('str'))
data['Unique']
```

In [ ]:

```
def upper_list(*args):
    return [string.upper() for string in args]
```

In [ ]:

```
#THIS QUERY FUNCTION SENDS THE RESULTS TO A SEPECIFIED FOLDER AND SAVES EACH Candidate-


import twint
import emoji
import nest_asyncio
import re
nest_asyncio.apply()





Cands=list(data['Unique'].astype('str'))


elect="election"

since = datetime.datetime(2020, 10, 18)
until = datetime.datetime(2020, 11, 2)









def search():
    c = twint.Config()
    tweets=pd.DataFrame()
    df=pd.DataFrame()
    for cand in Cands:
        print(cand)
        c = twint.Config()
        c.Limit = 100
        c.Search = cand
        c.Min_Likes = 5
        c.Since = str(since)
        c.Until = str(until)
        #near each state listed in the search key (states are all uppercase)
        c.Near = upper_list(cand)
        c.Filter_retweets = True
        c.Lang = 'en'
        # add this lines to your script
        c.Store_csv = True
        c.Output = os.path.join('C:/Users/Mary Jane/Downloads/candidates/{}.csv'.format
        twint.run.Search(c)
```

In [ ]:

```
#the more times you run it the more tweets are compiled.. each run yields about 700 twe

search()
```

In [ ]:

```
import os
import glob


globbed_files = glob.glob("C:/Users/Mary Jane/Downloads/candidates/*.csv")

tweets=[]
for csv in globbed_files:
    senate_texts = pd.read_csv(csv)
    senate_texts['CAND'] = os.path.basename(csv)
    tweets.append(senate_texts)

CORPUS = pd.concat(tweets, ignore_index=True)
os.chdir("C:/Users/Mary Jane/Downloads")
CORPUS.to_csv("CORPUS_Term.csv")
```

In [ ]:

```
DOWNLOAD_DIR = "C:/Users/Mary Jane/Downloads/"

filename3 = "CORPUS_Term.csv"
#create a dataframe with the data from the CSV file

senate_tweets = pd.read_csv(DOWNLOAD_DIR+filename3)
```

In [ ]:

```
#run multiple times to get more tweets (returns 702 tweets per loop)
senate_tweets=senate_tweets.drop_duplicates()
senate_tweets
```

In [ ]:

```
senate_tweets=senate_tweets.drop('Unnamed: 0',axis=1)
```

In [ ]:

```
senate_tweets[['CAND','file']] = senate_tweets['CAND'].str.split(',',expand=True)
```

In [ ]:

```
senate_tweets['CAND']=senate_tweets['CAND'].str.upper()
COMPOSITE=senate_tweets.merge(data, left_on='CAND', right_on='candidate',how='inner')
COMPOSITE
```

In [ ]:

```
import emoji
import re

#remove links
COMPOSITE["tweet"] =COMPOSITE["tweet"].apply(lambda x: re.sub(r'https?:\/\/\S*', '',x,f
#translate emojies
COMPOSITE["tweet"] = COMPOSITE["tweet"].apply(lambda x: emoji.demojize(x))
COMPOSITE
```

In [ ]:

In [ ]:

```
CORPUS=COMPOSITE
```

In [ ]:

```
#remove mentions and hastags (hastags will be added back later)
import re

CORPUS["tweet"] = CORPUS["tweet"].apply(lambda x: re.sub("@[A-Za-z0-9_#]+","",x))
CORPUS["tweet"] = CORPUS["tweet"].apply(lambda x: re.sub("#[A-Za-z0-9_#]+","",x))
CORPUS
```

In [ ]:

```
CORPUS['tweet']=CORPUS['tweet'].str.lower()
CORPUS
```

In [ ]:

In [ ]:

```
CORPUS["hashtags"]=CORPUS['hashtags'].str.lower()
CORPUS
```

In [ ]:

```
CORPUS['hashtags'] = [''.join(map(str, l)) for l in CORPUS['hashtags']]
CORPUS['tweet'] = CORPUS[['tweet', 'hashtags']].apply(lambda x: ''.join(x), axis=1)
CORPUS
```

In [ ]:

```
CORPUS['tweet']
```

In [ ]:

```
os.chdir("C:/Users/Mary Jane/Downloads")
```

```
CORPUS.to_csv('CLeanedCorpus.csv')
```

In [ ]:

```
CORPUS.columns
```

In [ ]:

```
CORPUS=CORPUS[['id', 'conversation_id', 'created_at', 'date', 'time',
       'timezone', 'user_id', 'username', 'name', 'place', 'tweet','language','replies_
```

In [ ]:

```
os.chdir("C:/Users/Mary Jane/Downloads")
CORPUS.to_csv('FINAL_TERM_CORPUS.csv')
```

In [ ]:

```
CORPUS['tweet']=CORPUS['tweet'].astype('str')
CORPUS['tweet']
```

In [ ]:

```
import re

CORPUS['tweet']=CORPUS['tweet'].str.replace('\d+', '')
CORPUS['tweet']
```

In [ ]:

```
VECTOR=CORPUS[['tweet','candidate','state','party_simplified','candidatevotes_x','total
```

In [ ]:

```
VECTOR['tweet']
```

In [ ]:

```
VECTOR
```

In [ ]:

```
#to unpcak tweet column
os.chdir("C:/Users/Mary Jane/Downloads")
VECTOR.to_csv('termVector.csv')
```

In [ ]:

```
DOWNLOAD_DIR = "C:/Users/Mary Jane/Downloads/"
filename10 = "termVector.csv"

#create a dataframe with the data from the CSV file
VECTOR = pd.read_csv(DOWNLOAD_DIR+filename10)
```

In [ ]:

```
VECTOR['BAG'] = VECTOR['tweet'].str.replace('[^\w\s]','')
VECTOR
```

In [ ]:

```
#to split hashtags into seprate words

from wordsegment import load, segment
load()

VECTOR["BAG"] = VECTOR["BAG"].apply(lambda x: ' '.join(segment(x)))
VECTOR
```

In [ ]:

```
#spell check (takes a while to run)
from textblob import TextBlob

VECTOR['BAG']=VECTOR.BAG.apply(lambda txt: ''.join(TextBlob(txt).correct()))
VECTOR['BAG']
```

In [ ]:

```
#tokenize to remove words/names that could lead to overfitting

from nltk.tokenize import  word_tokenize
from nltk.corpus import stopwords


VECTOR['BAG']=VECTOR['BAG'].apply(word_tokenize)

VECTOR['BAG']
```

In [ ]:

```
#remove stop words
from nltk.corpus import stopwords

stops= set(stopwords.words('english'))



VECTOR['BAG']=VECTOR['BAG'].apply(lambda x: [item for item in x if item not in stops])
VECTOR['BAG']
```

In [ ]:

```
VECTOR['BAG']
```

In [ ]:

```
#remove names based on list of common names found at links beloiw
#https://www.census.gov/topics/population/genealogy/data.html
#https://github.com/smashew/NameDatabases/blob/master/NamesDatabases/first%20names/all.


DOWNLOAD_DIR = "C:/Users/Mary Jane/Downloads/"
filename4 = "Common names.csv"

#create a dataframe with the data from the CSV file
names_ = pd.read_csv(DOWNLOAD_DIR+filename4)
names=data['candidate'].str.split(" +", 2, expand=True)
names=names.stack().reset_index()
names=names[0]
names=pd.DataFrame(names)
names=names.loc[names[0] != 'VOTES']
names
```

In [ ]:

```
names_=names_['NAME']
names_=pd.DataFrame(names_)
names_
```

In [ ]:

```
names=np.array(names)
```

```
names_=np.array(names_)

NAMES=np.concatenate((names,names_),axis=0)

names=pd.DataFrame(NAMES)

names=names.rename(columns={0: "NAME"})

names
```

In [ ]:

```
VECTOR['BAG']
```

In [ ]:

```
names['NAME']=names['NAME'].str.lower()
NAMES=set(names['NAME'])

VECTOR['BAG']=VECTOR['BAG'].apply(lambda x: [item for item in x if item not in NAMES])
VECTOR['BAG']
```

In [ ]:

```
#we dont want to muddy the waters with the calssifiers by allowing a state name to dict
#we want to measure how any old tweet, when stripped down to its skeleton (bag of words

DOWNLOAD_DIR = "C:/Users/Mary Jane/Downloads/"
filename6 = "STATES.csv"

#create a dataframe with the data from the CSV file
states = pd.read_csv(DOWNLOAD_DIR+filename6)
states['State']=states['State'].str.lower()
STATES=set(states['State'])

VECTOR['BAG']=VECTOR['BAG'].apply(lambda x: [item for item in x if item not in STATES])
VECTOR['BAG']
```

In [ ]:

```
VECTOR
```

In [ ]:

```
os.chdir("C:/Users/Mary Jane/Downloads/")
```

In [ ]:

```
#to unpcak tweet column
os.chdir("C:/Users/Mary Jane/Downloads")
VECTOR.to_csv('FINALVEC.csv')
```

In [ ]:

```
#export to unpack individual lists in each row

DOWNLOAD_DIR = "C:/Users/Mary Jane/Downloads/"
filename5 = "FINALVEC.csv"

#create a dataframe with the data from the CSV file
VECTOR = pd.read_csv(DOWNLOAD_DIR+filename5)
```

In [ ]:

```
VECTOR
```

In [ ]:

```
VECTOR['BAG'] = VECTOR['BAG'].str.replace('[^\w\s]','')
VECTOR
```

In [ ]:

```
VECTOR['BAG']
```

In [ ]:

```
import nltk
nltk.download('vader_lexicon')



from nltk.sentiment.vader import SentimentIntensityAnalyzer



sid = SentimentIntensityAnalyzer()





VECTOR['compound'] = [sid.polarity_scores(x)['compound'] for x in VECTOR['BAG']]
VECTOR['neg'] = [sid.polarity_scores(x)['neg'] for x in VECTOR['BAG']]
VECTOR['neu'] = [sid.polarity_scores(x)['neu'] for x in VECTOR['BAG']]
VECTOR['pos'] = [sid.polarity_scores(x)['pos'] for x in VECTOR['BAG']]


VECTOR.loc[VECTOR.compound>0,'sentiment_type']='POSITIVE'
VECTOR.loc[VECTOR.compound==0,'sentiment_type']='NEUTRAL'
VECTOR.loc[VECTOR.compound<0,'sentiment_type']='NEGATIVE'
```

In [ ]:

```
VECTOR.sentiment_type.value_counts().plot(kind='bar',title="sentiment analysis")
```

In [ ]:

```
#Polarity Score

VECTOR.compound.describe()
```

In [ ]:

```
DEMS=VECTOR.loc[VECTOR['party_simplified'] == 'DEMOCRAT']
REPS=VECTOR.loc[VECTOR['party_simplified'] == 'REPUBLICAN']
```

In [ ]:

```
DEMS.compound.describe()
```

In [ ]:

```
REPS.compound.describe()
```

In [ ]:

```
os.chdir("C:/Users/Mary Jane/Downloads")
VECTOR.to_csv('FINAL_TERM_SENTIMENT_CLEANED.csv')
```

In [ ]:

```
win=VECTOR.loc[VECTOR['winner'] == 1]
loss=VECTOR.loc[VECTOR['winner'] == 0]
```

In [ ]:

```
win.compound.describe()
```

In [ ]:

```
loss.compound.describe()
```

In [ ]:

```
VECTOR.columns
```

In [ ]:

```
CORR=VECTOR[[
       'candidatevotes_x', 'totalvotes', 'winner','compound', 'neg',
       'neu', 'pos']]
```

In [ ]:

```
CORR2=DEMS[[
       'candidatevotes_x', 'totalvotes', 'winner','compound', 'neg',
       'neu', 'pos']]
DEMS
```

In [ ]:

```
CORR3=REPS[[
       'candidatevotes_x', 'totalvotes', 'winner','compound', 'neg',
       'neu', 'pos']]

REPS
```

In [ ]:

```
#might indicate more negative campaigning towards most probablistic candidate

corrMatrix = CORR.corr()
print (corrMatrix)
```

In [ ]:

```
#Democrats

corrMatrix = CORR2.corr()
print (corrMatrix)
```

In [ ]:

```
#Republicans

corrMatrix = CORR3.corr()
print (corrMatrix)
```

In [ ]:

```
from sklearn.feature_extraction.text import CountVectorizer


vec = CountVectorizer()
X = vec.fit_transform(VECTOR.BAG)
tdm = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

In [ ]:

```
tdm
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
label_encoder_a = LabelEncoder()
label_encoder_b = LabelEncoder()
label_encoder_c = LabelEncoder()
label_encoder_d = LabelEncoder()

#republican=3, Democrats=0, 2=other, 1=libertarian
PARTY = label_encoder_a.fit_transform(VECTOR['party_simplified'])
SENTIMENT = label_encoder_b.fit_transform(VECTOR['sentiment_type'])
OUTCOME=VECTOR['winner']
STATE=label_encoder_c.fit_transform(VECTOR['state'])
CANDIDATE=label_encoder_d.fit_transform(VECTOR['candidate'])
```

In [ ]:

```
VECTOR['sentiment_type']
```

In [ ]:

```
PARTY = pd.DataFrame(PARTY)
SENTIMENT = pd.DataFrame(SENTIMENT)
OUTCOME=pd.DataFrame(OUTCOME)
STATE=pd.DataFrame(STATE)
CANDIDATE=pd.DataFrame(CANDIDATE)
CANDIDATE
```

In [ ]:

```
PARTY=PARTY.rename(columns={0: "PARTY"})
SENTIMENT=SENTIMENT.rename(columns={0: "SENTIMENT"})
OUTCOME=OUTCOME.rename(columns={"winner": "OUTCOME"})
STATE=STATE.rename(columns={0: "STATE"})
CANDIDATE=CANDIDATE.rename(columns={0: "CANDIDATE"})
OUTCOME
```

In [ ]:

```
tdm
```

In [ ]:

```
TDM=PARTY.merge(tdm,left_index=True, right_index=True)
TDM=SENTIMENT.merge(TDM,left_index=True, right_index=True)
TDM=STATE.merge(TDM,left_index=True, right_index=True)
TDM=CANDIDATE.merge(TDM,left_index=True, right_index=True)
TDM=OUTCOME.merge(TDM,left_index=True, right_index=True)
TDM
```

In [ ]:

```
TDM
```

In [ ]:

```
TDM['Tweet_count']= TDM.groupby(["CANDIDATE"])["id"].transform("count")
```

In [ ]:

```
TDM
```

In [ ]:

```
TDM.columns[3:]
```

In [ ]:

```
corr = TDM.corr()
c1 = corr.abs().unstack()
```

In [ ]:

```
TDM_CORR=c1.sort_values(ascending = False)
TDM_CORR
```

In [ ]:

```
TDM_CORR=pd.DataFrame(TDM_CORR)
TDM_CORR=TDM_CORR.reset_index()
TDM_CORR
```

In [ ]:

```
OUTCOME_CORR=TDM_CORR.loc[TDM_CORR['level_0'] == 'OUTCOME']
OUTCOME_CORR=OUTCOME_CORR.loc[OUTCOME_CORR['level_1'] != 'PARTY']
OUTCOME_CORR=OUTCOME_CORR.loc[OUTCOME_CORR['level_1'] != 'CANDIDATE']

OUTCOME_CORR=OUTCOME_CORR.reset_index()
OUTCOME_CORR
```

In [ ]:

```
OUTCOME_CORR['compound'] = [sid.polarity_scores(x)['compound'] for x in OUTCOME_CORR['l
OUTCOME_CORR.loc[OUTCOME_CORR.compound>0,'sentiment_type']='POSITIVE'
OUTCOME_CORR.loc[OUTCOME_CORR.compound==0,'sentiment_type']='NEUTRAL'
OUTCOME_CORR.loc[OUTCOME_CORR.compound<0,'sentiment_type']='NEGATIVE'
```

In [ ]:

```
OUTCOME_CORR.head(50)
```

In [ ]:

```
POLARS=OUTCOME_CORR.loc[OUTCOME_CORR['sentiment_type'] != 'NEUTRAL']
POLARS.head(50)
```

In [ ]:

```
POLARS.loc[POLARS['index'] == 'SENTIMENT']
```

In [ ]:

```
POLARS.plot.scatter(x = 'compound', y = 0, s = 5)
```

In [ ]:

```

```

In [ ]:

```
from sklearn.model_selection import StratifiedShuffleSplit

splitobj = StratifiedShuffleSplit(n_splits=1, test_size=.5, random_state=42)

#make a representative split of the data_combined dataframe based on Target
for train_index, test_index in splitobj.split(TDM, TDM['OUTCOME']):
    train_set = TDM.iloc[train_index]
    test_set = TDM.iloc[test_index]
```

In [ ]:

```
overfits=test_set[['CANDIDATE','STATE']]
```

In [ ]:

```

X_train = train_set.drop(['OUTCOME'], axis=1)
X_train=X_train.drop('CANDIDATE',axis=1)
X_train=X_train.drop('STATE',axis=1)



y_train = train_set[['OUTCOME']]
y_train
```

In [ ]:

```

X_test = test_set.drop(['OUTCOME'], axis=1)
X_test=X_test.drop('CANDIDATE',axis=1)
X_test=X_test.drop('STATE',axis=1)

y_test = test_set[['OUTCOME']]
y_test
```

In [ ]:

```

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
parameters = {"n_neighbors": range(2, 10)}
gridsearch = GridSearchCV(KNeighborsRegressor(), parameters)
gridsearch.fit(X_train, y_train)
gridsearch.best_params_
```

In [ ]:

```

knn_model = KNeighborsRegressor(n_neighbors=3,weights='distance')
knn_model.fit(X_train, y_train)
```

In [ ]:

```

from sklearn.metrics import mean_squared_error
from math import sqrt
test_preds = knn_model.predict(X_test)
test_preds=np.rint(test_preds)

mse = mean_squared_error(y_test, test_preds)
rmse = sqrt(mse)
rmse
```

In [ ]:

```

from sklearn.metrics import classification_report, confusion_matrix


print(confusion_matrix(y_test, test_preds))
```

In [ ]:

```

print(classification_report(y_test, test_preds))
```

In [ ]:

```

knn_model2 = KNeighborsRegressor(n_neighbors=2,weights='distance')
knn_model2.fit(X_train, y_train)
```

In [ ]:

```

from sklearn.metrics import mean_squared_error
from math import sqrt
test_preds2 = knn_model2.predict(X_test)
test_preds2=np.rint(test_preds2)

mse = mean_squared_error(y_test, test_preds2)
rmse = sqrt(mse)
rmse
```

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix


print(confusion_matrix(y_test, test_preds2))
```

In [ ]:

```
print(classification_report(y_test, test_preds2))
```

In [ ]:

```
X_test_=X_test.merge(overfits,left_index=True,right_index=True)
X_test_=X_test_.merge(y_test,left_index=True,right_index=True)
```

In [ ]:

In [ ]:

```
X_test_['KNN_Class'] = test_preds
```

In [ ]:

```
Grouped_Corr=X_test_.groupby(['PARTY','STATE','CANDIDATE']).mean()
```

In [ ]:

In [ ]:

In [ ]:

```
#republican=3, Democrats=0, 2=other, 1=libertarian

label_encoder_a.inverse_transform(X_test_['PARTY'])
```

In [ ]:

```
X_test_['PARTY']=label_encoder_a.inverse_transform(X_test_['PARTY'])
X_test_['STATE']=label_encoder_c.inverse_transform(X_test_['STATE'])
X_test_['CANDIDATE']=label_encoder_d.inverse_transform(X_test_['CANDIDATE'])

X_test_
```

In [ ]:

```
X_test_=X_test_.groupby(['PARTY','STATE','CANDIDATE']).mean()
```

In [ ]:

```
#to correct columns
os.chdir("C:/Users/Mary Jane/Downloads")
X_test_.to_csv('Candidate_Sumamrizations.csv')
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
#export to unpack individual lists in each row

DOWNLOAD_DIR = "C:/Users/Mary Jane/Downloads/"
filename11 = "Candidate_Sumamrizations.csv"

#create a dataframe with the data from the CSV file
X_test_ = pd.read_csv(DOWNLOAD_DIR+filename11)
X_test_['SENTIMENT']=X_test_['SENTIMENT'].astype('int64')
X_test_['SENTIMENT']=round(X_test_['SENTIMENT'])
X_test_['SENTIMENT']=label_encoder_b.inverse_transform(X_test_['SENTIMENT'])
X_test_
```

In [ ]:

```
X_test_
```

In [ ]:

```
X_test_.to_excel('Canidate_Sumamrizations_Dec 5th.xlsx')
```

In [ ]:

```
%matplotlib notebook

import re, seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap

# generate data
#46 Candidates in test set
n = 94
x = X_test_['CANDIDATE'].values
y = X_test_['SENTIMENT'].values
z = X_test_['PARTY'].values
a = X_test_['OUTCOME'].values

df = pd.DataFrame(zip(x,y,z), columns=list('XYZ'))
df['Z'] = np.linspace(1, 1000, n)

Z = np.log10(df['Z'])
Xuniques, X = np.unique(x, return_inverse=True)
Yuniques, Y = np.unique(y, return_inverse=True)
Zuniques, Z = np.unique(z, return_inverse=True)

fig = plt.figure()
```

```
ax = fig.add_subplot(1, 1, 1, projection='3d')
scatter=ax.scatter(X, Y, Z, s=20, c=a)
ax.set(zticks=range(len(Zuniques)), zticklabels=Zuniques,
       yticks=range(len(Yuniques)), yticklabels=Yuniques)


ax.set_xlabel('CANDIDATE')
ax.set_ylabel('SENTIMENT')
ax.set_zlabel('PARTY')



plt.show()

lgd=plt.legend(handles=scatter.legend_elements()[0],
           labels=['Loss','Win'],
           title="OUTCOME")

ax.azim = 20
ax.dist = 10
ax.elev = 21


fig.savefig("Candidate_OUTCOME.pdf".format(ax.azim, ax.dist, ax.elev,bbox_extra_artists
```

In [ ]:

```
%matplotlib notebook

import re, seaborn as sns
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap

# generate data
#46 Candidates in test set
n = 94
x = X_test_['CANDIDATE'].values
y = X_test_['SENTIMENT'].values
z = X_test_['PARTY'].values
a = X_test_['KNN_Class'].values

df = pd.DataFrame(zip(x,y,z), columns=list('XYZ'))
df['Z'] = np.linspace(1, 1000, n)

Z = np.log10(df['Z'])
Xuniques, X = np.unique(x, return_inverse=True)
Yuniques, Y = np.unique(y, return_inverse=True)
Zuniques, Z = np.unique(z, return_inverse=True)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection='3d')
scatter=ax.scatter(X, Y, Z, s=20, c=a)
ax.set(zticks=range(len(Zuniques)), zticklabels=Zuniques,
       yticks=range(len(Yuniques)), yticklabels=Yuniques)


ax.set_xlabel('CANDIDATE')
ax.set_ylabel('SENTIMENT')
ax.set_zlabel('PARTY')



plt.show()




ax.azim = 20
ax.dist = 10
ax.elev = 21


fig.savefig("KNN_Classified_Scatter.pdf".format(ax.azim, ax.dist, ax.elev), bbox_inches
```

In [ ]:

```
def report_best_scores(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

In [ ]:

```
#use ensembling to yield precise fit in feature importance
#optimize parameters with grid-search
from scipy.stats import uniform, randint
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold, RandomizedSea
from sklearn import tree as TR

XG=XGBRegressor(objective="binary:logistic", random_state=42)

params = {
    "colsample_bytree": uniform(0.9, 0.1),
    "gamma": uniform(0, 0.5),
    "learning_rate": uniform(0.003, 0.3), # default 0.1
    "max_depth": randint(3, 10), # default 3
    "n_estimators": randint(1, 200 ) ,
    "subsample": uniform(0.9, 0.1)
}

search = RandomizedSearchCV(XG, param_distributions=params, random_state=42, n_iter=5,

search.fit(X_train, y_train)

report_best_scores(search.cv_results_, 1)
```

In [ ]:

```
search.best_estimator_
```

In [ ]:

```
xgb_model = XG.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)
```

In [ ]:

```
y_pred
```

In [ ]:

```
y_pred=np.rint(y_pred)
```

In [ ]:

```
print(confusion_matrix(y_test, y_pred))
```

In [ ]:

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

In [ ]:

```
import matplotlib.pylab as plt
from matplotlib import pyplot
from xgboost import plot_importance
plot_importance(xgb_model, max_num_features=20) # top 20 most important features
plt.show()
plt.savefig("feature_importance.pdf", bbox_inches='tight')
```

In [ ]:

```
#using max depth=6

from sklearn.tree import DecisionTreeClassifier

DT=DecisionTreeClassifier (random_state=1234, max_depth=6)

TREE = DT.fit(X_train, y_train)

pred_Tree=TREE.predict(X_test)

print(confusion_matrix(y_test, pred_Tree))
```

In [ ]:

```
print(classification_report(y_test, pred_Tree))
```

In [ ]:

```
from matplotlib import pyplot as plt
from sklearn import tree




fig = plt.figure(figsize=(25,25))
_ = tree.plot_tree(DT,
                   feature_names=X_test.columns,
                   filled=True)

fig.savefig("Decision_Tree.pdf", bbox_inches='tight')
```

In [ ]:

In [ ]:

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

In [ ]:

```
pred_NB  =  classifier.predict(X_test)
print(confusion_matrix(y_test, pred_NB))
```

In [ ]:

```
print(classification_report(y_test, pred_NB))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: