**FOSS Lab Final Project**

**Student Details**

Name: Mantu Kumar Gupta

Roll No.: 205319008

Programme: M.Sc. (CS)

Semester- 3

**Aim and Motivation**

I have chosen to create this project to practice and use Logistic Regression, K-Nearest Neighbor, Support Vector Machine, MLP Classifier, and Deep Learning (1 convolutional layer) for various tasks and applications. Apart from implementing these algorithms, I have also visualized and analyzed with the help of word cloud and nltk.

**Dataset**

Data is provided by the police department of Kerala Government. The dataset consists of over 1000 observations with 16 features such as place name, additional information and place name.

**Methodologies**

Graphs: To get visual representation of feature available in the dataset.

Word Cloud: To see most relevant words present there in the dataset.

Maps: To locate cases observed in different regions of the state.

Natural Language Processing to get insights with respect to the semantic meanings of complaints available in the dataset using NLTK package.

Table: It is used to get summarise view of the data.

Classification: Implementations of algorithms such as KNN, SVM, Logistic Regression, MLP classifier and deep Learning.

**Required Libraries**

Numpy: NumPy is a core Python Linear Algebra library for Data Science used for faster array processing than the native Python lists with a bunch of handy methods.

Pandas:  Pandas is a software library written for the Python programming language for data manipulation and analysis.

Wordcloud: Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance.

Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

NLTK: Natural Language Processing with Python NLTK is one of the leading platforms for working with human language data and Python, the module NLTK is used for natural language processing. NLTK is literally an acronym for Natural Language Toolkit.

Geopandas: GeoPandas is an open source project to make working with geospatial data in python easier.

Descartes: The descartes library provides a nicer integration of Shapely geometry objects with Matplotlib.

Sklearn: Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms.

Keras: Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow.

**Colab Link**

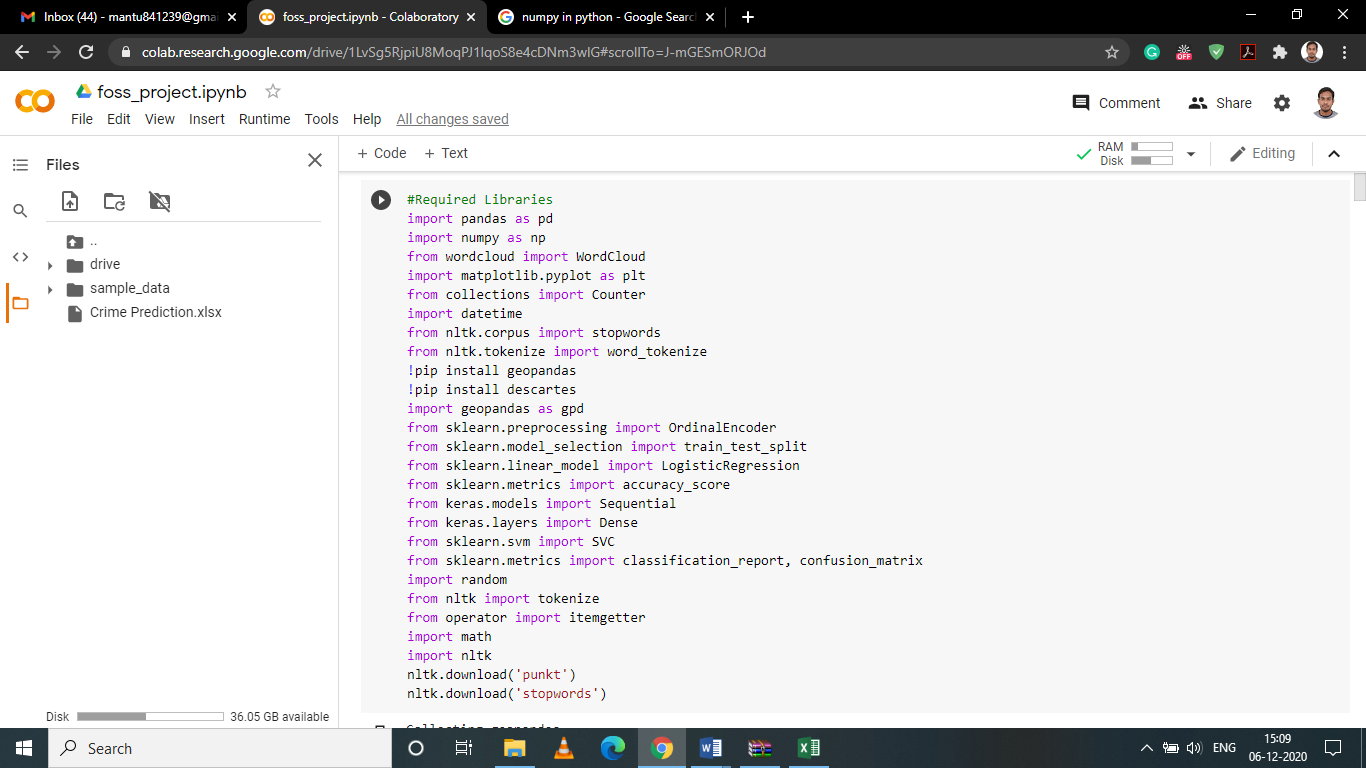
<https://colab.research.google.com/drive/1LvSg5RjpiU8MoqPJ1lqoS8e4cDNm3wIG?usp=sharing>

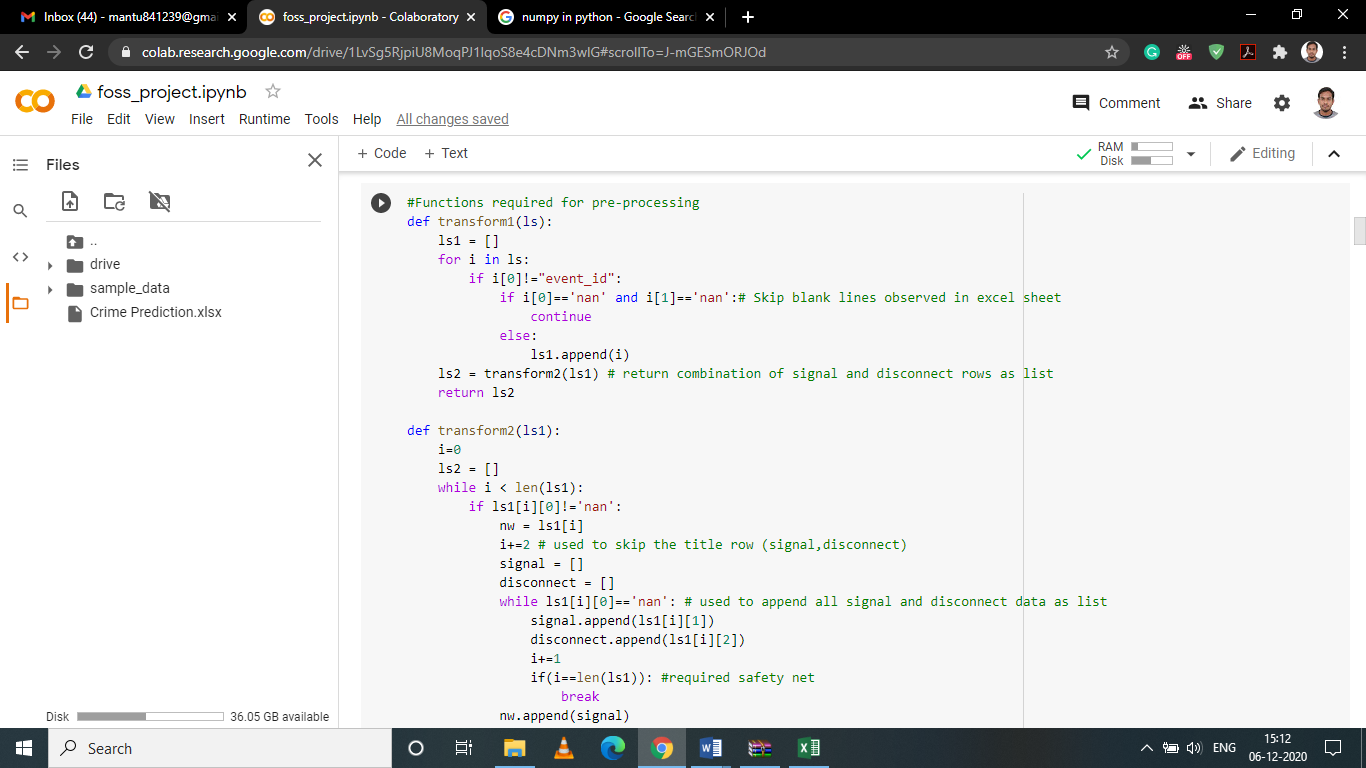
**Github Link**

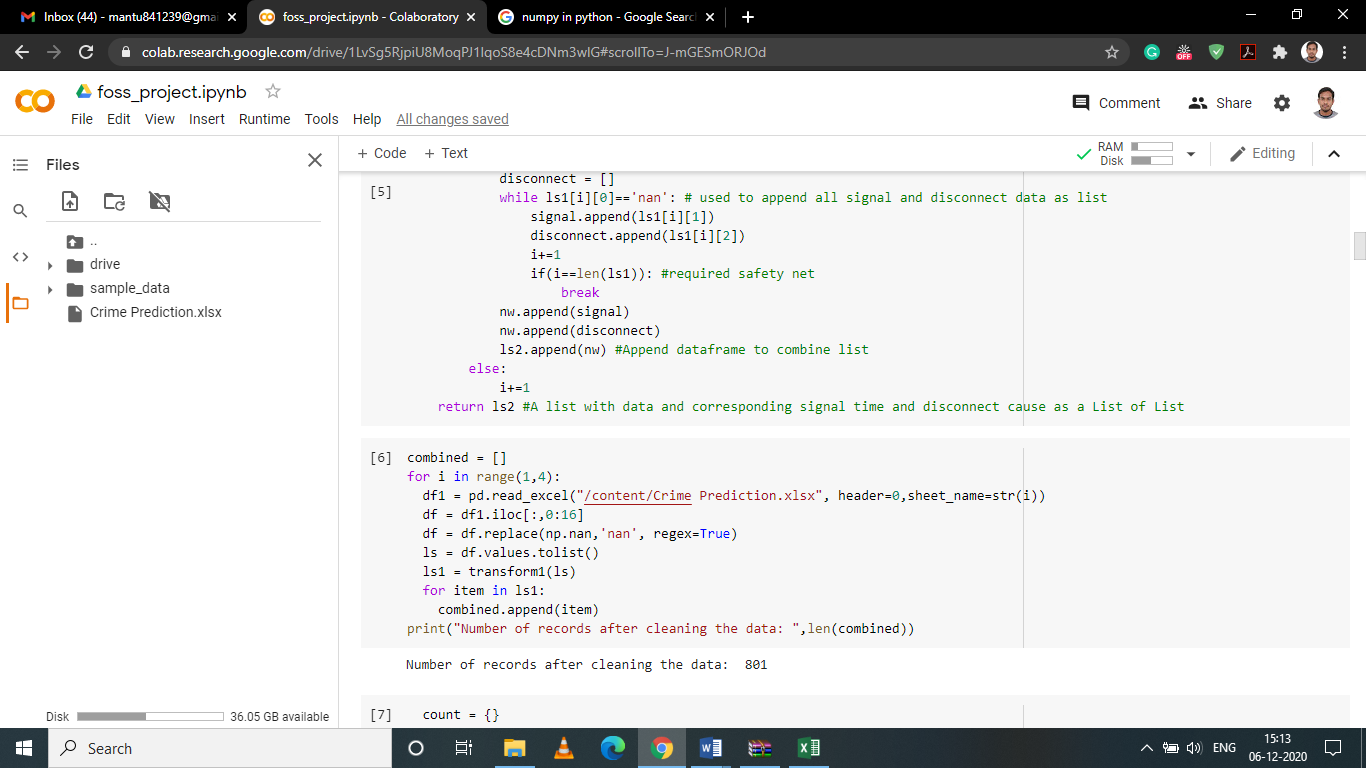
https://github.com/Debug-Entity9250/Foss\_final

**Snapshots of results along with explanation**

**Cleaning the dataset**

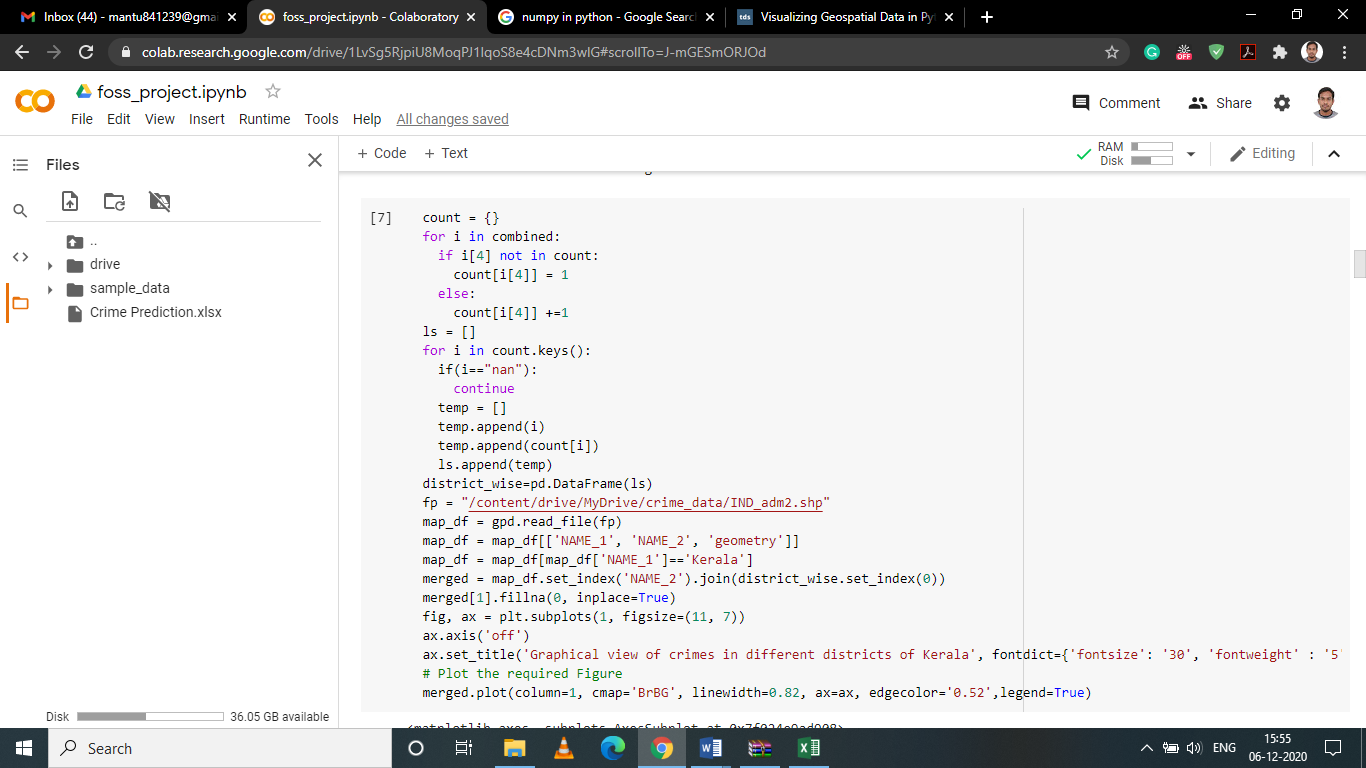
****

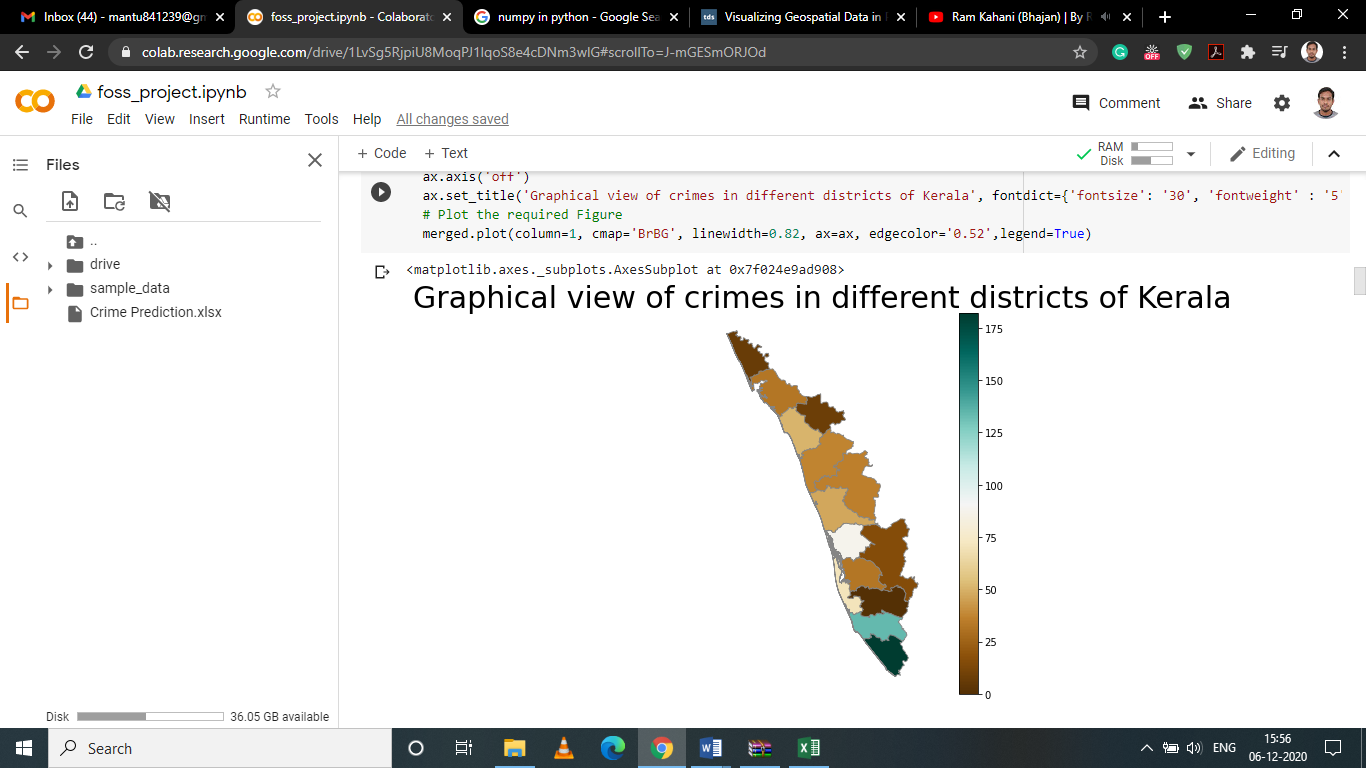
****

****

We have set of entries that are inconsistent. In order make it consistent we have to do modifications in the data. E.g. eliminating those rows that creates any kind of redundancy. These eliminations are not able to clean the data up to the mark. That is why we have to use cleaning with the help of python for the given dataset. We are using user defined functions to make data consistent. We are implementing these functions for each entry available in the excel sheet.

**Visualizing Geospatially**

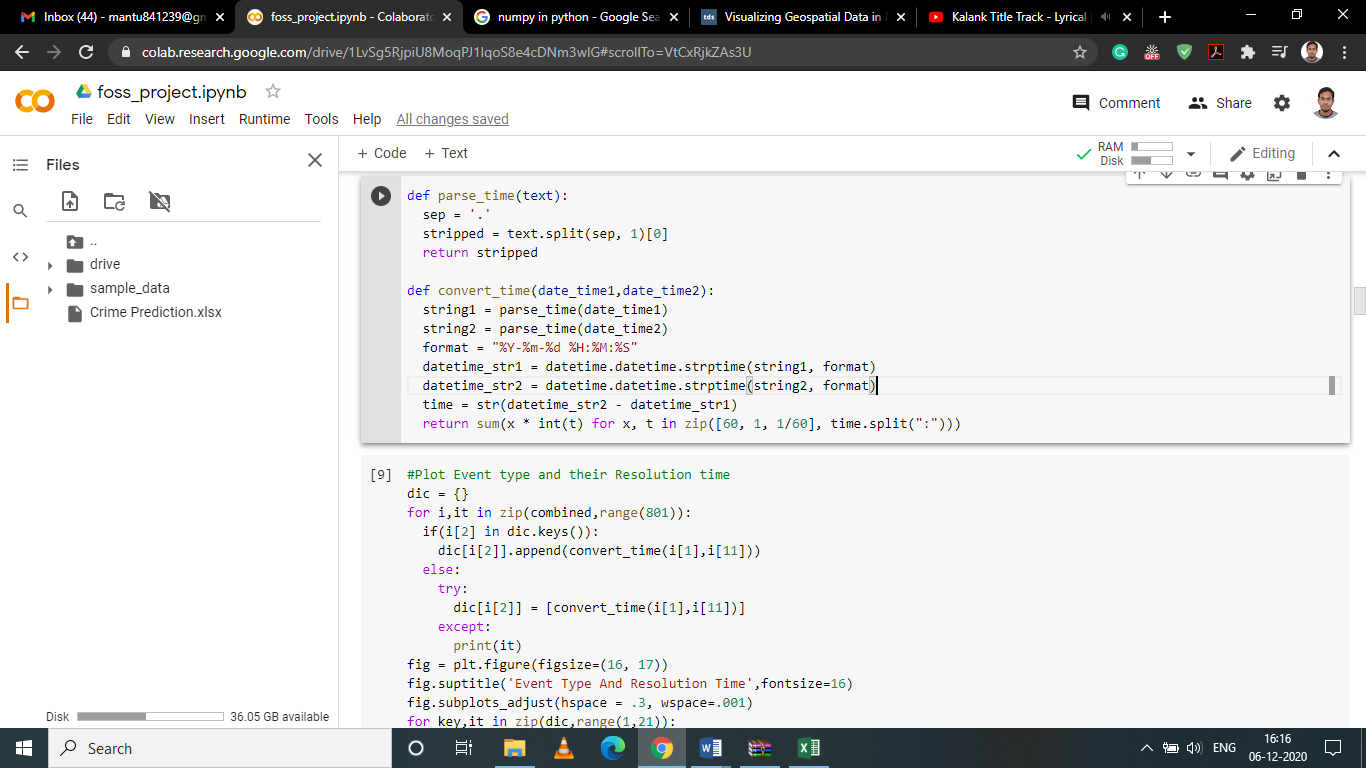
****

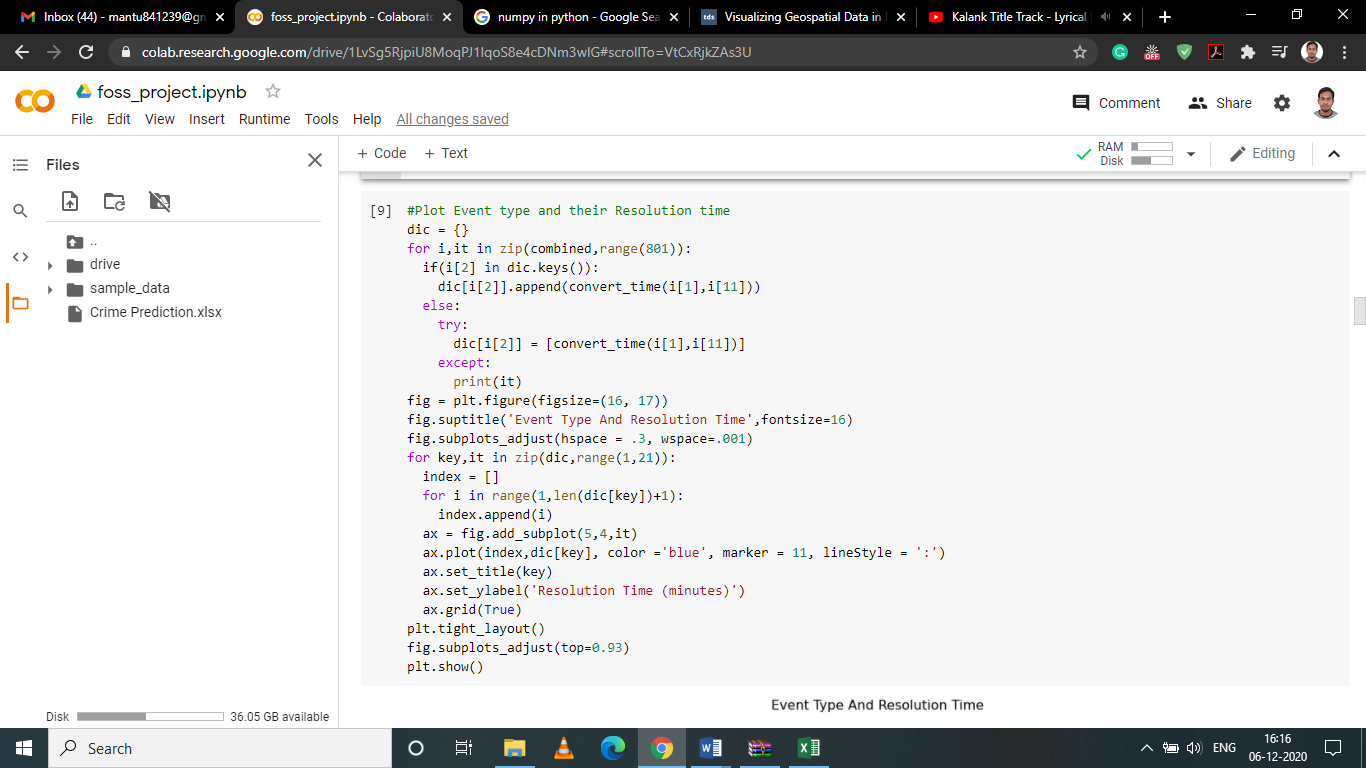
****

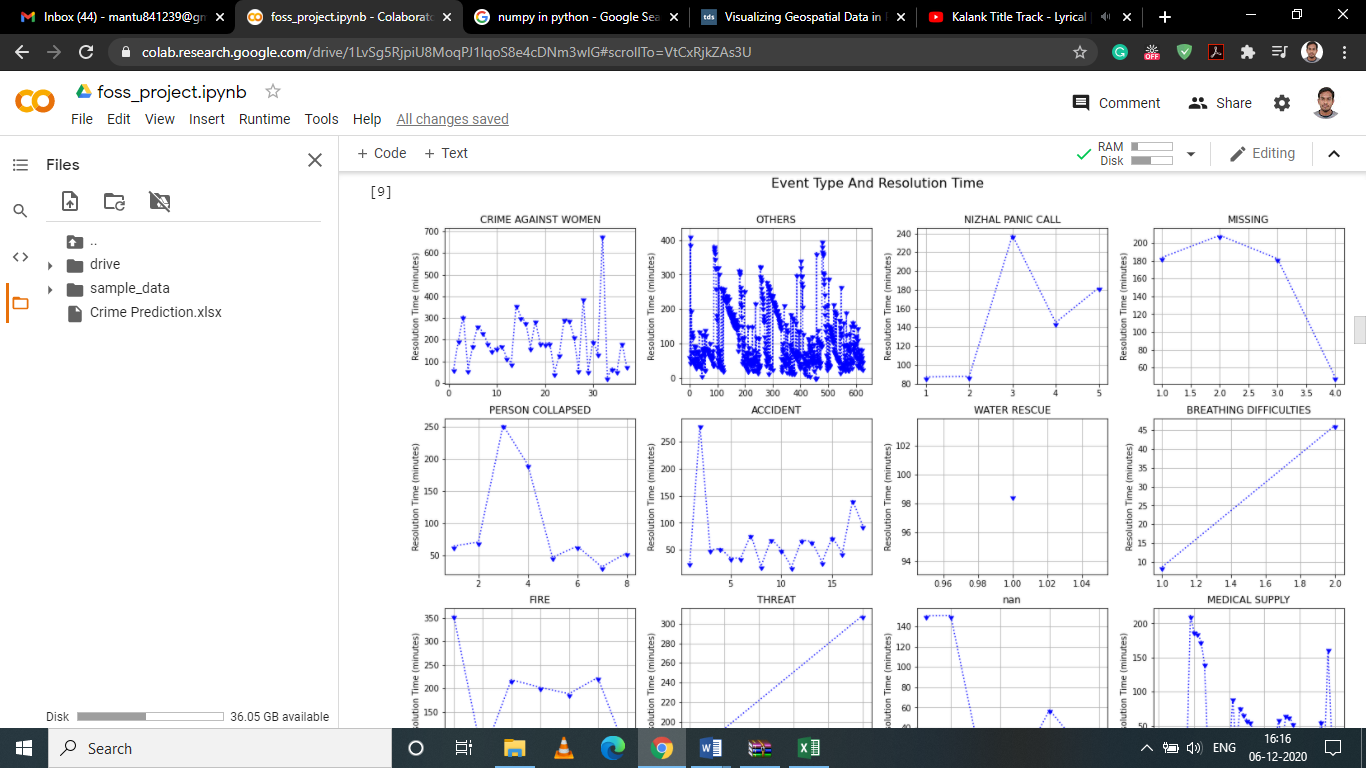
We are going to visualize the crimes reported in the sheet based on the districts of Kerala. For this purpose, we will be using the geopandas library which has the shape file of all districts in India. We shall combine the shape df with our own df of crimes reported and the district. The obtained result is a kind of map that represent crimes reported based on the different parameters in each district of the state. We observe that the brighter the color higher the number of calls received from the district.

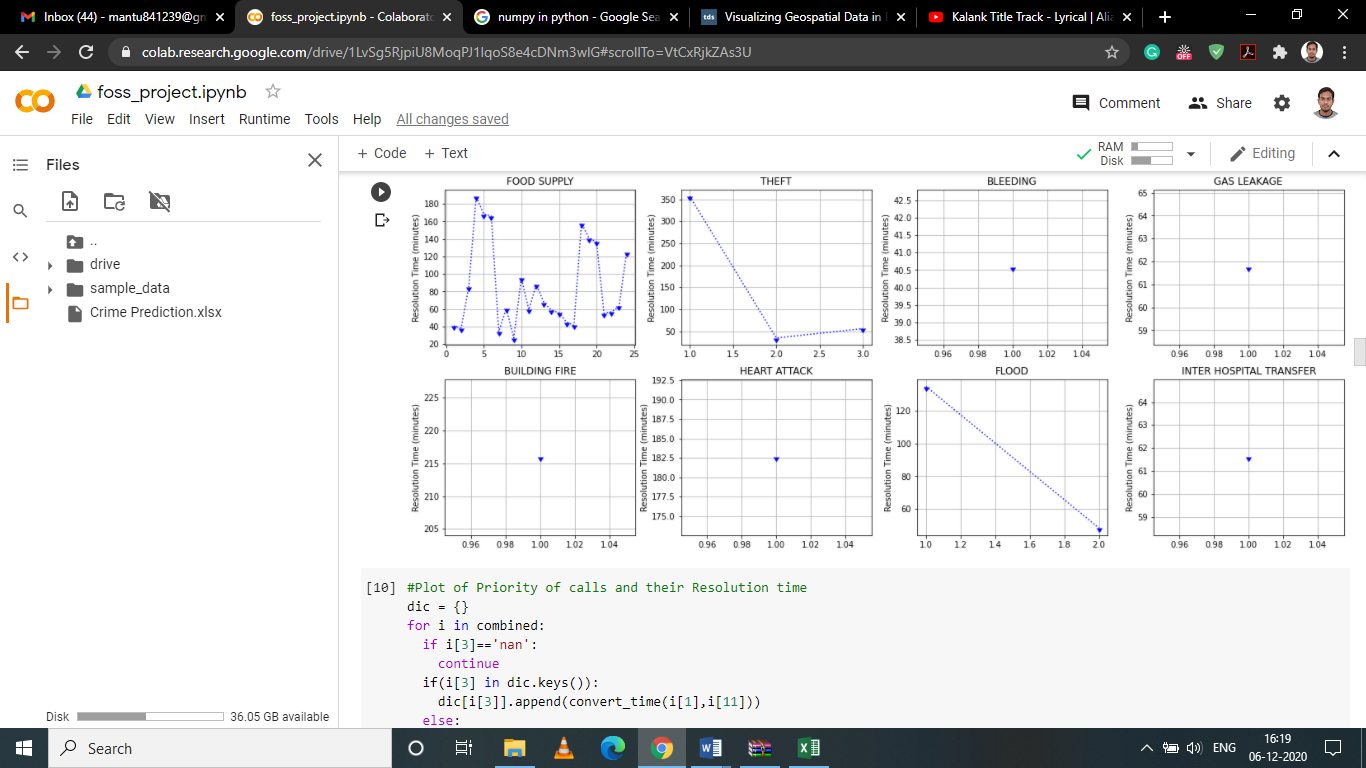
We observe that more calls were received from the southern part of the state than the northern. This can help officials to predict where police deployment and recruitment should be higher within the state.

**Graphs**

****

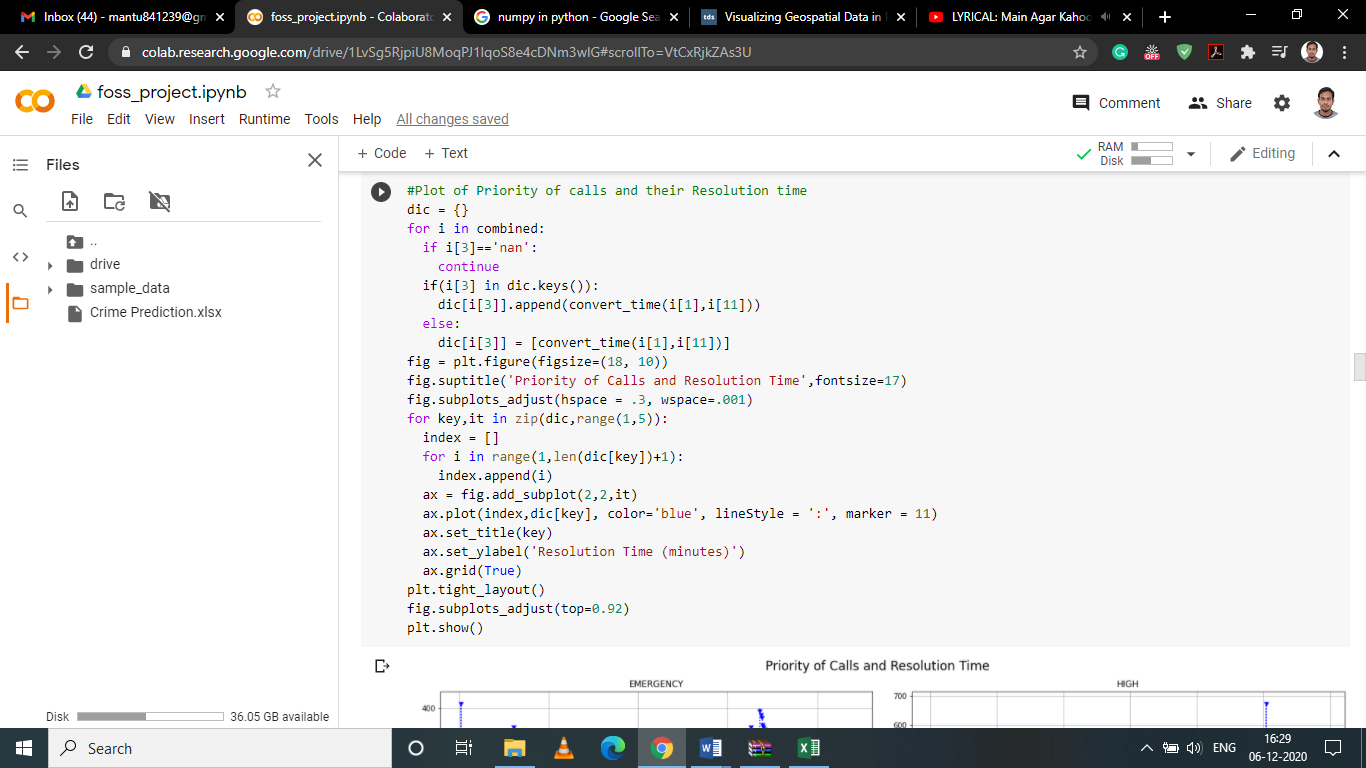
****

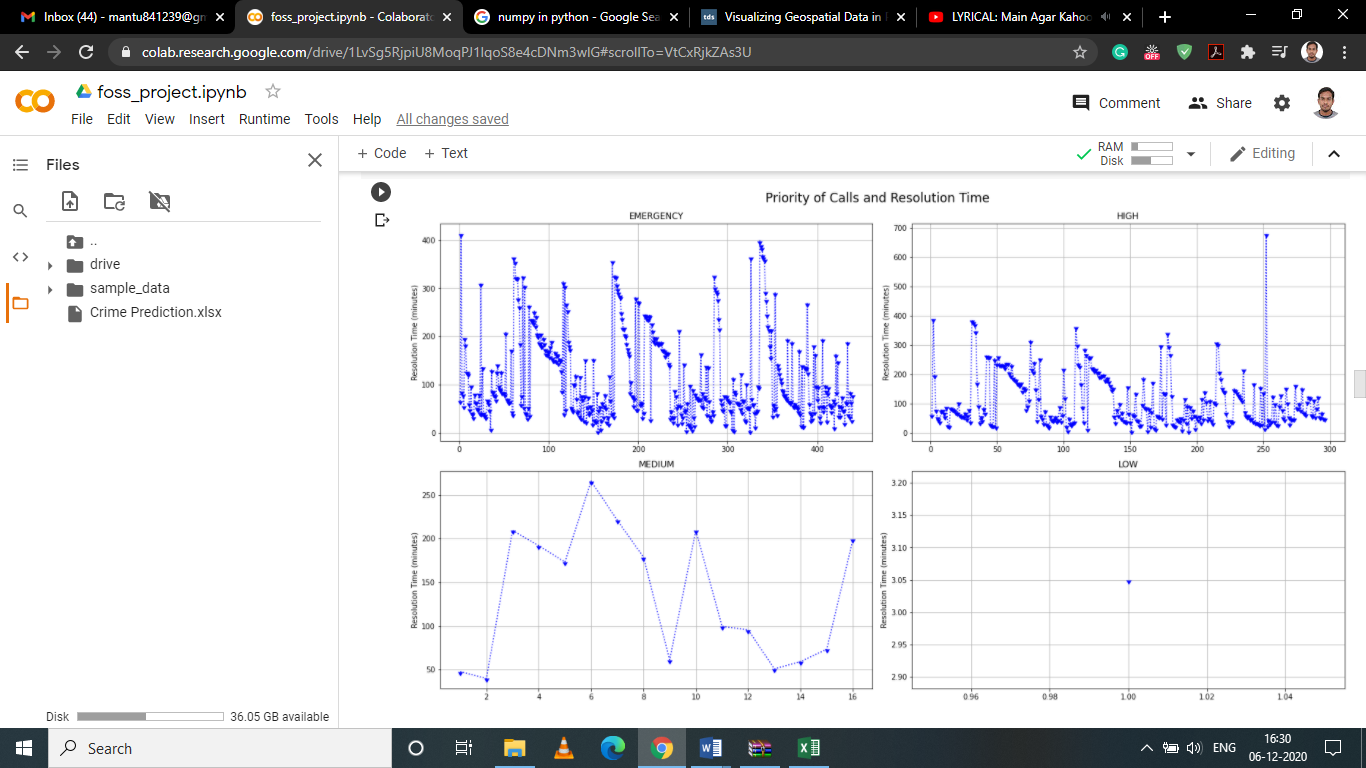
****

****

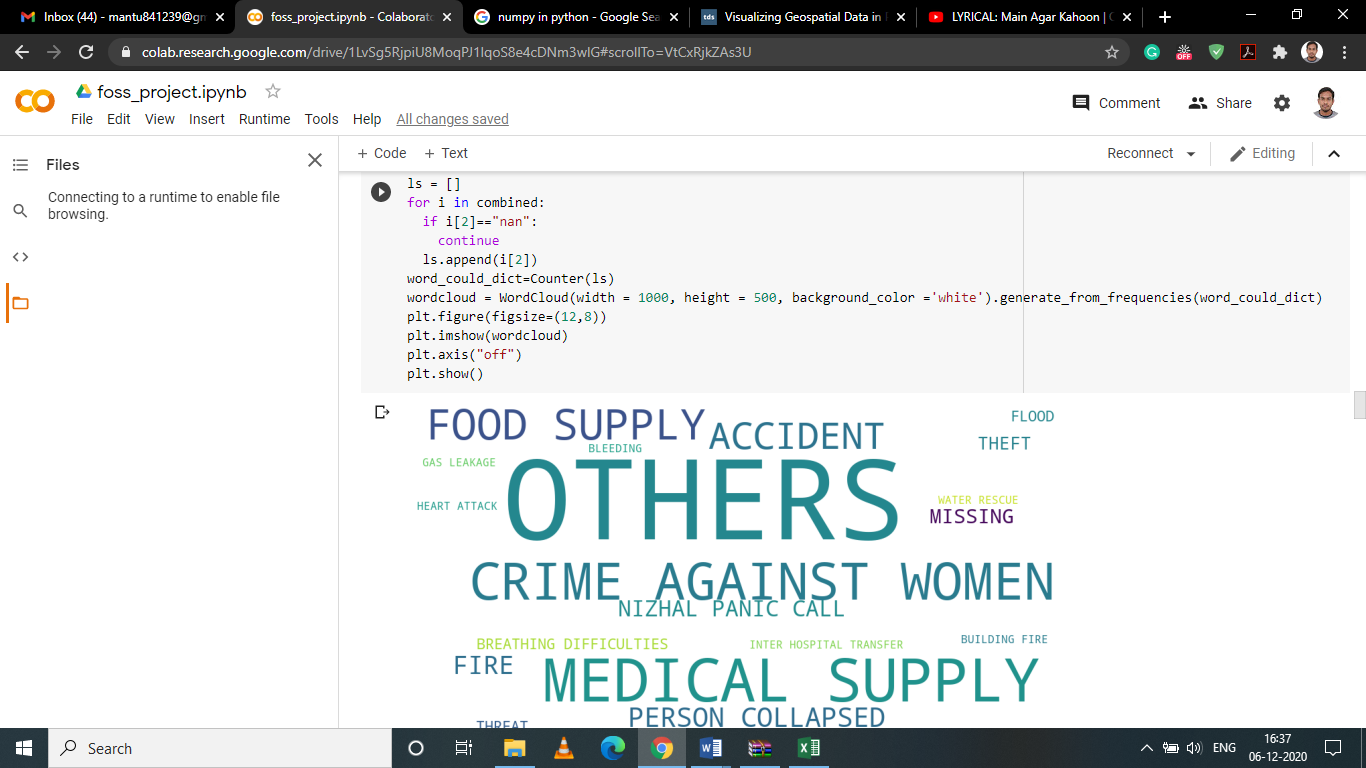
We have the start time and the closure time in the dataset. This helps us to get resolution time of calls. In order to get required answer, we have to clean the string because provided data is not in the standard form. We are going to use following functions and loop through the combined list to generate resolution times for all calls. We shall loop through the combined list collecting both the Event type and their Resolution times in a dictionary and plot the obtained result.

We are also implementing the same to plot the resolution time based on the Priority of calls.



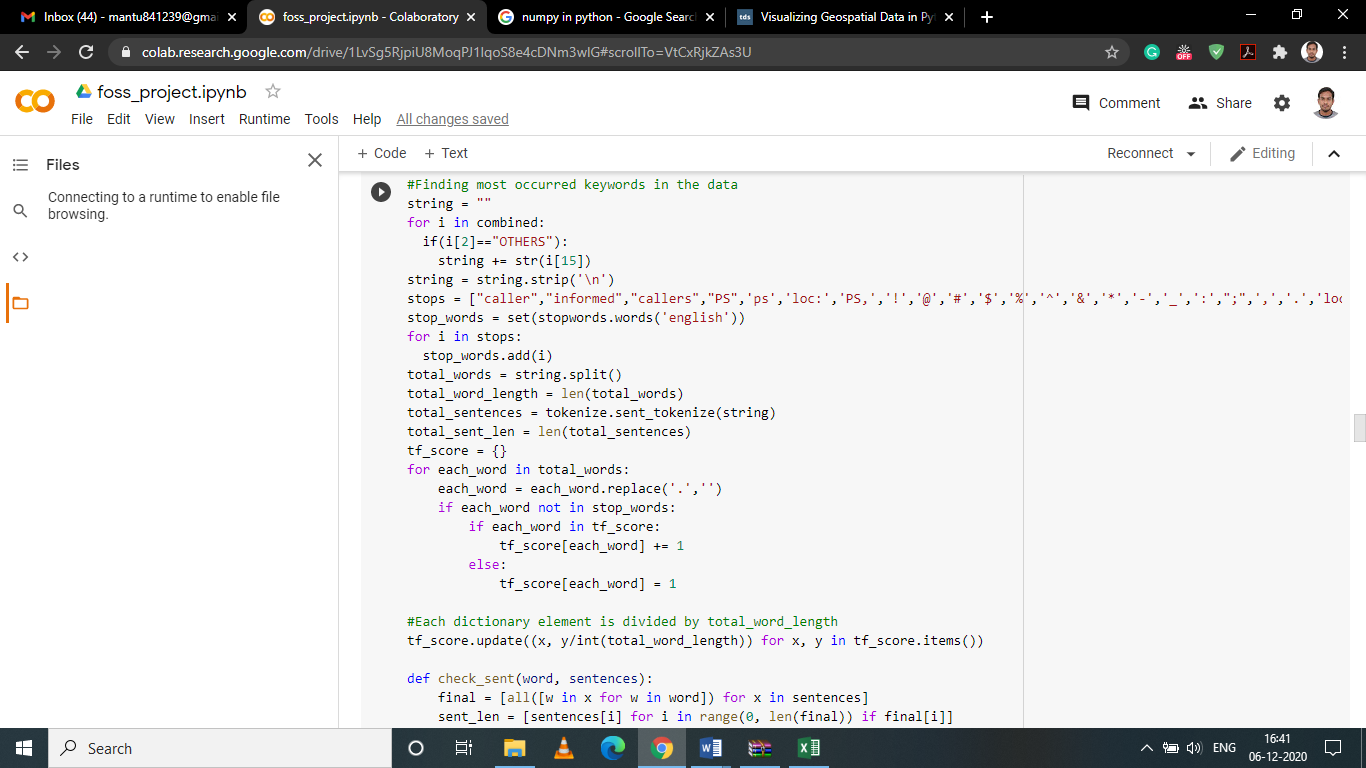


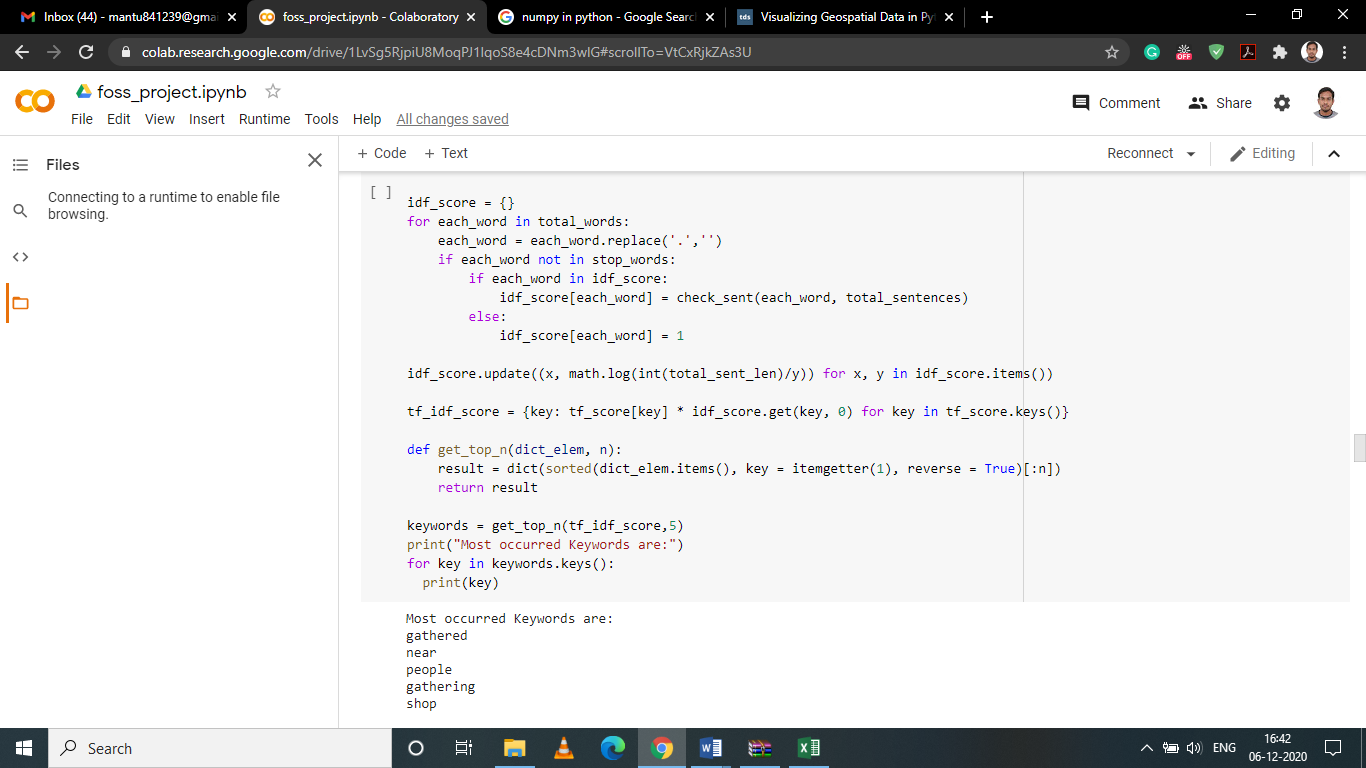
**Word Cloud**

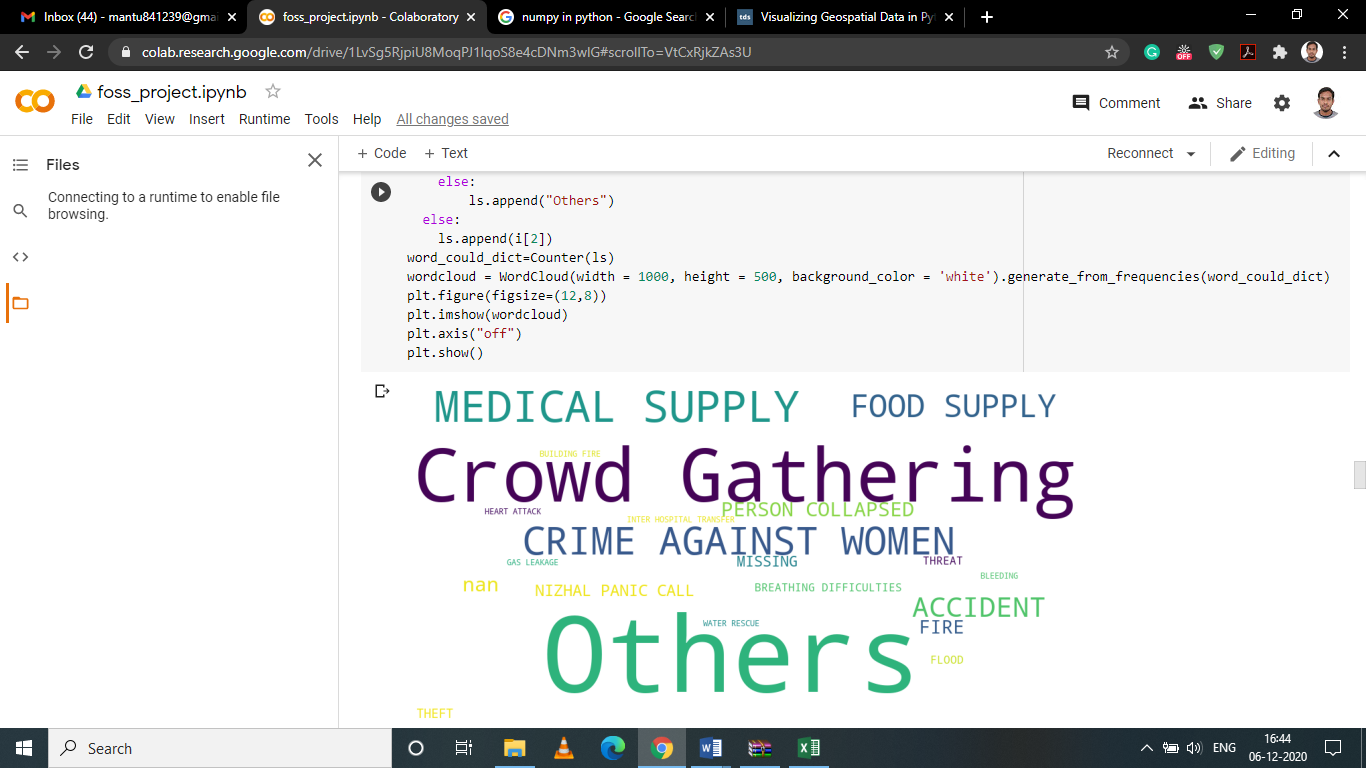


On the basis of frequency of event type, let us create a word cloud to visualize what are the main types of calls received during the period. We can observe different types of events being plotted based on their frequency. One observable point is that the most forthcoming outcome is of “OTHERS”. This means that most of the calls were not being classified properly. We can rectify this problem with the help of NLP.

**NLTK**

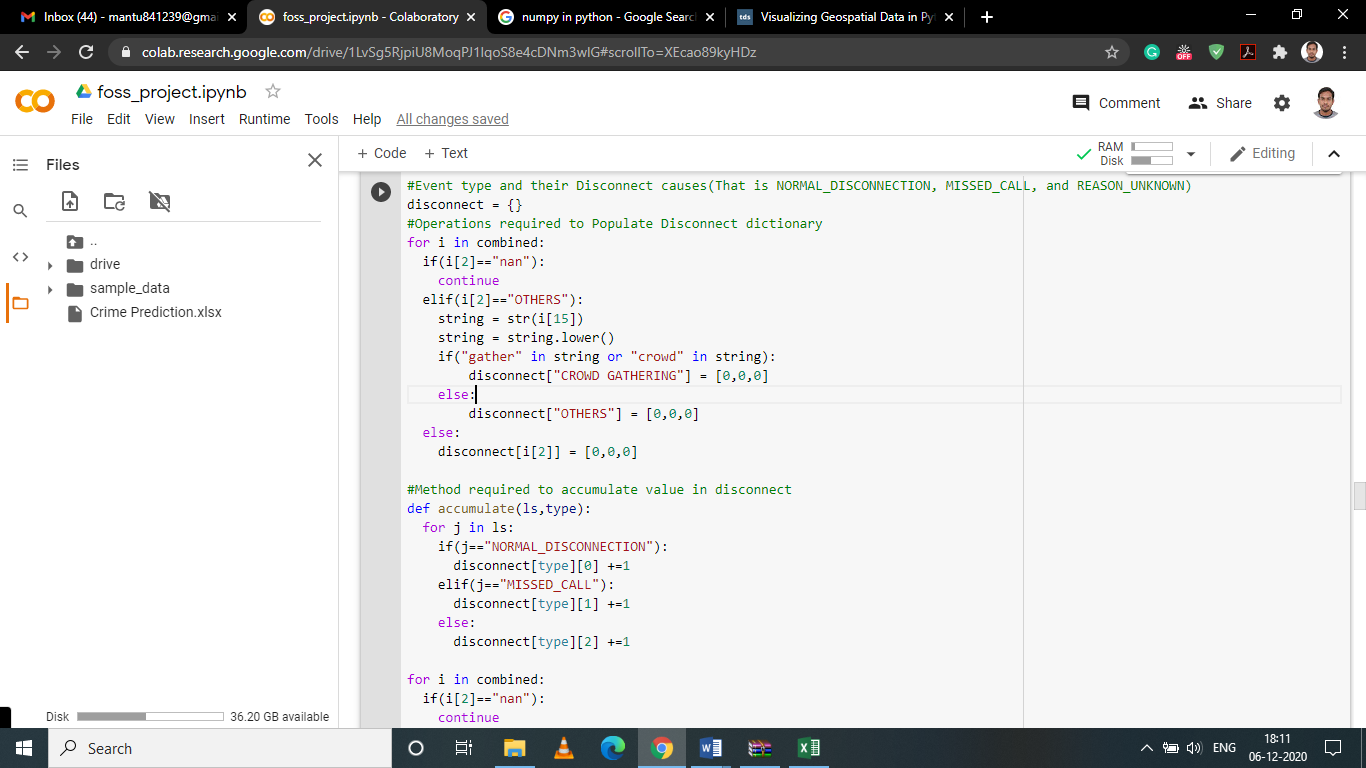
****

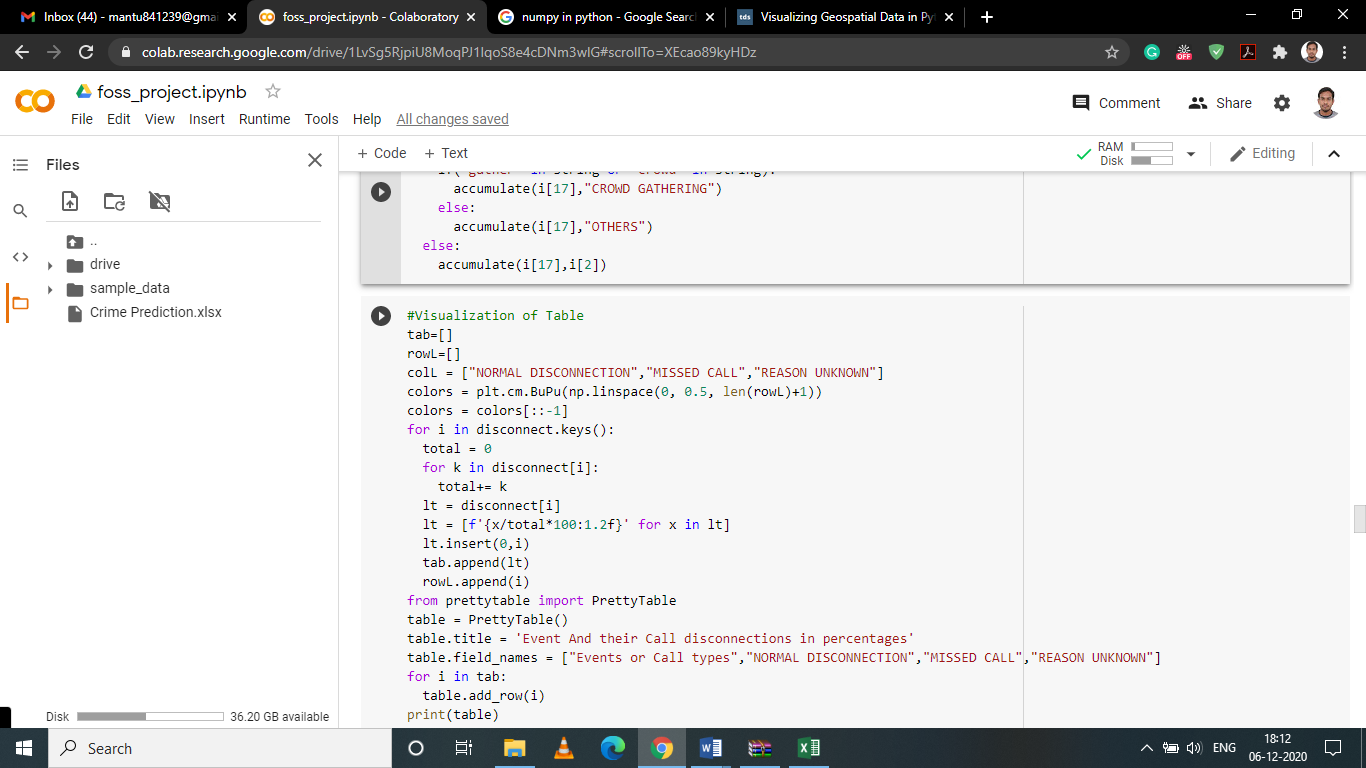
****

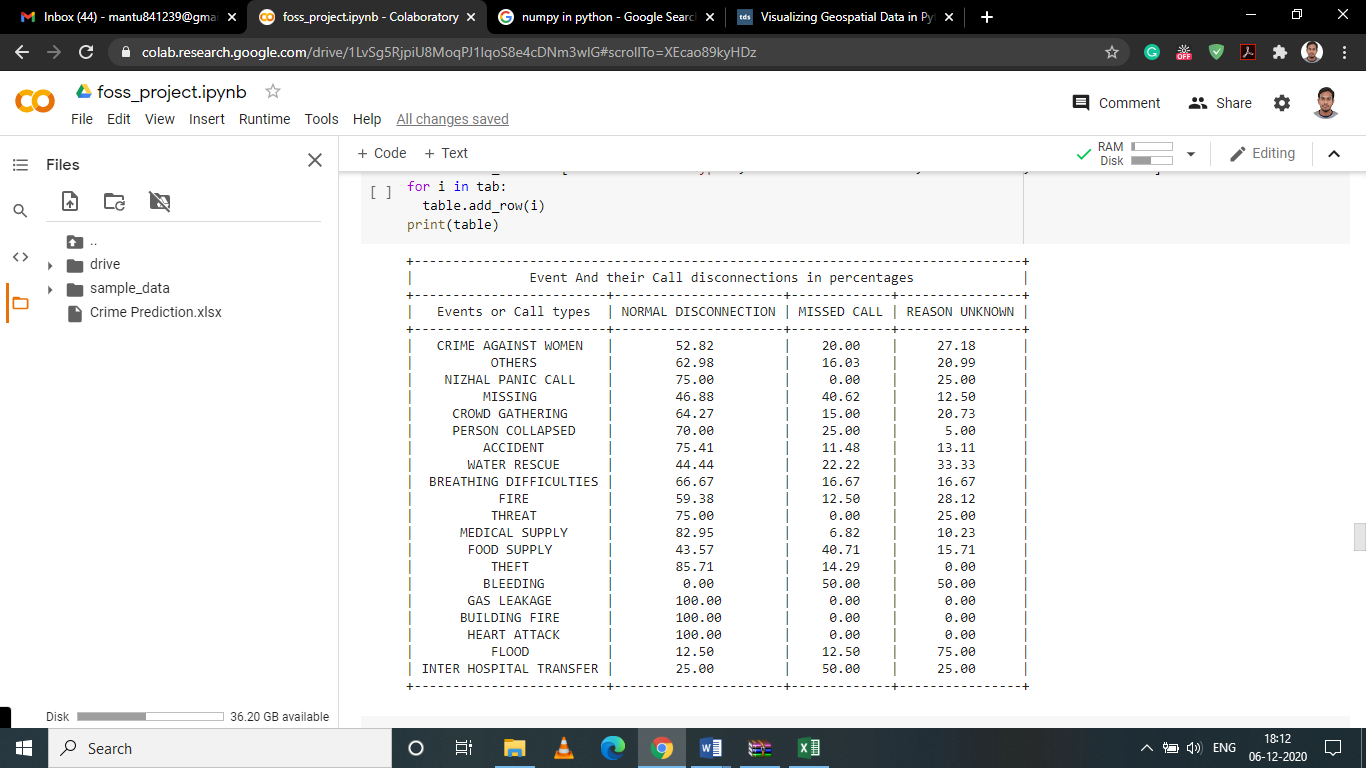


We observed that words such as “gathered”, “gathering”, “person” were used most often in calls. This should be true given that the calls were logged in the month of April. A nationwide lockdown was enforced due to the present Covid-19 situation. So it means many callers were complaining about Crowd Gathering.

**Table**





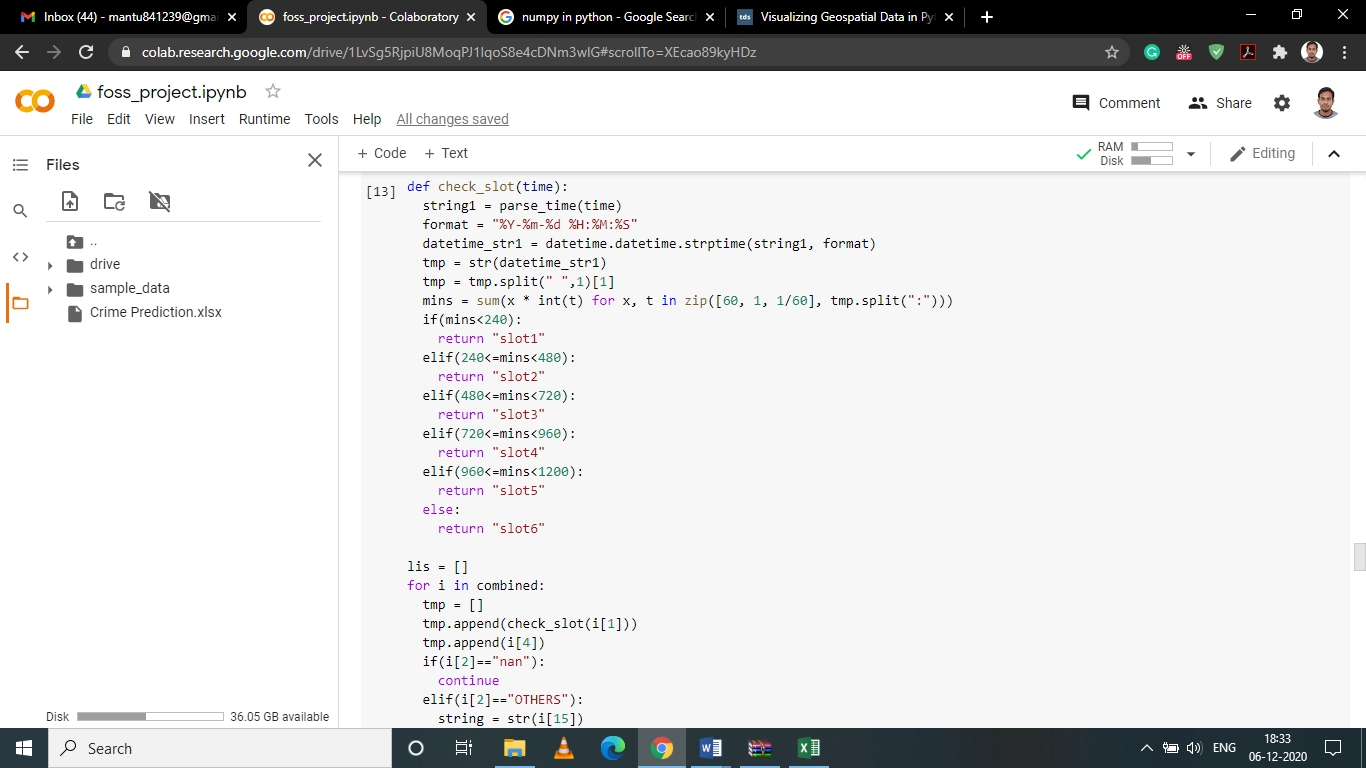


Now we are trying to replace the word others by crowd gathering wherever applicable. We can see immediate change in the output of the word cloud. Let us try to represent the percentage of calls and their disconnection types along with their event types. For this we will first build a list.

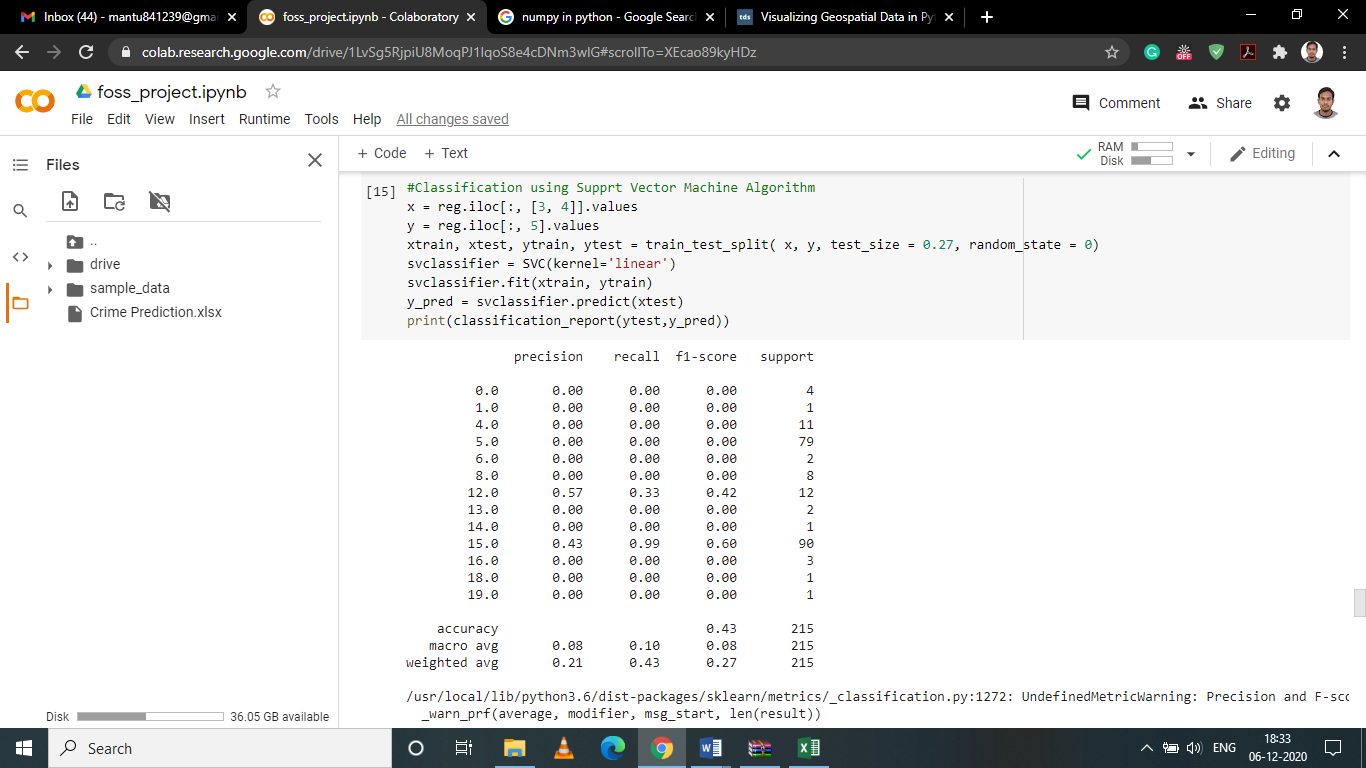
**Classification**

In the given sheet, we do not have any specific data of crimes to classify. Therefore, we need to make certain adjustments to run a classification model. For this purpose, I have selected three parameters, viz. call start time, district of the call, and the event reported in the call. The selection of these parameters is that if we can find good enough accuracy then we can predict what type of crime is reported generally at what times in a district. Suppose if we find that most of the calls for Crowd Gathering were purported at around 7 pm. So, we can deploy more police force during these times. Let us start by converting the call start time to different slots in the day and encoding district and types of crimes to perform classification.

check\_slot function used to get required features for analysis.

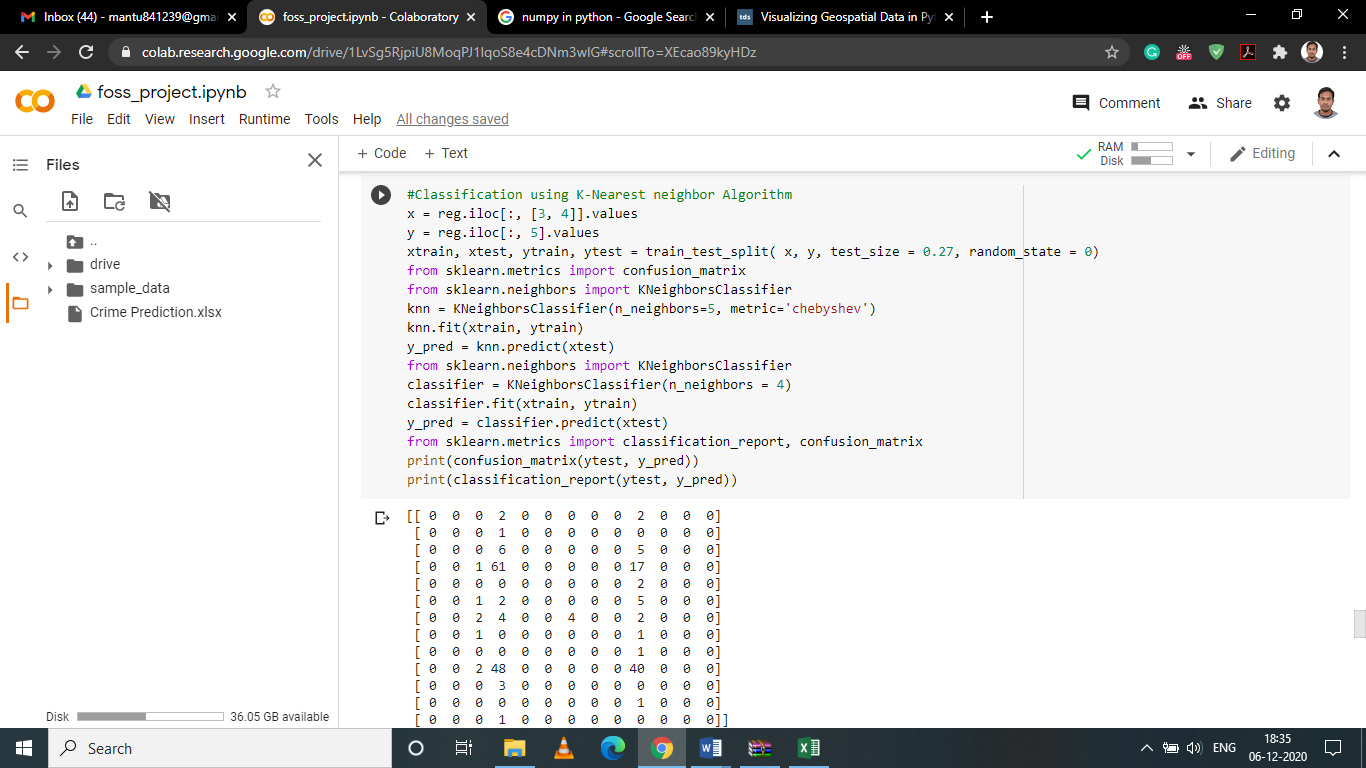


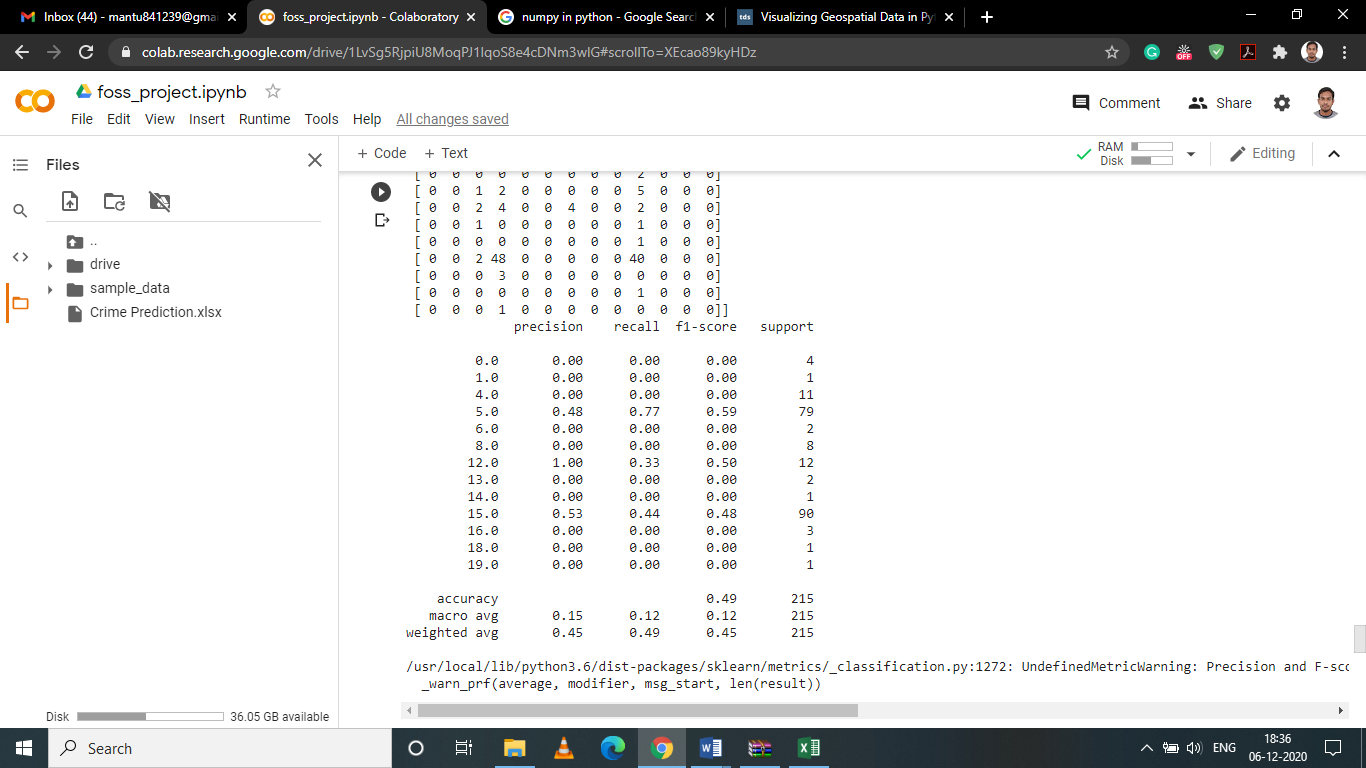
**SVM**



Using SVM classifier we get an accuracy of 43%.

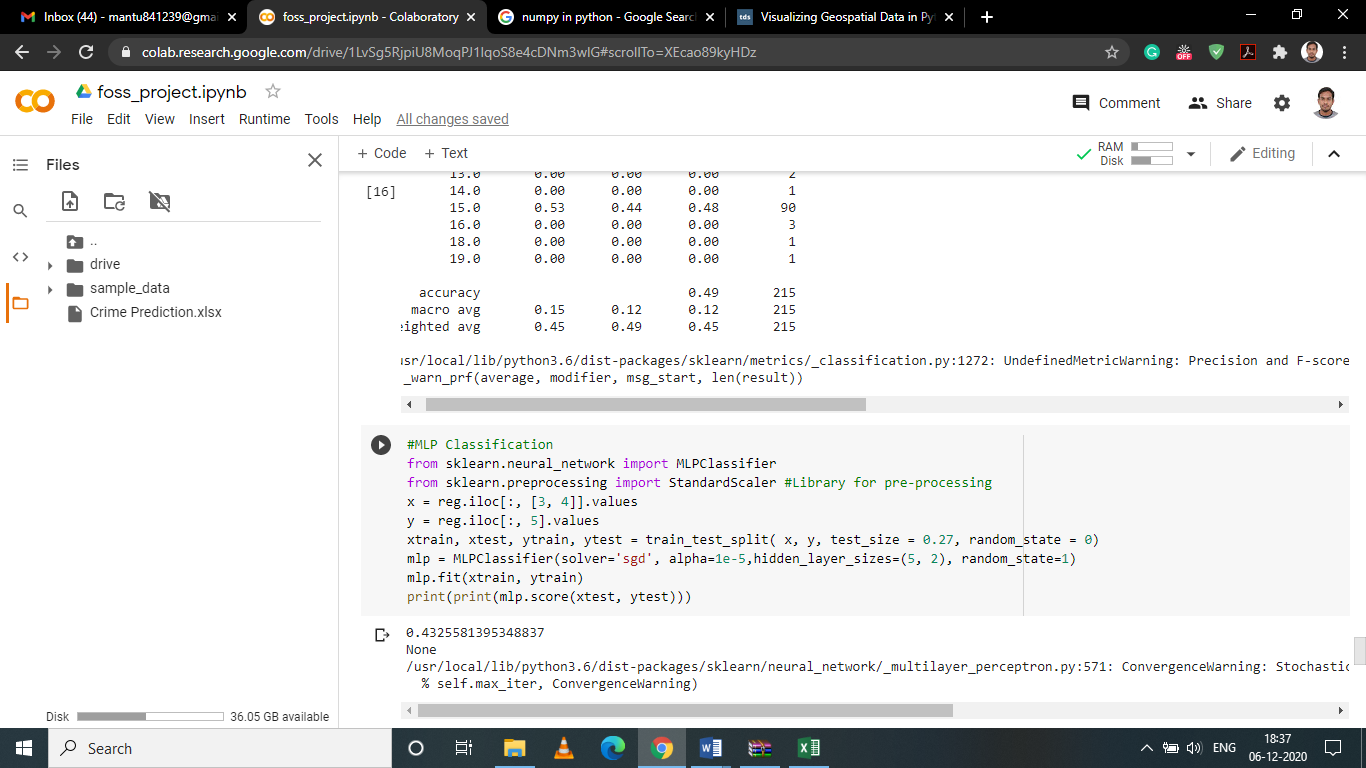
**KNN**





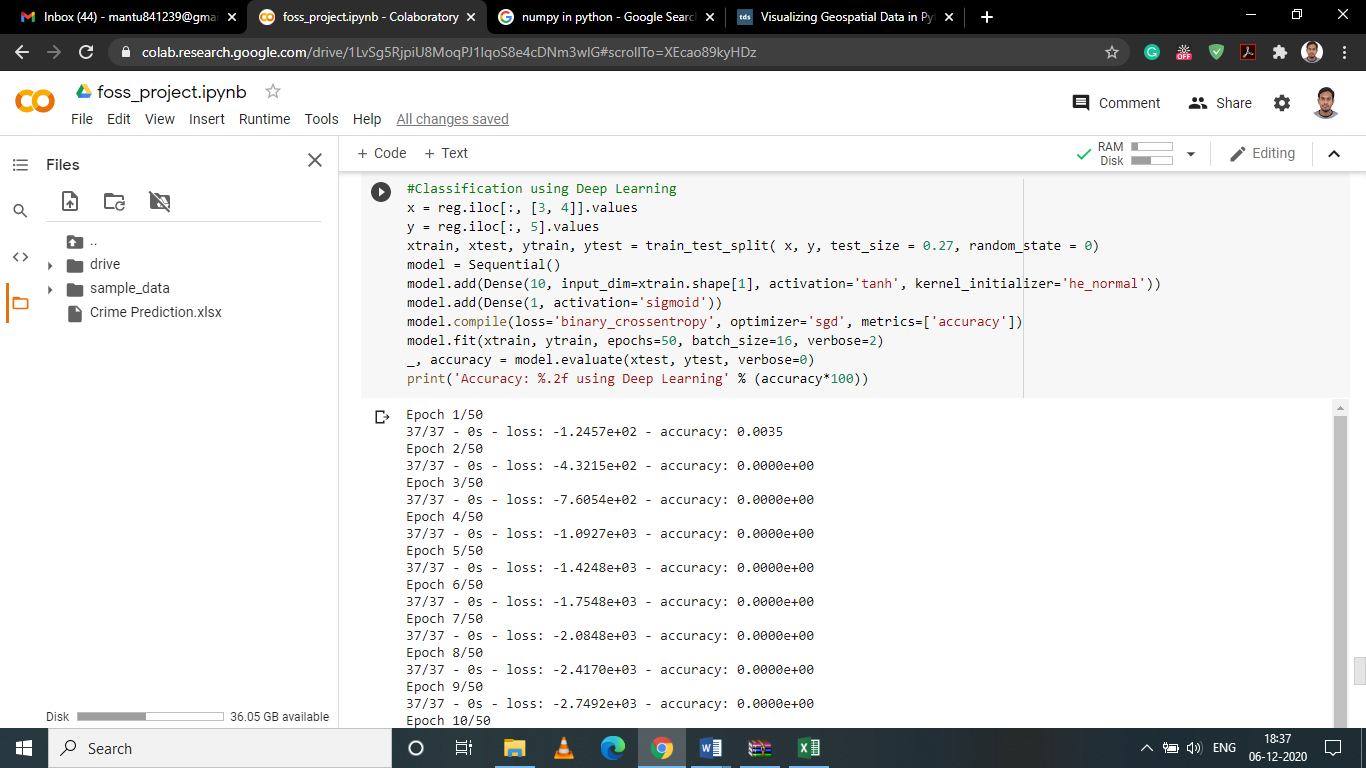
Using KNN we get an accuracy of 49%.

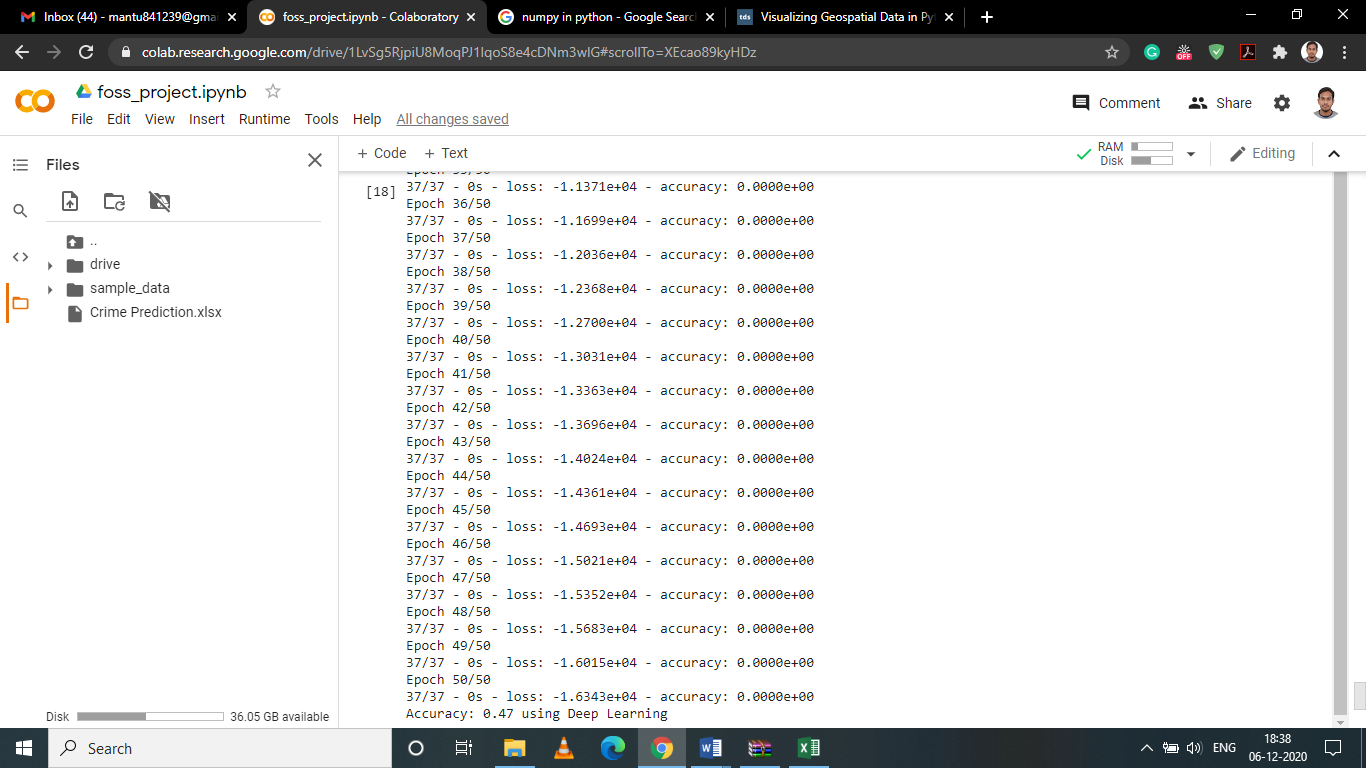
**MLP Classifier**



Using MLP classifier we get an accuracy of 43.25%.

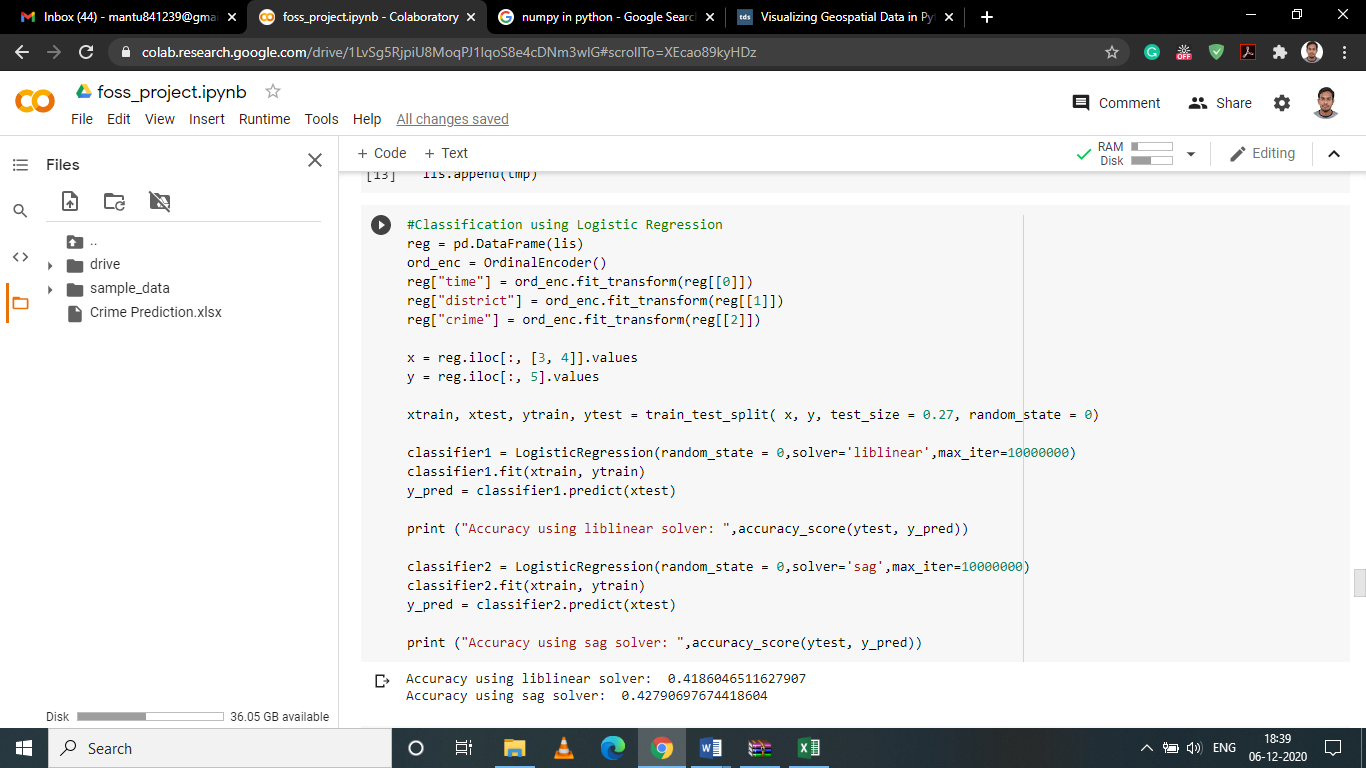
**Deep Learning (Simple Neural Network with 1 Convolutional Layer)**





Using simple neural network with 1 convolution layer we get an accuracy of 47%.

**Logistic Regression**



In this case we have two types of solver and their respective accuracies are given below:

Liblinear solver = 41.86%

Sag Solver = 42.79%

**Comparison Table**

|  |  |
| --- | --- |
| **Algorithm(s)** | **Accuracy(in %)** |
| **KNN** | **49** |
| **Logistic Regression** | **Liblinear solver = 41.86, Sag solver = 42.79** |
| **SVM** | **43** |
| **MLP** | **43.25** |
| **Deep Learning (1 convolutional layer)** | **47** |

**Based on the above table, we have KNN as the best model with an accuracy of 49%.**

**Python Code:**

#Required Libraries

import pandas as pd

import numpy as np

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from collections import Counter

import datetime

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

!pip install geopandas

!pip install descartes

import geopandas as gpd

from sklearn.preprocessing import OrdinalEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from keras.models import Sequential

from keras.layers import Dense

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, confusion\_matrix

import random

from nltk import tokenize

from operator import itemgetter

import math

import nltk

nltk.download('punkt')

nltk.download('stopwords')

#Functions required for pre-processing

def transform1(ls):

    ls1 = []

    for i in ls:

        if i[0]!="event\_id":

            if i[0]=='nan' and i[1]=='nan':# Skip blank lines observed in excel sheet

                continue

            else:

                ls1.append(i)

    ls2 = transform2(ls1) # return combination of signal and disconnect rows as list

    return ls2

def transform2(ls1):

    i=0

    ls2 = []

    while i < len(ls1):

        if ls1[i][0]!='nan':

            nw = ls1[i]

            i+=2 # used to skip the title row (signal,disconnect)

            signal = []

            disconnect = []

            while ls1[i][0]=='nan': # used to append all signal and disconnect data as list

                signal.append(ls1[i][1])

                disconnect.append(ls1[i][2])

                i+=1

                if(i==len(ls1)): #required safety net

                    break

            nw.append(signal)

            nw.append(disconnect)

            ls2.append(nw) #Append dataframe to combine list

        else:

            i+=1

    return ls2 #A list with data and corresponding signal time and disconnect cause as a List of List

combined = []

for i in range(1,4):

  df1 = pd.read\_excel("/content/Crime Prediction.xlsx", header=0,sheet\_name=str(i))

  df = df1.iloc[:,0:16]

  df = df.replace(np.nan,'nan', regex=True)

  ls = df.values.tolist()

  ls1 = transform1(ls)

  for item in ls1:

    combined.append(item)

print("Number of records after cleaning the data: ",len(combined))

  count = {}

  for i in combined:

    if i[4] not in count:

      count[i[4]] = 1

    else:

      count[i[4]] +=1

  ls = []

  for i in count.keys():

    if(i=="nan"):

      continue

    temp = []

    temp.append(i)

    temp.append(count[i])

    ls.append(temp)

  district\_wise=pd.DataFrame(ls)

  fp = "/content/drive/MyDrive/crime\_data/IND\_adm2.shp"

  map\_df = gpd.read\_file(fp)

  map\_df = map\_df[['NAME\_1', 'NAME\_2', 'geometry']]

  map\_df = map\_df[map\_df['NAME\_1']=='Kerala']

  merged = map\_df.set\_index('NAME\_2').join(district\_wise.set\_index(0))

  merged[1].fillna(0, inplace=True)

  fig, ax = plt.subplots(1, figsize=(11, 7))

  ax.axis('off')

  ax.set\_title('Graphical view of crimes in different districts of Kerala', fontdict={'fontsize': '30', 'fontweight' : '5'})

  # Plot the required Figure

  merged.plot(column=1, cmap='BrBG', linewidth=0.82, ax=ax, edgecolor='0.52',legend=True)

def parse\_time(text):

  sep = '.'

  stripped = text.split(sep, 1)[0]

  return stripped

def convert\_time(date\_time1,date\_time2):

  string1 = parse\_time(date\_time1)

  string2 = parse\_time(date\_time2)

  format = "%Y-%m-%d %H:%M:%S"

  datetime\_str1 = datetime.datetime.strptime(string1, format)

  datetime\_str2 = datetime.datetime.strptime(string2, format)

  time = str(datetime\_str2 - datetime\_str1)

  return sum(x \* int(t) for x, t in zip([60, 1, 1/60], time.split(":")))

#Plot Event type and their Resolution time

dic = {}

for i,it in zip(combined,range(801)):

  if(i[2] in dic.keys()):

    dic[i[2]].append(convert\_time(i[1],i[11]))

  else:

    try:

      dic[i[2]] = [convert\_time(i[1],i[11])]

    except:

      print(it)

fig = plt.figure(figsize=(16, 17))

fig.suptitle('Event Type And Resolution Time',fontsize=16)

fig.subplots\_adjust(hspace = .3, wspace=.001)

for key,it in zip(dic,range(1,21)):

  index = []

  for i in range(1,len(dic[key])+1):

    index.append(i)

  ax = fig.add\_subplot(5,4,it)

  ax.plot(index,dic[key], color ='blue', marker = 11, lineStyle = ':')

  ax.set\_title(key)

  ax.set\_ylabel('Resolution Time (minutes)')

  ax.grid(True)

plt.tight\_layout()

fig.subplots\_adjust(top=0.93)

plt.show()

#Plot of Priority of calls and their Resolution time

dic = {}

for i in combined:

  if i[3]=='nan':

    continue

  if(i[3] in dic.keys()):

    dic[i[3]].append(convert\_time(i[1],i[11]))

  else:

    dic[i[3]] = [convert\_time(i[1],i[11])]

fig = plt.figure(figsize=(18, 10))

fig.suptitle('Priority of Calls and Resolution Time',fontsize=17)

fig.subplots\_adjust(hspace = .3, wspace=.001)

for key,it in zip(dic,range(1,5)):

  index = []

  for i in range(1,len(dic[key])+1):

    index.append(i)

  ax = fig.add\_subplot(2,2,it)

  ax.plot(index,dic[key], color='blue', lineStyle = ':', marker = 11)

  ax.set\_title(key)

  ax.set\_ylabel('Resolution Time (minutes)')

  ax.grid(True)

plt.tight\_layout()

fig.subplots\_adjust(top=0.92)

plt.show()

#WordCloud of the given dataset

ls = []

for i in combined:

  if i[2]=="nan":

    continue

  ls.append(i[2])

word\_could\_dict=Counter(ls)

wordcloud = WordCloud(width = 1000, height = 500, background\_color ='white').generate\_from\_frequencies(word\_could\_dict)

plt.figure(figsize=(12,8))

plt.imshow(wordcloud)

plt.axis("off")

plt.show()

#Finding most occurred keywords in the data

string = ""

for i in combined:

  if(i[2]=="OTHERS"):

    string += str(i[15])

string = string.strip('\n')

stops = ["caller","informed","callers","PS",'ps','loc:','PS,','!','@','#','$','%','^','&','\*','-','\_',':',";",',','.','loc;','SAYS','IN','fish']

stop\_words = set(stopwords.words('english'))

for i in stops:

  stop\_words.add(i)

total\_words = string.split()

total\_word\_length = len(total\_words)

total\_sentences = tokenize.sent\_tokenize(string)

total\_sent\_len = len(total\_sentences)

tf\_score = {}

for each\_word in total\_words:

    each\_word = each\_word.replace('.','')

    if each\_word not in stop\_words:

        if each\_word in tf\_score:

            tf\_score[each\_word] += 1

        else:

            tf\_score[each\_word] = 1

#Each dictionary element is divided by total\_word\_length

tf\_score.update((x, y/int(total\_word\_length)) for x, y in tf\_score.items())

def check\_sent(word, sentences):

    final = [all([w in x for w in word]) for x in sentences]

    sent\_len = [sentences[i] for i in range(0, len(final)) if final[i]]

    return int(len(sent\_len))

idf\_score = {}

for each\_word in total\_words:

    each\_word = each\_word.replace('.','')

    if each\_word not in stop\_words:

        if each\_word in idf\_score:

            idf\_score[each\_word] = check\_sent(each\_word, total\_sentences)

        else:

            idf\_score[each\_word] = 1

idf\_score.update((x, math.log(int(total\_sent\_len)/y)) for x, y in idf\_score.items())

tf\_idf\_score = {key: tf\_score[key] \* idf\_score.get(key, 0) for key in tf\_score.keys()}

def get\_top\_n(dict\_elem, n):

    result = dict(sorted(dict\_elem.items(), key = itemgetter(1), reverse = True)[:n])

    return result

keywords = get\_top\_n(tf\_idf\_score,5)

print("Most occurred Keywords are:")

for key in keywords.keys():

  print(key)

#Updated word cloud based on NLTK analysis

ls = []

for i in combined:

  if(i[2]=="OTHERS"):

    string = str(i[15])

    string = string.lower()

    if("gather" in string or "crowd" in string):

        ls.append("Crowd Gathering")

    else:

        ls.append("Others")

  else:

    ls.append(i[2])

word\_could\_dict=Counter(ls)

wordcloud = WordCloud(width = 1000, height = 500, background\_color = 'white').generate\_from\_frequencies(word\_could\_dict)

plt.figure(figsize=(12,8))

plt.imshow(wordcloud)

plt.axis("off")

plt.show()

#Event type and their Disconnect causes(That is NORMAL\_DISCONNECTION, MISSED\_CALL, and REASON\_UNKNOWN)

disconnect = {}

#Operations required to Populate Disconnect dictionary

for i in combined:

  if(i[2]=="nan"):

    continue

  elif(i[2]=="OTHERS"):

    string = str(i[15])

    string = string.lower()

    if("gather" in string or "crowd" in string):

        disconnect["CROWD GATHERING"] = [0,0,0]

    else:

        disconnect["OTHERS"] = [0,0,0]

  else:

    disconnect[i[2]] = [0,0,0]

#Method required to accumulate value in disconnect

def accumulate(ls,type):

  for j in ls:

    if(j=="NORMAL\_DISCONNECTION"):

      disconnect[type][0] +=1

    elif(j=="MISSED\_CALL"):

      disconnect[type][1] +=1

    else:

      disconnect[type][2] +=1

for i in combined:

  if(i[2]=="nan"):

    continue

  elif(i[2]=="OTHERS"):

    string = str(i[15])

    string = string.lower()

    if("gather" in string or "crowd" in string):

      accumulate(i[17],"CROWD GATHERING")

    else:

      accumulate(i[17],"OTHERS")

  else:

    accumulate(i[17],i[2])

#Visualization of Table

tab=[]

rowL=[]

colL = ["NORMAL DISCONNECTION","MISSED CALL","REASON UNKNOWN"]

colors = plt.cm.BuPu(np.linspace(0, 0.5, len(rowL)+1))

colors = colors[::-1]

for i in disconnect.keys():

  total = 0

  for k in disconnect[i]:

    total+= k

  lt = disconnect[i]

  lt = [f'{x/total\*100:1.2f}' for x in lt]

  lt.insert(0,i)

  tab.append(lt)

  rowL.append(i)

from prettytable import PrettyTable

table = PrettyTable()

table.title = 'Event And their Call disconnections in percentages'

table.field\_names = ["Events or Call types","NORMAL DISCONNECTION","MISSED CALL","REASON UNKNOWN"]

for i in tab:

  table.add\_row(i)

print(table)

def check\_slot(time):

  string1 = parse\_time(time)

  format = "%Y-%m-%d %H:%M:%S"

  datetime\_str1 = datetime.datetime.strptime(string1, format)

  tmp = str(datetime\_str1)

  tmp = tmp.split(" ",1)[1]

  mins = sum(x \* int(t) for x, t in zip([60, 1, 1/60], tmp.split(":")))

  if(mins<240):

    return "slot1"

  elif(240<=mins<480):

    return "slot2"

  elif(480<=mins<720):

    return "slot3"

  elif(720<=mins<960):

    return "slot4"

  elif(960<=mins<1200):

    return "slot5"

  else:

    return "slot6"

lis = []

for i in combined:

  tmp = []

  tmp.append(check\_slot(i[1]))

  tmp.append(i[4])

  if(i[2]=="nan"):

    continue

  elif(i[2]=="OTHERS"):

    string = str(i[15])

    string = string.lower()

    if("gather" in string or "crowd" in string):

        tmp.append("Crowd Gathering")

    else:

        tmp.append("Others")

  else:

    tmp.append(i[2])

  lis.append(tmp)

#Classification using Logistic Regression

reg = pd.DataFrame(lis)

ord\_enc = OrdinalEncoder()

reg["time"] = ord\_enc.fit\_transform(reg[[0]])

reg["district"] = ord\_enc.fit\_transform(reg[[1]])

reg["crime"] = ord\_enc.fit\_transform(reg[[2]])

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.27, random\_state = 0)

classifier1 = LogisticRegression(random\_state = 0,solver='liblinear',max\_iter=10000000)

classifier1.fit(xtrain, ytrain)

y\_pred = classifier1.predict(xtest)

print ("Accuracy using liblinear solver: ",accuracy\_score(ytest, y\_pred))

classifier2 = LogisticRegression(random\_state = 0,solver='sag',max\_iter=10000000)

classifier2.fit(xtrain, ytrain)

y\_pred = classifier2.predict(xtest)

print ("Accuracy using sag solver: ",accuracy\_score(ytest, y\_pred))

#Classification using Supprt Vector Machine Algorithm

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.27, random\_state = 0)

svclassifier = SVC(kernel='linear')

svclassifier.fit(xtrain, ytrain)

y\_pred = svclassifier.predict(xtest)

print(classification\_report(ytest,y\_pred))

#Classification using K-Nearest neighbor Algorithm

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.27, random\_state = 0)

from sklearn.metrics import confusion\_matrix

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5, metric='chebyshev')

knn.fit(xtrain, ytrain)

y\_pred = knn.predict(xtest)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 4)

classifier.fit(xtrain, ytrain)

y\_pred = classifier.predict(xtest)

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(ytest, y\_pred))

print(classification\_report(ytest, y\_pred))

#MLP Classification

from sklearn.neural\_network import MLPClassifier

from sklearn.preprocessing import StandardScaler #Library for pre-processing

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.27, random\_state = 0)

mlp = MLPClassifier(solver='sgd', alpha=1e-5,hidden\_layer\_sizes=(5, 2), random\_state=1)

mlp.fit(xtrain, ytrain)

print(print(mlp.score(xtest, ytest)))

#Classification using Deep Learning

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.27, random\_state = 0)

model = Sequential()

model.add(Dense(10, input\_dim=xtrain.shape[1], activation='tanh', kernel\_initializer='he\_normal'))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='sgd', metrics=['accuracy'])

model.fit(xtrain, ytrain, epochs=50, batch\_size=16, verbose=2)

\_, accuracy = model.evaluate(xtest, ytest, verbose=0)

print('Accuracy: %.2f using Deep Learning' % (accuracy\*100))