**FOSS Lab – Final Project**

Student Details:

Praful Kamble, 205319006

Dataset:

Kerala State 100 Helpline dataset

Description:

The dataset consists of over 1000 observations with 16 features such as call time, event type and district.

Task:

Our aim is to classify, visualize and analyze the crime data provided in the sheet.

Methodologies:

1. Graphs
2. Word Clouds
3. Maps
4. NLP using NLTK
5. Table
6. Classification using Logistic Regression, Deep Learning

Libraries and Requirements:

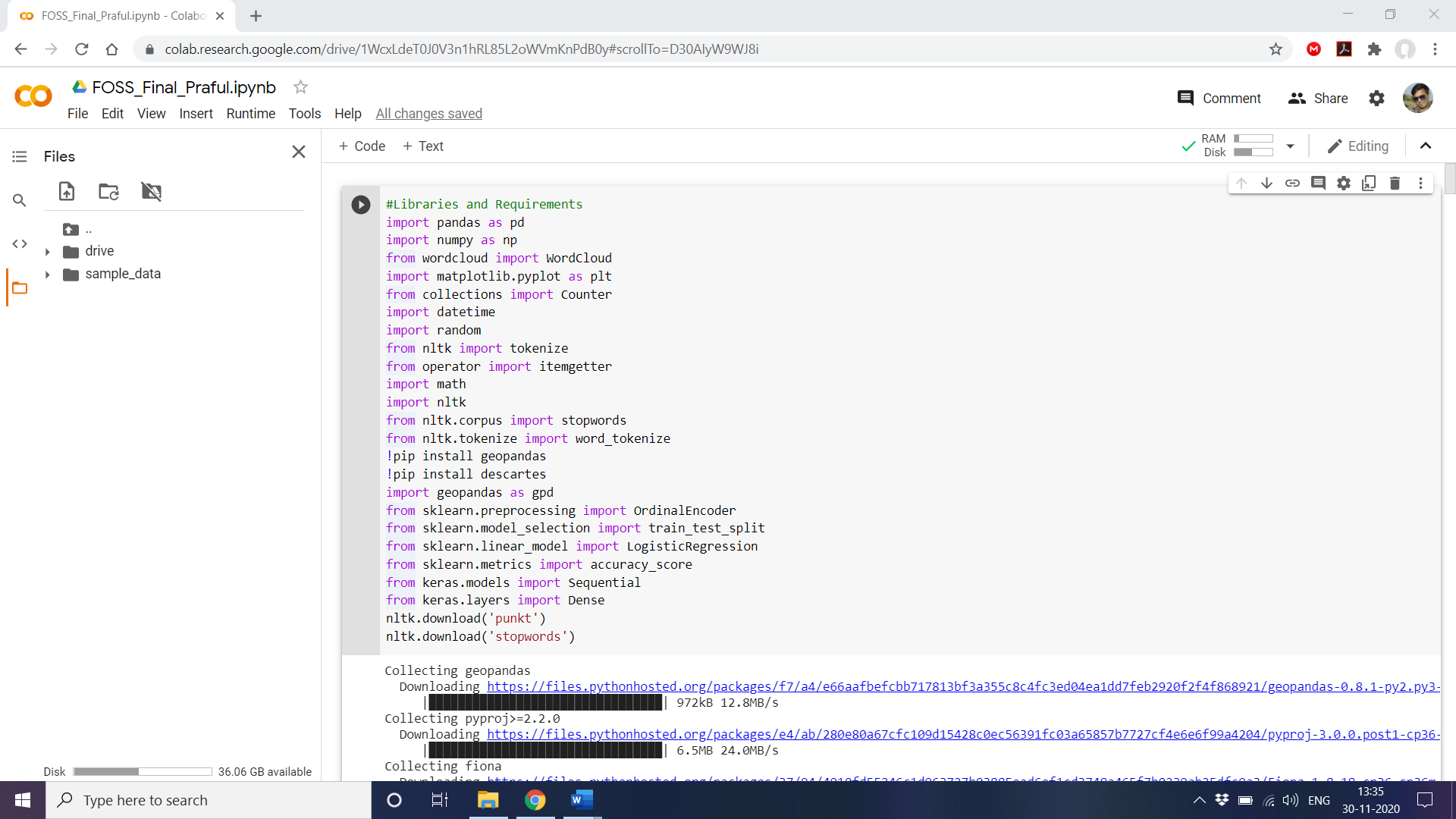
1. Pandas
2. Numpy
3. Matplotlib
4. Wordcloud
5. NLTK
6. Geopandas
7. Descartes
8. Sklearn
9. Keras

Github:

<https://github.com/praful055/FossFinal/blob/main/FOSS_Final_Praful.ipynb>

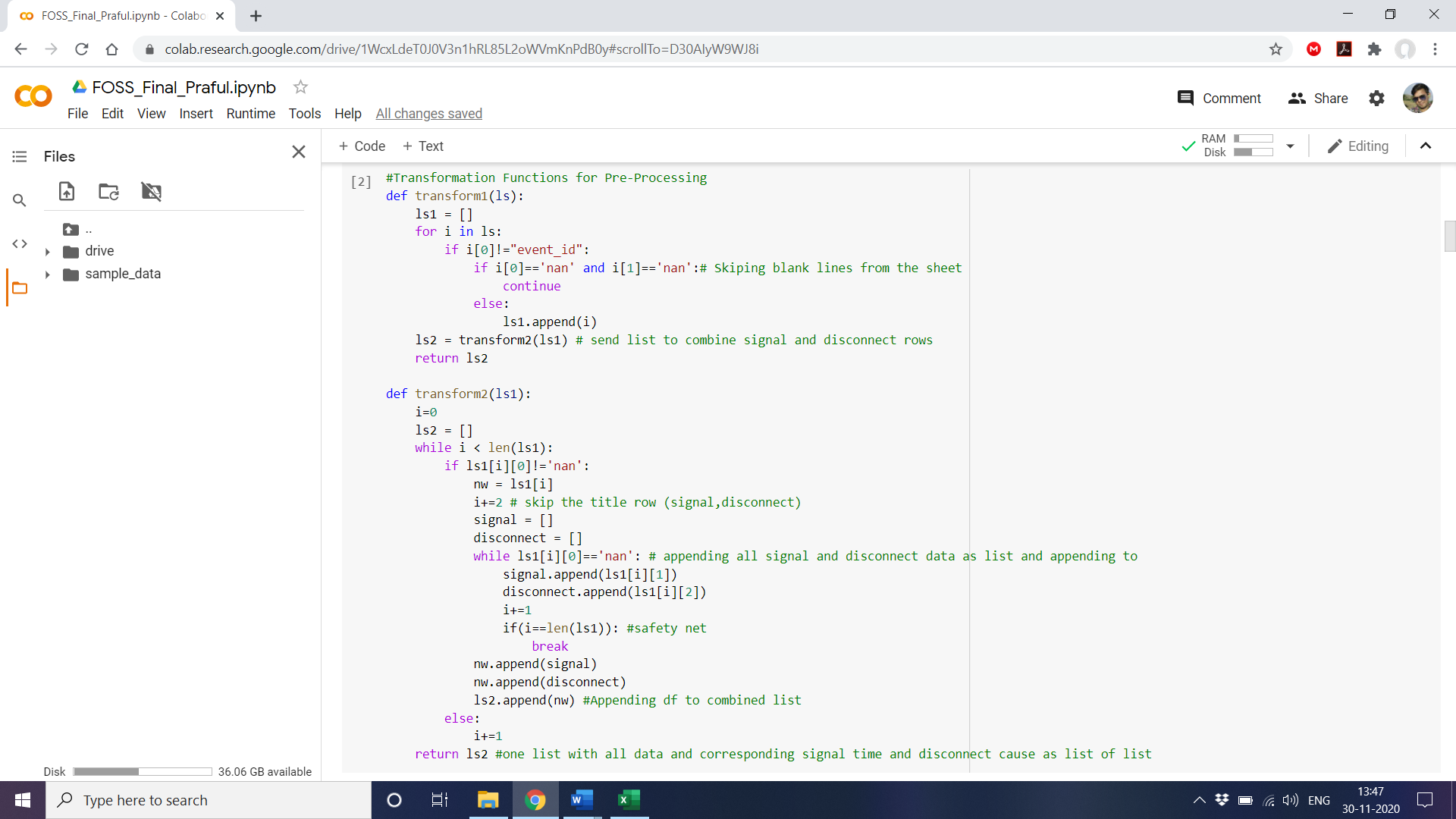
Colab:

<https://colab.research.google.com/drive/1WcxLdeT0J0V3n1hRL85L2oWVmKnPdB0y?usp=sharing>

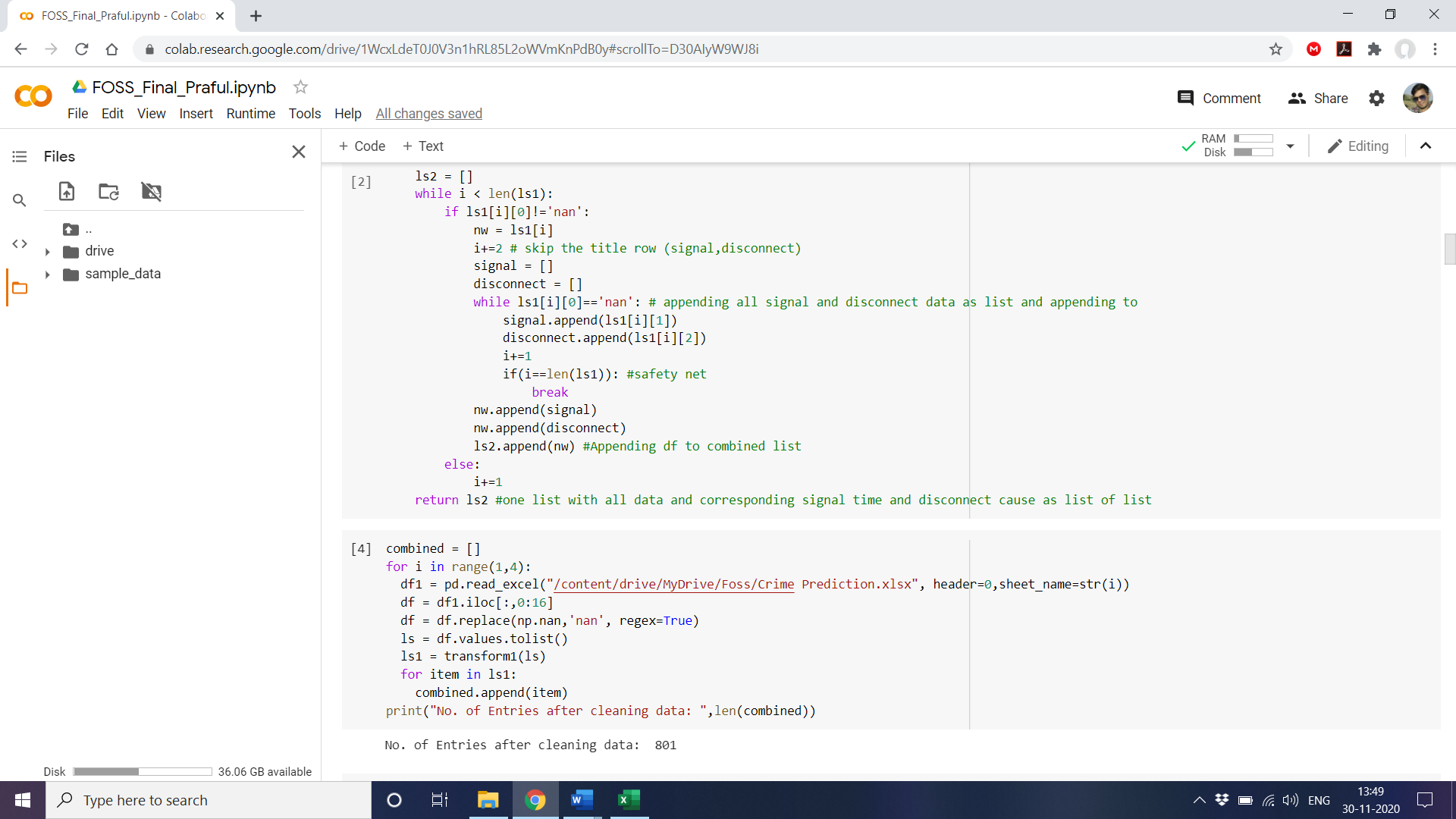


**Pre-Processing**

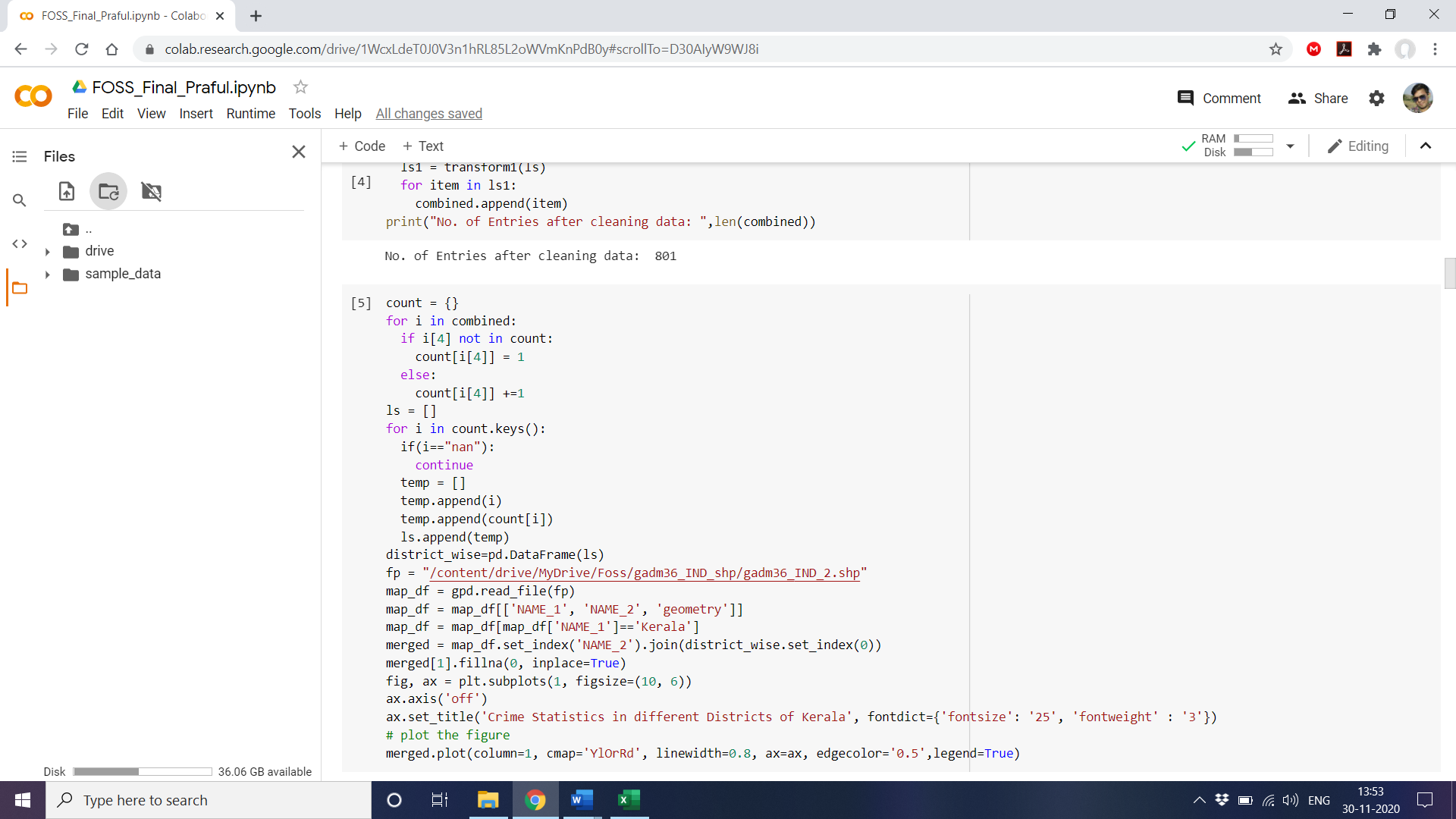
The data provided to us consists of many irregularities such as multiple header rows, multiple duplicate rows with same values and different ids. We can do some clean up such as deleting repetietive value using our sheet itself in excel. However, this is not good enough to process the data. The important processing must be done using python.

 For this we create two functions viz. transform1 and transform2 which does all the cleaning.

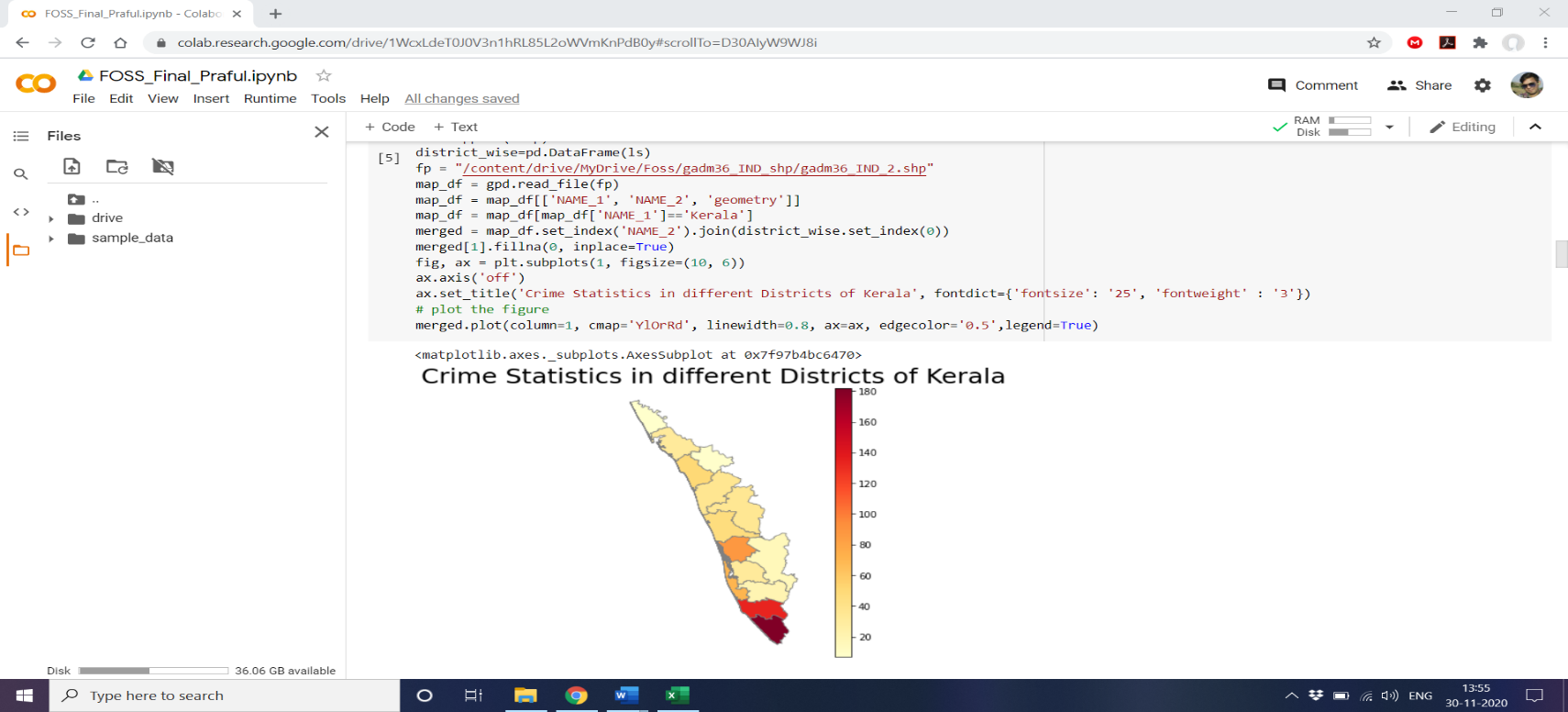
We loop through all the sheets while calling the transforming functions and creating a combined list with all cleaned data. We get a total of 801 values after processing the data.



**Geo Visualization**

Let us try to visualize the crimes reported in the sheet based on the different district in 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.

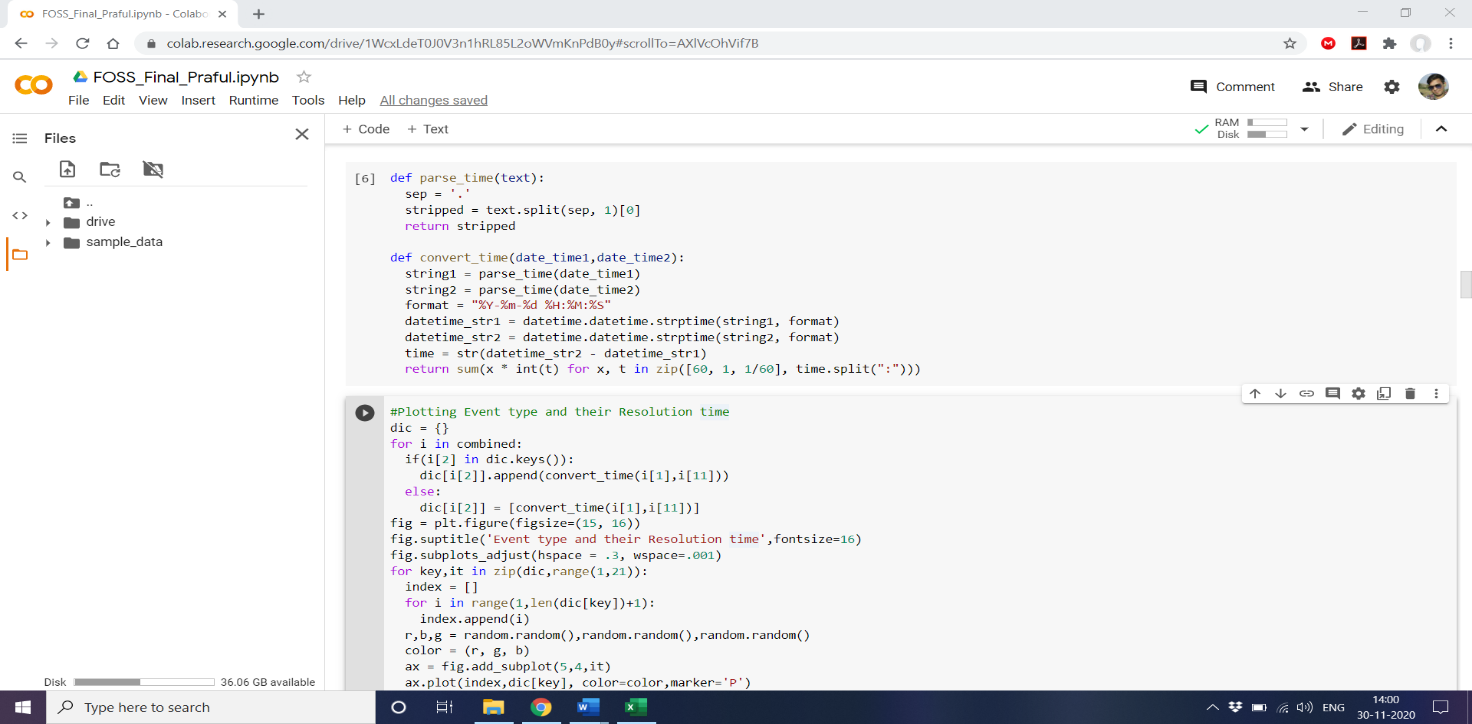
The result is sort of a heat map of crimes reported differentiated based on the values in each district. The brighter the color higher the number of calls received from the district.

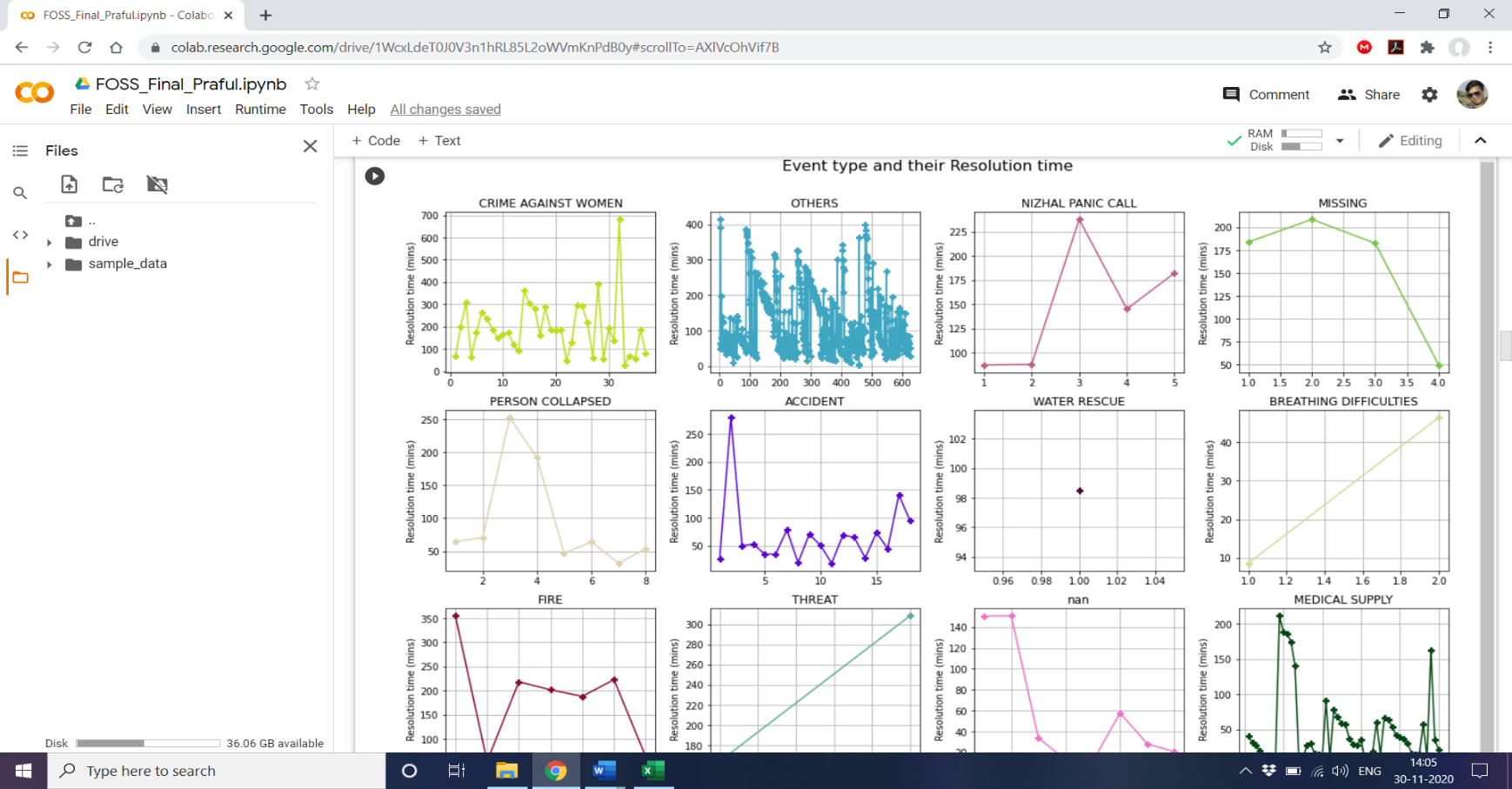


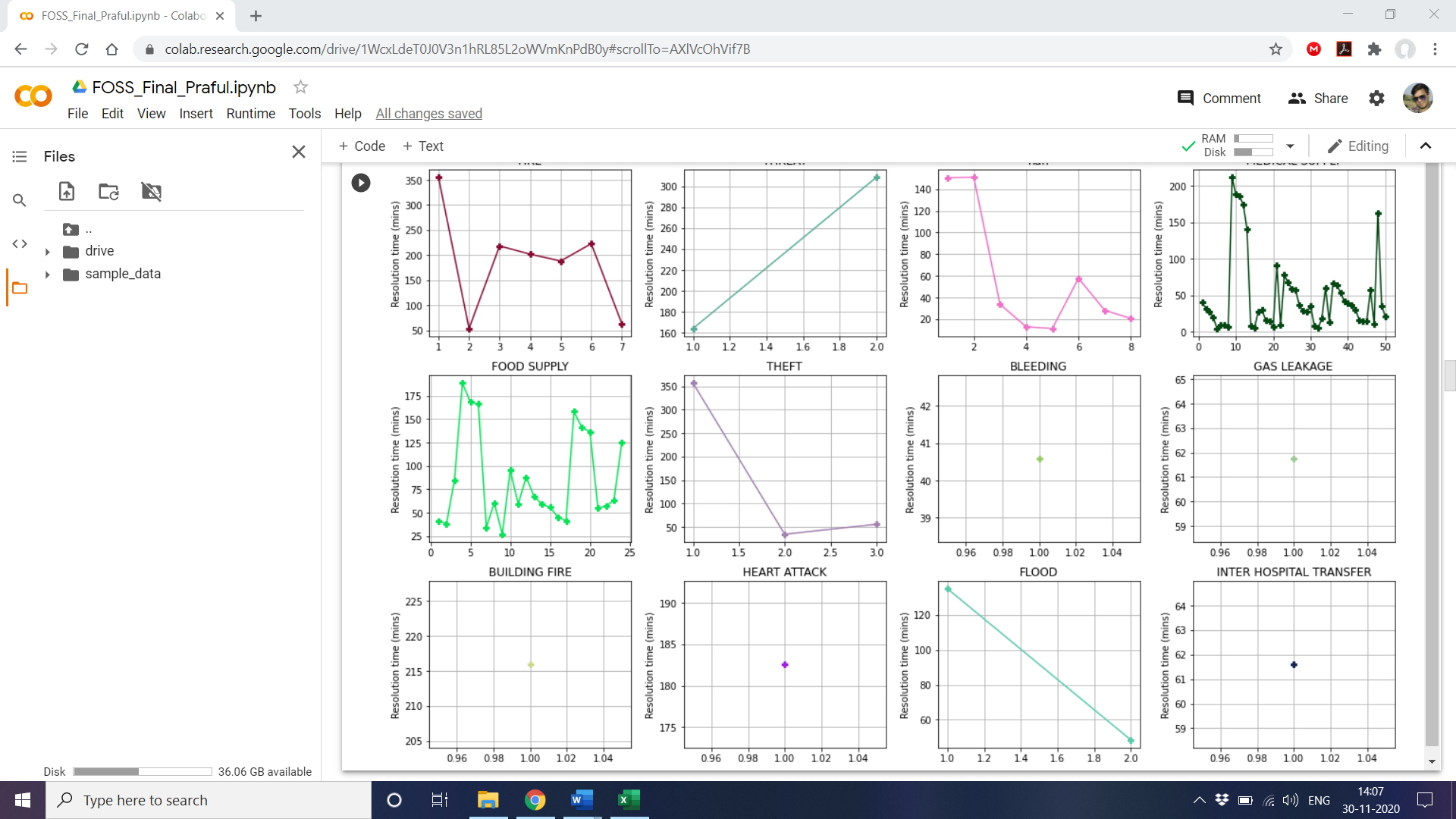
We can clearly see that more calls were received from the southern part of the state than the northern. Thiruvananthapuram, Kollam and Kottayam were amongst the district with highest call ratio. This can help us to predict where police deployment and recruitment should be higher within the state.

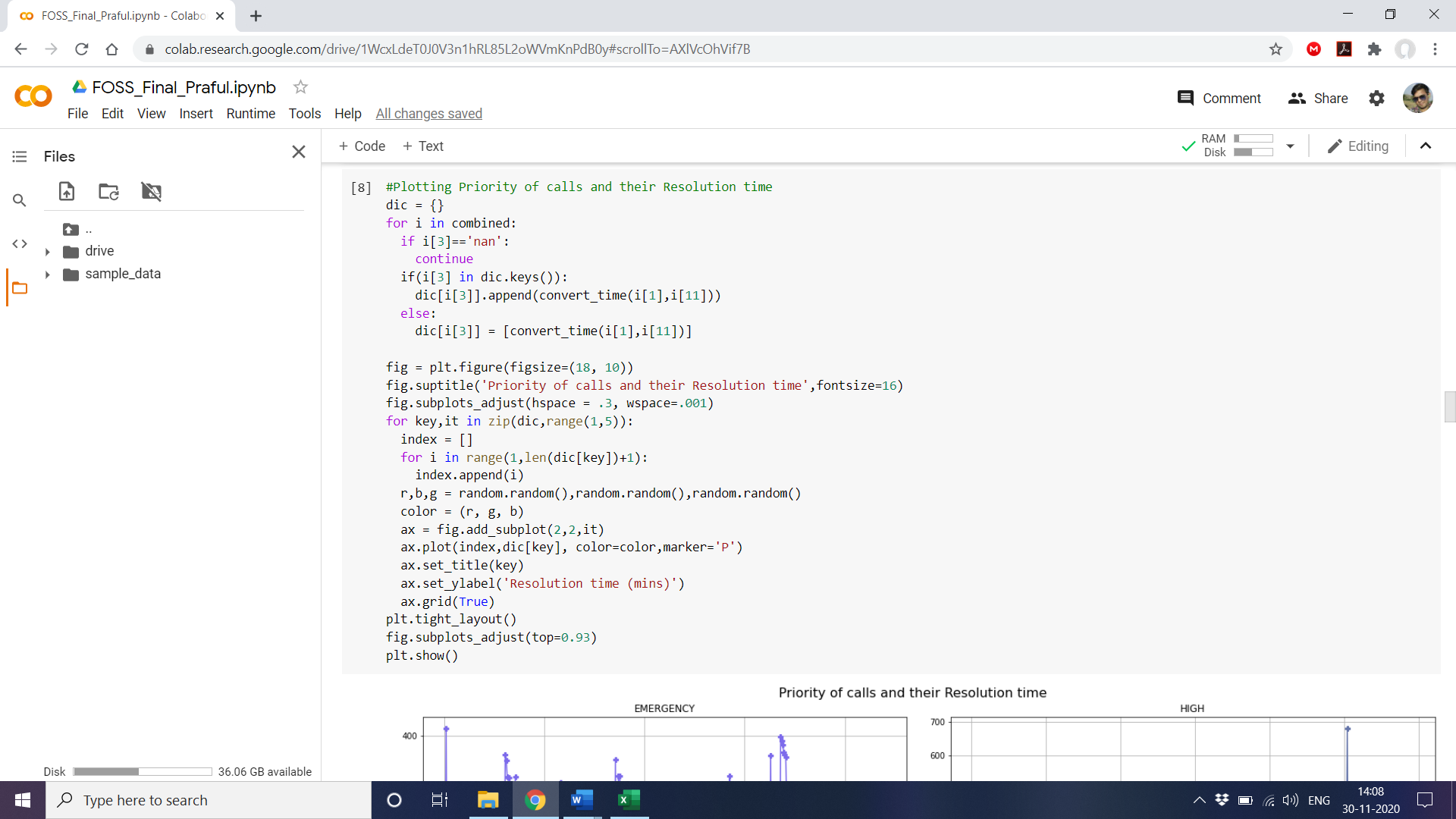
**Graphs**

We have been provided the start time (time when a call is first received) and the closure time (time when the issue has been marked as closed) in the dataset. From this we can extract the Resolution time of the call. For this purpose, we need to clean up the string since it is not in the standard form. We will be using the following two 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 same.

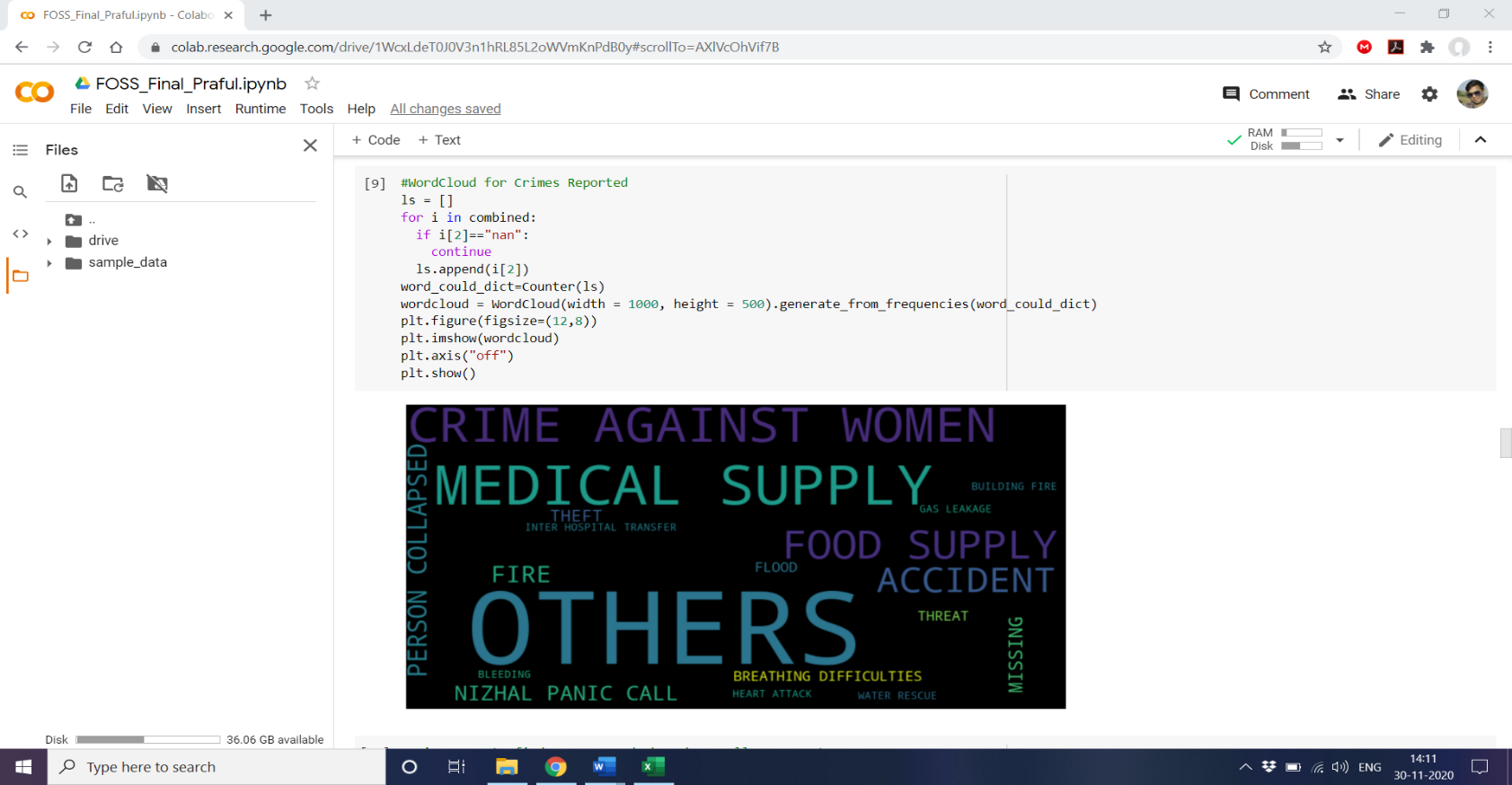


Let us use continue with the same methodology to plot the resolution time based on the Priority of calls.



**Word Cloud**

Based on the frequency of event types, let’s create a word cloud to visualize what are the major types of calls received during the period.

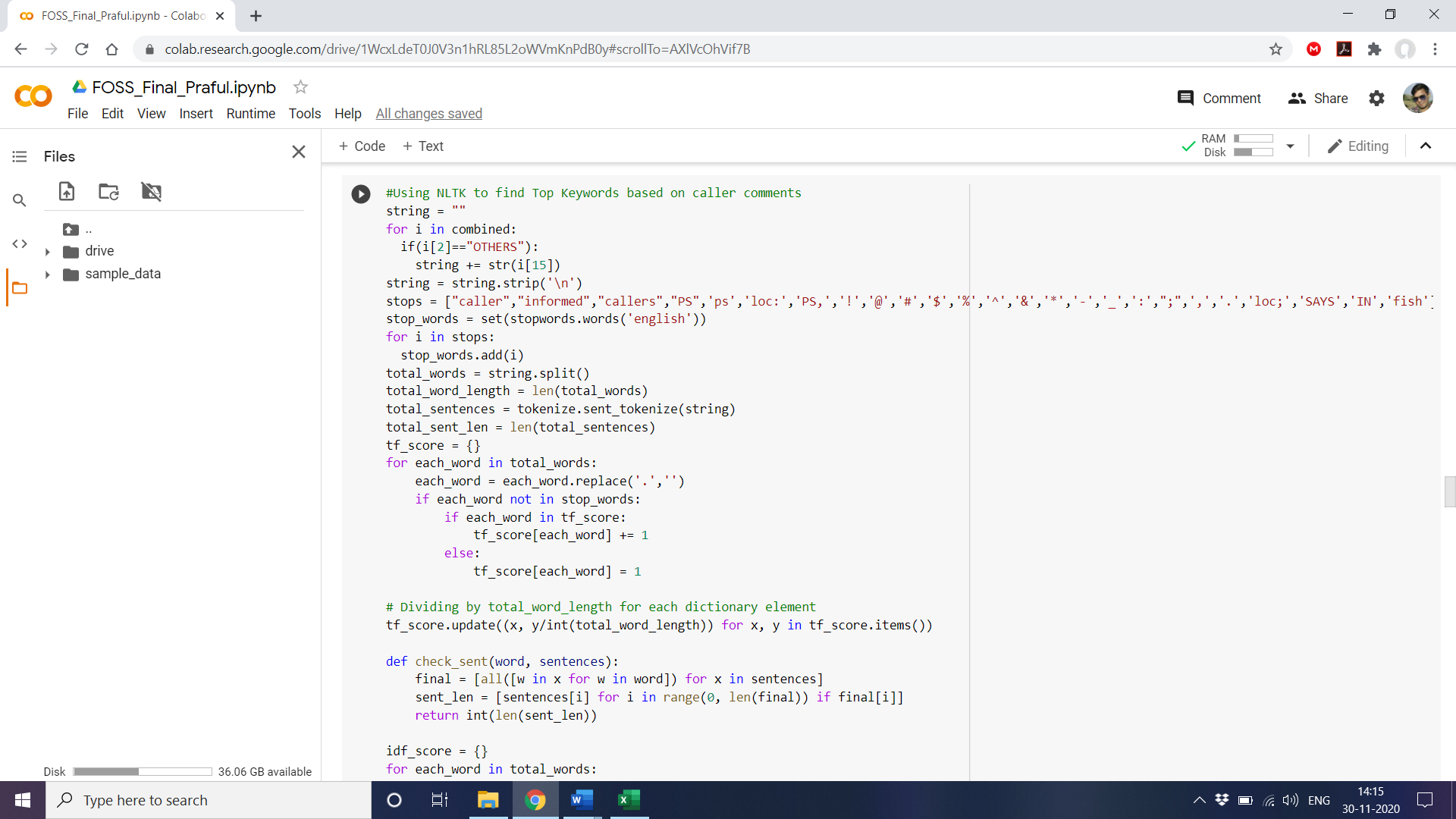


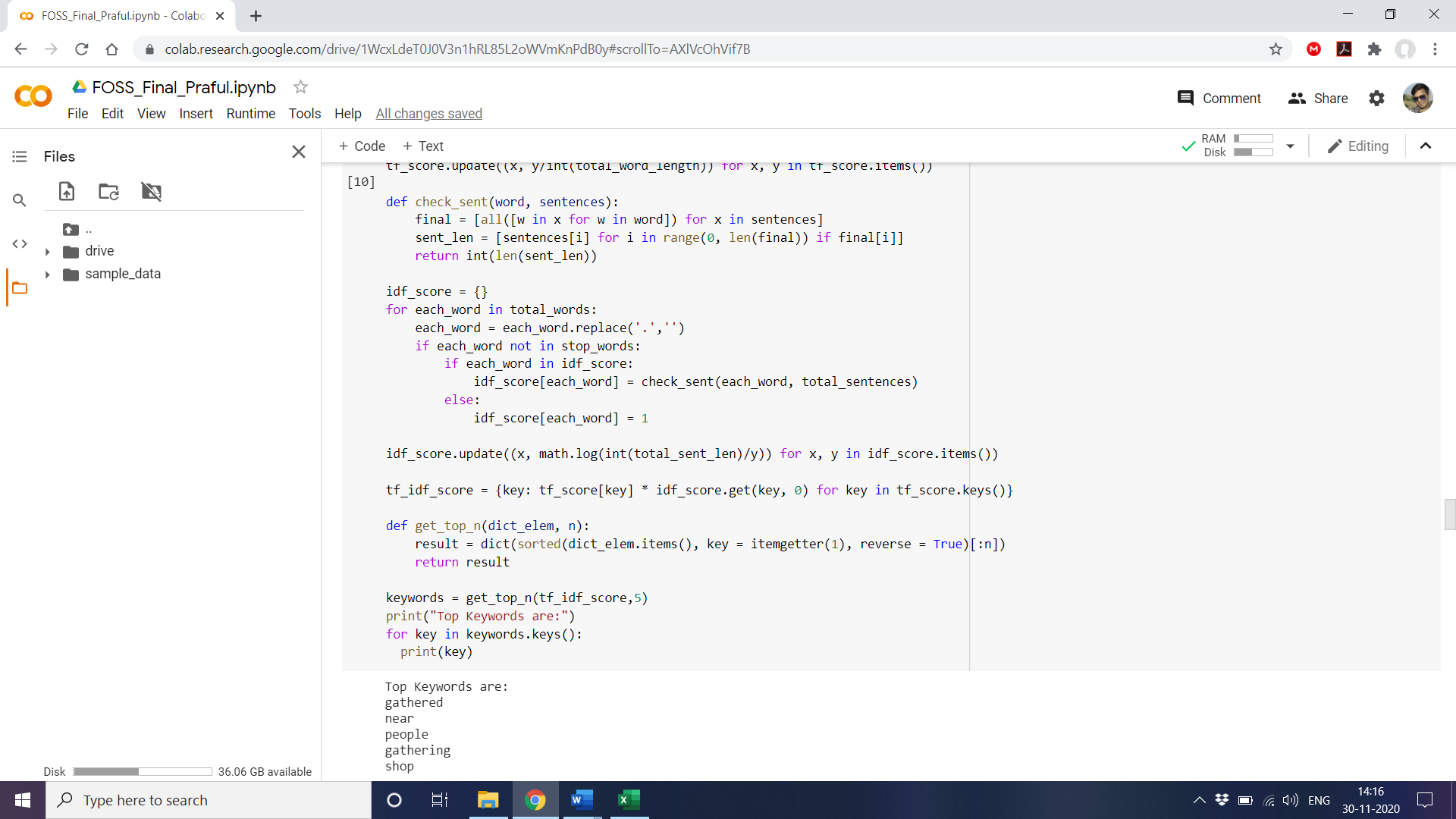
We can see different types of events being plotted based on their frequency. One observable point is that the most forthcoming outcome is of “OTHERS”.

This implies that most of the calls were not being classified properly or no standard classification for these calls was available. Lets us try and rectify this issue.

**NLP**

We use the comments which were logged by the user during the calls. We combine all these comments in a sinlge string and then use nltk to find the Top Keywords which were used by the caller. We use stop words in order to avoid punctutations and others standard words to produce better results.

***NLTK***

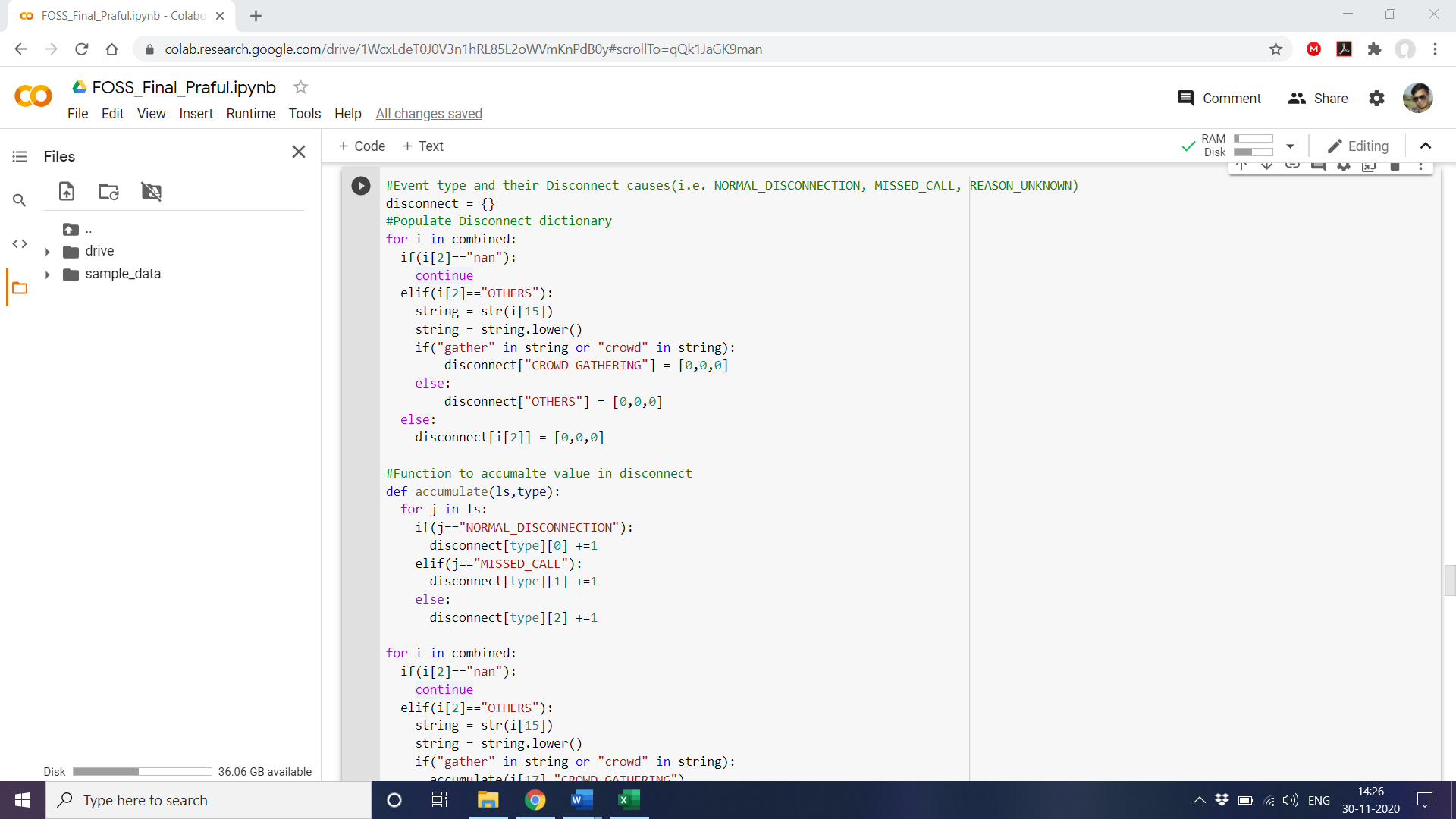


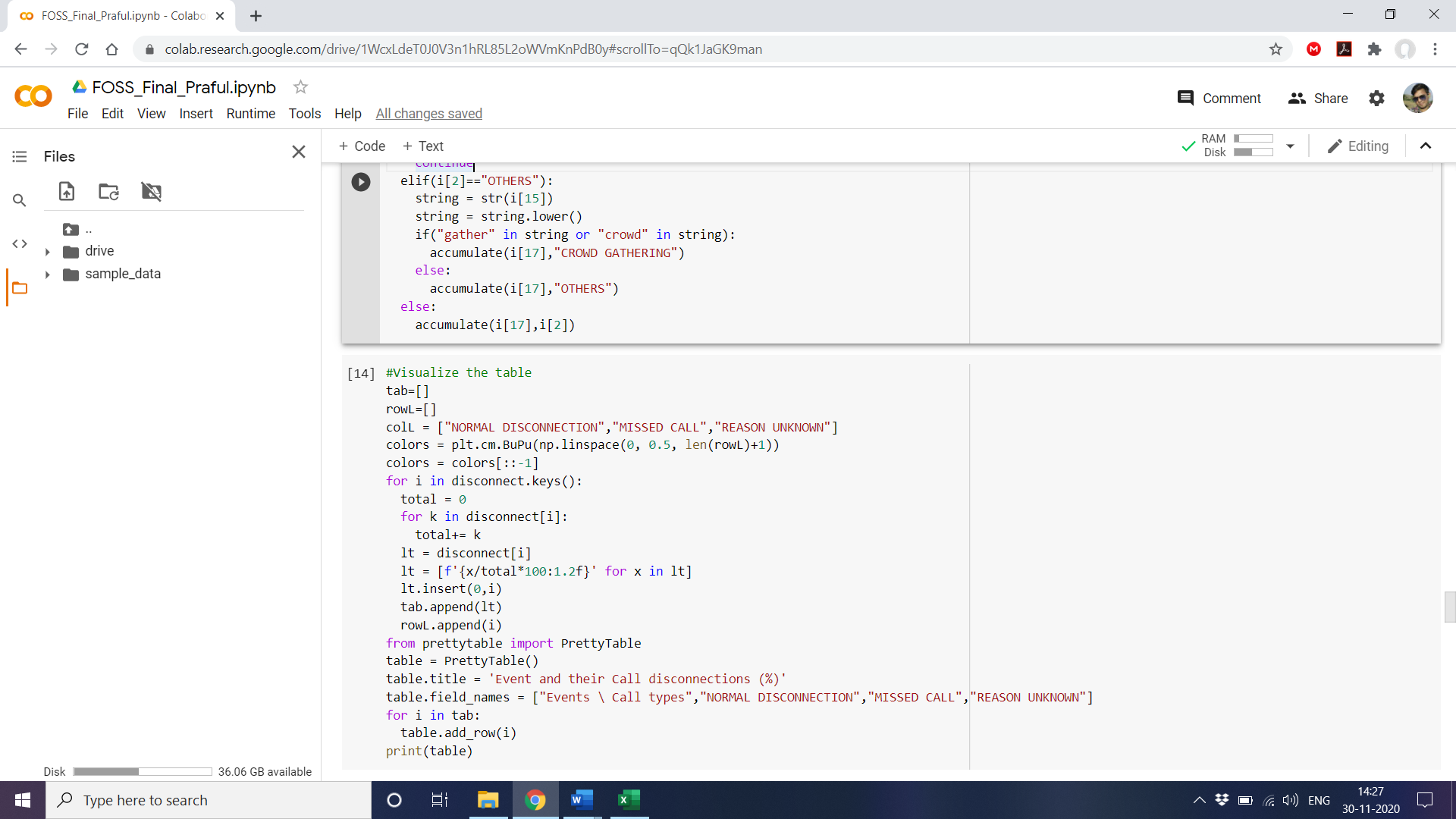
We find that words such as “gathered”, “gathering”, “people” were used most often in calls. This should be true given that the calls were logged in the month of April. A national lockdown was enforced in the nation due to the present Covid-19 situation. So, it is apt that many callers were complaining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

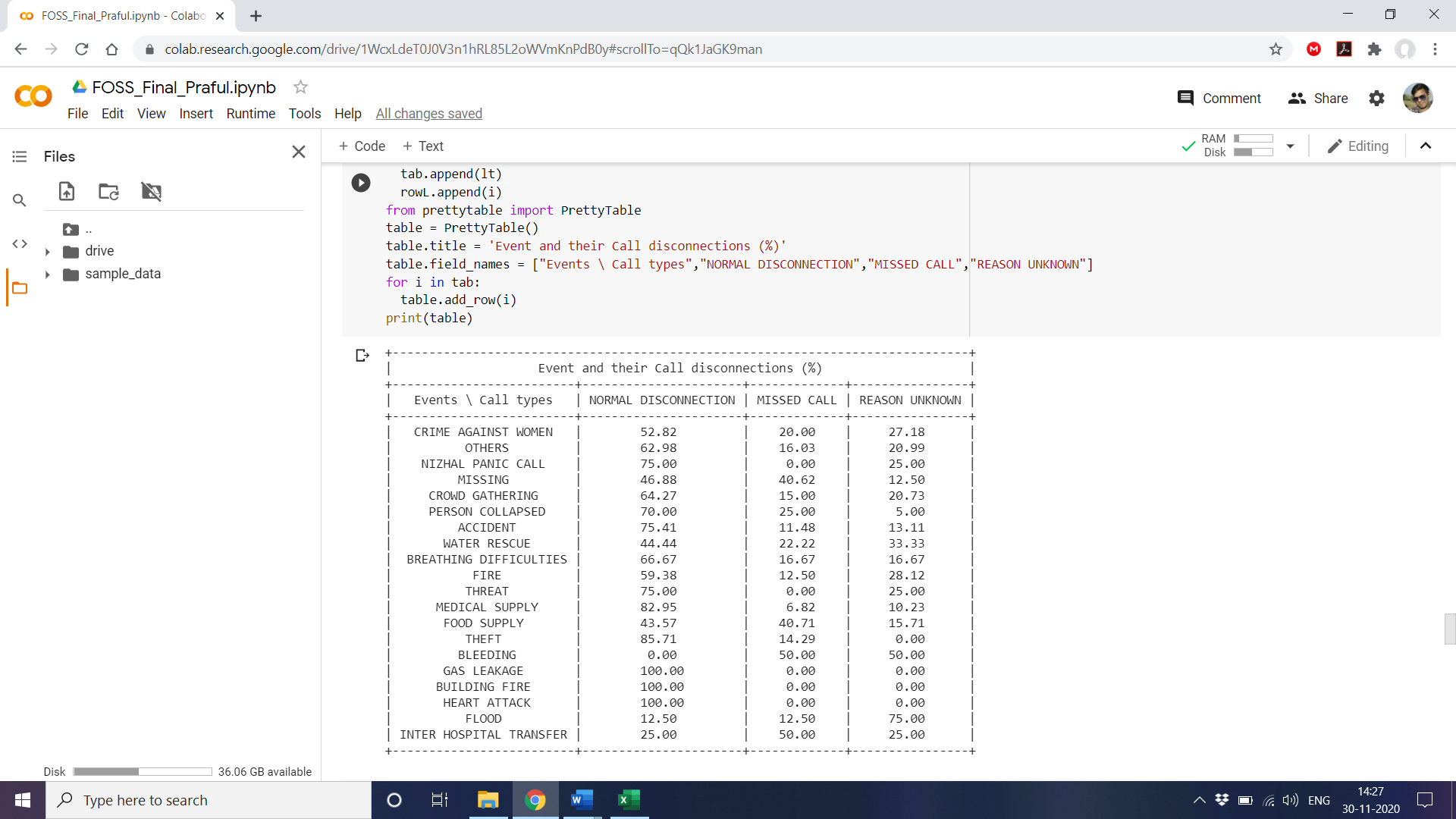
Now that we have identified the event types of many calls listed as “OTHERS” let us redraw the word cloud changing “others” to “Crowd Gathering” wherever applicable. We can see immediate change in the output of the word cloud.



**Table**

 Let us try to represent the percantage of calls and their disconnection types along with their event types. For this we will first build a list.



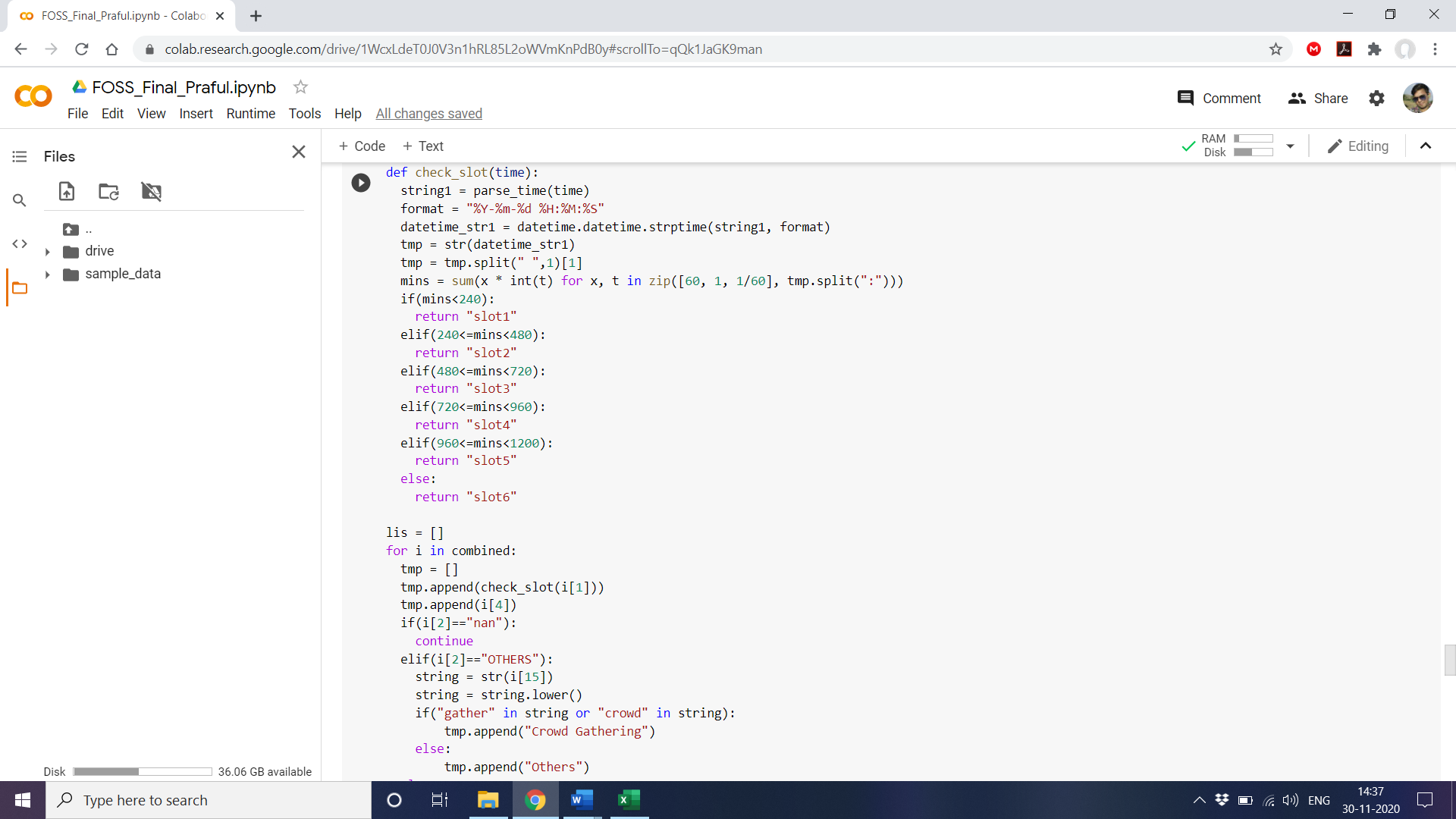
The final output:

**Classification**

In this sheet we do not have any straightforward 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.

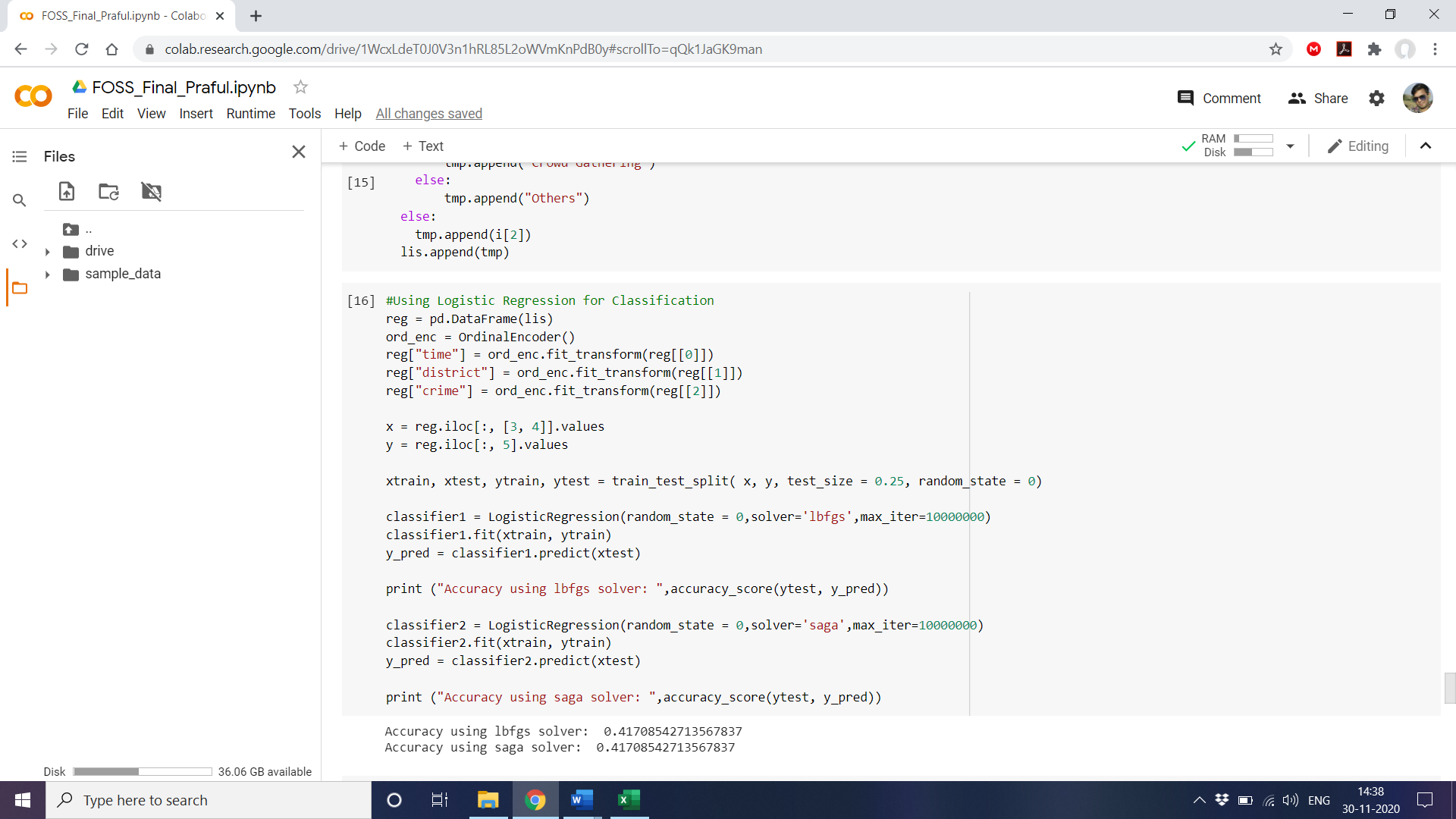
The reason behind selection of these parameters is that if we can find good enough accuracy than we can predict what type of crime is reported generally at what times in a district. Say if we find that most of the calls for Crowd Gathering were purported at around 7 pm in the evening. So, we can deploy 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 in order to perform classification.

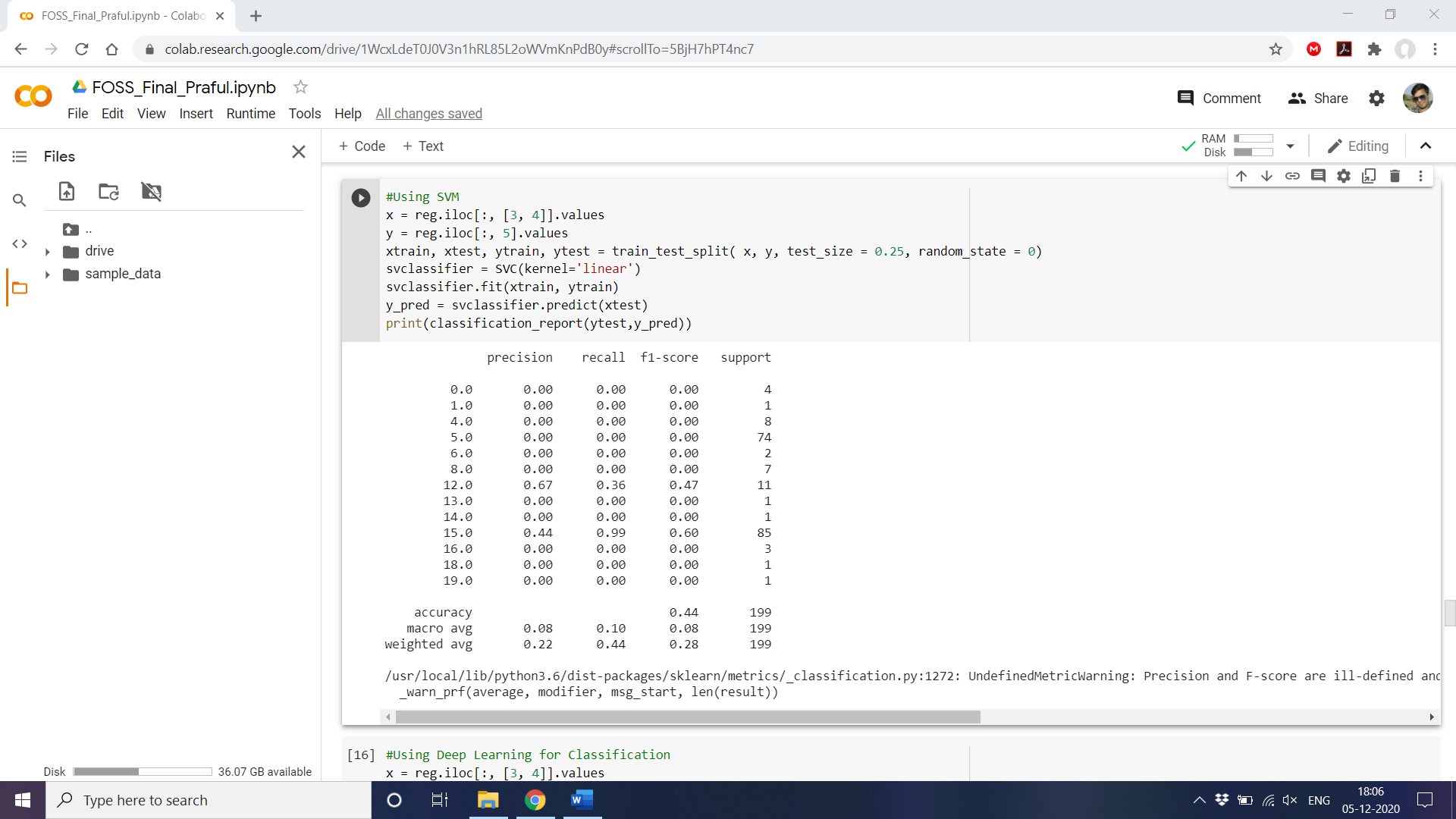


The check slot function will do the same for us and in lis we collect the three features we will work on.

**Logistic Regression**

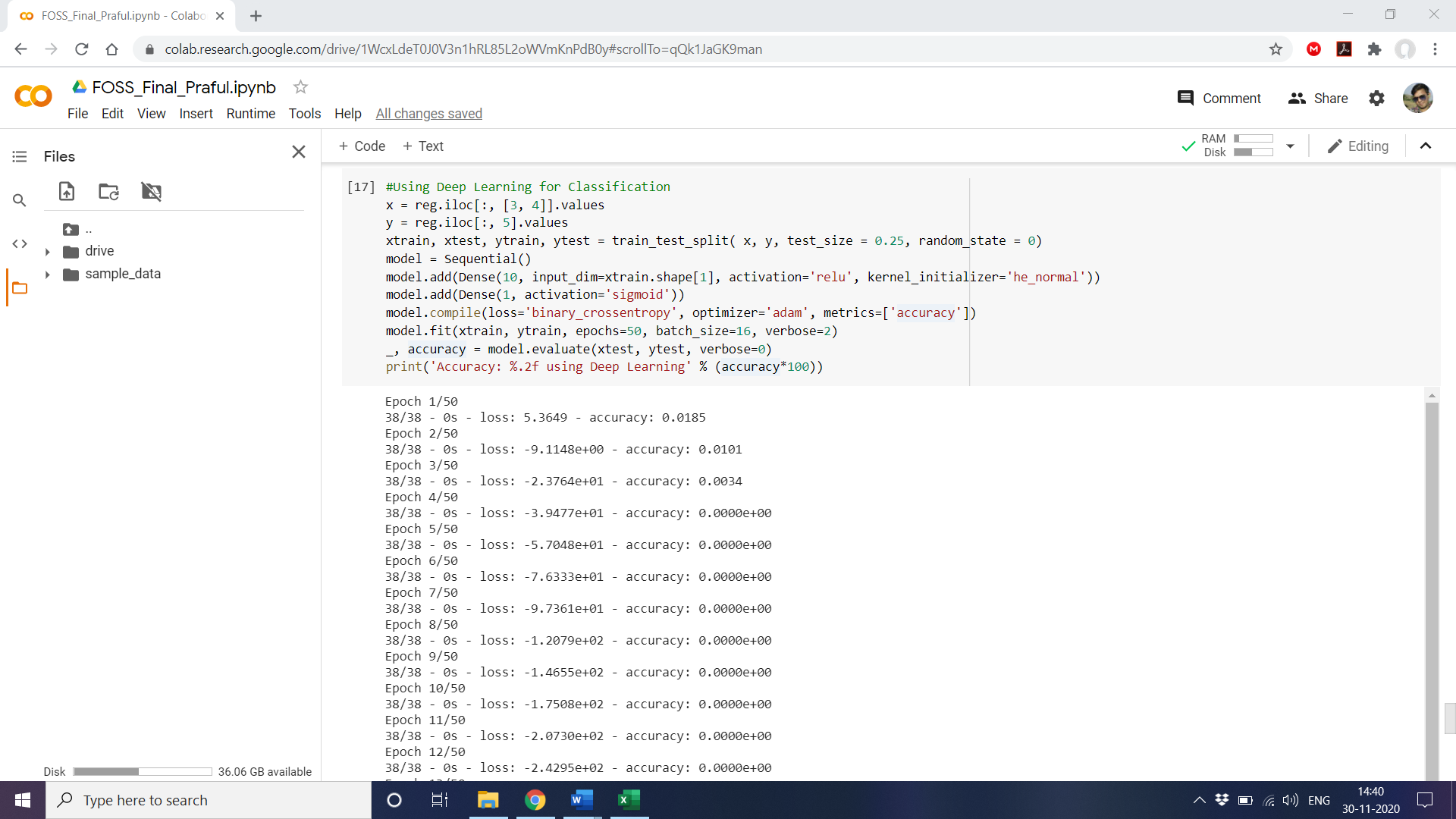
Fit the data in logit model for classification.

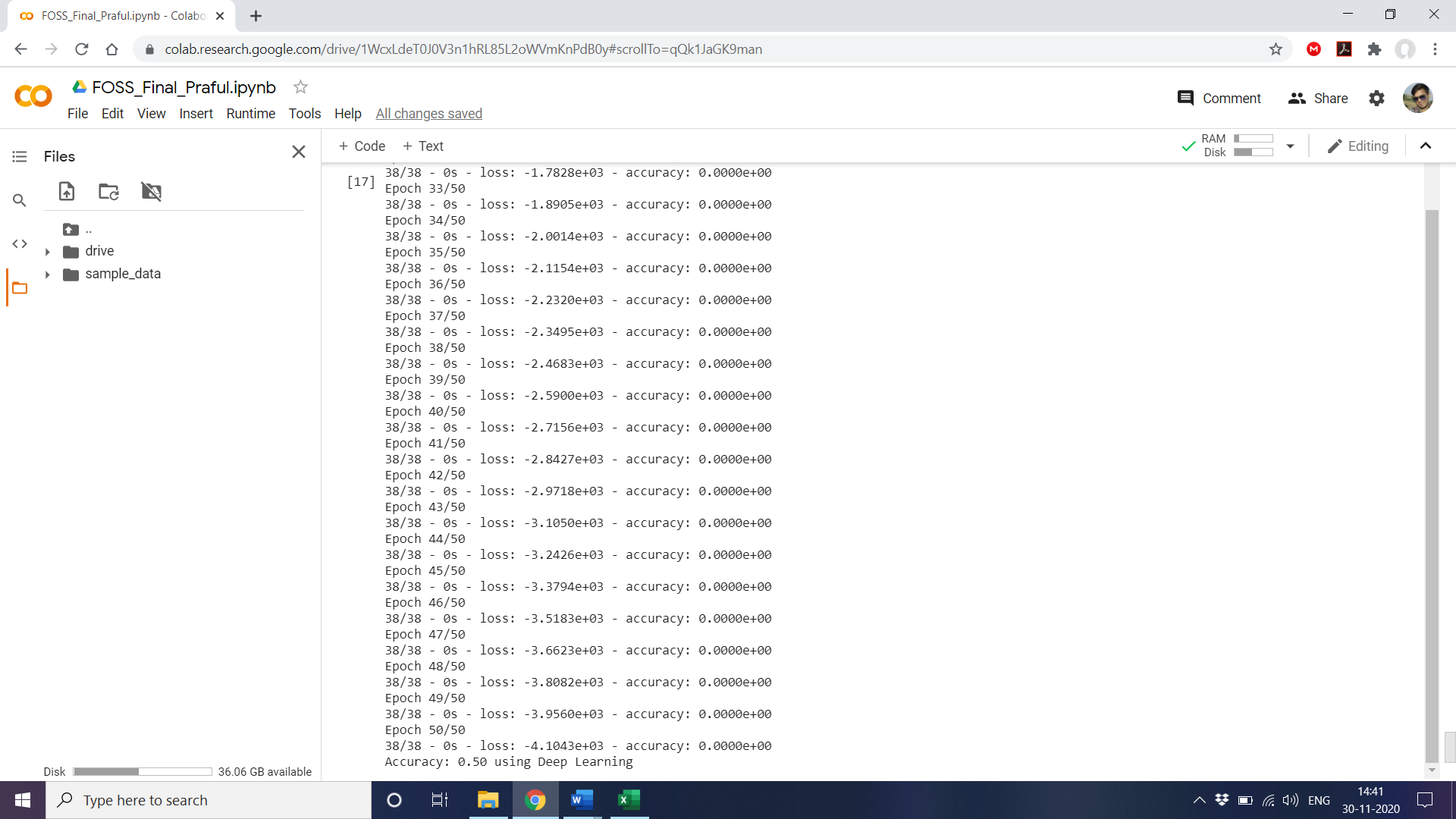
Using both the lbfgs and Saga solver for Logistic regression we get the same accuracy of 41.70%.

**SVM**

Using SVM classifier we get an accuracy of 44% which is higher than Logistic Regression but still low. So let us try to fit the data in a Deep Learning model to see if the accuracy can be improved.

**Deep Learning**

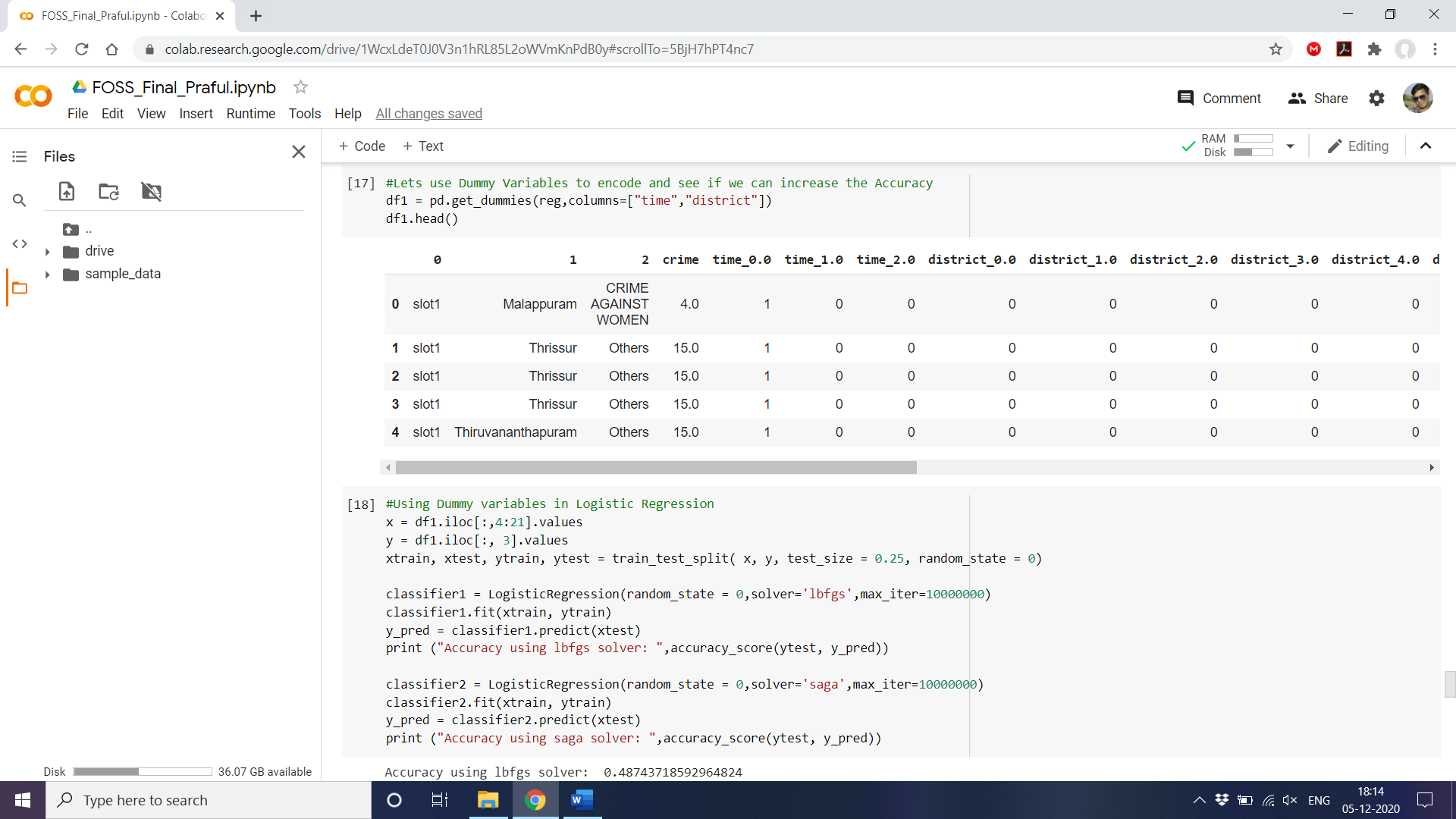
Let us use a simple Neural Network with 1 convolution layer to see if there is any change in the accuracy.

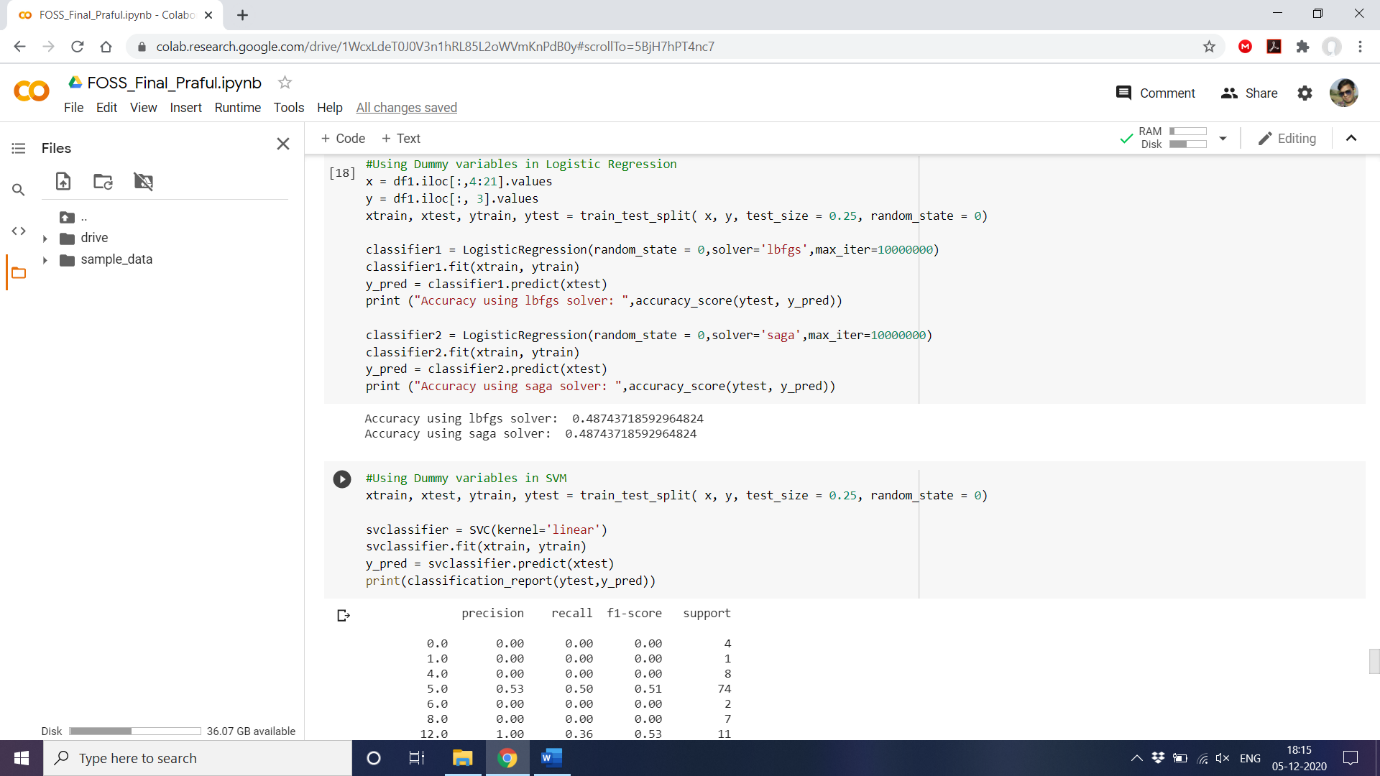


**Dummy Variables**

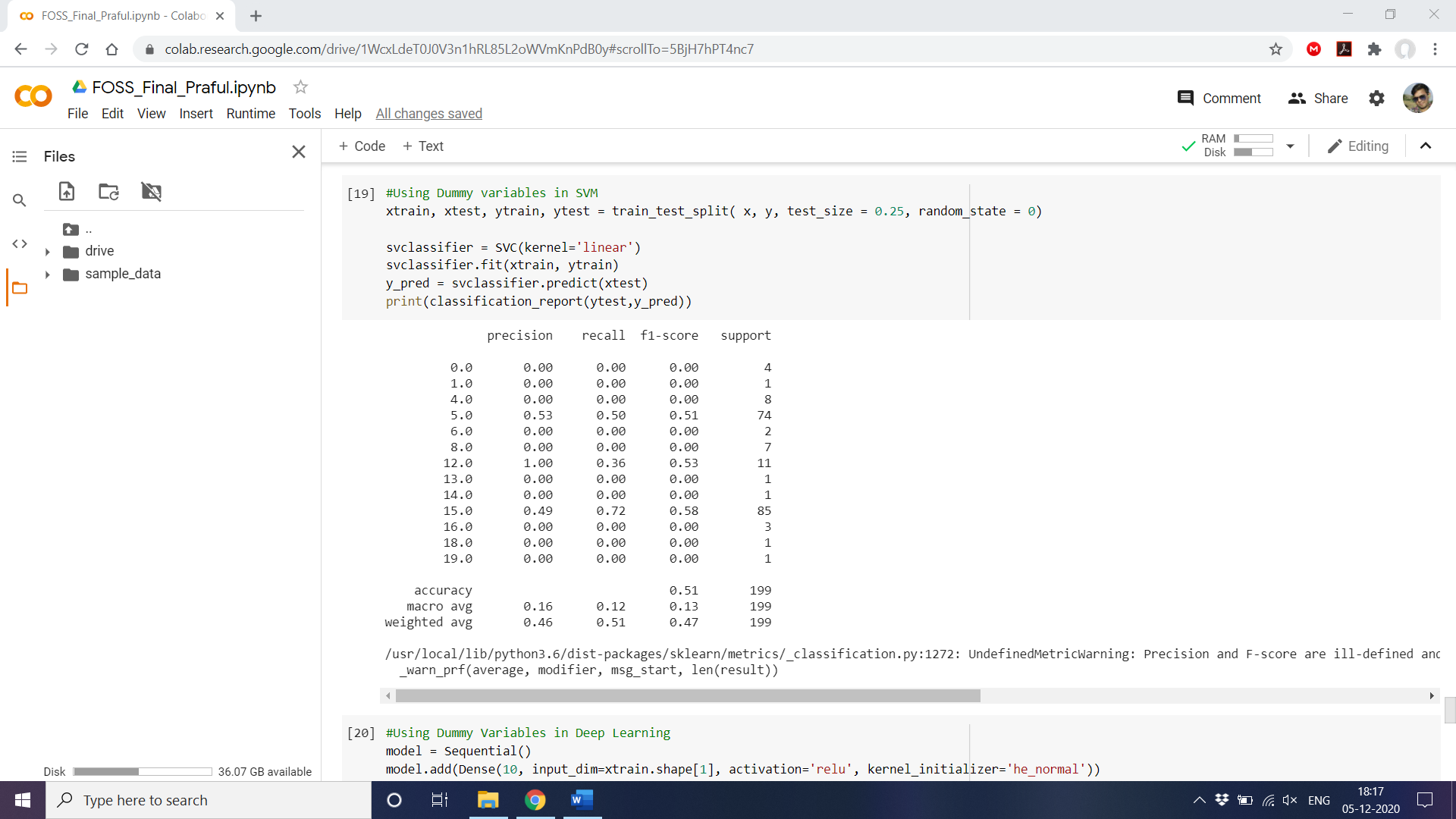
Nominal variables, or variables that describe a characteristic using two or more categories, are commonplace in quantitative research, but are not always useable in their categorical form. A common workaround for using these variables in a regression analysis is dummy coding. A Dummy variable is an artificial variable created to represent an attribute with two or more distinct categories/levels. A dummy variable takes values 1 or 0.

We can use dummy variables in our classification models to encode the features time and distict. Let us see if this leads to any increase in accuracy in Logistic, SVM or Deep Learning.

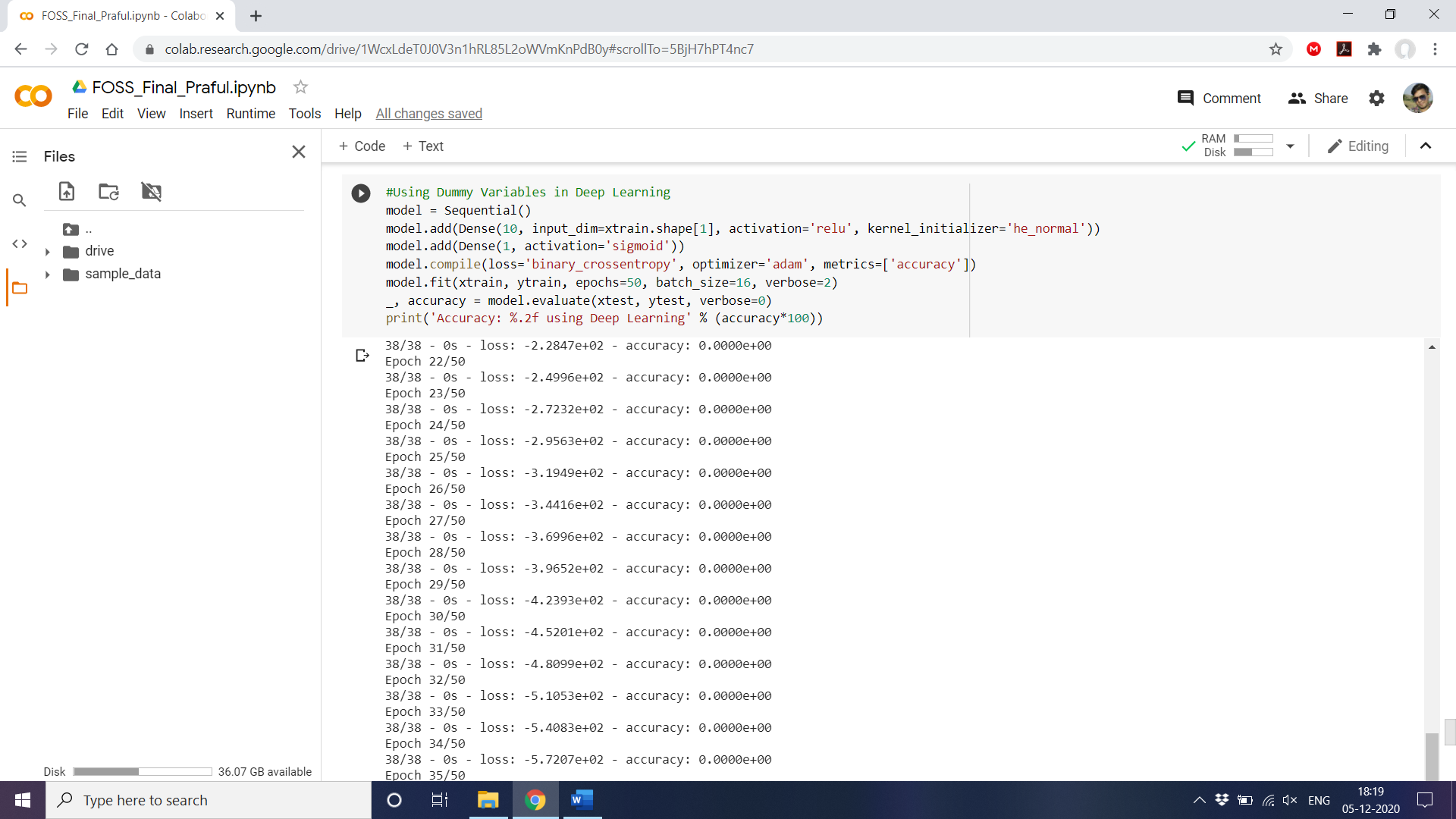


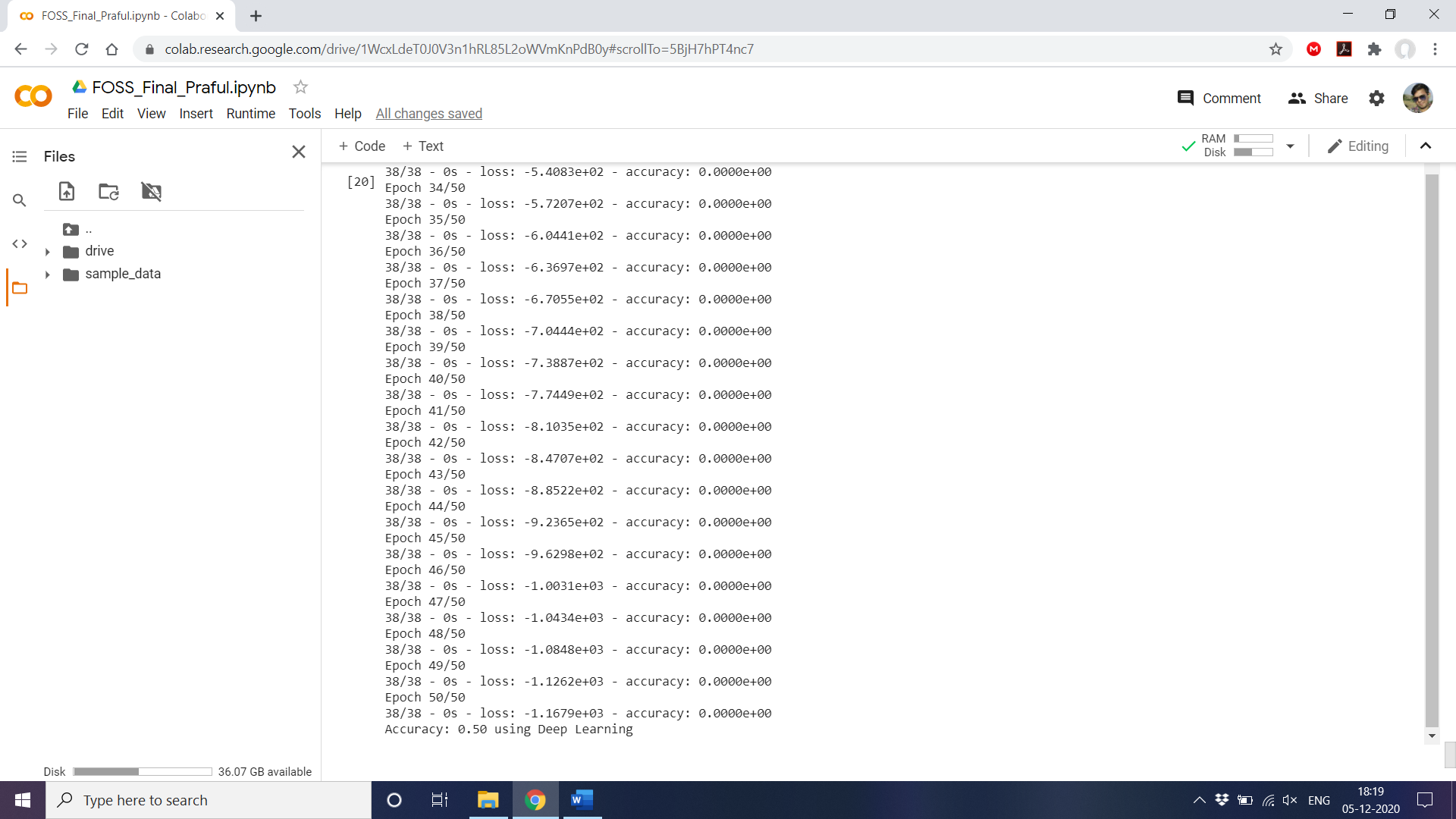
**Dummy with Logistic Regression**

Compared to our original Logistic Regression which had an accuracy of 41%, the model with Dummy variables gives an **improved** accuracy of 48.7%.

**Dummy with SVM**

This version gives accuracy of 51%, **higher** than even deep learning model.

**Dummy and Deep Learning**



There is no change in the accuracy of Deep Learning even after the introduction of Dummy Variables.

**Conclusion**

|  |  |
| --- | --- |
| **Model** | **Accuracy (%)** |
| Logistic Regression | 41.70 |
| SVM | 44.00 |
| Deep Learning | 50.00 |
| Dummy with Logistic Regression | 48.74 |
| Dummy with SVM | 51.00 |
| Dummy with Deep Learning | 50.00 |

SVM encoded with Dummy variables turns out to be our best model with an accuracy of 51%. This is an improvement over boht Logistic Regression and Dummy Variables but is still some way off to be considered a good model.

The main reason behind this is lack of good feautes in the data, lack of data values, under fitting. The model does show promise but still needs a lot of training data and features to be considered viable.

**Code**

#Libraries and Requirements

import pandas as pd

import numpy as np

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from collections import Counter

import datetime

import random

from nltk import tokenize

from operator import itemgetter

import math

import nltk

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

nltk.download('punkt')

nltk.download('stopwords')

#Transformation Functions for Pre-Processing

def transform1(ls):

    ls1 = []

    for i in ls:

        if i[0]!="event\_id":

            if i[0]=='nan' and i[1]=='nan':# Skiping blank lines from the sheet

                continue

            else:

                ls1.append(i)

    ls2 = transform2(ls1) # send list to combine signal and disconnect rows

    return ls2

def transform2(ls1):

    i=0

    ls2 = []

    while i < len(ls1):

        if ls1[i][0]!='nan':

            nw = ls1[i]

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

            signal = []

            disconnect = []

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

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

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

                i+=1

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

                    break

            nw.append(signal)

            nw.append(disconnect)

            ls2.append(nw) #Appending df to combined list

        else:

            i+=1

    return ls2 #one list with all data and corresponding signal time and disconnect cause as list of list

combined = []

for i in range(1,4):

  df1 = pd.read\_excel("/content/drive/MyDrive/Foss/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("No. of Entries after cleaning 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/Foss/gadm36\_IND\_shp/gadm36\_IND\_2.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=(10, 6))

ax.axis('off')

ax.set\_title('Crime Statistics in different Districts of Kerala', fontdict={'fontsize': '25', 'fontweight' : '3'})

# plot the figure

merged.plot(column=1, cmap='YlOrRd', linewidth=0.8, ax=ax, edgecolor='0.5',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(":")))

#Plotting Event type and their Resolution time

dic = {}

for i in combined:

  if(i[2] in dic.keys()):

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

  else:

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

fig = plt.figure(figsize=(15, 16))

fig.suptitle('Event type and their 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)

  r,b,g = random.random(),random.random(),random.random()

  color = (r, g, b)

  ax = fig.add\_subplot(5,4,it)

  ax.plot(index,dic[key], color=color,marker='P')

  ax.set\_title(key)

  ax.set\_ylabel('Resolution time (mins)')

  ax.grid(True)

plt.tight\_layout()

fig.subplots\_adjust(top=0.93)

plt.show()

#Plotting 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 their Resolution time',fontsize=16)

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)

  r,b,g = random.random(),random.random(),random.random()

  color = (r, g, b)

  ax = fig.add\_subplot(2,2,it)

  ax.plot(index,dic[key], color=color,marker='P')

  ax.set\_title(key)

  ax.set\_ylabel('Resolution time (mins)')

  ax.grid(True)

plt.tight\_layout()

fig.subplots\_adjust(top=0.93)

plt.show()

#WordCloud for Crimes Reported

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).generate\_from\_frequencies(word\_could\_dict)

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

plt.imshow(wordcloud)

plt.axis("off")

plt.show()

#Using NLTK to find Top Keywords based on caller comments

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

# Dividing by total\_word\_length for each dictionary element

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("Top Keywords are:")

for key in keywords.keys():

  print(key)

#New Word Cloud based on nltk results

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).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(i.e. NORMAL\_DISCONNECTION, MISSED\_CALL, REASON\_UNKNOWN)

disconnect = {}

#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]

#Function to accumalte 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])

#Visualize the 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 (%)'

table.field\_names = ["Events \ 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)

#Using Logistic Regression for Classification

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 = 0.25, random\_state = 0)

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

classifier1.fit(xtrain, ytrain)

y\_pred = classifier1.predict(xtest)

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

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

classifier2.fit(xtrain, ytrain)

y\_pred = classifier2.predict(xtest)

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

#Using SVM

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.25, random\_state = 0)

svclassifier = SVC(kernel='linear')

svclassifier.fit(xtrain, ytrain)

y\_pred = svclassifier.predict(xtest)

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

#Using Deep Learning for Classification

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.25, random\_state = 0)

model = Sequential()

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

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

model.compile(loss='binary\_crossentropy', optimizer='adam', 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))

#Lets use Dummy Variables to encode and see if we can increase the Accuracy

df1 = pd.get\_dummies(reg,columns=["time","district"])

df1.head()

#Using Dummy variables in Logistic Regression

x = df1.iloc[:,4:21].values

y = df1.iloc[:, 3].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.25, random\_state = 0)

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

classifier1.fit(xtrain, ytrain)

y\_pred = classifier1.predict(xtest)

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

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

classifier2.fit(xtrain, ytrain)

y\_pred = classifier2.predict(xtest)

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

#Using Dummy variables in SVM

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.25, random\_state = 0)

svclassifier = SVC(kernel='linear')

svclassifier.fit(xtrain, ytrain)

y\_pred = svclassifier.predict(xtest)

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

#Using Dummy Variables in Deep Learning

model = Sequential()

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

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

model.compile(loss='binary\_crossentropy', optimizer='adam', 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))