**Kerala State Crime Prediction**

**FOSS Lab**

**A Project Report**

**Submitted in Partial fulfillment of**

**Master’s Degree in Computer Science**

**Session: 2020-2021**

**Submitted By: Under the Guidance of:**

**Abhijeet Kumar Dr. B. Janet**

**M.Sc. (Computer Science) Department of Computer Applications**

**Roll No: 205319001 National Institute of Technology,Trichy**

**Department of Computer Applications**

**National Institute of Technology**

**Truchirapalli-60001**

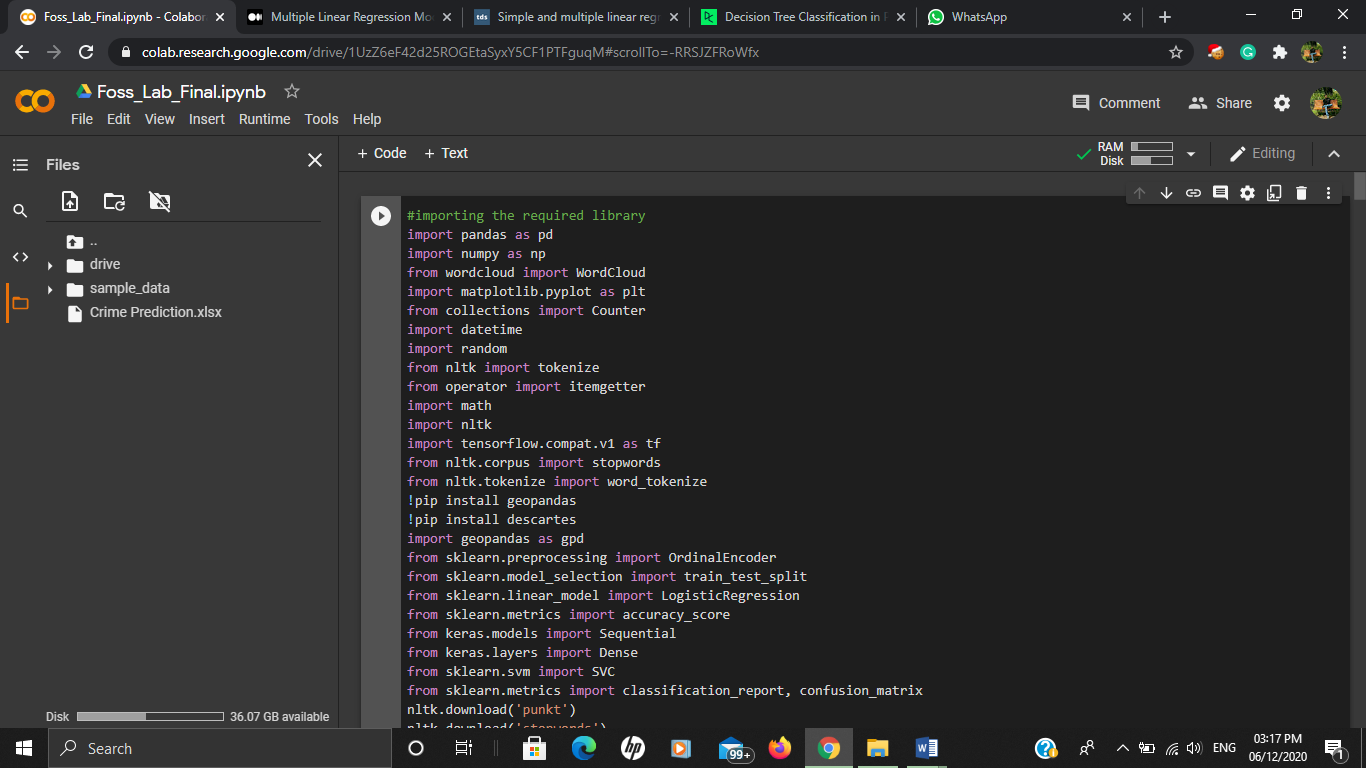
**Github link:** <https://github.com/abhijeetjee01/FOSS_Lab_Final>

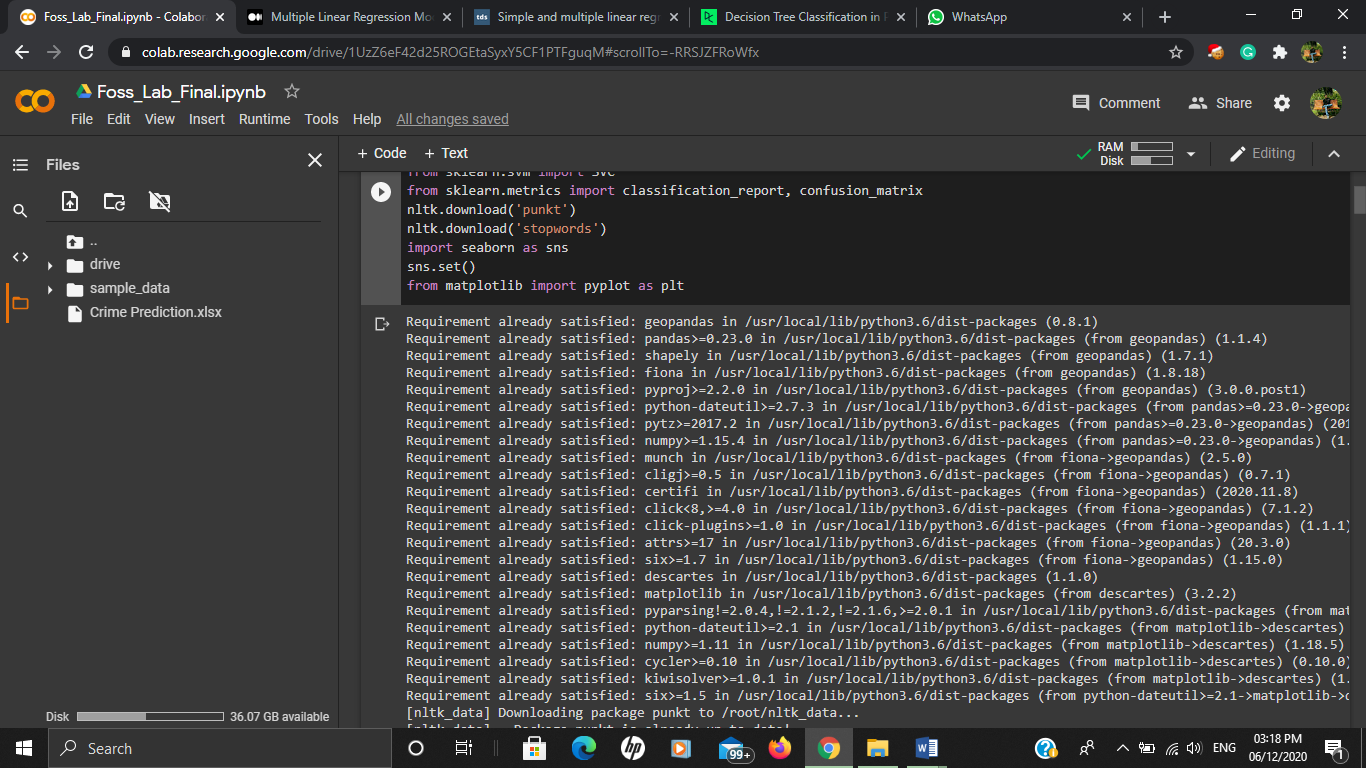
**Colab Link:** <https://colab.research.google.com/drive/1UzZ6eF42d25ROGEtaSyxY5CF1PTFguqM#scrollTo=UWZSpGS1AeNY>

**Data Preprocessing and Visualization**

STEP 1

**Importing Libraries.**





Step2:

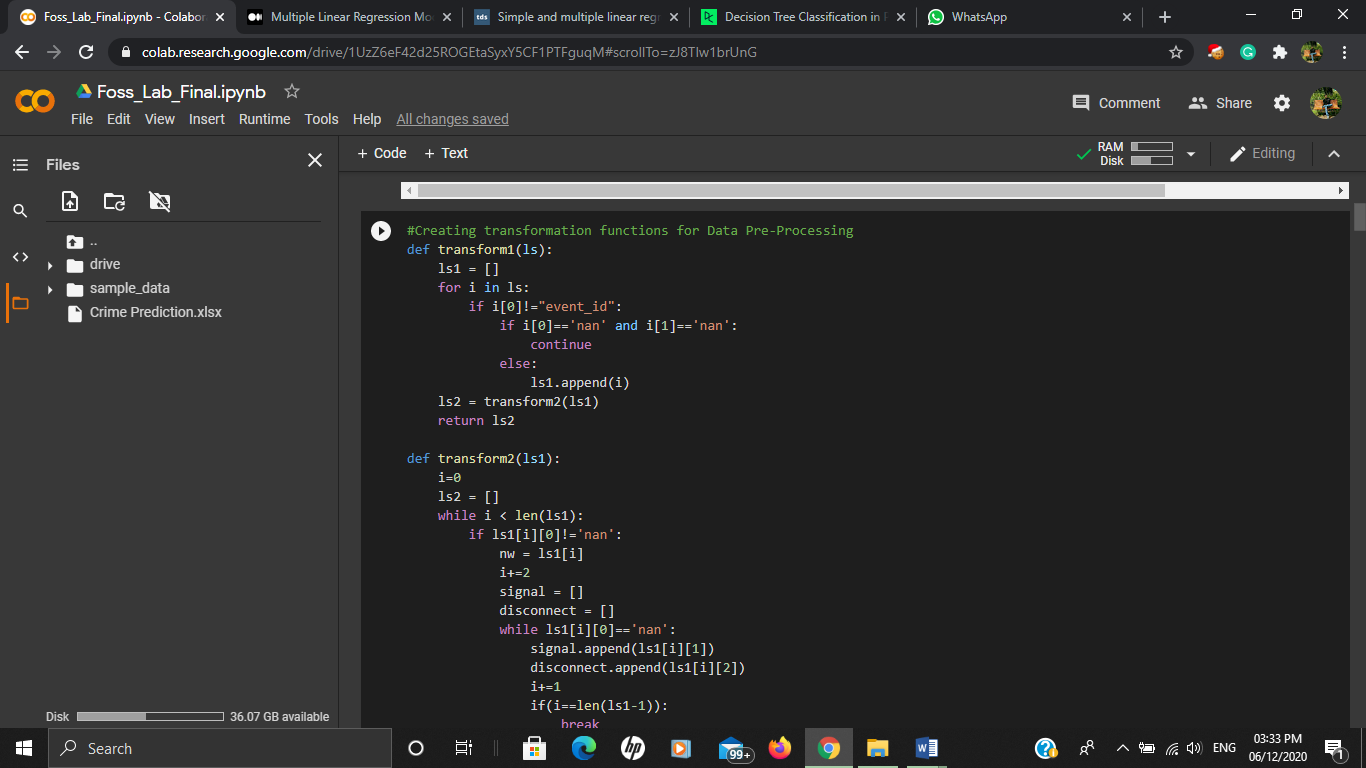
**Cleaning and processing data**

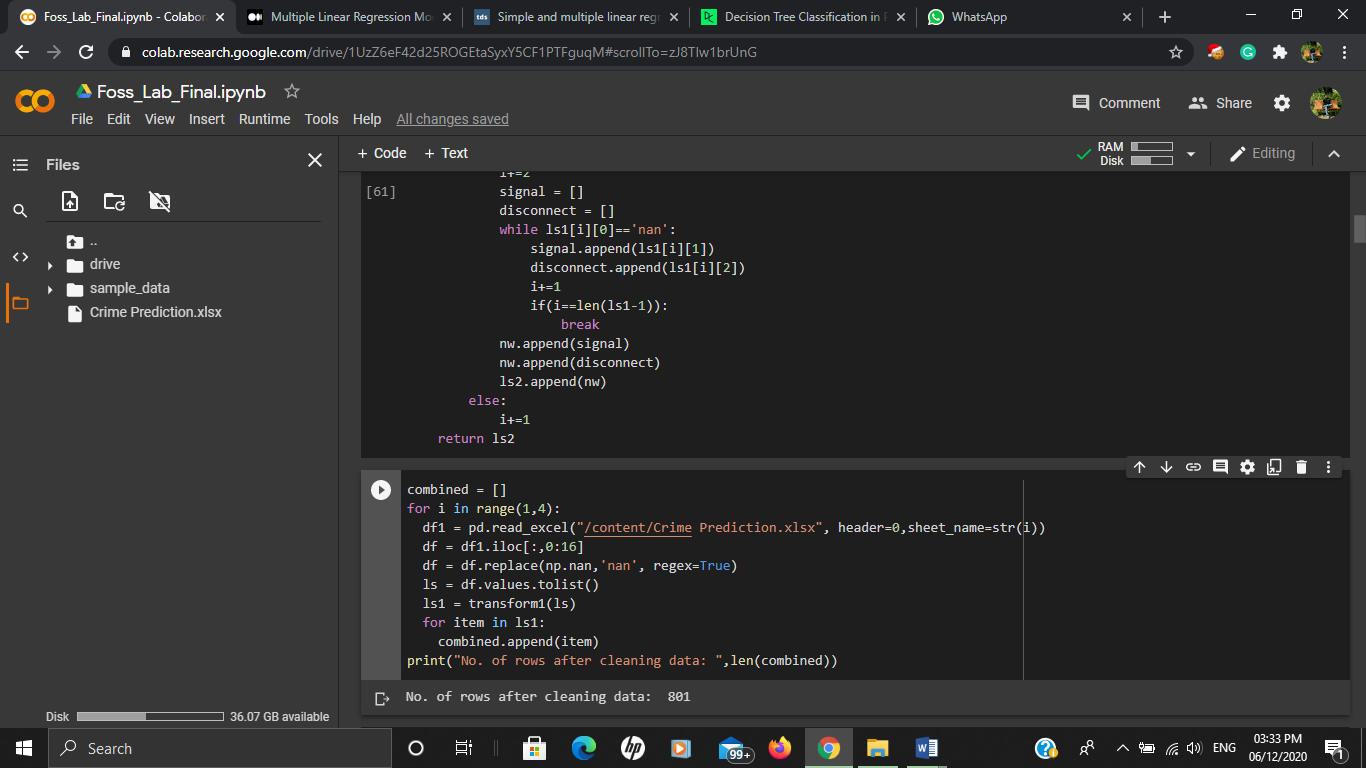
Data pre-processing is a data mining technique which is used to transform the raw data in a useful and efficient format.

This data had many irrelevant, repetitive and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data and removal of repetition data etc.

Here after analysing the data, I manually find the repetitive irrelevant rows and removed that from sheet1.

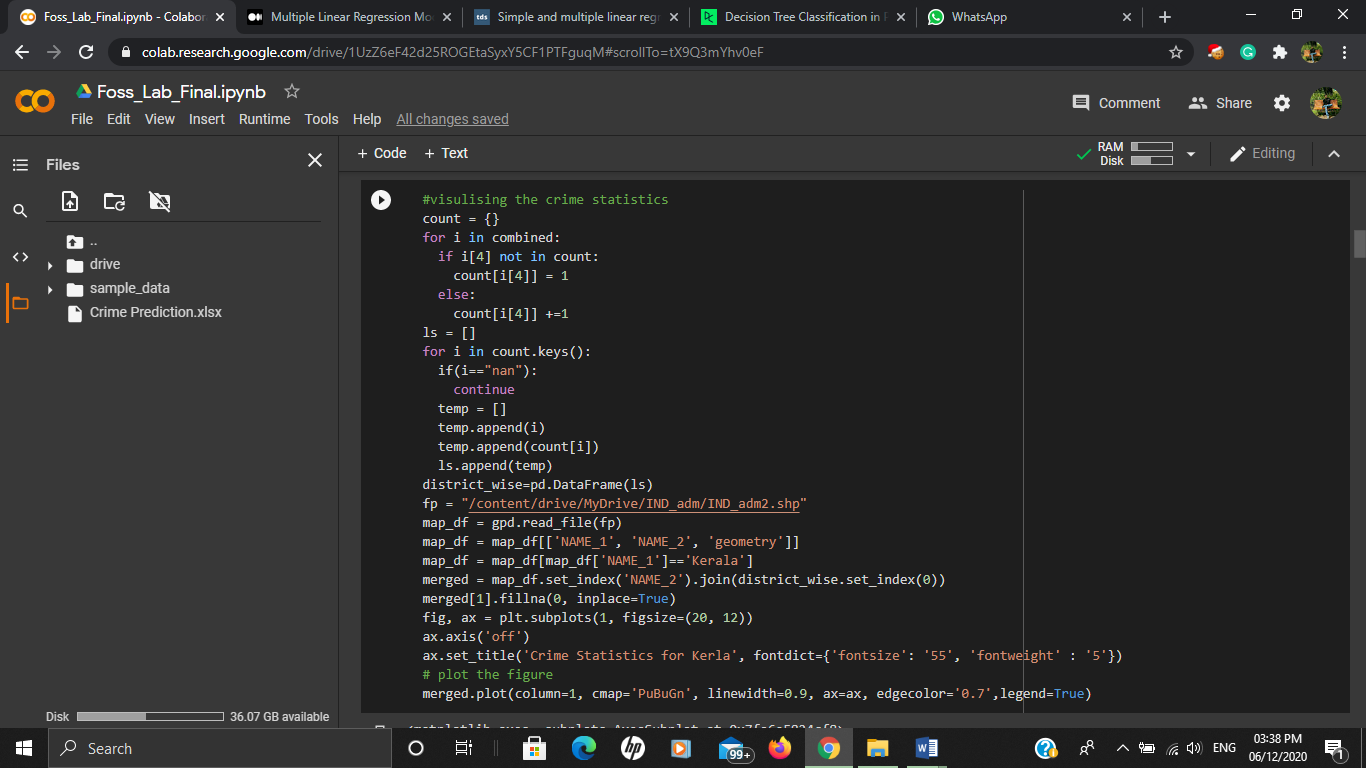
After removing data manually I created two transform function that iterates through each sheet and removes irrevelent data and noise from it.

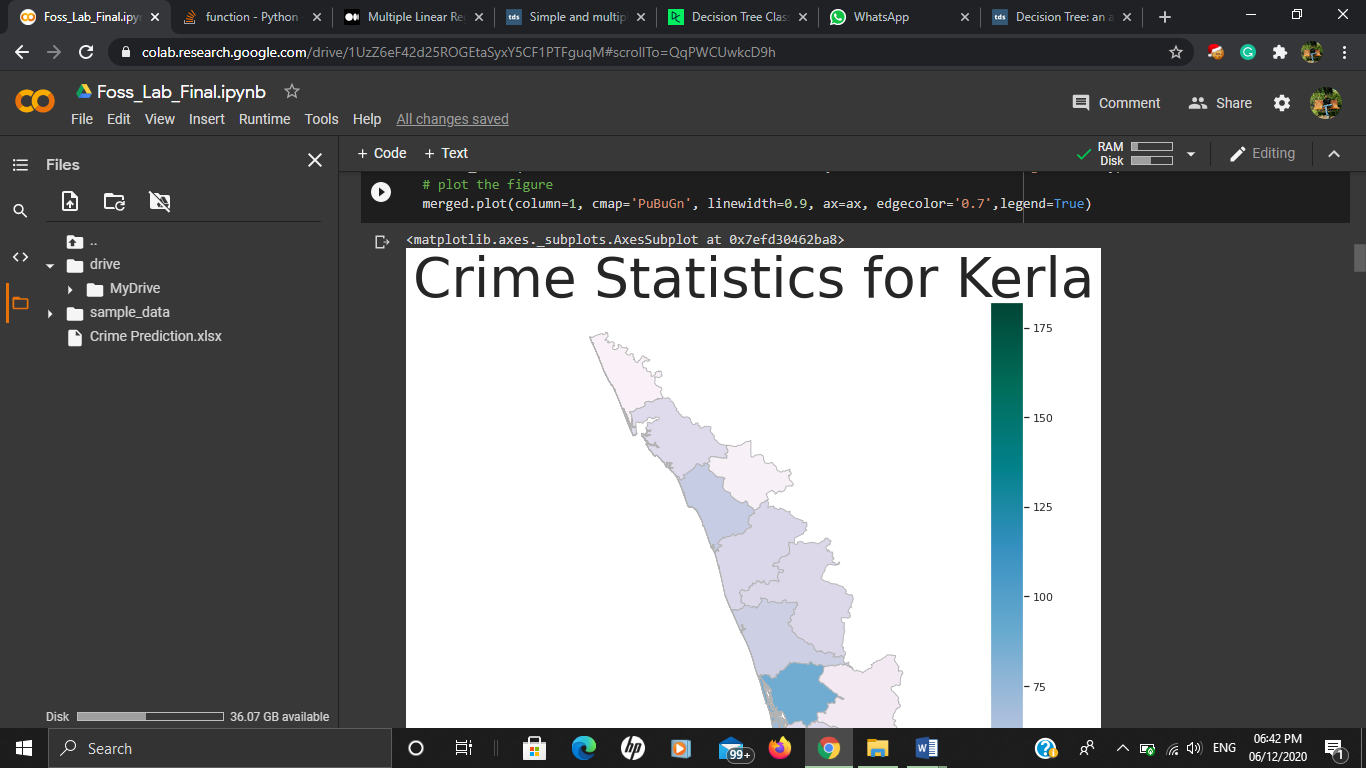


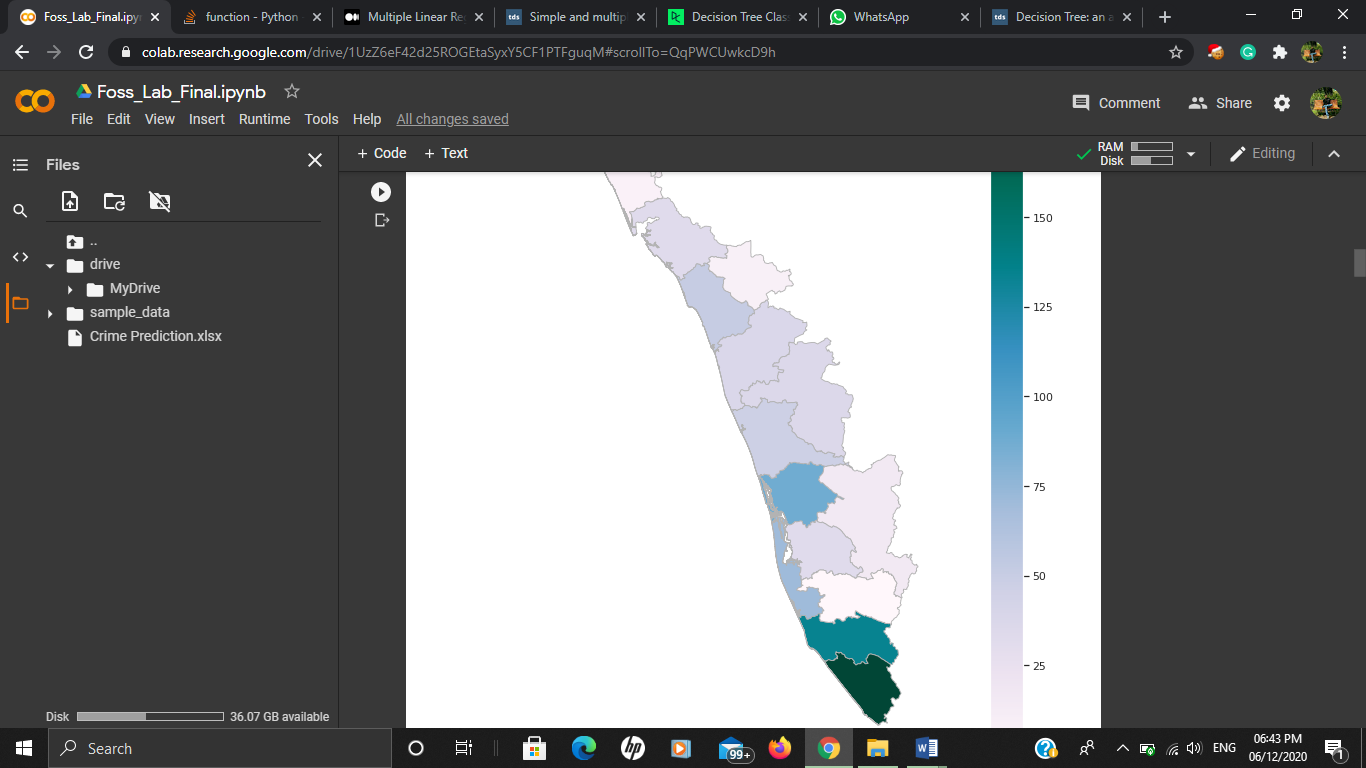


Step3:

Visualizing crime statistics in different parts of Kerala using using geopanda library provided in python. It also incorporates the shape of the dataframe with crime dataframe.

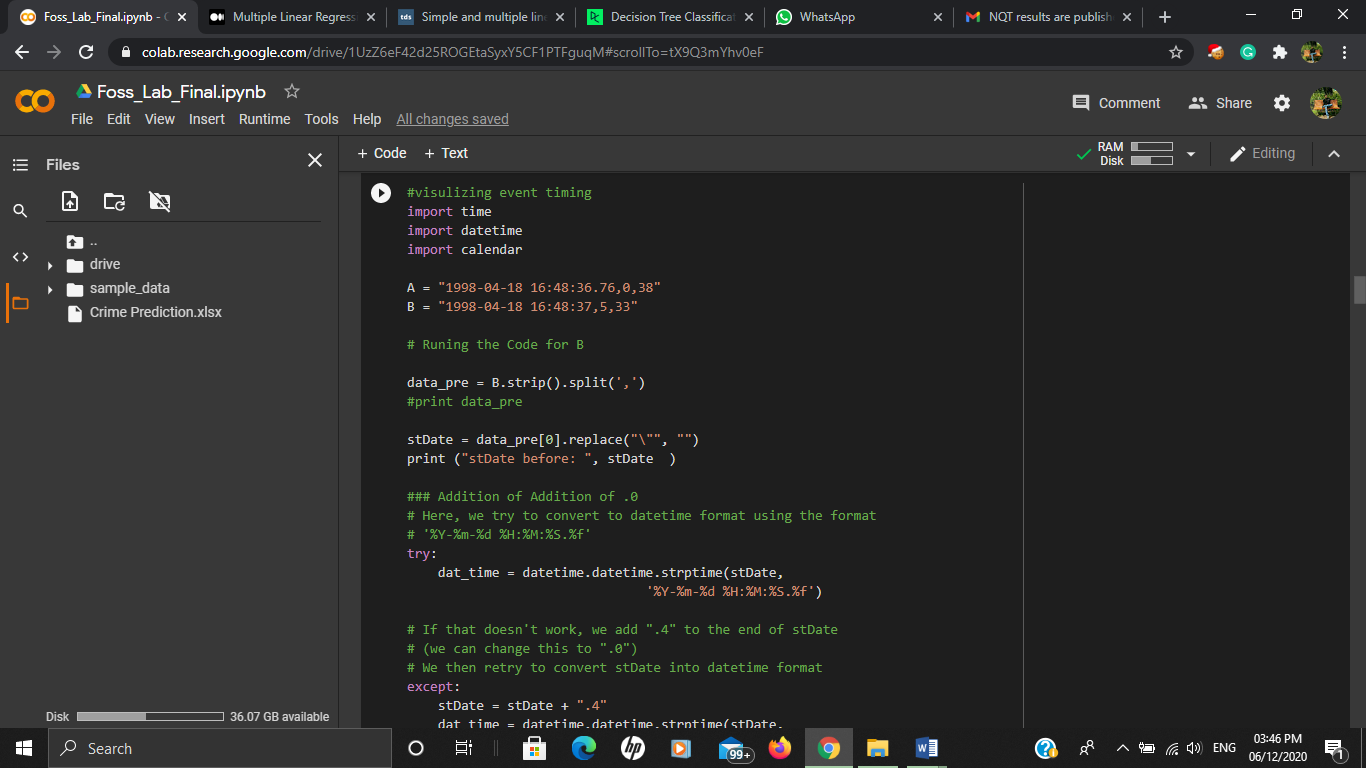


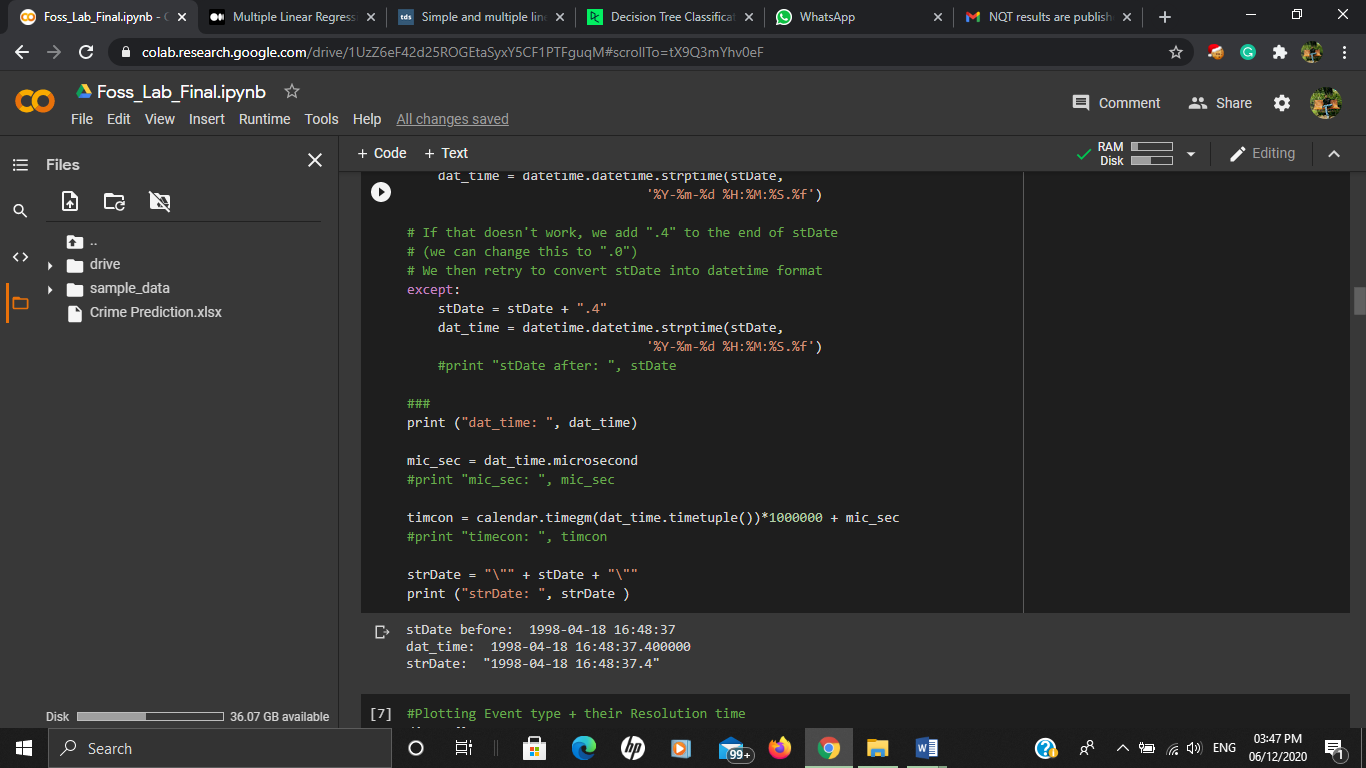




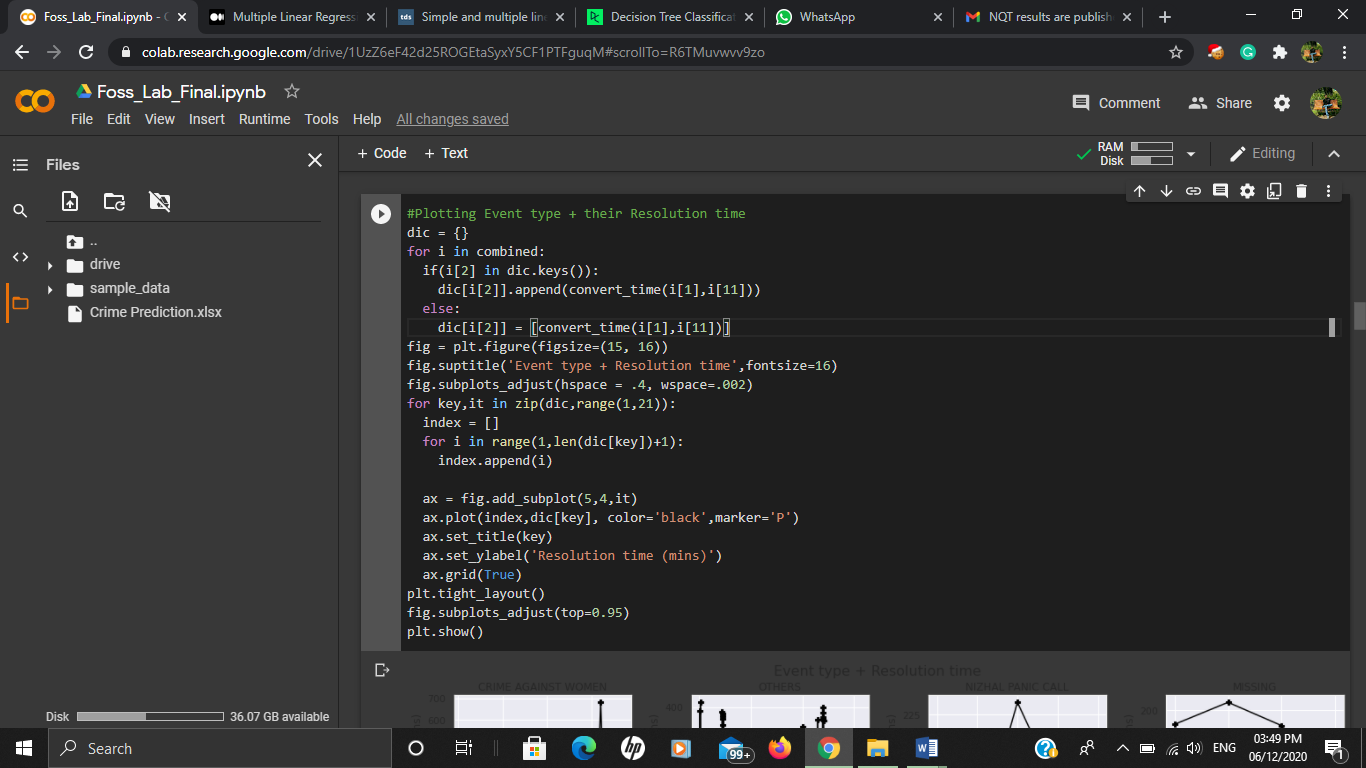
Step4:

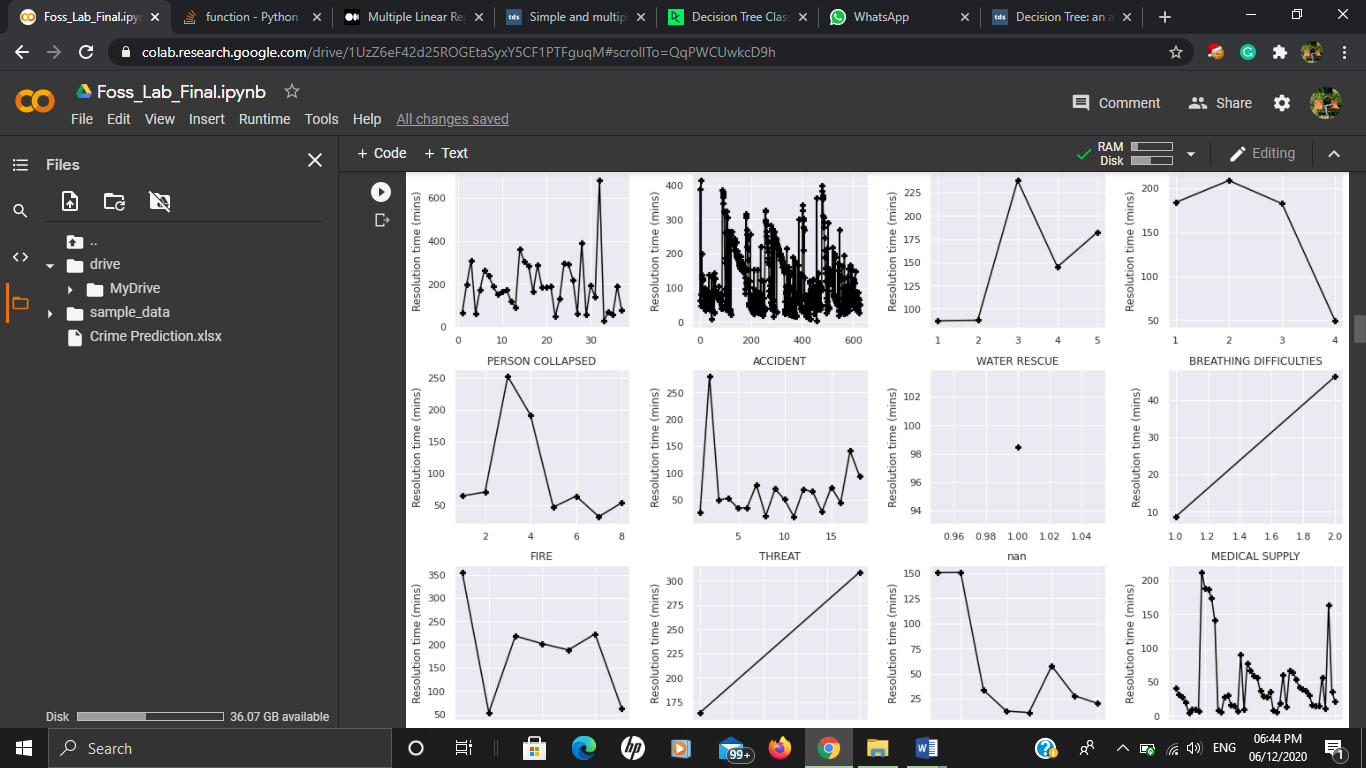
Visualizing crimes with their event time after cleaning and bringing time is their standard form.



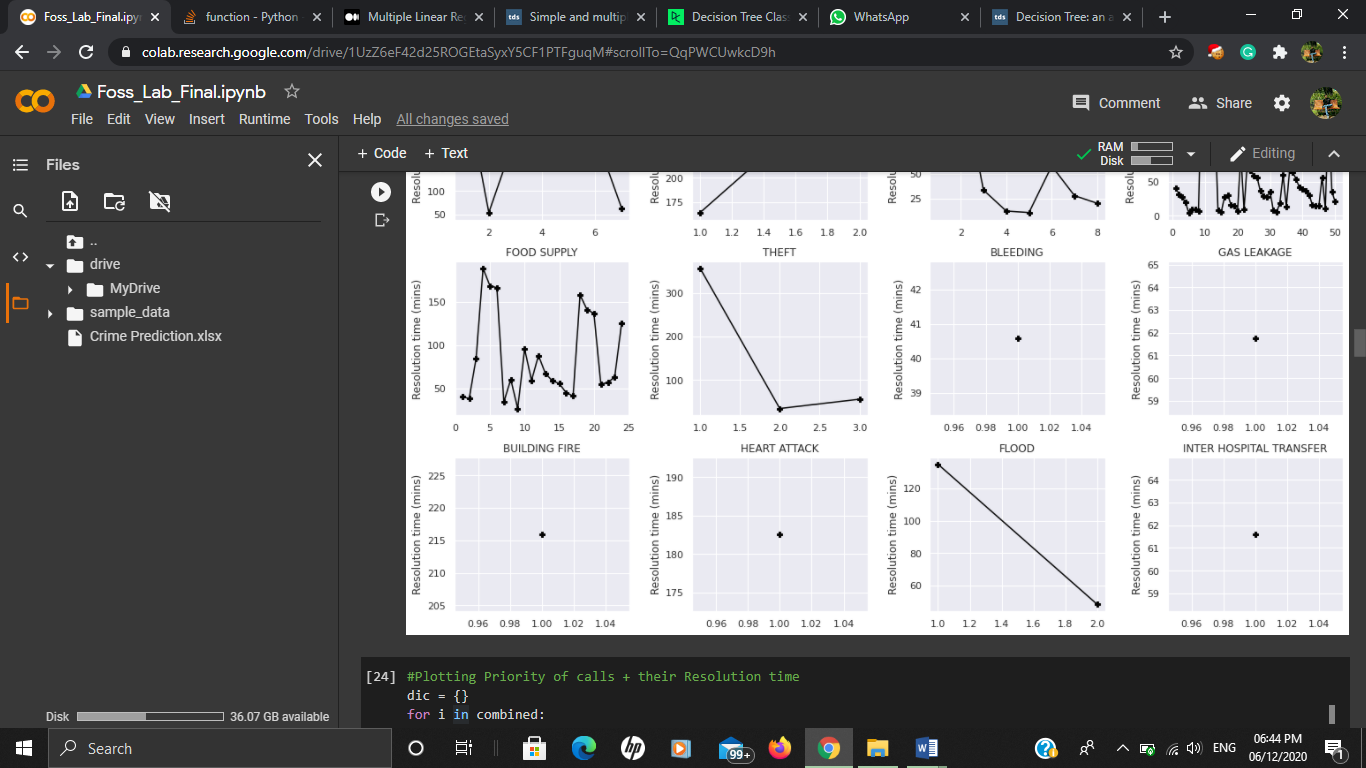


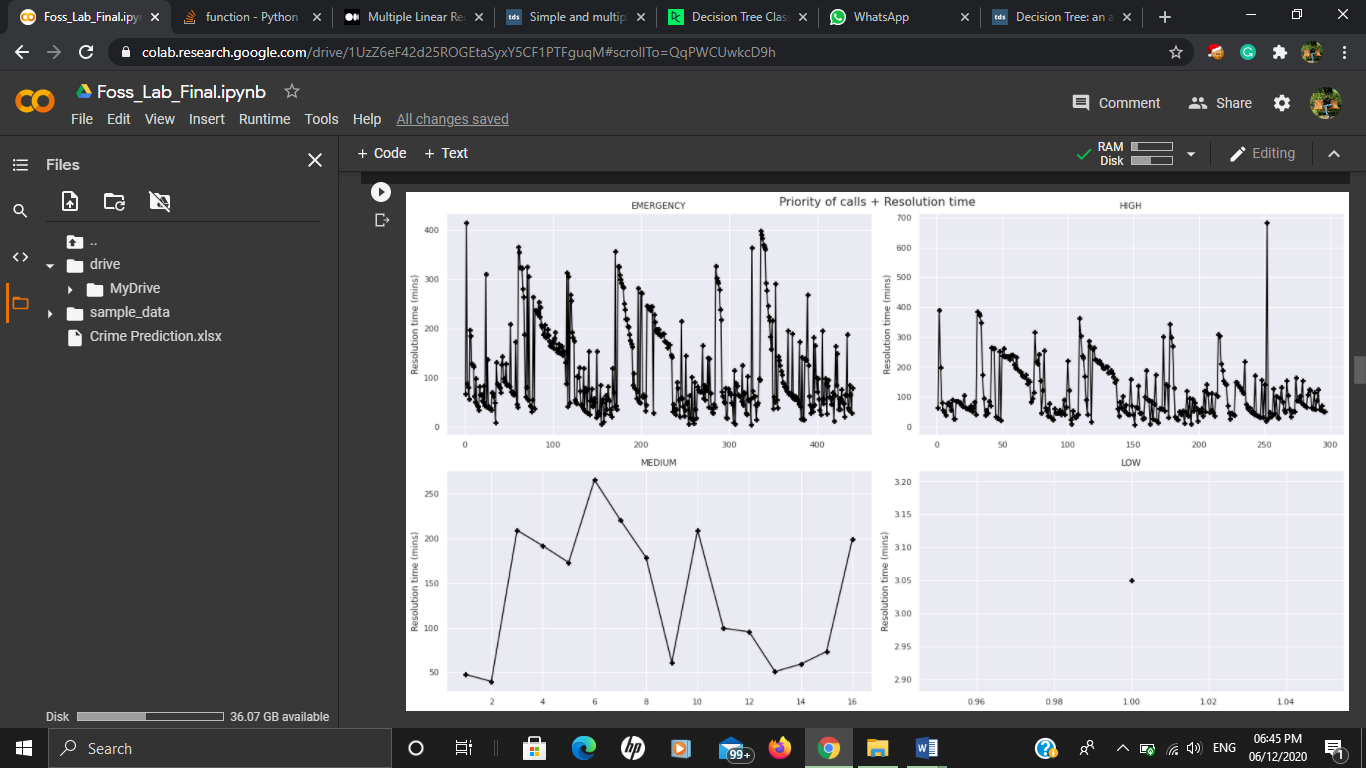
#**Plotting Event type + their Resolution time**





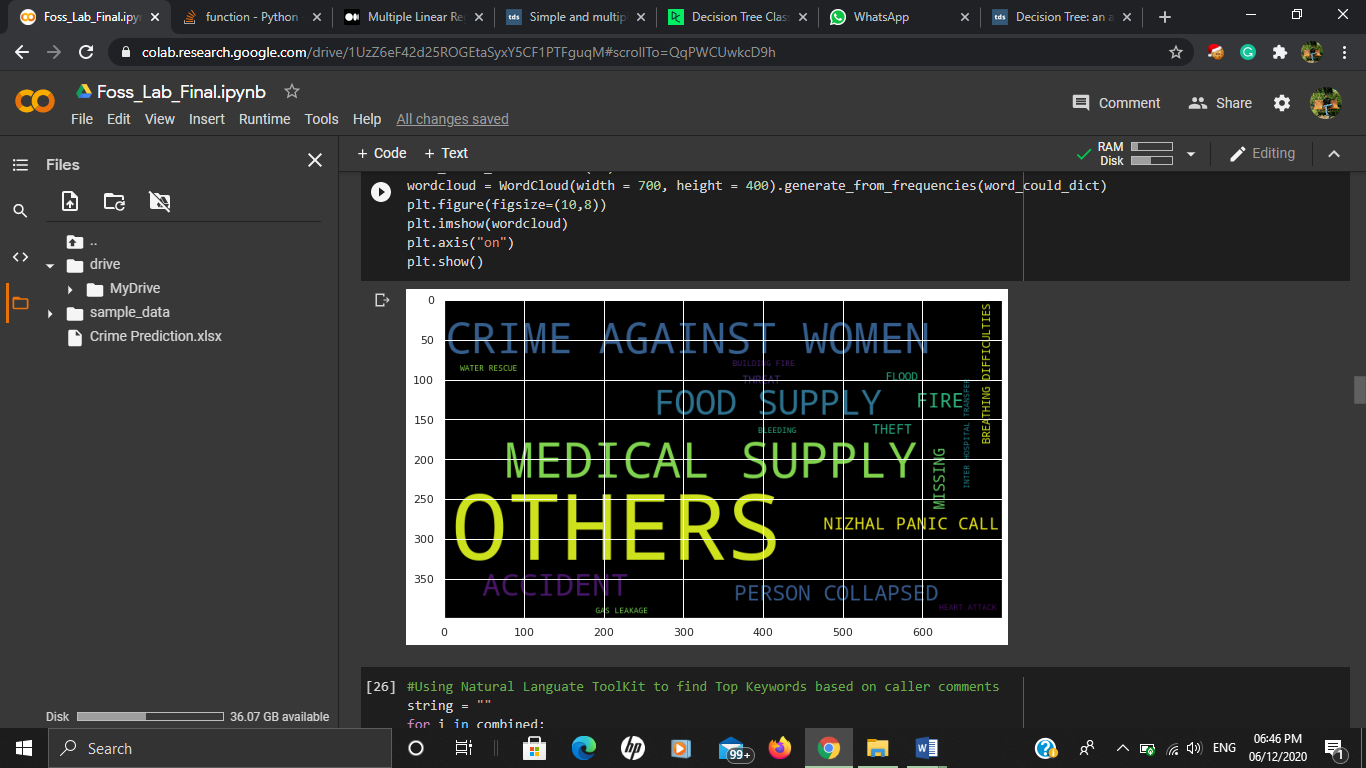
#**Plotting Priority of calls with respect to their Resolution time**





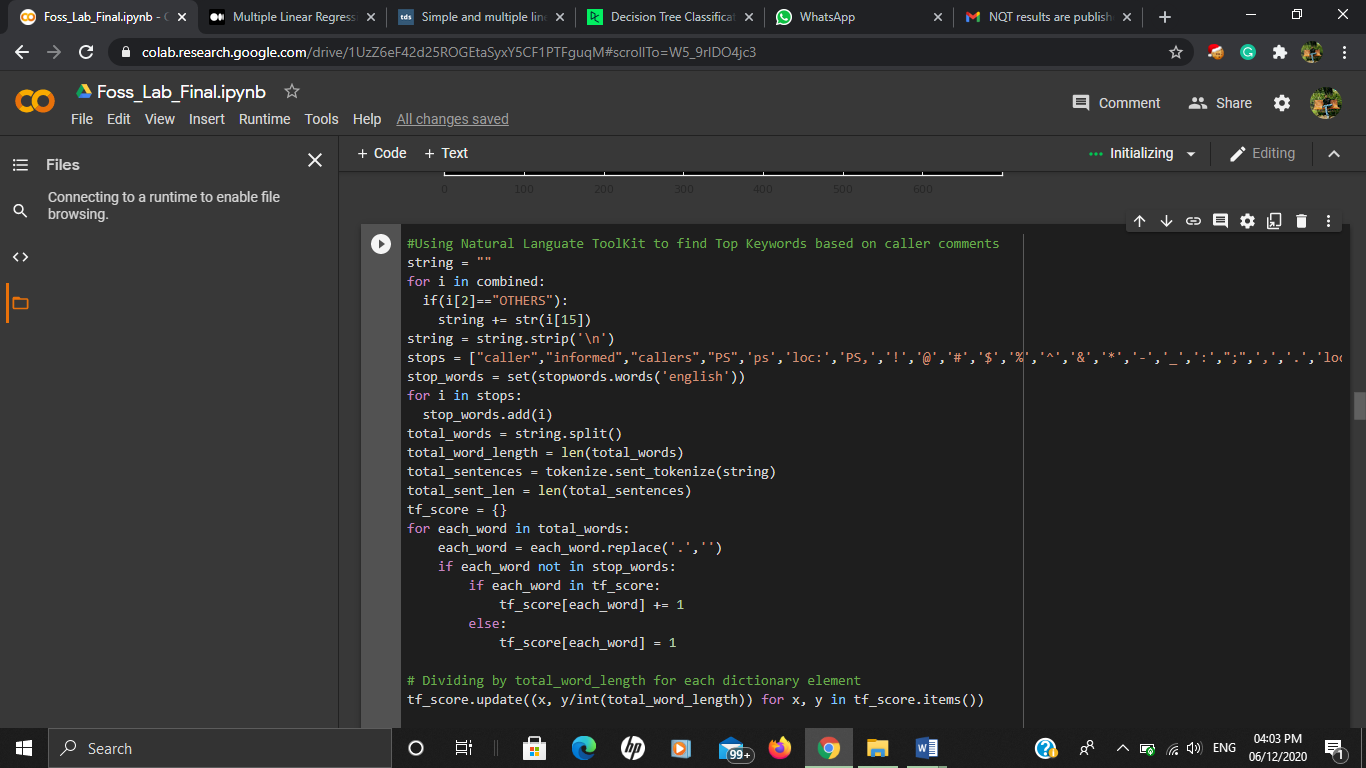
Step5:

**Creating word cloud for crime reported**

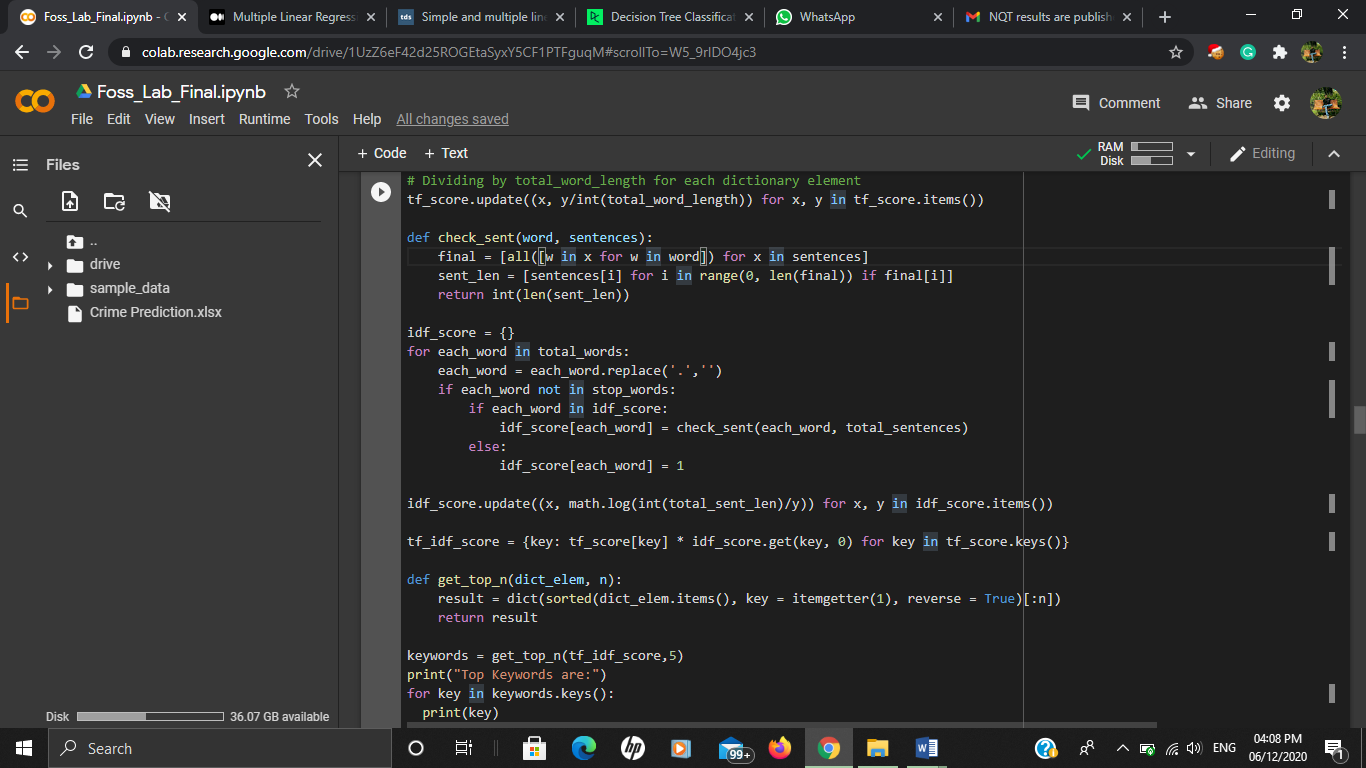


Step6:

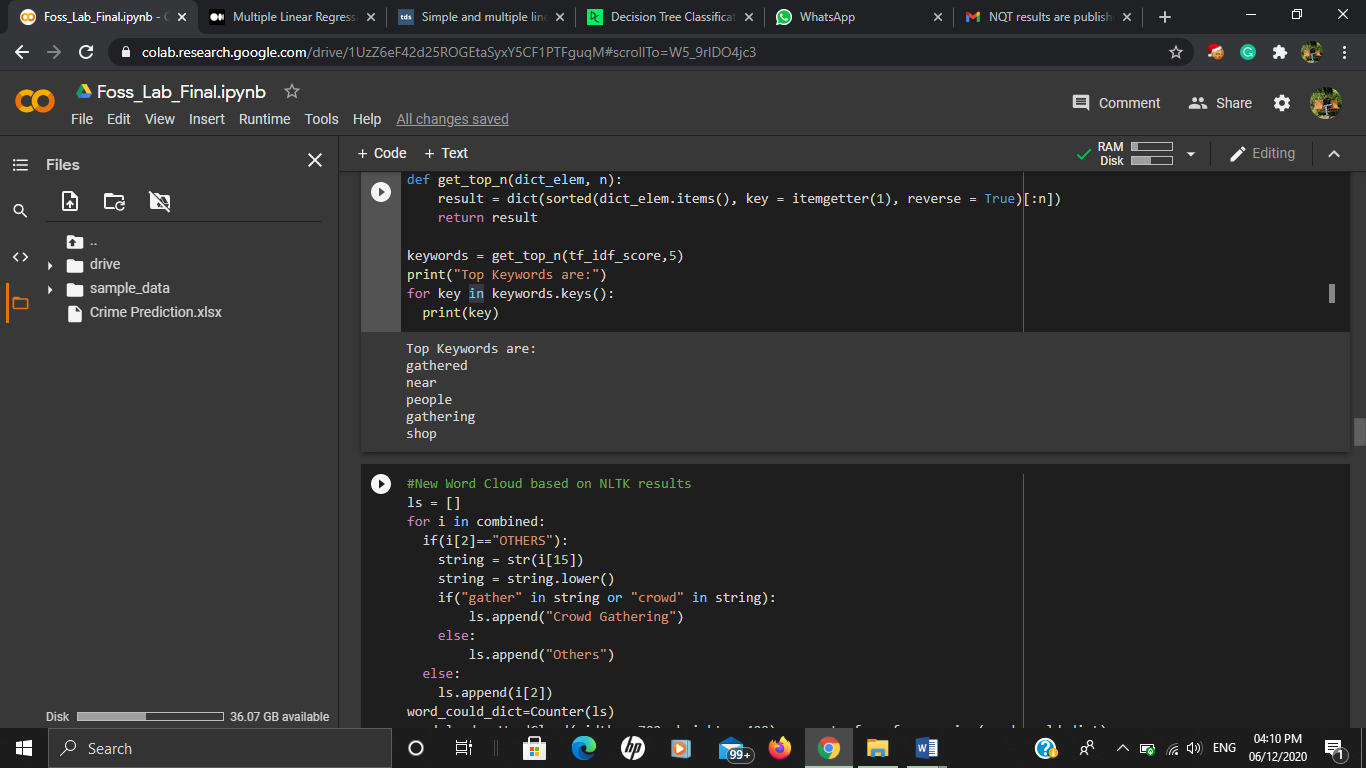
**Using Natural Languate ToolKit (NLTK) to find Top Keywords based on caller comments.**



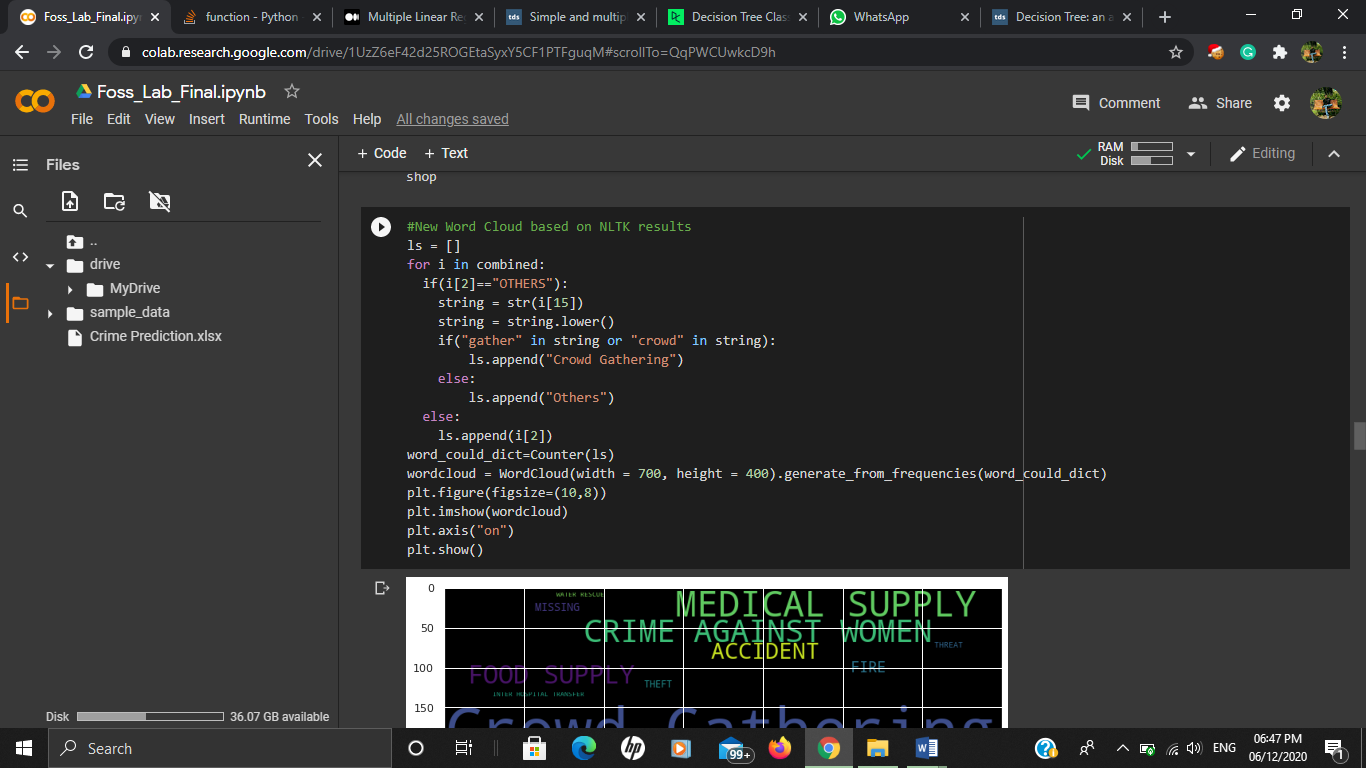
# **Dividing by total\_word\_length for each dictionary element**

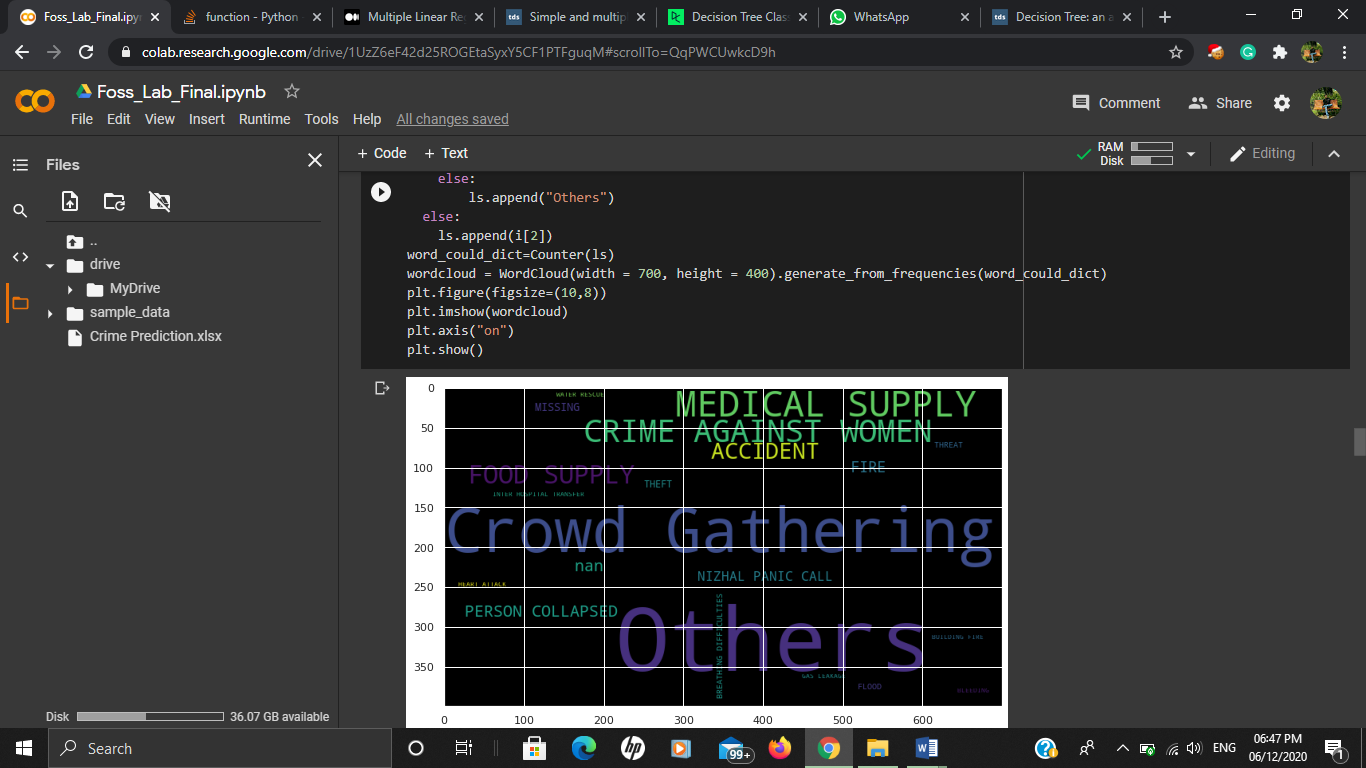


**Frequent occurring word output. We can see some words like gather, near, people gathering shop**



We can also visualize using word cloud.





**CLASSIFICATION**

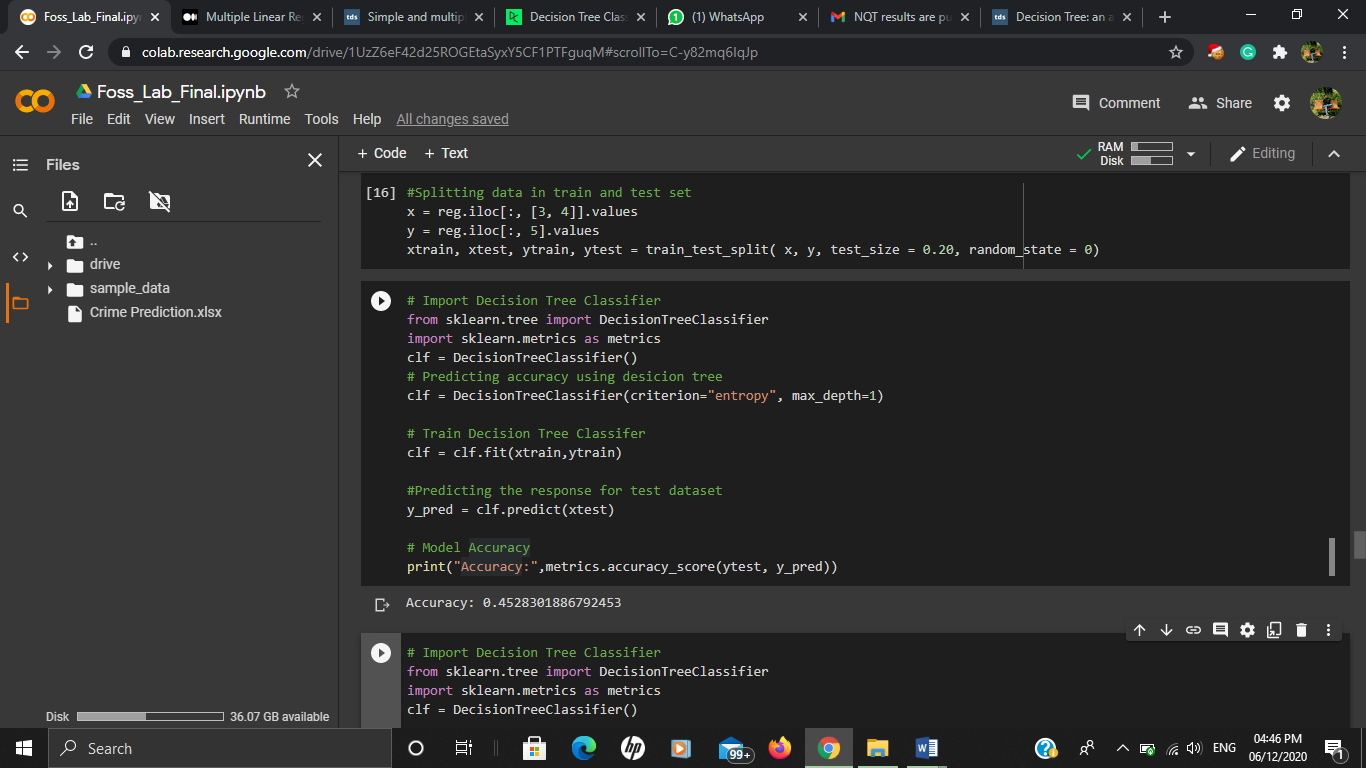
**Decision Tree**

When training a dataset to classify a variable, the idea of the Decision Tree is to divide the data into smaller datasets based on a certain feature value until the target variables all fall under one category. While the human brain decides to pick the “splitting feature” based on the experience (i.e. the cloudy sky), a computer splits the dataset based on the maximum information gain.

Here in this excel sheet we have not been given straight forward data to classify. Here we considering call start time, place of call, time of event reported.

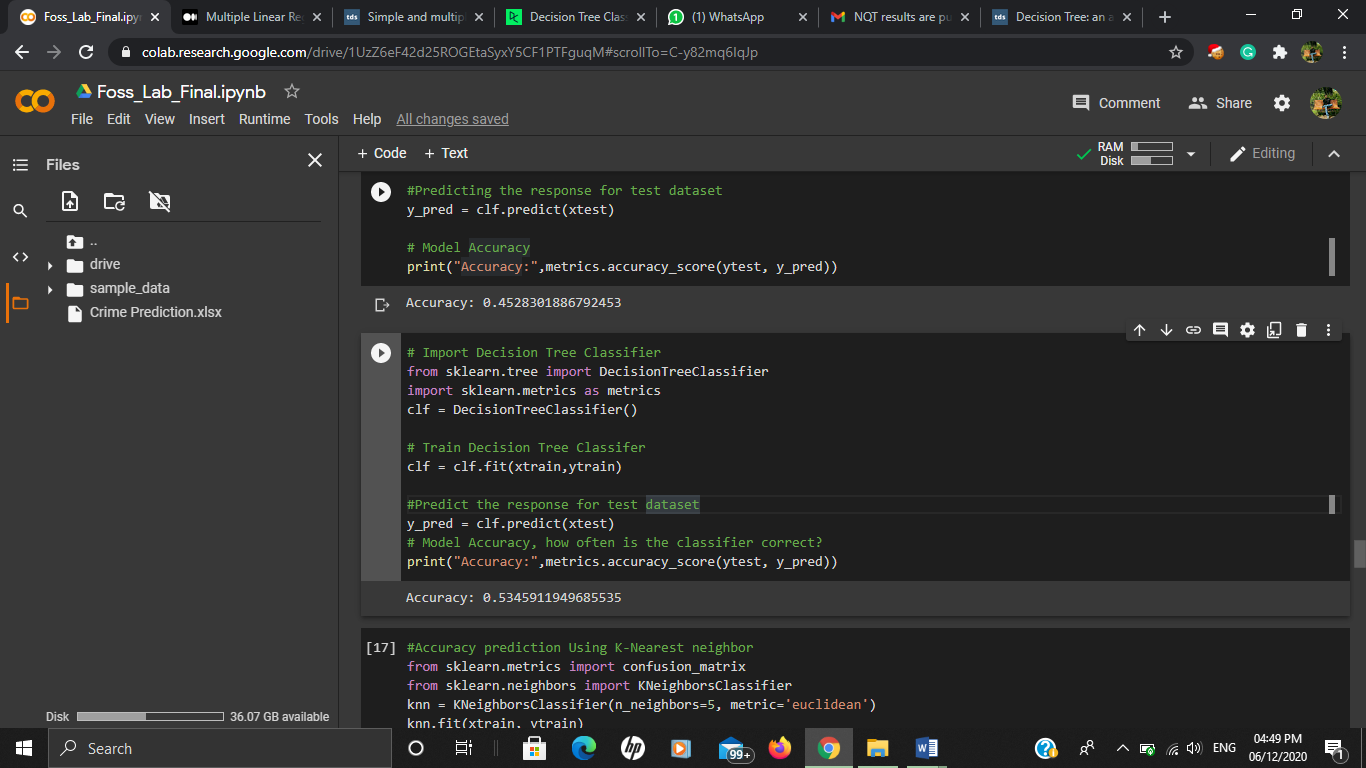
**Accuracy prediction using different criterion and max\_depth.**

We got accuracy of **45%**



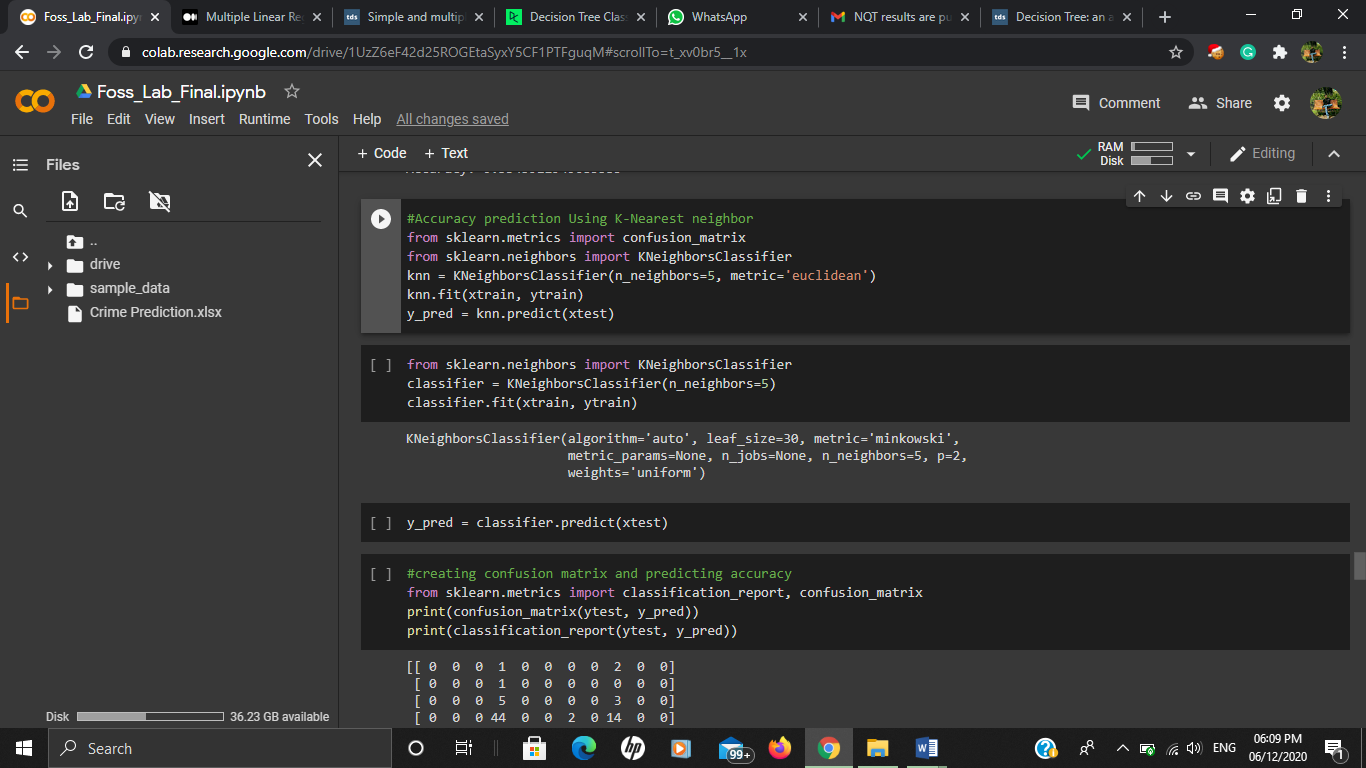
**Accuracy prediction without criterion and max\_depth**

We got accuracy of **53%**

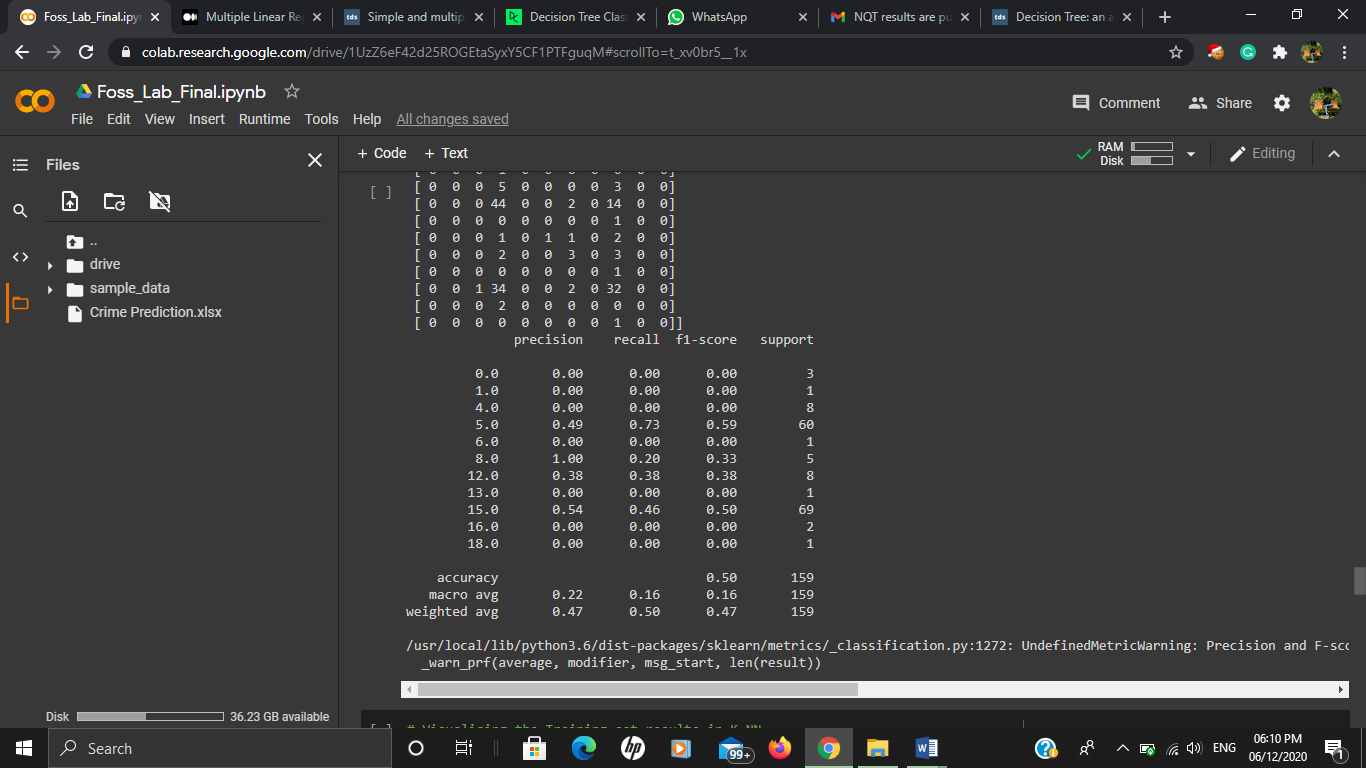


**Accuracy prediction Using K-Nearest neighbour**

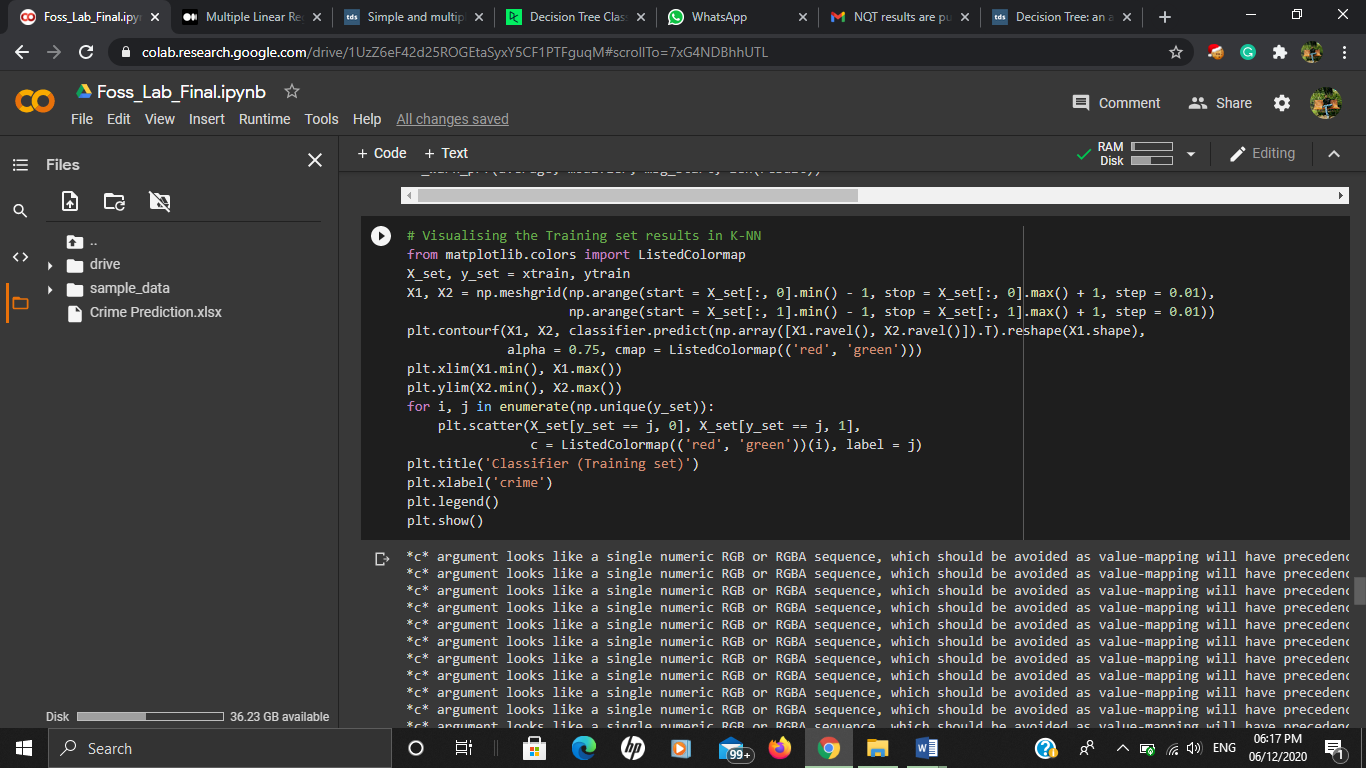
**Accuracy Gain=50%**

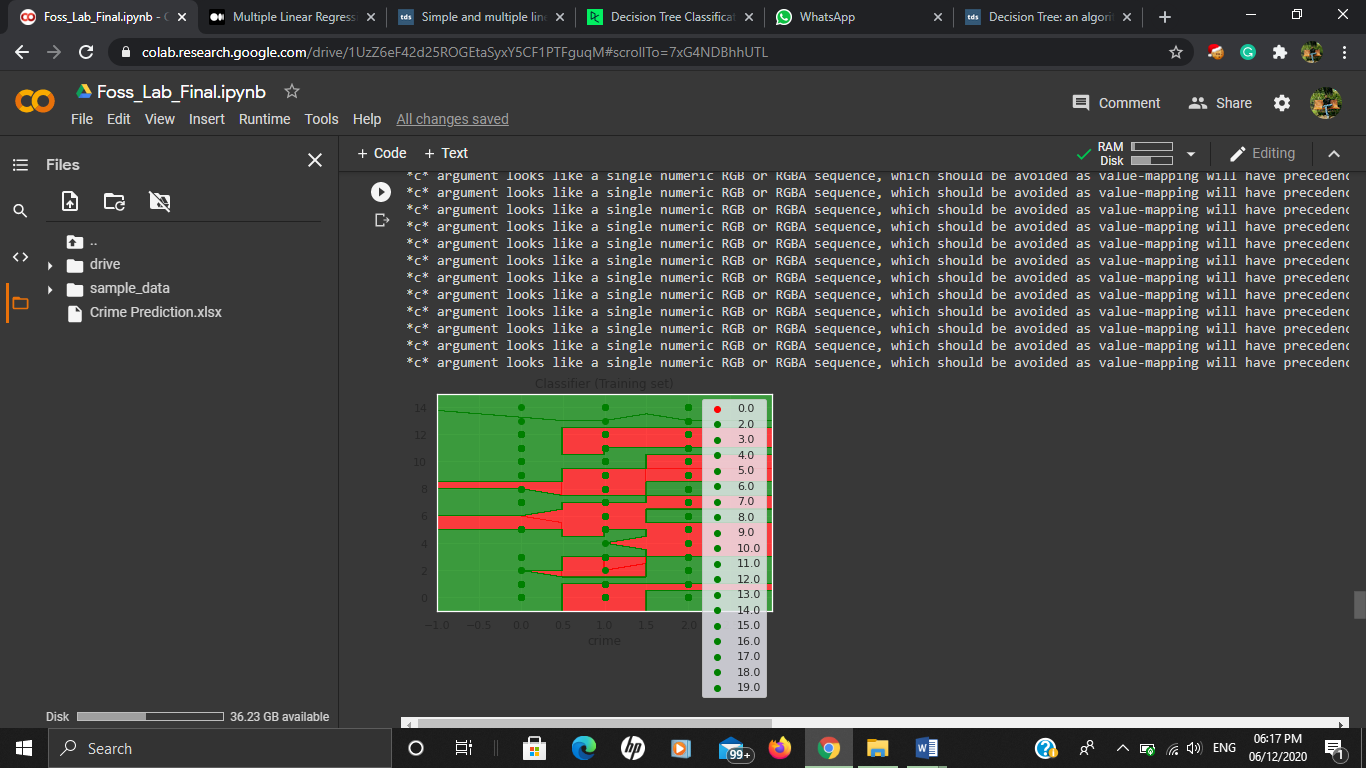


**Confusion Matrix and accuracy.**

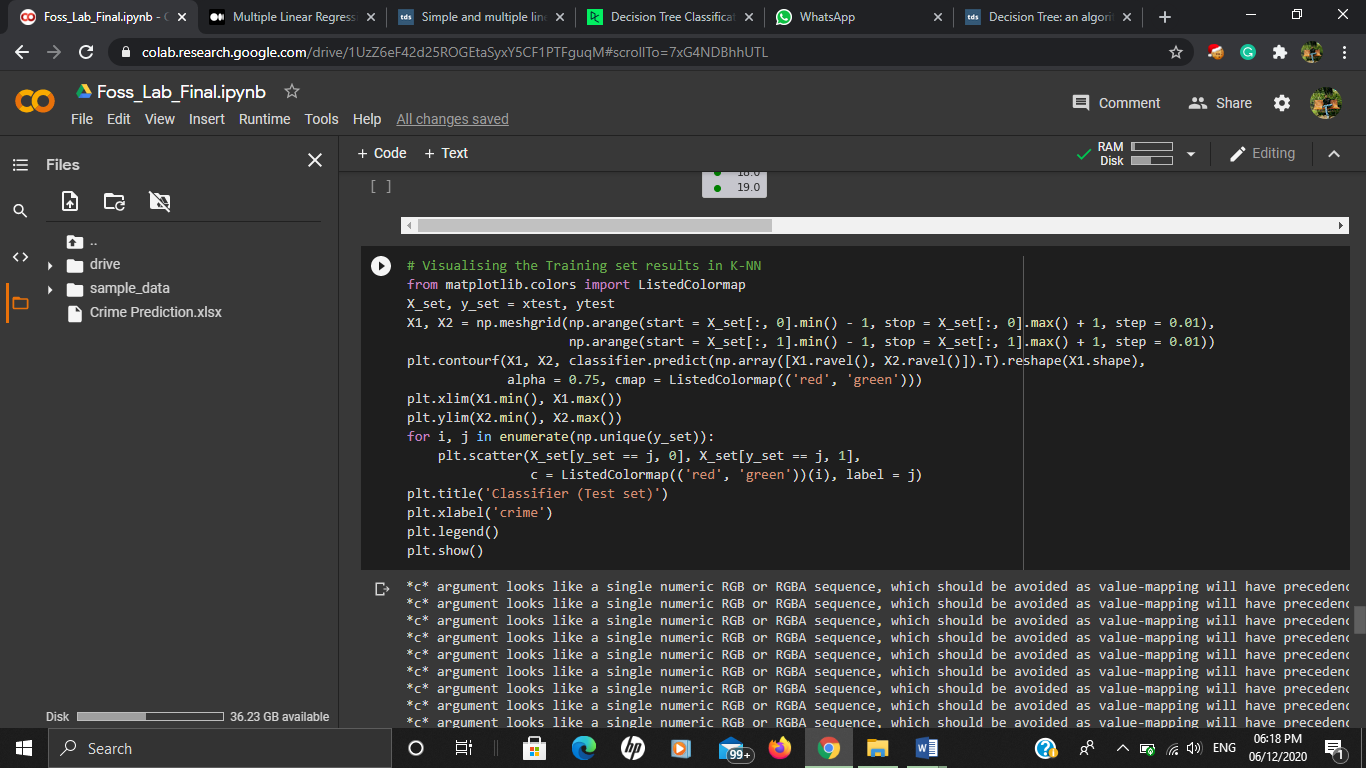


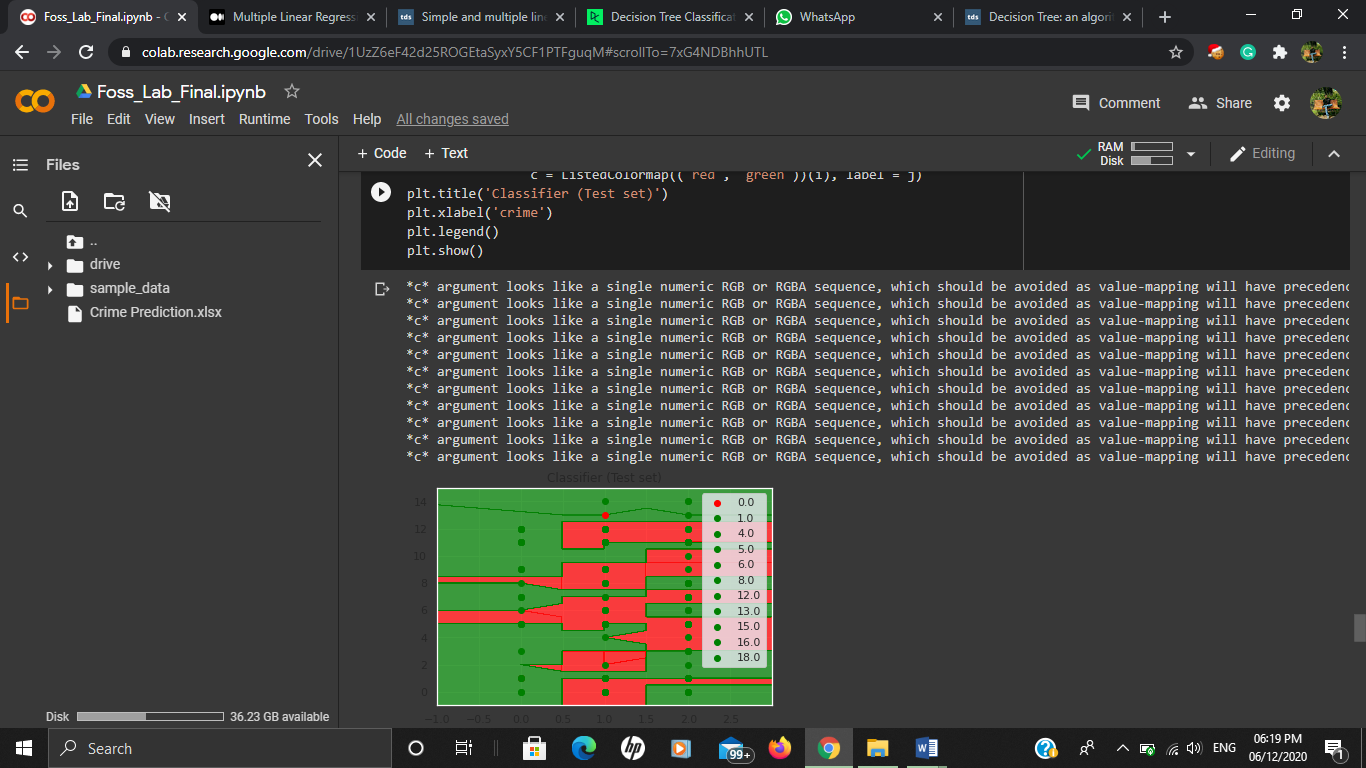
**Visualizing Training data Using K-NN**





**Visualizing Test data Using K-NN**

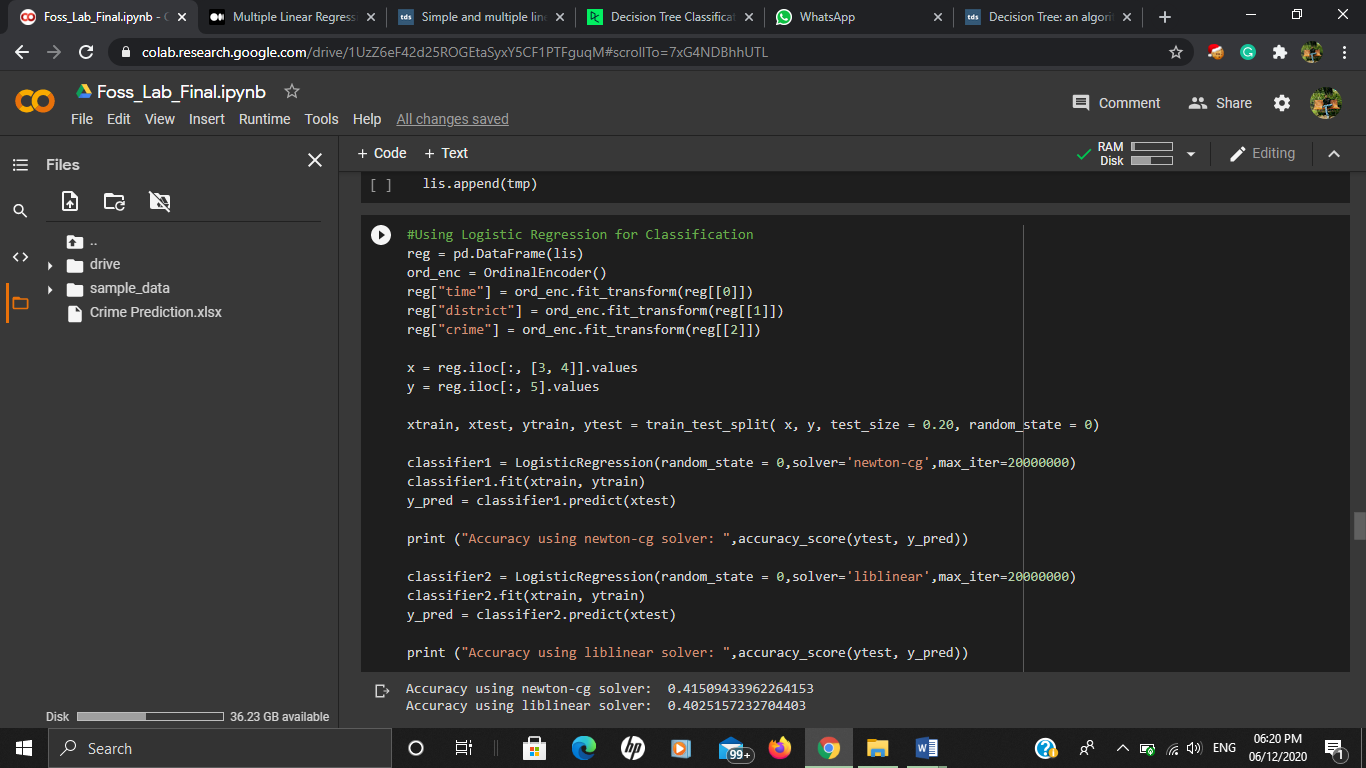




**Accuracy Prediction using Logistic Regression Classification**

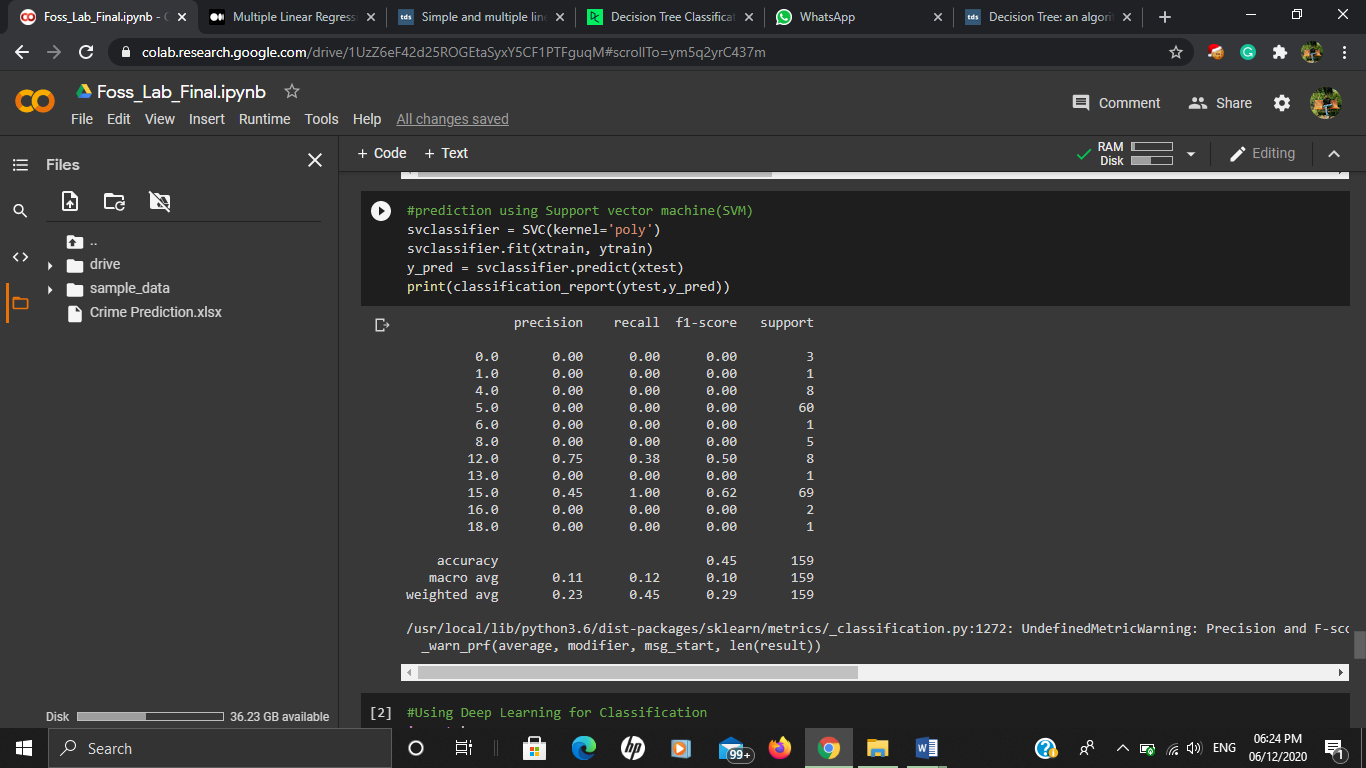
Accuracy using newton-cg solver: **41%**

Accuracy using liblinear solver: **40%**



**Accuracy Prediction using Support Vector Machine**

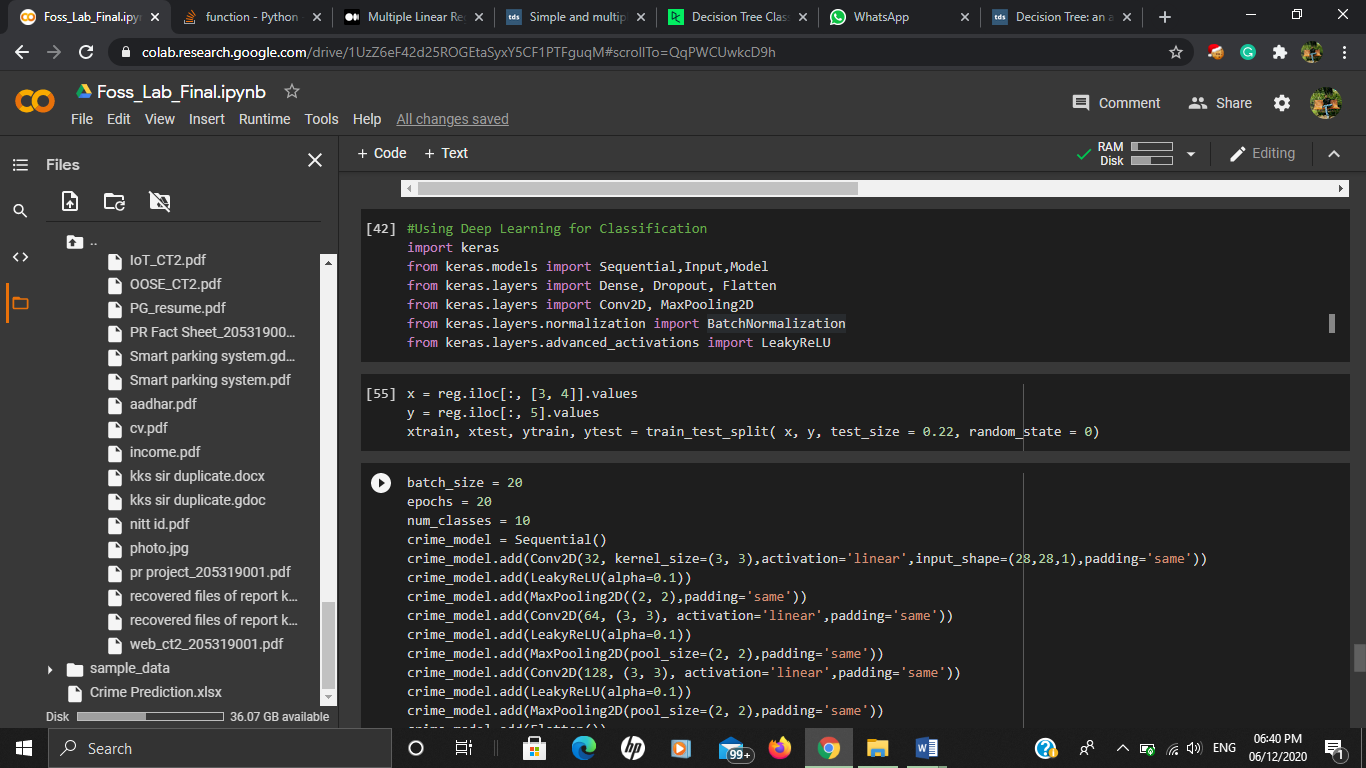
**Accuracy Gain=45%**

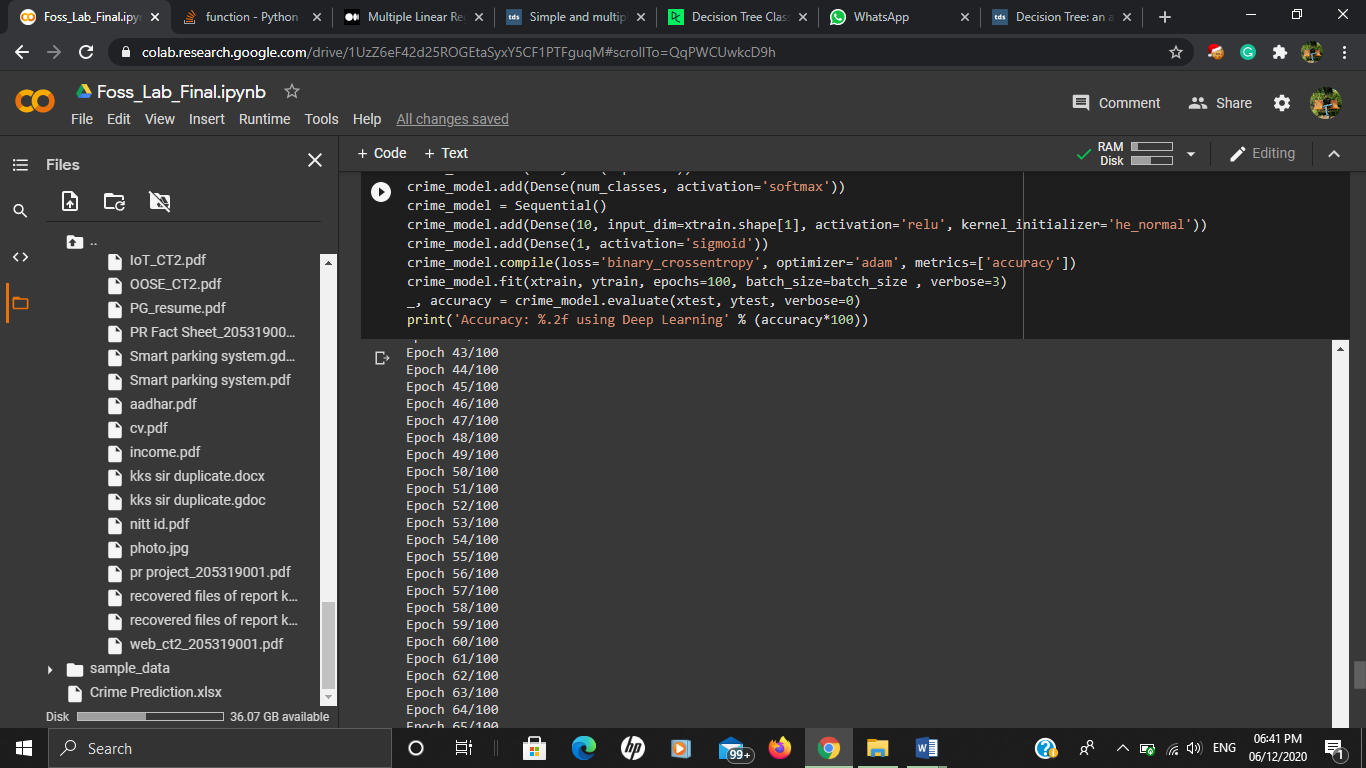


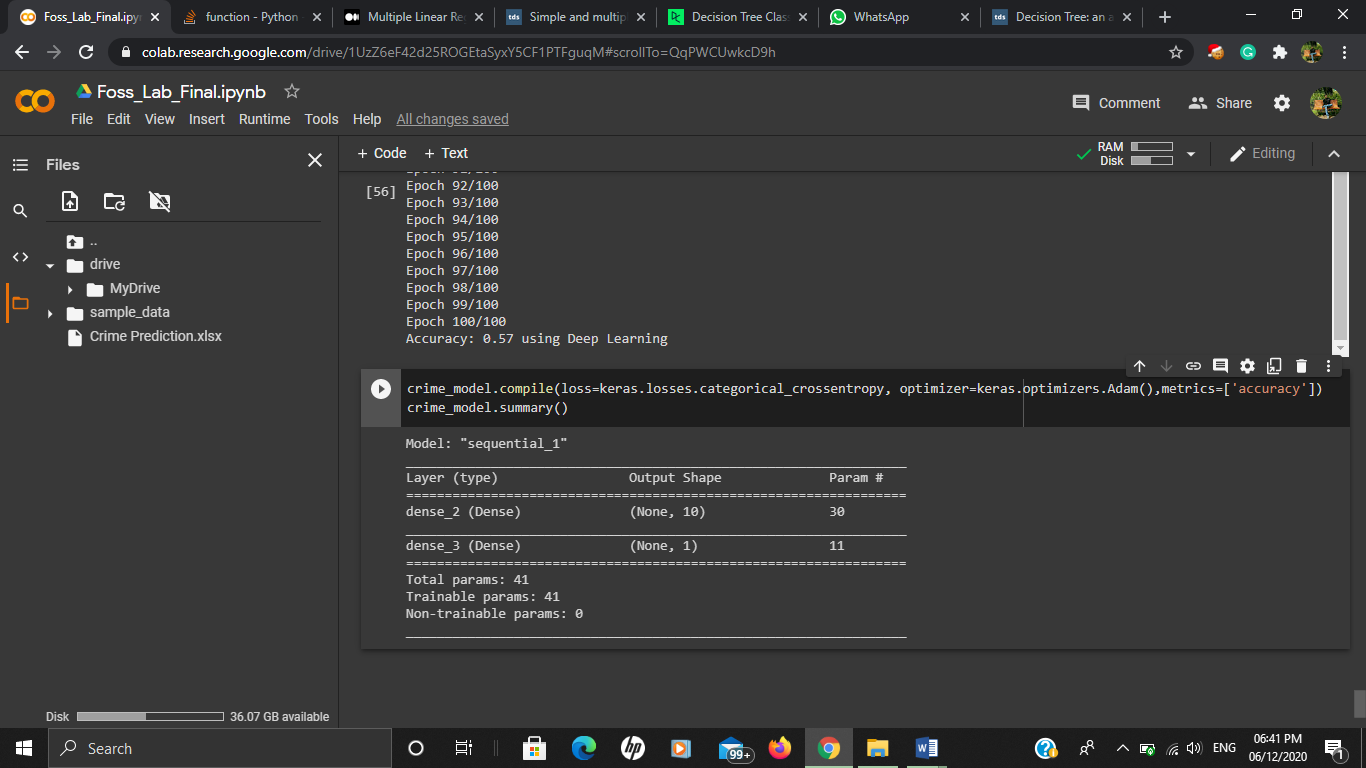
**Accuracy Using Deep Learning Model**

**Accuracy=57%**

**We can see that after using deep learning accuracy**







**Conclusion**

**Accuracy Comparison Table**

|  |  |
| --- | --- |
| **Model Name** | **Accuracy(%)** |
| Decision Tree(With Criterion) | 45% |
| Decision Tree(Without Criterion) | 53% |
| KNN | 50% |
| Logistic Regression With newton-cg | 41% |
| Logistic Regression with liblinear | 40% |
| Deep Learning | 57% |
|  |  |

Deep Learning Model is giving best accuracy of **57%** after that decision tree is giving good accuracy of **53%.**

**Nobility**

This different applied algorithm helps in finding the accurate model for prediction. Different government agencies can apply this kind of model in real life to improve the crime detection crime detection method.

**Code**

#importing the required library

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

import tensorflow.compat.v1 as tf

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

nltk.download('punkt')

nltk.download('stopwords')

import seaborn as sns

sns.set()

from matplotlib import pyplot as plt

#Creating transformation functions for Data Pre-Processing

def transform1(ls):

    ls1 = []

    for i in ls:

        if i[0]!="event\_id":

            if i[0]=='nan' and i[1]=='nan':

                continue

            else:

                ls1.append(i)

    ls2 = transform2(ls1)

    return ls2

def transform2(ls1):

    i=0

    ls2 = []

    while i < len(ls1):

        if ls1[i][0]!='nan':

            nw = ls1[i]

            i+=2

            signal = []

            disconnect = []

            while ls1[i][0]=='nan':

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

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

                i+=1

                if(i==len(ls1)):

                    break

            nw.append(signal)

            nw.append(disconnect)

            ls2.append(nw)

        else:

            i+=1

    return ls2

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("No. of rows after cleaning data: ",len(combined))

  #visulising the crime statistics

  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/IND\_adm/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=(20, 12))

  ax.axis('off')

  ax.set\_title('Crime Statistics for Kerla', fontdict={'fontsize': '55', 'fontweight' : '5'})

  # plot the figure

  merged.plot(column=1, cmap='PuBuGn', linewidth=0.9, ax=ax, edgecolor='0.7',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(":")))

#visulizing event timing

import time

import datetime

import calendar

A = "1998-04-18 16:48:36.76,0,38"

B = "1998-04-18 16:48:37,5,33"

# Runing the Code for B

data\_pre = B.strip().split(',')

#print data\_pre

stDate = data\_pre[0].replace("\"", "")

print ("stDate before: ", stDate  )

### Addition of Addition of .0

# Here, we try to convert to datetime format using the format

# '%Y-%m-%d %H:%M:%S.%f'

try:

    dat\_time = datetime.datetime.strptime(stDate,

                               '%Y-%m-%d %H:%M:%S.%f')

# If that doesn't work, we add ".4" to the end of stDate

# (we can change this to ".0")

# We then retry to convert stDate into datetime format

except:

    stDate = stDate + ".4"

    dat\_time = datetime.datetime.strptime(stDate,

                               '%Y-%m-%d %H:%M:%S.%f')

    #print "stDate after: ", stDate

###

print ("dat\_time: ", dat\_time)

mic\_sec = dat\_time.microsecond

#print "mic\_sec: ", mic\_sec

timcon = calendar.timegm(dat\_time.timetuple())\*1000000 + mic\_sec

#print "timecon: ", timcon

strDate = "\"" + stDate + "\""

print ("strDate: ", strDate )

#Plotting Event type + 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 + Resolution time',fontsize=16)

fig.subplots\_adjust(hspace = .4, wspace=.002)

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='black',marker='P')

  ax.set\_title(key)

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

  ax.grid(True)

plt.tight\_layout()

fig.subplots\_adjust(top=0.95)

plt.show()

#Plotting Priority of calls + 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 + Resolution time',fontsize=16)

fig.subplots\_adjust(hspace = .4, wspace=.002)

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='black',marker='P')

  ax.set\_title(key)

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

  ax.grid(True)

plt.tight\_layout()

fig.subplots\_adjust(top=0.95)

plt.show()

#Creating WordCloud for Crime Reported

ls = []

for i in combined:

  if i[2]=="nan":

    continue

  ls.append(i[2])

word\_could\_dict=Counter(ls)

wordcloud = WordCloud(width = 700, height = 400).generate\_from\_frequencies(word\_could\_dict)

plt.figure(figsize=(10,8))

plt.imshow(wordcloud)

plt.axis("on")

plt.show()

#Using Natural Languate ToolKit 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 = 700, height = 400).generate\_from\_frequencies(word\_could\_dict)

plt.figure(figsize=(10,8))

plt.imshow(wordcloud)

plt.axis("on")

plt.show()

#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 = 0.20, random\_state = 0)

classifier1 = LogisticRegression(random\_state = 0,solver='newton-cg',max\_iter=20000000)

classifier1.fit(xtrain, ytrain)

y\_pred = classifier1.predict(xtest)

print ("Accuracy using newton-cg solver: ",accuracy\_score(ytest, y\_pred))

classifier2 = LogisticRegression(random\_state = 0,solver='liblinear',max\_iter=20000000)

classifier2.fit(xtrain, ytrain)

y\_pred = classifier2.predict(xtest)

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

#Splitting data in train and test set

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.20, random\_state = 0)

# Import Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

import sklearn.metrics as metrics

clf = DecisionTreeClassifier()

# Predicting accuracy using desicion tree

clf = DecisionTreeClassifier(criterion="entropy", max\_depth=1)

# Train Decision Tree Classifer

clf = clf.fit(xtrain,ytrain)

#Predicting the response for test dataset

y\_pred = clf.predict(xtest)

# Model Accuracy

print("Accuracy:",metrics.accuracy\_score(ytest, y\_pred))

# Import Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

import sklearn.metrics as metrics

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer

clf = clf.fit(xtrain,ytrain)

#Predict the response for test dataset

y\_pred = clf.predict(xtest)

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(ytest, y\_pred))

# Import Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

import sklearn.metrics as metrics

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer

clf = clf.fit(xtrain,ytrain)

#Predict the response for test dataset

y\_pred = clf.predict(xtest)

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(ytest, y\_pred))

#Accuracy prediction Using K-Nearest neighbor

from sklearn.metrics import confusion\_matrix

from sklearn.neighbors import KNeighborsClassifier

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

knn.fit(xtrain, ytrain)

y\_pred = knn.predict(xtest)

y\_pred = classifier.predict(xtest)

#creating confusion matrix and predicting accuracy

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

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

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

# Visualising the Training set results in K-NN

from matplotlib.colors import ListedColormap

X\_set, y\_set = xtrain, ytrain

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Training set)')

plt.xlabel('crime')

plt.legend()

plt.show()

# Visualising the Training set results in K-NN

from matplotlib.colors import ListedColormap

X\_set, y\_set = xtest, ytest

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Test set)')

plt.xlabel('crime')

plt.legend()

plt.show()

#prediction using Support vector machine(SVM)

svclassifier = SVC(kernel='poly')

svclassifier.fit(xtrain, ytrain)

y\_pred = svclassifier.predict(xtest)

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

#Using Deep Learning for Classification

import keras

from keras.models import Sequential,Input,Model

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras.layers.normalization import BatchNormalization

from keras.layers.advanced\_activations import LeakyReLU

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.22, random\_state = 0)

batch\_size = 20

epochs = 20

num\_classes = 10

crime\_model = Sequential()

crime\_model.add(Conv2D(32, kernel\_size=(3, 3),activation='linear',input\_shape=(28,28,1),padding='same'))

crime\_model.add(LeakyReLU(alpha=0.1))

crime\_model.add(MaxPooling2D((2, 2),padding='same'))

crime\_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))

crime\_model.add(LeakyReLU(alpha=0.1))

crime\_model.add(MaxPooling2D(pool\_size=(2, 2),padding='same'))

crime\_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))

crime\_model.add(LeakyReLU(alpha=0.1))

crime\_model.add(MaxPooling2D(pool\_size=(2, 2),padding='same'))

crime\_model.add(Flatten())

crime\_model.add(Dense(128, activation='linear'))

crime\_model.add(LeakyReLU(alpha=0.1))

crime\_model.add(Dense(num\_classes, activation='softmax'))

crime\_model = Sequential()

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

crime\_model.add(Dense(1, activation='sigmoid'))

crime\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

crime\_model.fit(xtrain, ytrain, epochs=100, batch\_size=batch\_size , verbose=3)

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

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

crime\_model.compile(loss=keras.losses.categorical\_crossentropy, optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

crime\_model.summary()