**Foss Lab Final Project**

Student details-

Name- Aniket Shukla

Roll no- 205319002

Std - M.Sc(III sem)

**Objective: Classification, Visualization, and Analysis of the Kerala state crime dataset.**

**About Dataset:** Dataset provided is with the name “Kerala State 100 Helpline dataset”.

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

**Python Libraries used:**

1. **Pandas:** Pandas are used for data manipulation and analysis.
2. **Numpy:** Numpy is used for performing mathematical operations on arrays such as trigonometric, statistical, and algebraic.
3. **Matplotlib:** Matlotlib is used for 2D graphics in python.
4. **NLTK:** This toolkit is one of the most powerful NLP libraries which contains packages to make machines understand human language and reply to it with an appropriate response.
5. **Geopandas:** Geoplot is a **Python** library providing a selection of easy-to-use geospatial visualizations.
6. **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.
7. **Sklearn:** It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
8. **Keras:** **Keras** is an open-source library that provides a **Python** interface for artificial neural networks. **Keras** acts as an interface for the TensorFlow library.

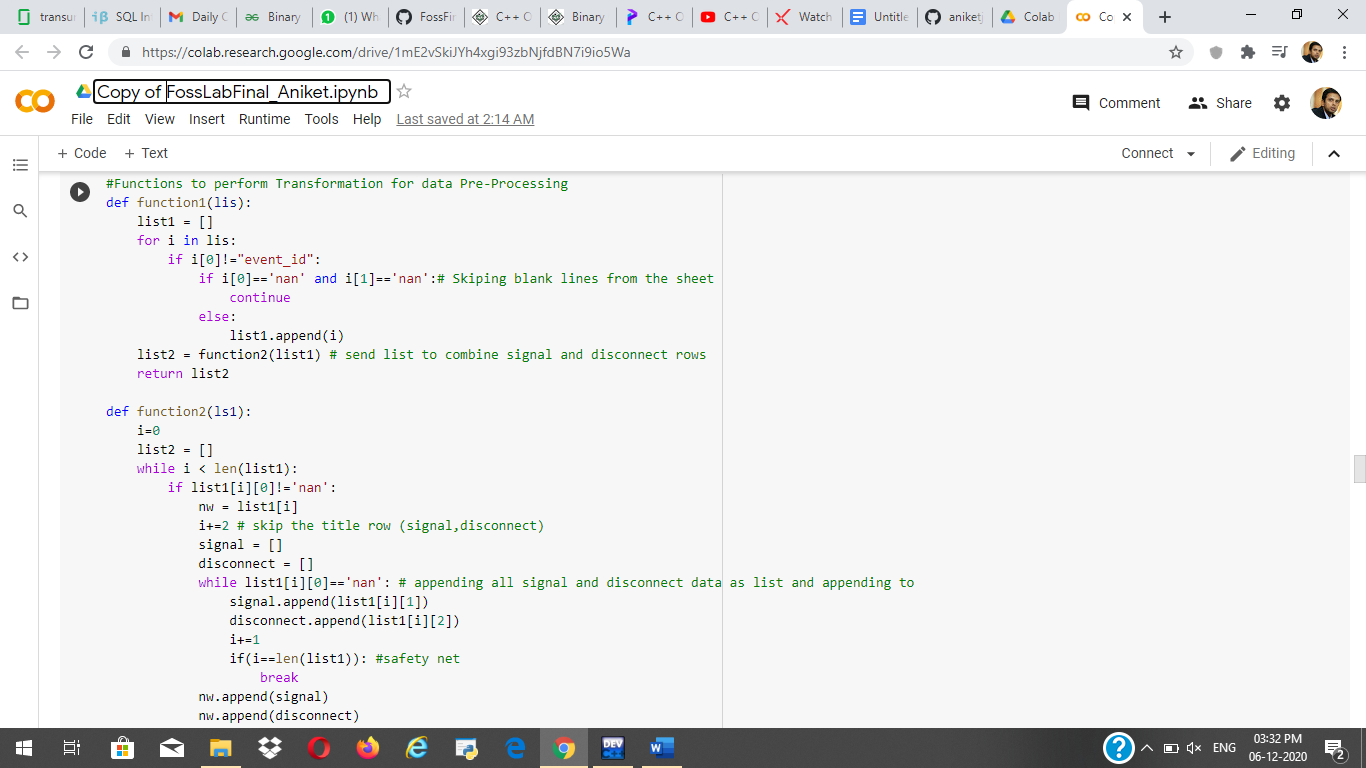
**Github link:** <https://github.com/aniketjee01/FossLabFinalLab>

**Steps to perform the project:**

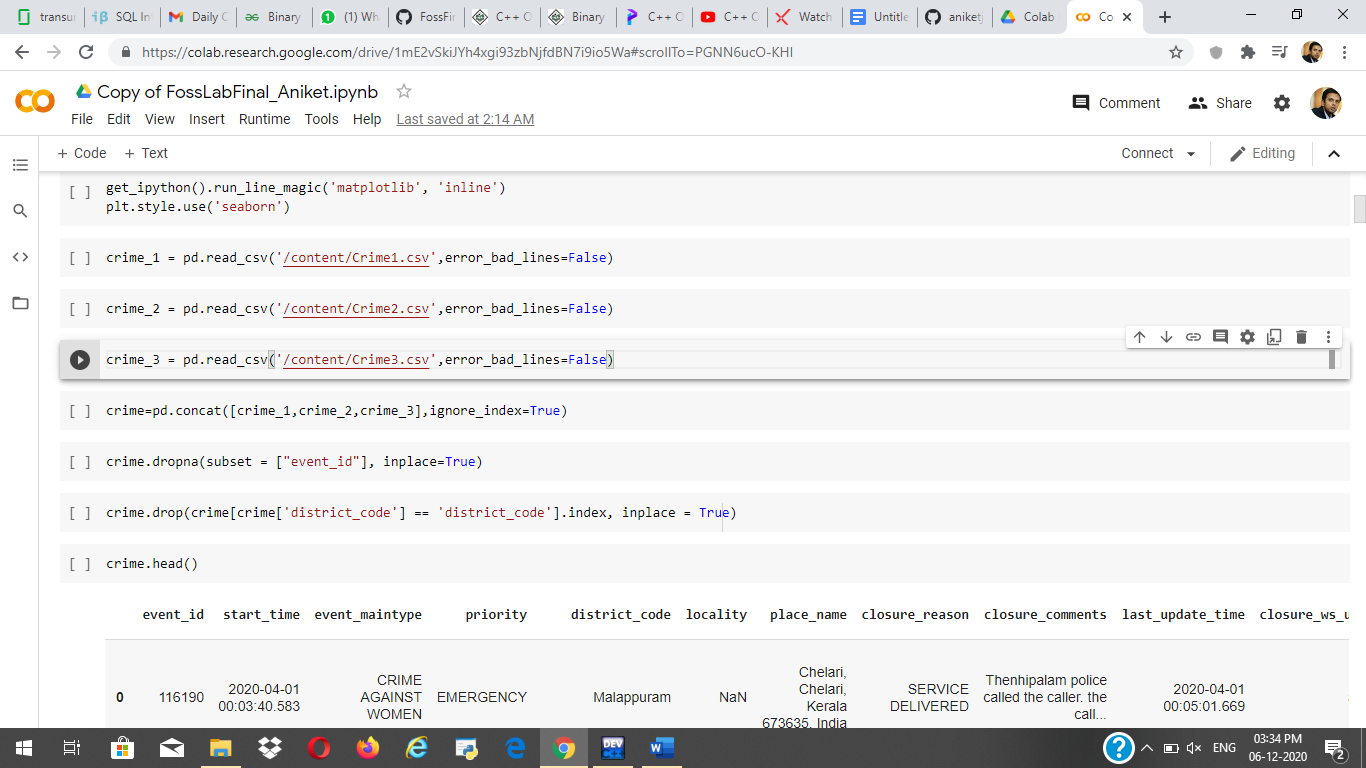
1. **Data Preprocessing:** The dataset received was in raw format. There were redundancies, duplicates, and a number of white spaces left. We have done data preprocessing with two different kinds of methods. The first one is that we loop through all the sheets while calling the transforming functions and creating a combined list with all cleaned data. We get a total of 801 values after processing the data.

The second one is that we have made three different CSV files of the excel sheets given and then concatenated each of them into a single file.

First method-

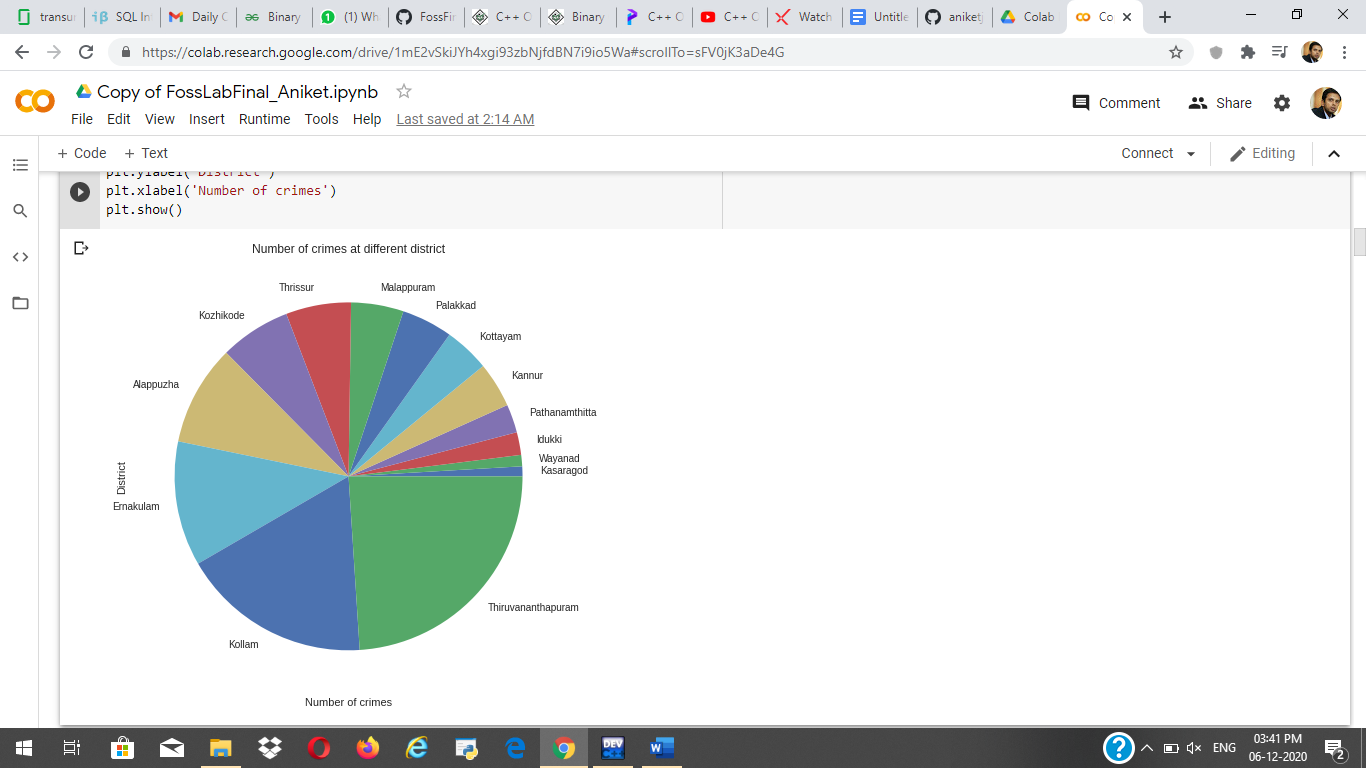


Second Method:

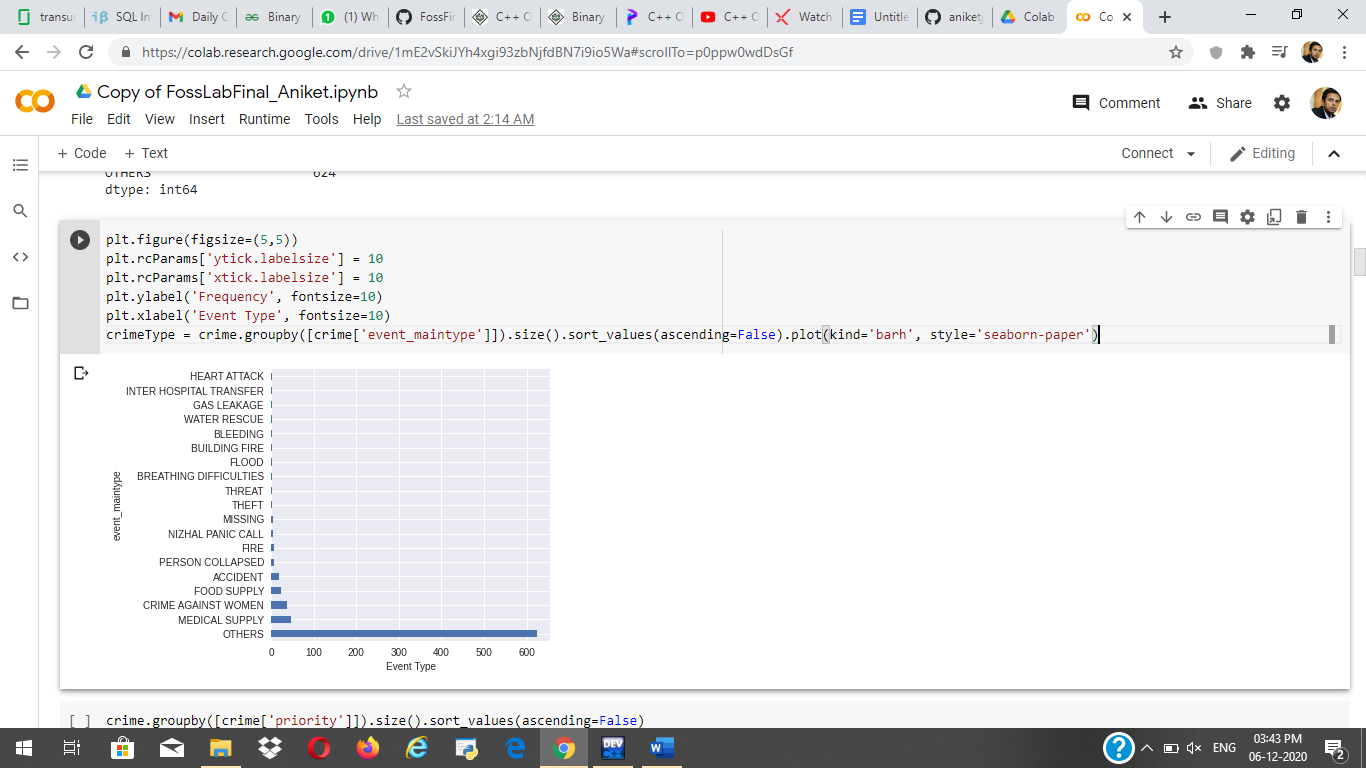


1. **Data Visualization:**

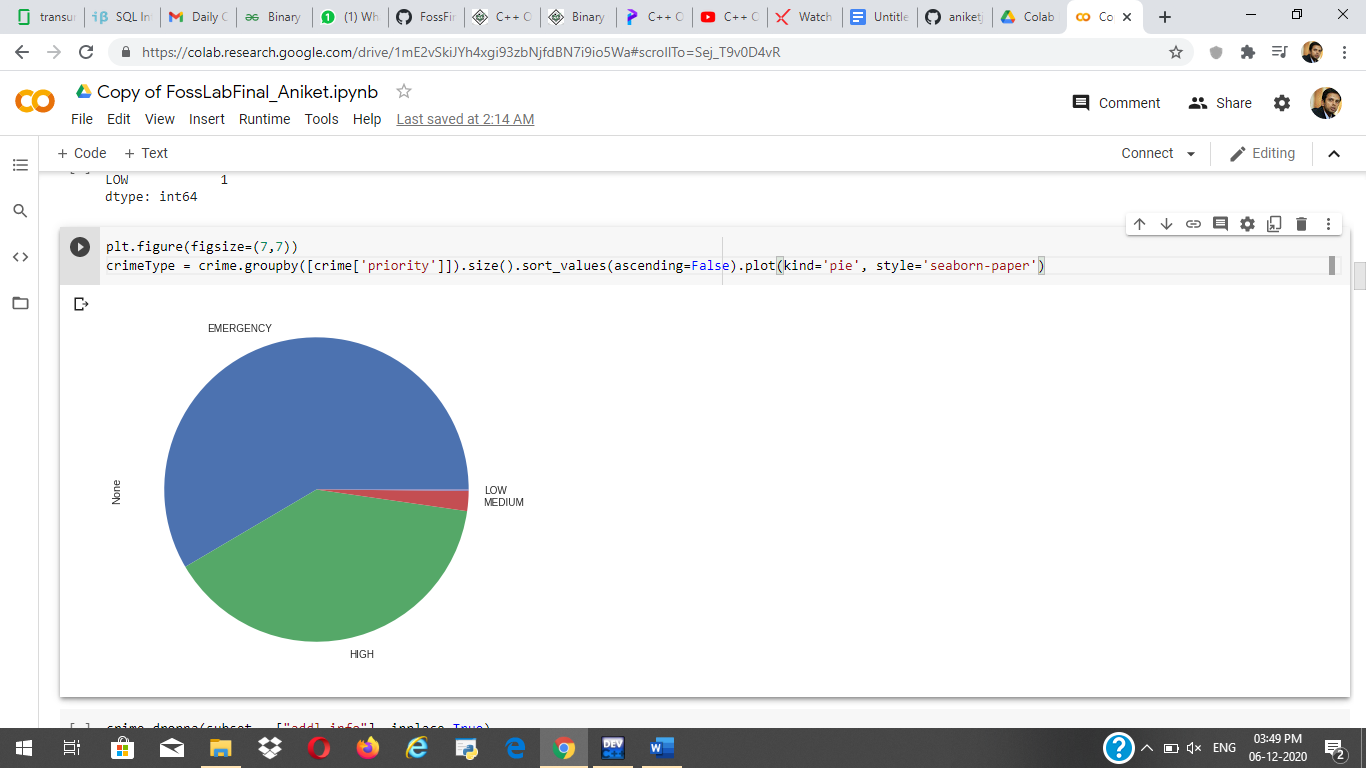
i) Number of crimes in the different districts of Kerala.



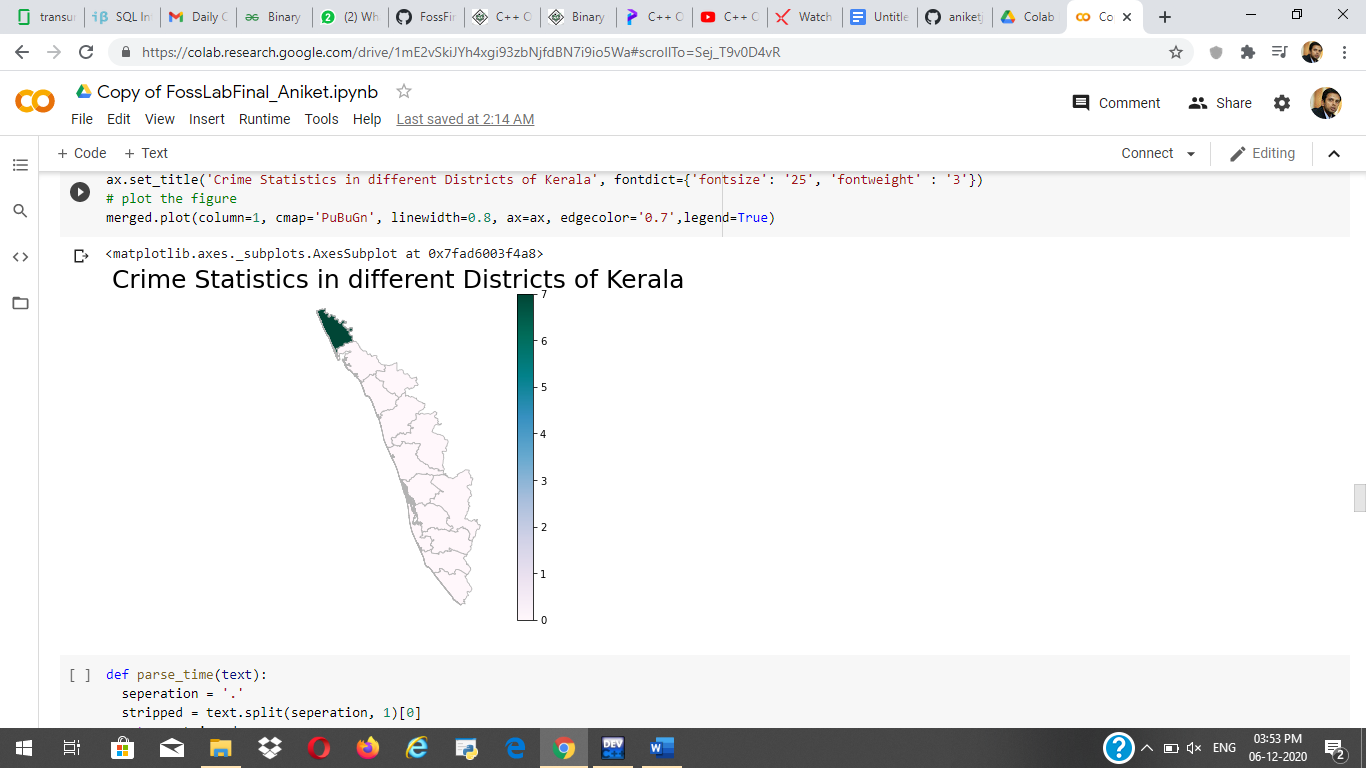
ii) Visualization on the basis of crime:



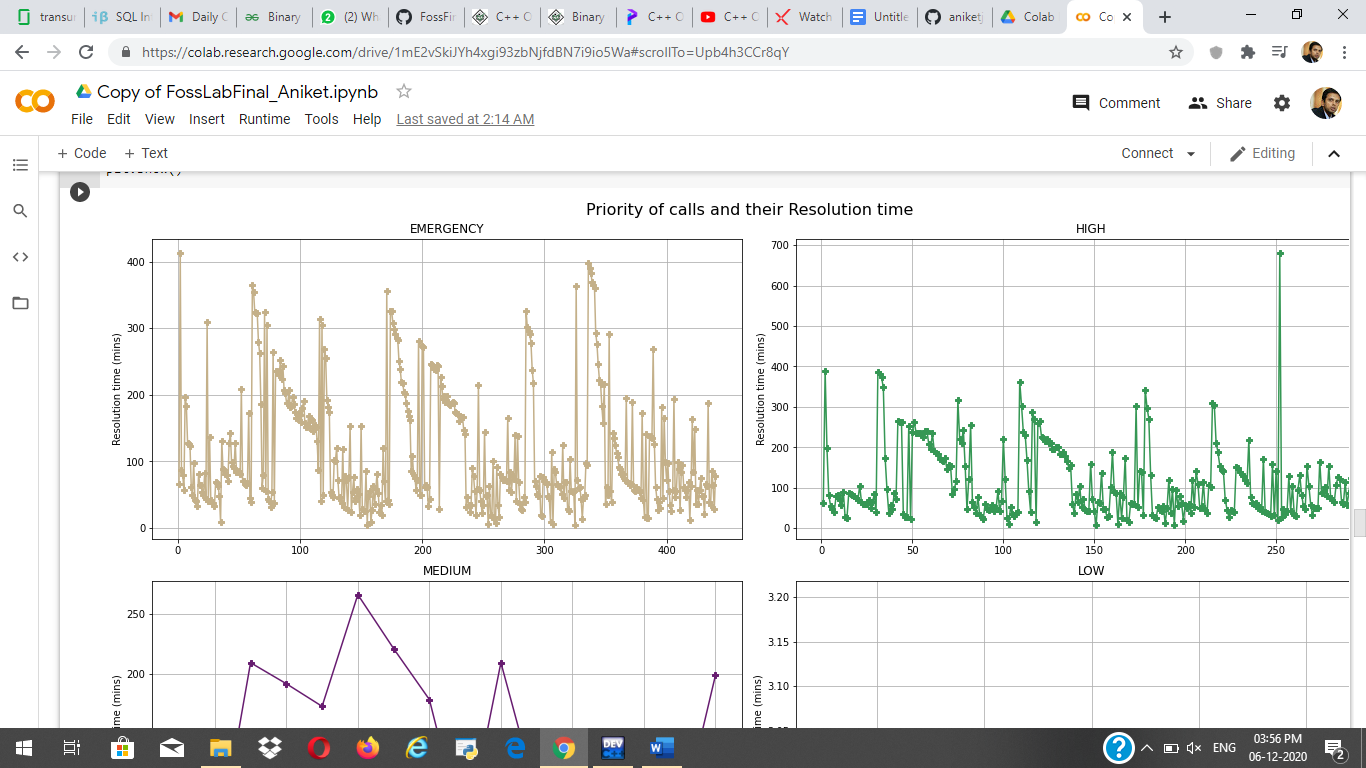
iii) Visualization on the basis of service required:



iii) Crime statistics showed on the map:



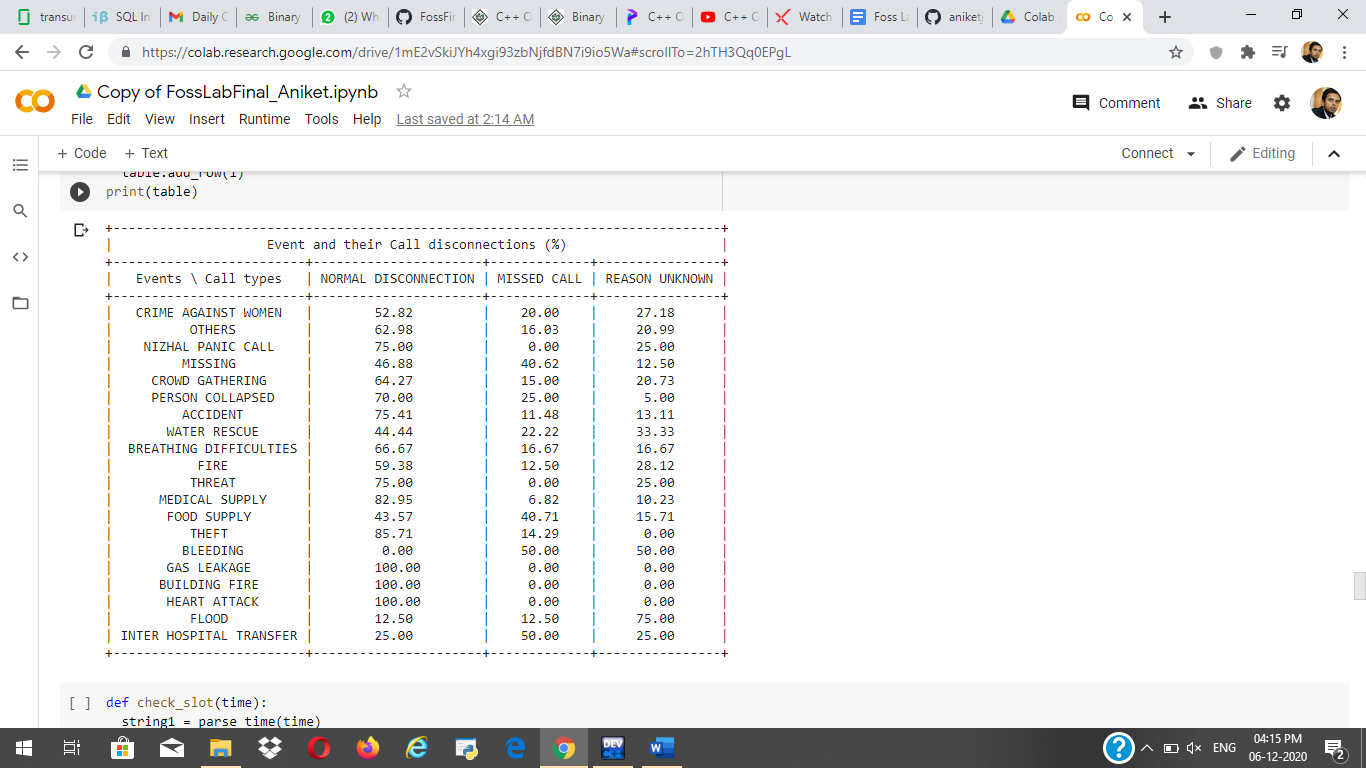
iv) Priority of calls and their Resolution time



v) Cloud view

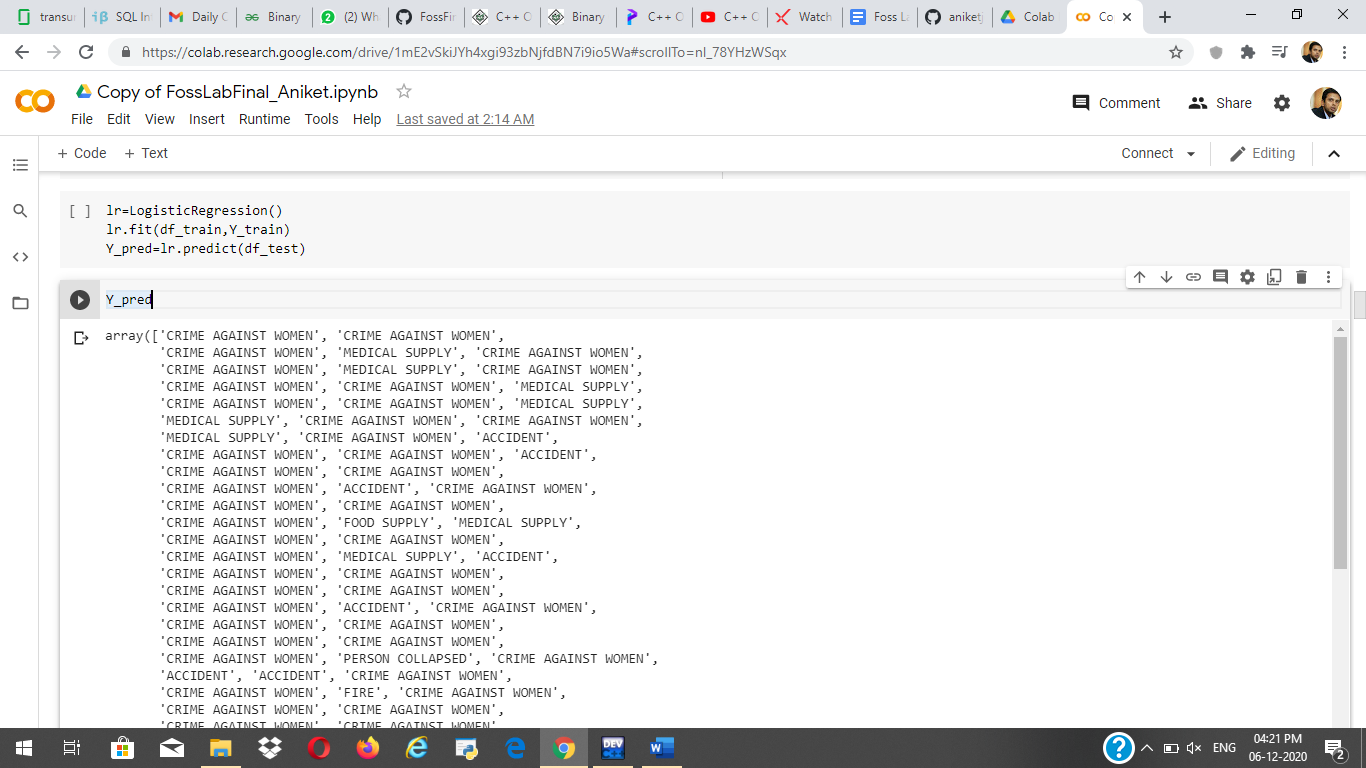


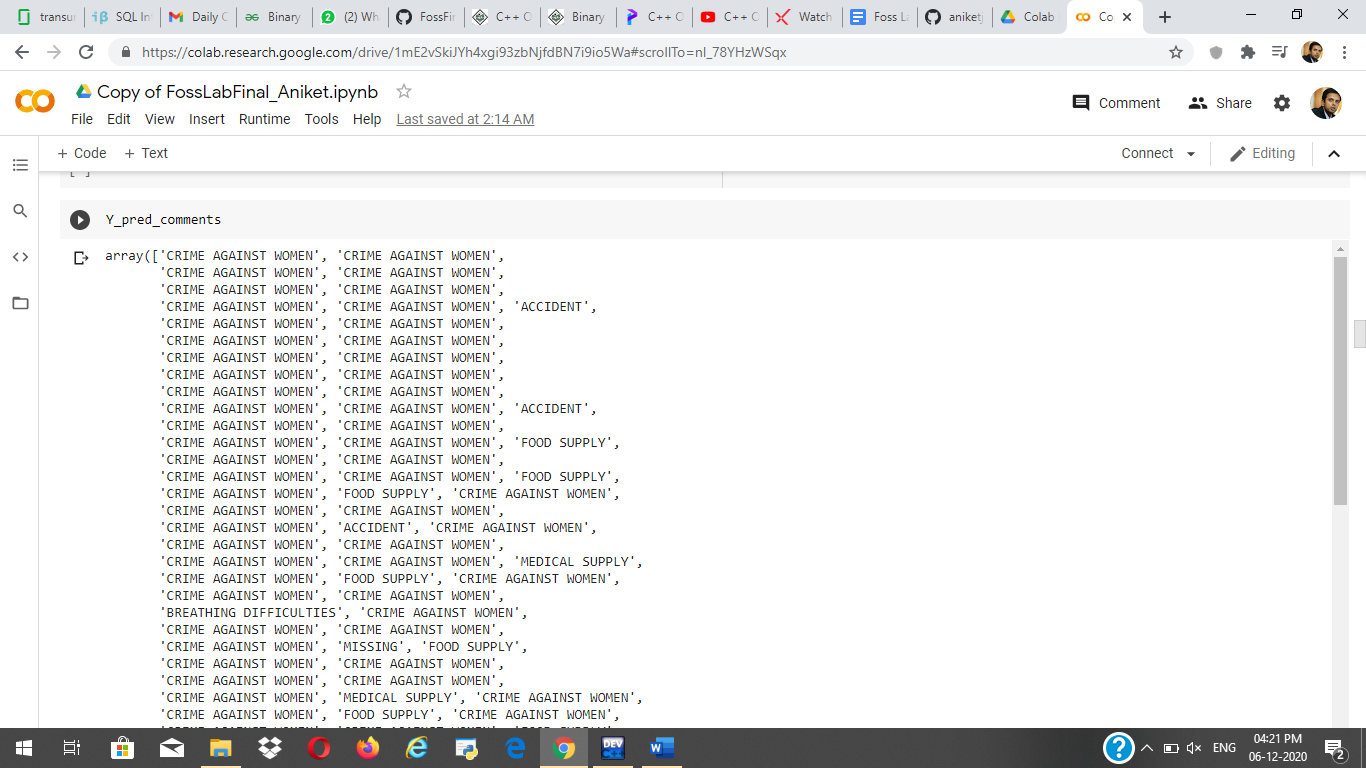
vi) Table view:

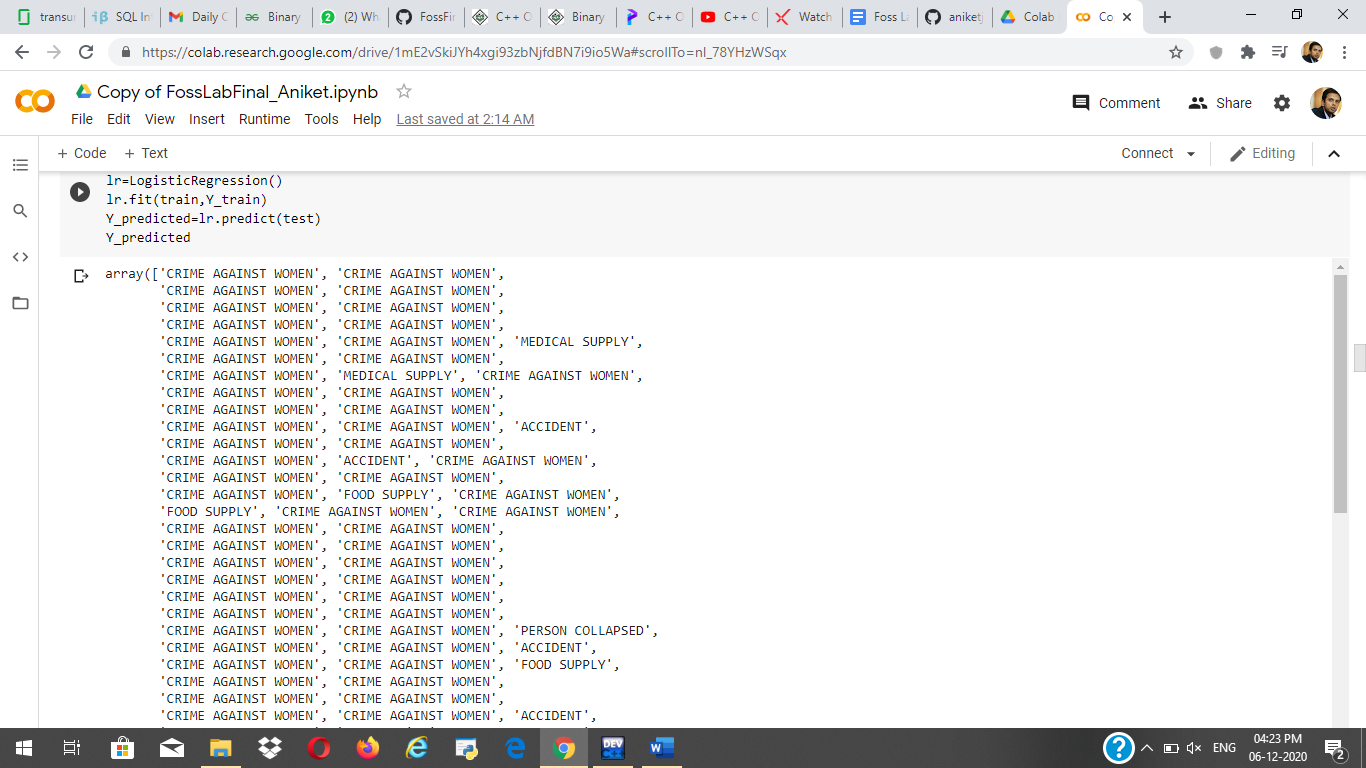


1. Prediction:

i) Logistic regression:







Printing the elements and their frequencies:



Final code and accuracy:

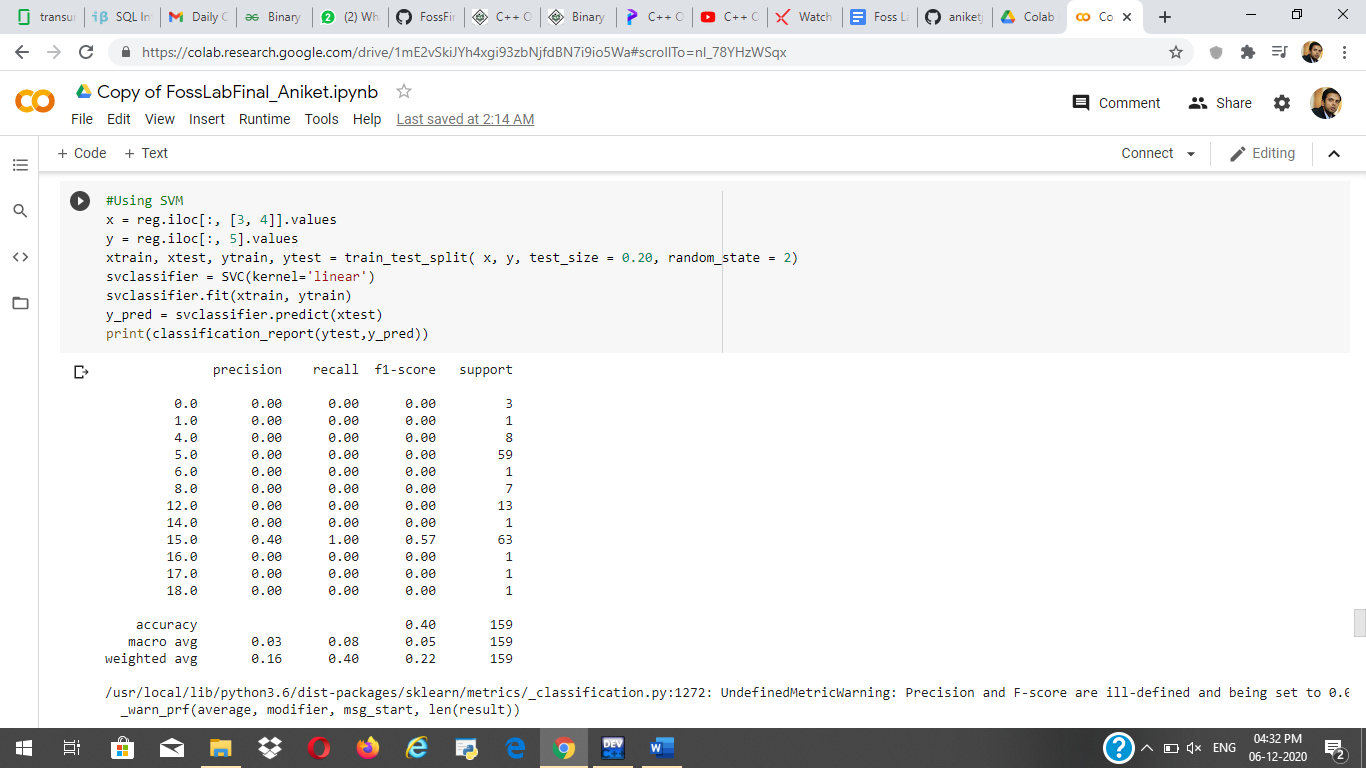
Accuracy of lbfgs logistics regression model: 41.72%

Accuracy of saga logistics regression model: 41.72%

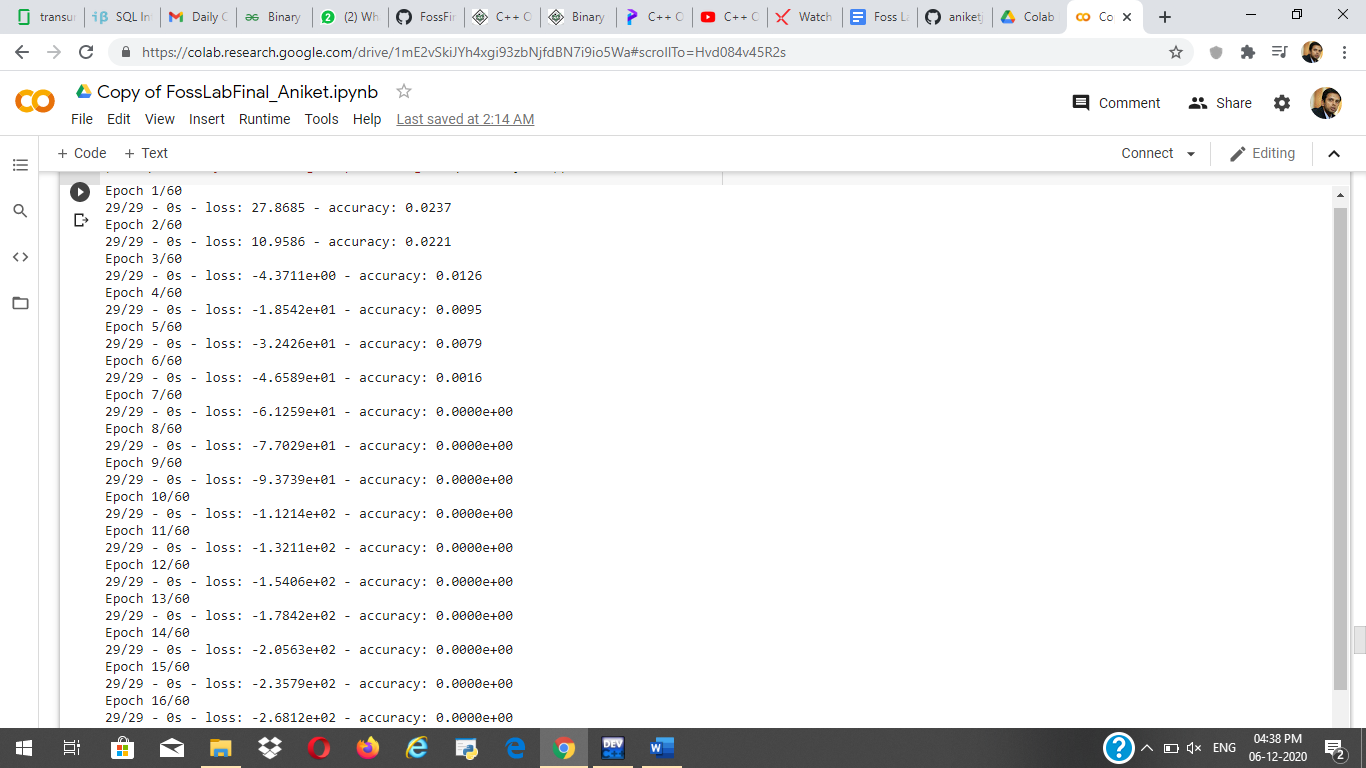


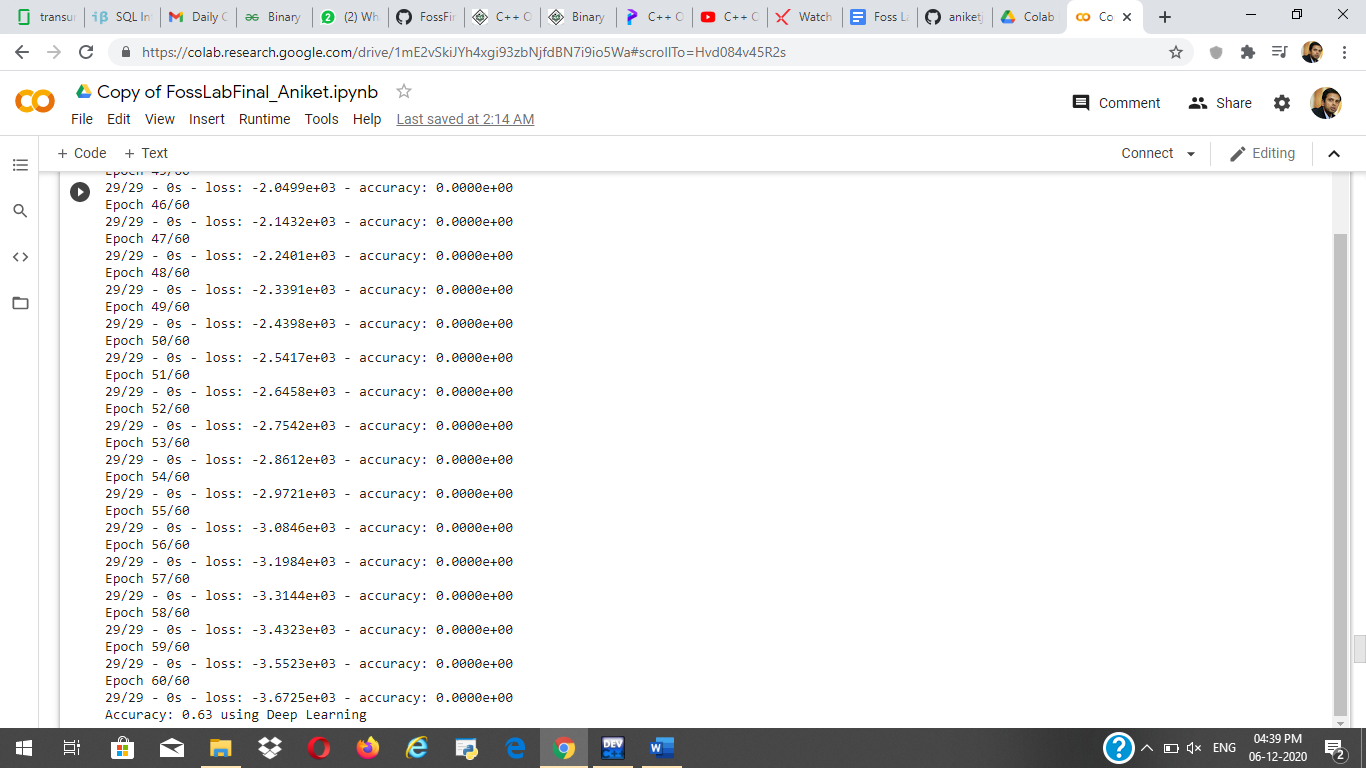
ii) Using SVM :

Accuracy of the SVM model: 40%



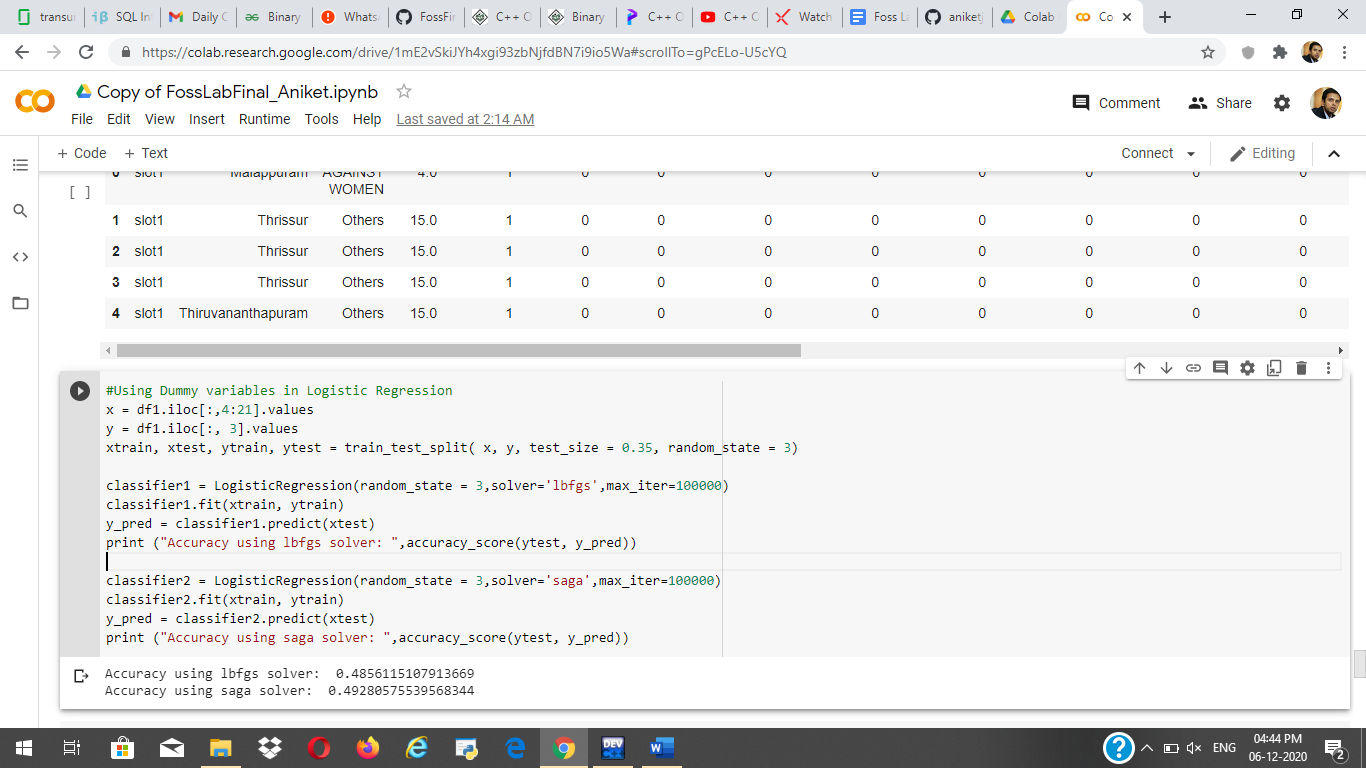
iii) **Using Deep Learning for Classification:**

****

****

**Accuracy by using deep learning classification: 63%**

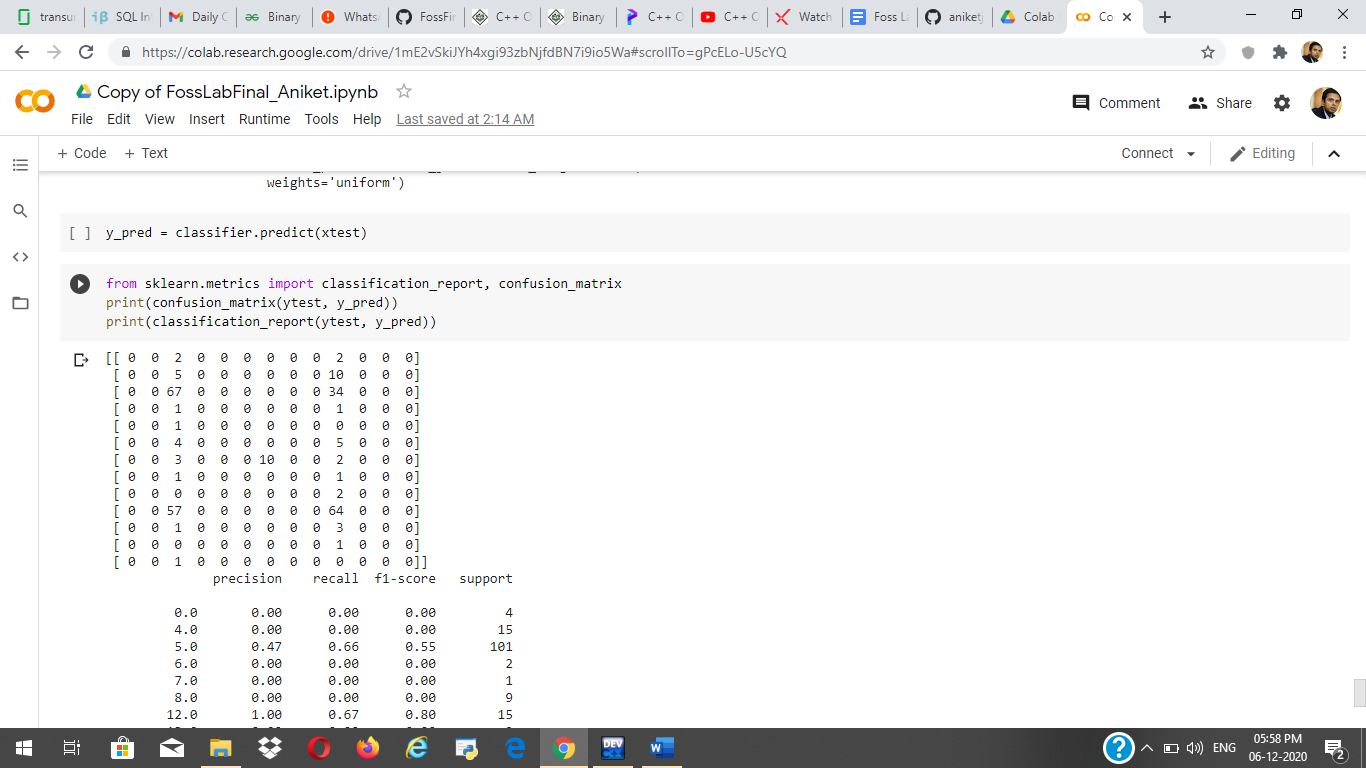
**iv) Using dummy variables for logistic regression:**

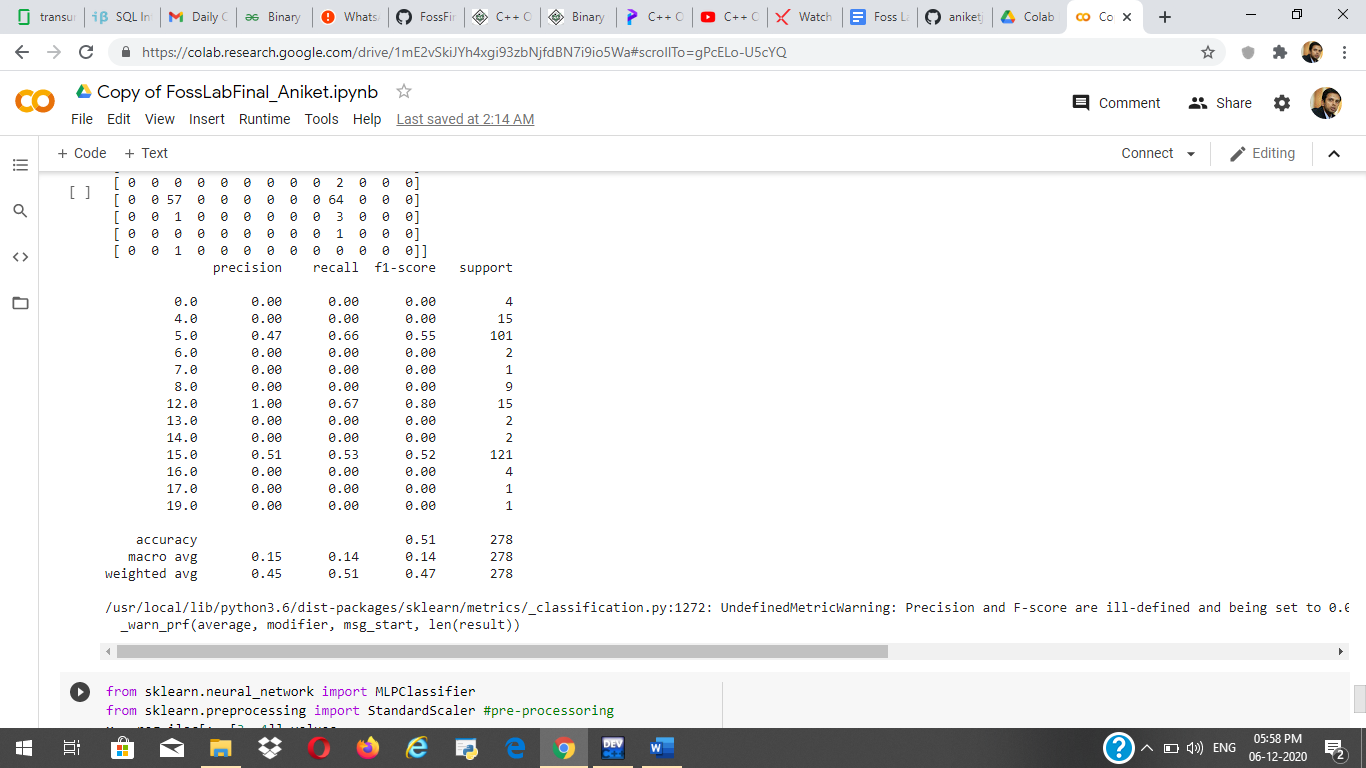


Accuracy using lbfgs solver: 48.5%

Accuracy using saga solver: 49.2%

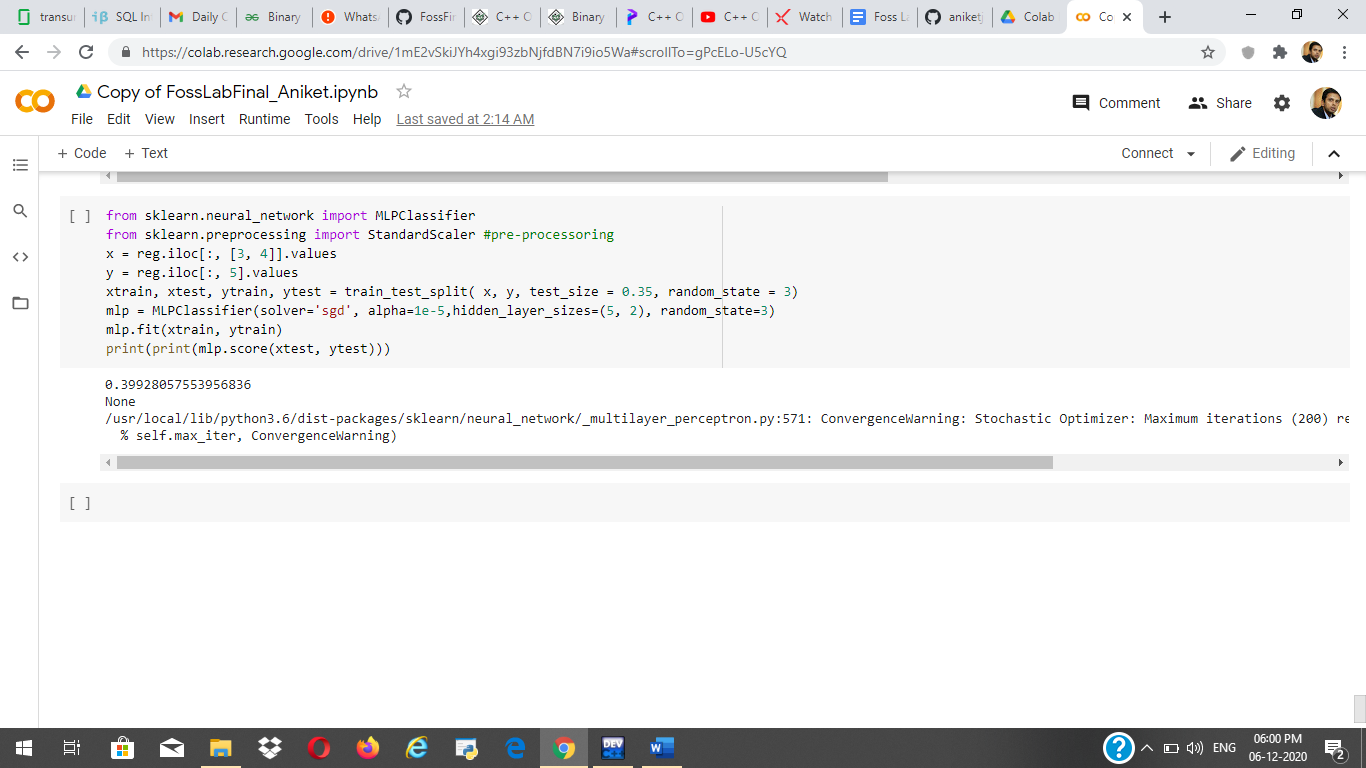
v) K-nearest neighbors:





Accuray by using random forest method is: 51%

vi) MLP classifier:



Accuracy by using MLP classifier: 39.99%

**Conclusion**

|  |  |
| --- | --- |
| **Model** | **Accuracy (%)** |
| **Deep Learning Classifier** | **63.00** |
| **SVM** | **40.00** |
| **Logistic Regression** | **41.72** |
| **Dummy with Logistic Regression** | **48.50, 49.20** |
| **MLP Classifier** | **39.00** |
| **KNN** | **51.00** |

Deep Learning Classifier model gave the best accuracy i.e. 63%.

No any model is giving decent accuracy because there are lack of good features, data value and there is data underfitting. Models presented here can performed better if the quality of the dataset is improved further.

Code:

*#Libraries and Requirements*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**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

**from** **wordcloud** **import** WordCloud

**import** **matplotlib.pyplot** **as** **plt**

**from** **collections** **import** Counter

**from** **wordcloud** **import** WordCloud

**import** **matplotlib.pyplot** **as** **plt**

**from** **collections** **import** Counter

!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')

get\_ipython().run\_line\_magic('matplotlib', 'inline')

plt.style.use('seaborn')

crime\_1 = pd.read\_csv('/content/Crime1.csv',error\_bad\_lines=**False**)

crime\_2 = pd.read\_csv('/content/Crime2.csv',error\_bad\_lines=**False**)

crime\_3 = pd.read\_csv('/content/Crime3.csv',error\_bad\_lines=**False**)

crime=pd.concat([crime\_1,crime\_2,crime\_3],ignore\_index=**True**)

crime.dropna(subset = ["event\_id"], inplace=**True**)

crime.drop(crime[crime['district\_code'] == 'district\_code'].index, inplace = **True**)

crime.head()

crime.shape

crime.groupby([crime['district\_code']]).size().sort\_values(ascending=**False**)

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

crime.groupby([crime['district\_code']]).size().sort\_values(ascending=**True**).plot(kind='pie', style='dark')

plt.title('Number of crimes at different district')

plt.ylabel('District')

plt.xlabel('Number of crimes')

plt.show()

crime.groupby([crime['event\_maintype']]).size().sort\_values(ascending=**True**)

plt.figure(figsize=(5,5))

plt.rcParams['ytick.labelsize'] = 10

plt.rcParams['xtick.labelsize'] = 10

plt.ylabel('Frequency', fontsize=10)

plt.xlabel('Event Type', fontsize=10)

crimeType = crime.groupby([crime['event\_maintype']]).size().sort\_values(ascending=**False**).plot(kind='barh', style='seaborn-paper')

crime.groupby([crime['priority']]).size().sort\_values(ascending=**False**)

plt.figure(figsize=(7,7))

crimeType = crime.groupby([crime['priority']]).size().sort\_values(ascending=**False**).plot(kind='pie', style='seaborn-paper')

crime.dropna(subset = ["addl\_info"], inplace=**True**)

crime.dropna(subset = ["event\_maintype"], inplace=**True**)

crime.isna().sum()

crime\_temp=crime.copy()

X = crime[crime['event\_maintype'] != "OTHERS"]

X\_Others = crime[crime['event\_maintype'] == "OTHERS"]

**from** **sklearn.feature\_extraction.text** **import** CountVectorizer

**from** **sklearn.model\_selection** **import** cross\_val\_score, cross\_val\_predict

**from** **sklearn** **import** metrics

**from** **sklearn** **import** linear\_model

**from** **sklearn.linear\_model** **import** LogisticRegression

X\_train=X.addl\_info

Y\_train=X.event\_maintype

X\_test=X\_Others.addl\_info

cvec=CountVectorizer()

cvec= CountVectorizer(stop\_words='english').fit(X\_train)

df\_train =pd.DataFrame(cvec.transform(X\_train).todense(),columns=cvec.get\_feature\_names())

df\_test =pd.DataFrame(cvec.transform(X\_test).todense(),columns=cvec.get\_feature\_names())

lr=LogisticRegression()

lr.fit(df\_train,Y\_train)

Y\_pred=lr.predict(df\_test)

Y\_pred

X\_train\_comments=X.closure\_comments

Y\_train\_comments=X.event\_maintype

X\_test\_comments=X\_Others.closure\_comments

cvec= CountVectorizer(stop\_words='english').fit(X\_train\_comments)

df\_train\_comments =pd.DataFrame(cvec.transform(X\_train\_comments).todense(),columns=cvec.get\_feature\_names())

df\_test\_comments =pd.DataFrame(cvec.transform(X\_test\_comments).todense(),columns=cvec.get\_feature\_names())

lr=LogisticRegression()

lr.fit(df\_train\_comments,Y\_train\_comments)

Y\_pred\_comments=lr.predict(df\_test\_comments)

Y\_pred\_comments

train=pd.concat([df\_train,df\_train\_comments],axis=1)

test=pd.concat([df\_test,df\_test\_comments],axis=1)

lr=LogisticRegression()

lr.fit(train,Y\_train)

Y\_predicted=lr.predict(test)

Y\_predicted

**import** **collections**

elements\_count = collections.Counter(Y\_predicted)

*# printing the element and the frequency*

**for** key, value **in** elements\_count.items():

print(f"**{**key**}**: **{**value**}**")

Event\_type = elements\_count.keys()

Frequency = elements\_count.values()

plt.title("Predicted Event Type",fontsize=15)

plt.xlabel("Event Type",fontsize=15)

plt.xticks(rotation=90,fontsize=15)

plt.ylabel("Frequency",fontsize=30)

plt.yticks(fontsize=15)

plt.bar(Event\_type,Frequency)

crime\_temp=crime.copy()

**import** **nltk**

nltk.download('stopwords')

**from** **nltk.corpus** **import** stopwords

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

**from** **collections** **import** defaultdict

**import** **seaborn** **as** **sns**

**from** **collections** **import** Counter

**from** **nltk.stem** **import** PorterStemmer

**import** **nltk**

**from** **nltk.corpus** **import** wordnet **as** wn

**from** **nltk.stem.wordnet** **import** WordNetLemmatizer

**from** **nltk.corpus** **import** stopwords

**from** **nltk.tokenize** **import** word\_tokenize

**from** **wordcloud** **import** WordCloud, STOPWORDS

**import** **nltk**

**def** plot\_wordcloud(text):

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

**def** \_preprocess\_text(text):

corpus=[]

stem=PorterStemmer()

lem=WordNetLemmatizer()

**for** news **in** text:

words=[w **for** w **in** word\_tokenize(news) **if** (w **not** **in** stop)]

words=[lem.lemmatize(w) **for** w **in** words **if** len(w)>2]

corpus.append(words)

**return** corpus

corpus=\_preprocess\_text(text)

wordcloud = WordCloud(

background\_color='yellow',

stopwords=set(STOPWORDS),

max\_words=80,

max\_font\_size=25,

scale=3,

random\_state=2)

wordcloud=wordcloud.generate(str(corpus))

fig = plt.figure(1, figsize=(12, 12))

plt.axis('off')

plt.imshow(wordcloud)

plt.show()

plot\_wordcloud(crime['addl\_info'])

**import** **seaborn** **as** **sns**

**from** **collections** **import** Counter

**def** plot\_top\_non\_stopwords\_barchart(text):

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

new= text.str.split()

new=new.values.tolist()

corpus=[word **for** i **in** new **for** word **in** i]

counter=Counter(corpus)

most=counter.most\_common()

x, y=[], []

**for** word,count **in** most[:40]:

**if** (word **not** **in** stop):

x.append(word)

y.append(count)

sns.barplot(x=y,y=x)

plot\_top\_non\_stopwords\_barchart(crime['addl\_info'])

district\_wise=crime.groupby('district\_code')[['district\_code']].count()

district\_wise.columns.values[0] = 'freq'

district\_wise

**import** **seaborn** **as** **sns**

**from** **sklearn.metrics** **import** silhouette\_score,silhouette\_samples

**from** **sklearn.cluster** **import** KMeans

**from** **sklearn.preprocessing** **import** StandardScaler

std=StandardScaler()

loc=district\_wise

loc=std.fit\_transform(loc)

wcss=[]

sil=[]

**for** i **in** range(2,11):

clf=KMeans(n\_clusters=i,init='k-means++',random\_state=42)

clf.fit(loc)

labels=clf.labels\_

centroids=clf.cluster\_centers\_

sil.append(silhouette\_score(loc, labels, metric='euclidean'))

wcss.append(clf.inertia\_)

clf\_final=KMeans(n\_clusters=6,init='k-means++',random\_state=50)

clf\_final.fit(loc)

district\_wise["Clusters"]=clf\_final.predict(loc)

cluster\_summary=pd.concat([district\_wise[district\_wise["Clusters"]==1],district\_wise[district\_wise["Clusters"]==2],district\_wise[district\_wise["Clusters"]==3],district\_wise[district\_wise["Clusters"]==4],district\_wise[district\_wise["Clusters"]==0].head(15)])

cluster\_summary.style.background\_gradient(cmap='BuGn')

*#Functions to perform Transformation for data Pre-Processing*

**def** function1(lis):

list1 = []

**for** i **in** lis:

**if** i[0]!="event\_id":

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

**continue**

**else**:

list1.append(i)

list2 = function2(list1) *# send list to combine signal and disconnect rows*

**return** list2

**def** function2(ls1):

i=0

list2 = []

**while** i < len(list1):

**if** list1[i][0]!='nan':

nw = list1[i]

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

signal = []

disconnect = []

**while** list1[i][0]=='nan': *# appending all signal and disconnect data as list and appending to*

signal.append(list1[i][1])

disconnect.append(list1[i][2])

i+=1

**if**(i==len(list1)):

**break**

nw.append(signal)

nw.append(disconnect)

list2.append(nw) *#Appendinging df to combined list*

**else**:

i+=1

**return** list2

data = []

**for** i **in** range(1,4):

df1 = pd.read\_excel("/content/Crime Prediction 1.xlsx", header=0,sheet\_name=str(i))

df = df1.iloc[:,0:16]

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

lis = df.values.tolist()

list1 = transform1(lis)

**for** item **in** list1:

data.append(item)

print("total number of Entries after data preprocessing: ",len(data))

count = {}

**for** i **in** data:

**if** i[4] **not** **in** count:

count[i[4]] = 1

**else**:

count[i[4]] +=1

lis = []

**for** i **in** count.keys():

**if**(i=="nan"):

**continue**

t = []

t.append(i)

t.append(count[i])

lis.append(temp)

district\_wise=pd.DataFrame(lis)

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=(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='PuBuGn', linewidth=0.8, ax=ax, edgecolor='0.7',legend=**True**)

**def** parse\_time(text):

seperation = '.'

stripped = text.split(seperation, 1)[0]

**return** stripped

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

string1 = date\_time1.split('.',1)[0]

string2 = date\_time2.split('.',1)[0]

*#print(string1)*

*#print(string2)*

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(":")))

dic = {}

**for** i **in** data:

**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()

*#Using NLTK to find Top Keywords based on caller comments*

string = ""

**for** i **in** data:

**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)

*#Event type and their Disconnect causes(i.e. NORMAL\_DISCONNECTION, MISSED\_CALL, REASON\_UNKNOWN)*

disconnect = {}

*#Populate Disconnect dictionary*

**for** i **in** data:

**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** data:

**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** data:

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 = 0.35, random\_state = 3)

classifier1 = LogisticRegression(random\_state = 3,solver='lbfgs',max\_iter=100000)

classifier1.fit(xtrain, ytrain)

y\_pred = classifier1.predict(xtest)

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

classifier2 = LogisticRegression(random\_state = 3,solver='saga',max\_iter=100000)

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 = 0.20, random\_state = 2)

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 = 0.20, random\_state = 2)

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=60, batch\_size=22, 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 = 0.35, random\_state = 3)

classifier1 = LogisticRegression(random\_state = 3,solver='lbfgs',max\_iter=100000)

classifier1.fit(xtrain, ytrain)

y\_pred = classifier1.predict(xtest)

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

classifier2 = LogisticRegression(random\_state = 3,solver='saga',max\_iter=100000)

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 = 0.35, random\_state = 3)

svclassifier = SVC(kernel='linear')

svclassifier.fit(xtrain, ytrain)

y\_pred = svclassifier.predict(xtest)

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

*#Using K-Nearest neighbor*

**from** **sklearn.metrics** **import** confusion\_matrix

**from** **sklearn.neighbors** **import** KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=7, metric='manhattan')

knn.fit(xtrain, ytrain)

y\_pred = knn.predict(xtest)

**from** **sklearn.neighbors** **import** KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=7)

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))

**from** **sklearn.neural\_network** **import** MLPClassifier

**from** **sklearn.preprocessing** **import** StandardScaler *#pre-processoring*

x = reg.iloc[:, [3, 4]].values

y = reg.iloc[:, 5].values

xtrain, xtest, ytrain, ytest = train\_test\_split( x, y, test\_size = 0.35, random\_state = 3)

mlp = MLPClassifier(solver='sgd', alpha=1e-5,hidden\_layer\_sizes=(5, 2), random\_state=3)

mlp.fit(xtrain, ytrain)

print(print(mlp.score(xtest, ytest)))