

Foss Lab – Final Report

Name : Sameer Kumar Kaushik

Roll No. : 205319012

Dataset:

Kerala State 100 Helpline dataset

Discription :

The dataset consists of over 900 observations with 16 features such as call time, event type and district.

Task:

classify, visualize and analyze the crime data based on the crime sheet.

Methodologies:

1. Word Clouds
2. NLP using NLTK
3. Classification using Logistic Regression
4. SVM
5. Gradient Boosting Classifier
6. Random Forest Classifier
7. Multi Layer Perceptron Classifier
8. Deep Learning

Libraries and Requirements:

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

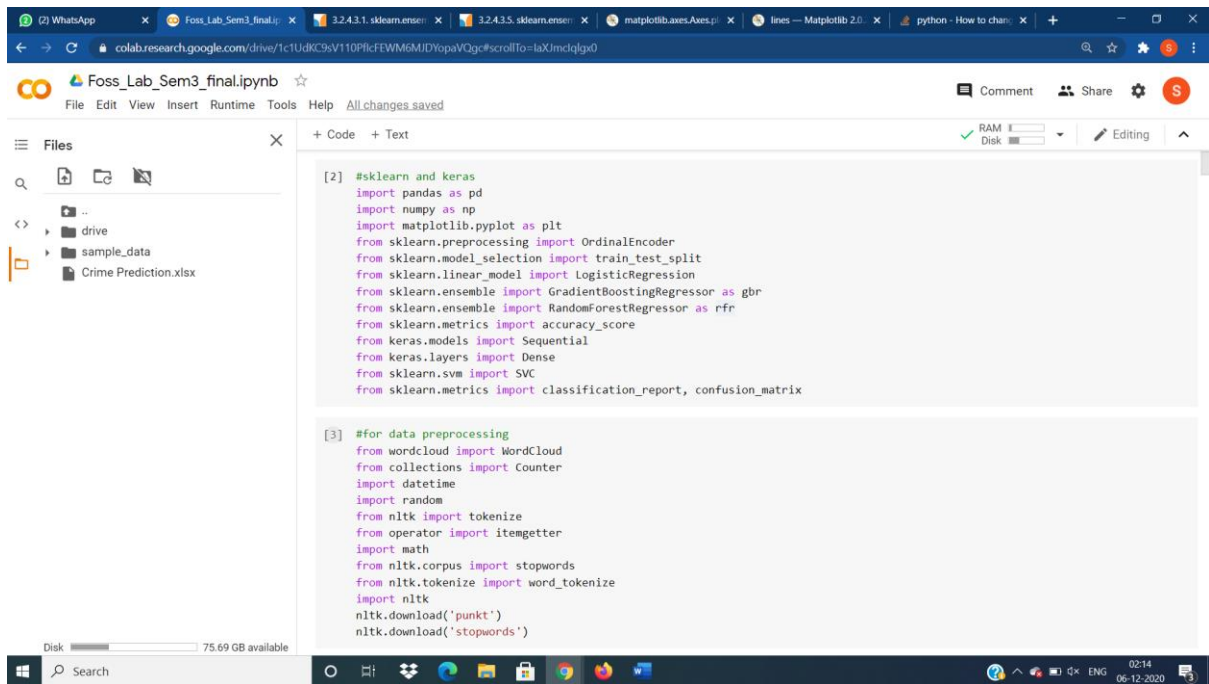
Colab link:

https://colab.research.google.com/drive/1c1UdKC9sV110PfIcFEWM6MJDYopaVQgc#scrollTo=L-Gf6BPv9_yk

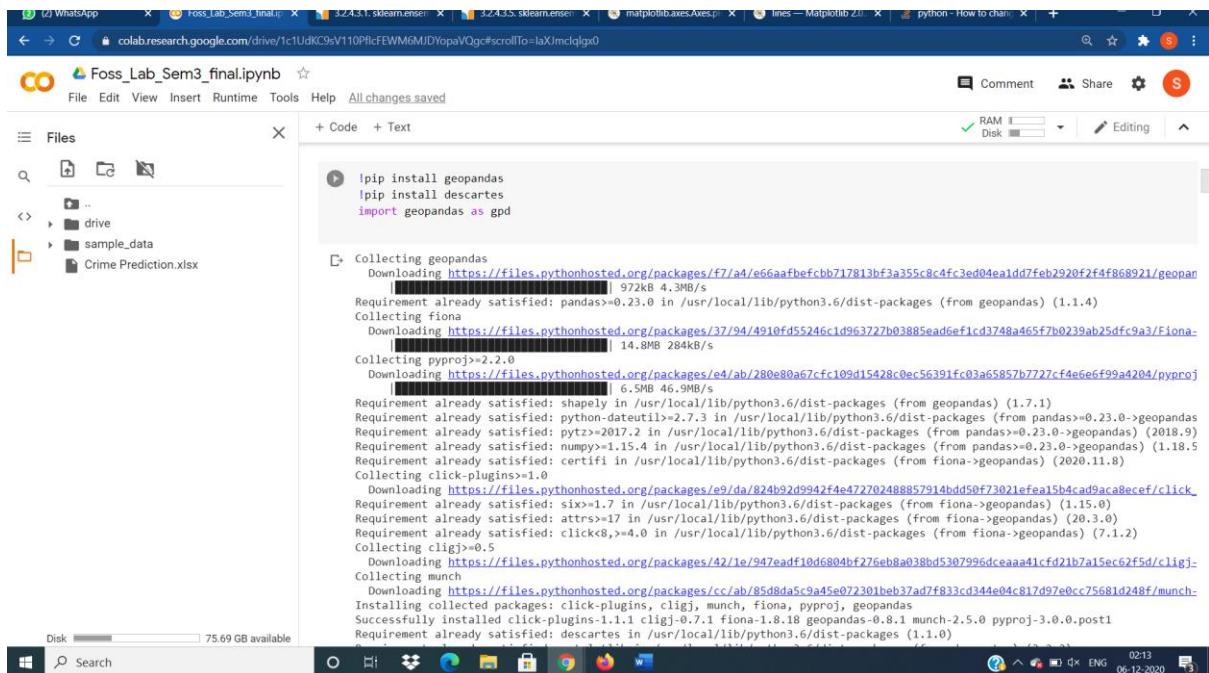
Git Hub Link:

https://github.com/sameerkaushik007/Foss_Lab_Final

Import different type of libraries



Install geopandas library for geo map of Kerala region

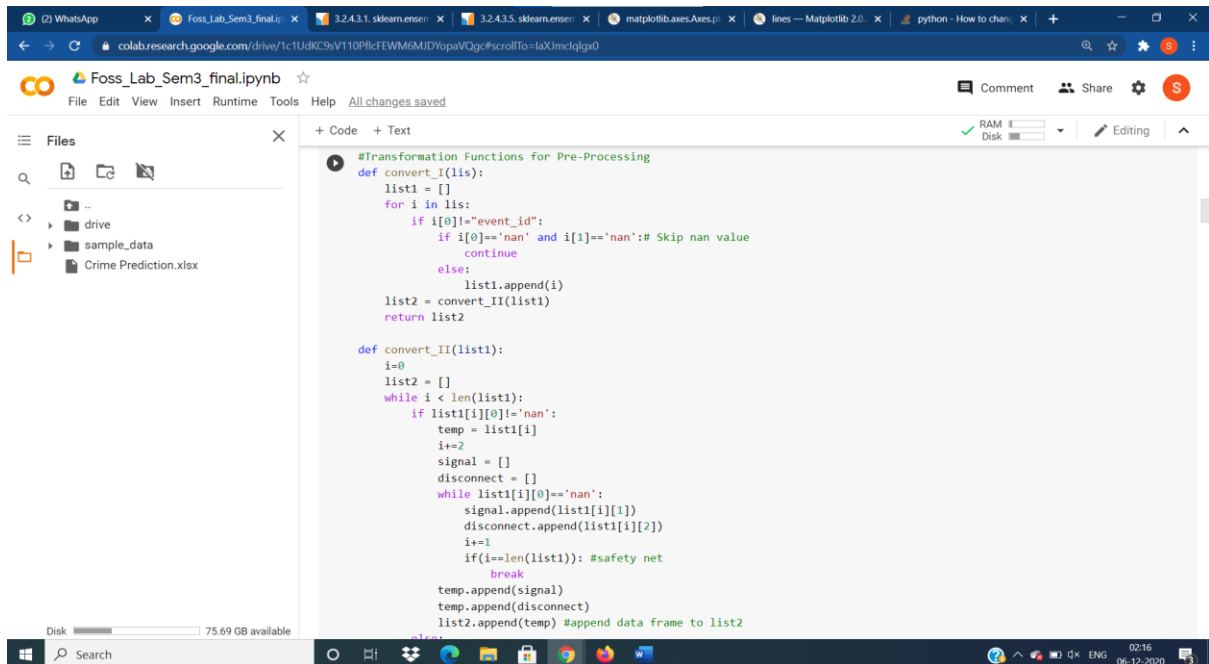


Pre-Processing

There is multiple noise data and multiple duplicate row value so. We try to clean up irrelevant data using pre-processing technique.

Create transformation function for pre-processing using convert_I and convert_II function

Avoid the nun value

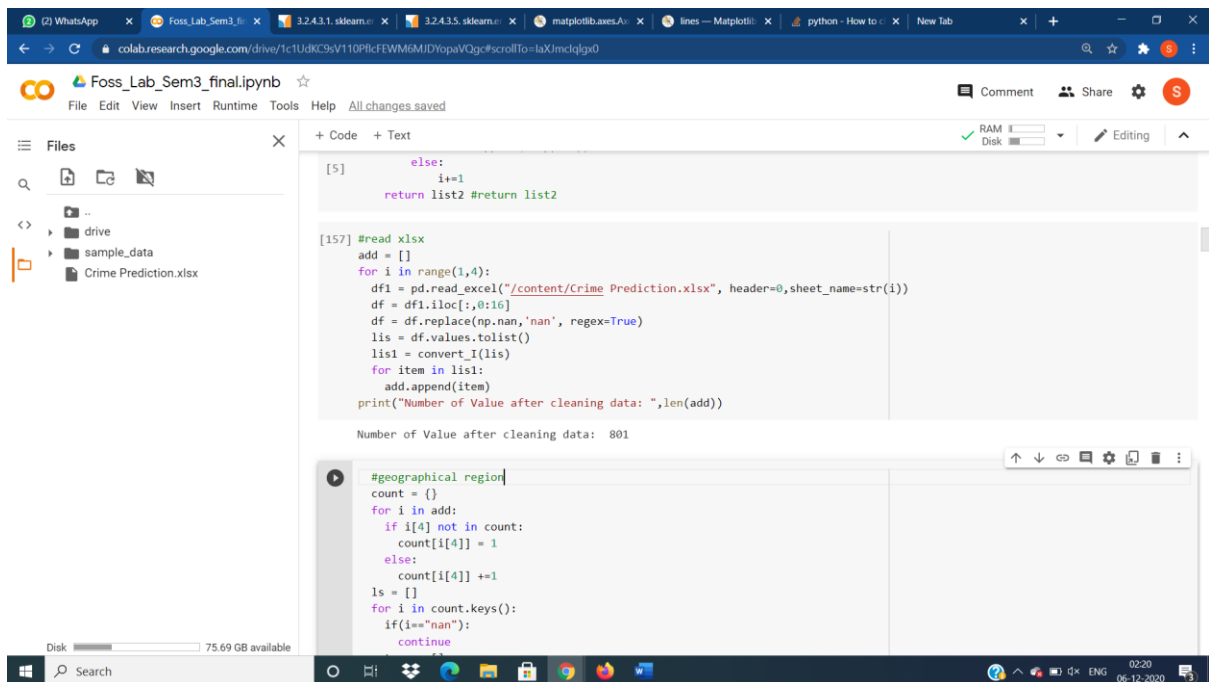


The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'sample_data' containing a file 'Crime Prediction.xlsx'. The code editor contains the following Python code:

```
#Transformation Functions for Pre-Processing
def convert_I(lis):
    list1 = []
    for i in lis:
        if i[0]!="event_id":
            if i[0]=="nan" and i[1]=="nan":# Skip nan value
                continue
            else:
                list1.append(i)
    list2 = convert_II(list1)
    return list2

def convert_II(list1):
    i=0
    list2 = []
    while i < len(list1):
        if list1[i][0]!="nan":
            temp = list1[i]
            i+=2
            signal = []
            disconnect = []
            while list1[i][0]=="nan":
                signal.append(list1[i][1])
                disconnect.append(list1[i][2])
                i+=1
            if(i==len(list1)): #safety net
                break
            temp.append(signal)
            temp.append(disconnect)
            list2.append(temp) #append data frame to list2
```

read crime prediction xl file and in xl sheet there is 16 column



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'sample_data' containing a file 'Crime Prediction.xlsx'. The code editor contains the following Python code:

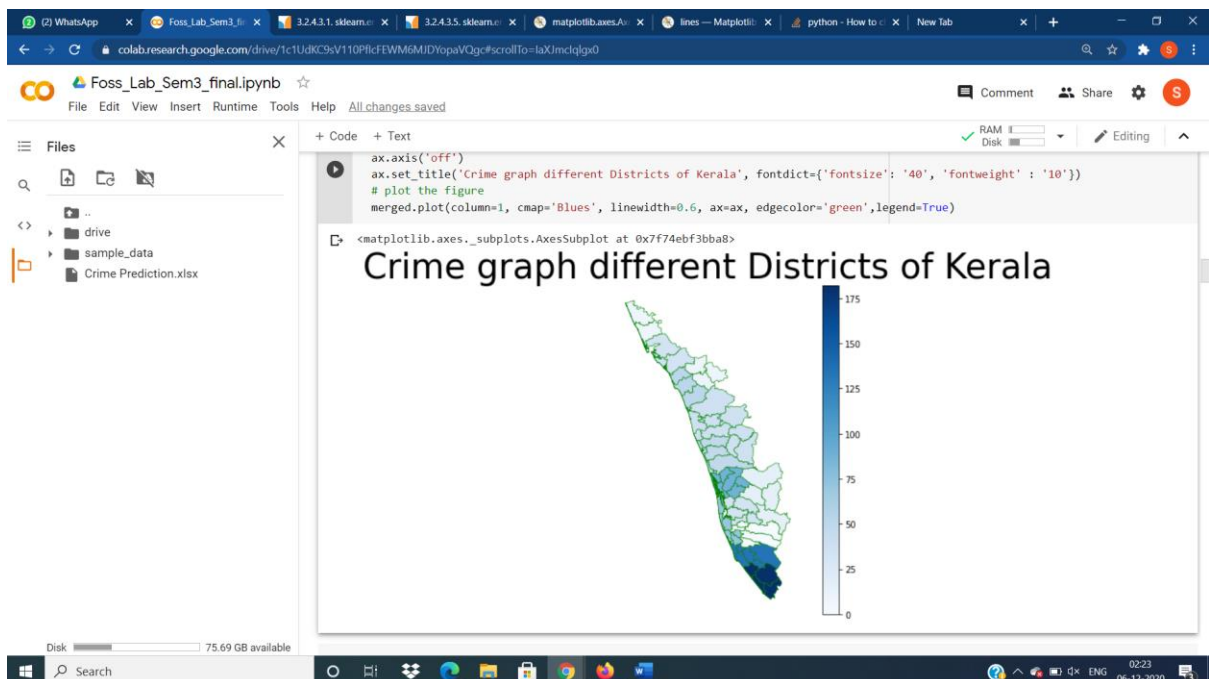
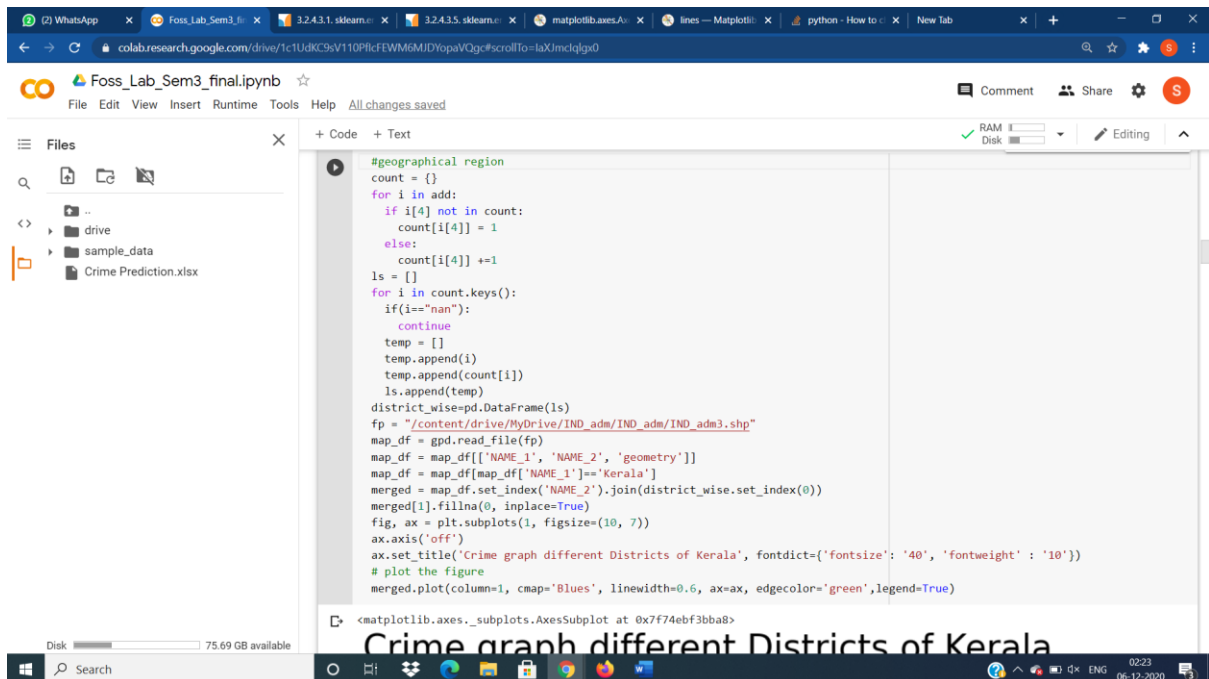
```
[5]
    else:
        i+=1
    return list2 #return list2

[157] #read xlsx
add = []
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)
    lis = df.values.tolist()
    lis1 = convert_I(lis)
    for item in lis1:
        add.append(item)
print("Number of Value after cleaning data: ",len(add))

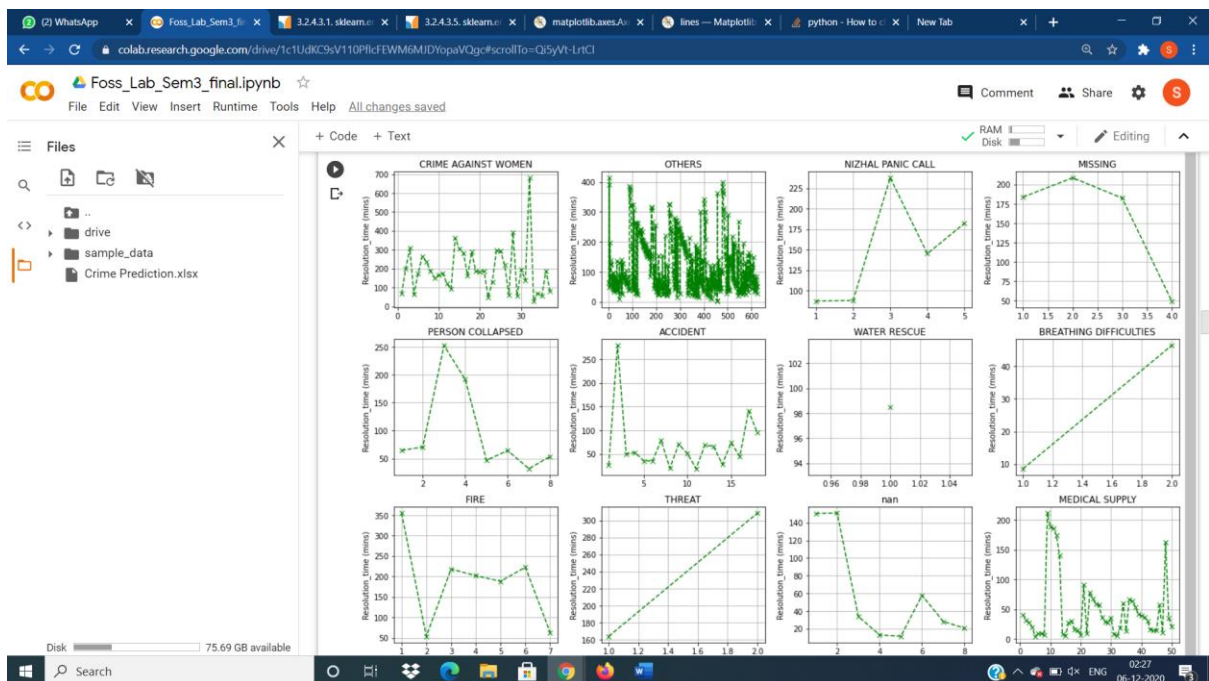
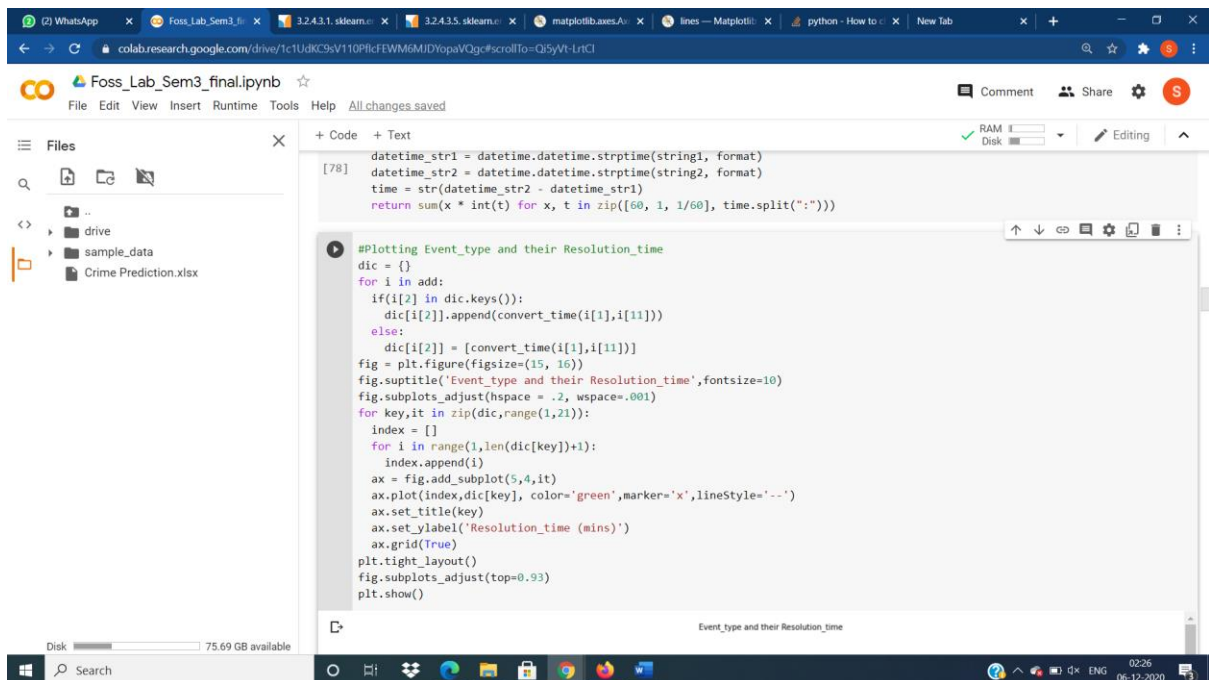
Number of Value after cleaning data: 801

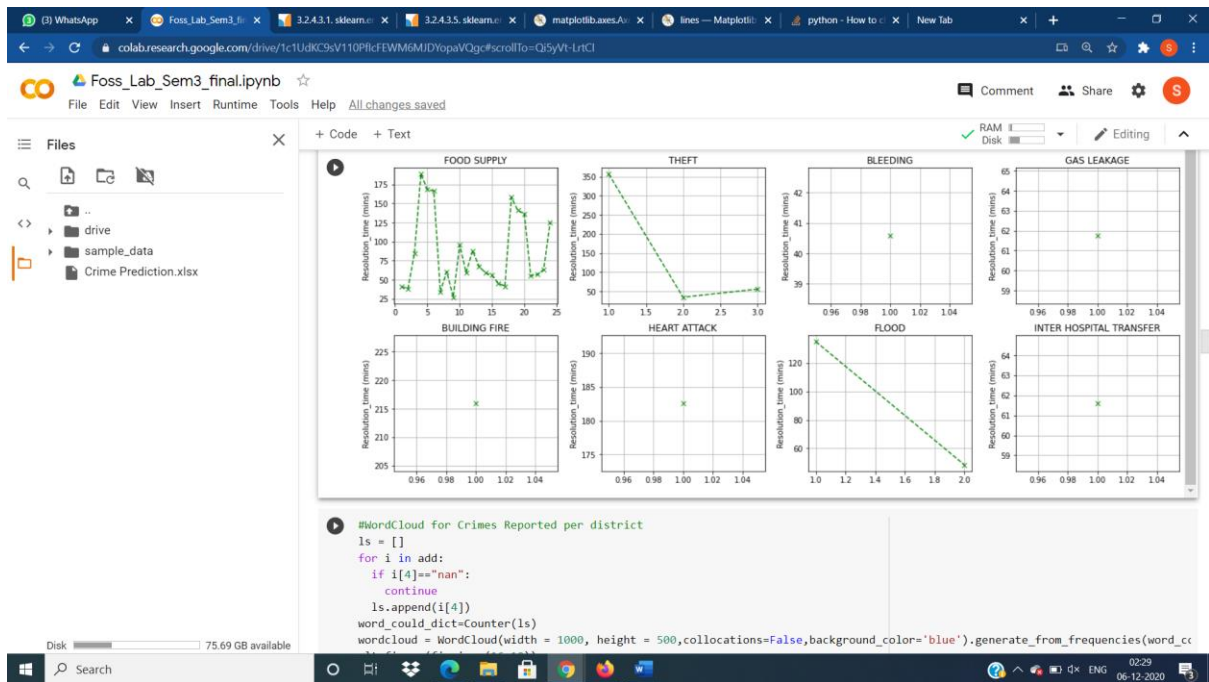
#geographical region
count = {}
for i in add:
    if i[4] not in count:
        count[i[4]] = 1
    else:
        count[i[4]] +=1
ls = []
for i in count.keys():
    if(i!="nan"):
        continue
```

Create geographical region of Kerala district with crime rate

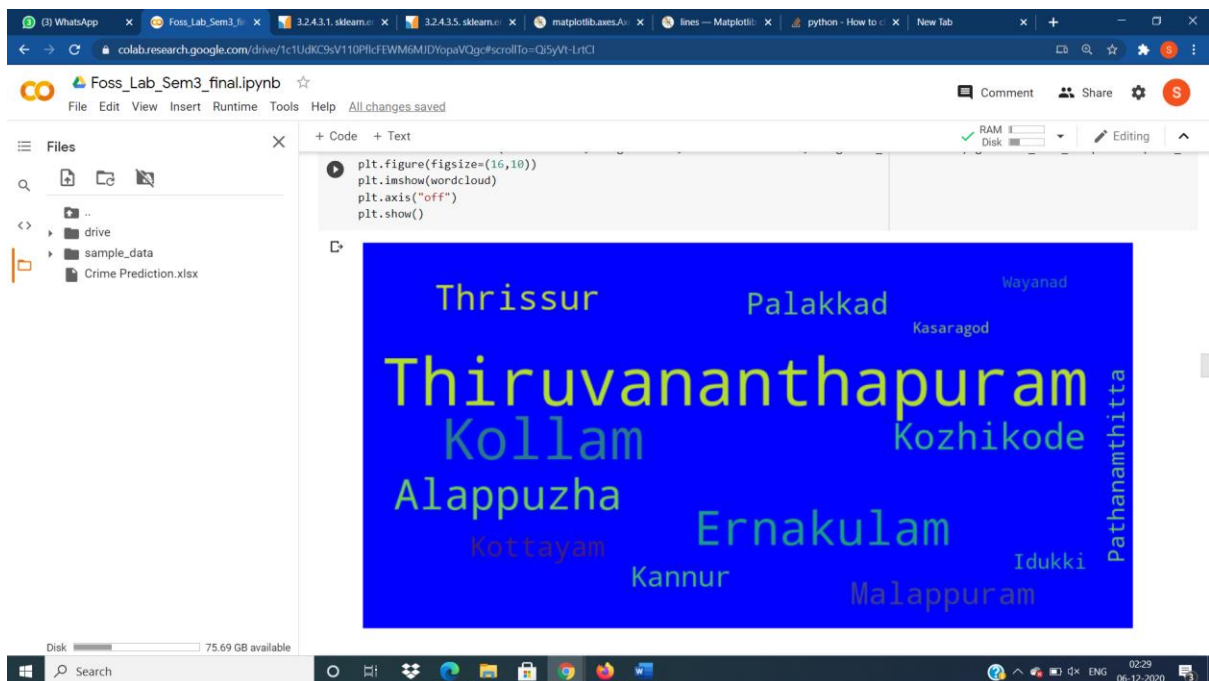


Plot event Vs Time graph

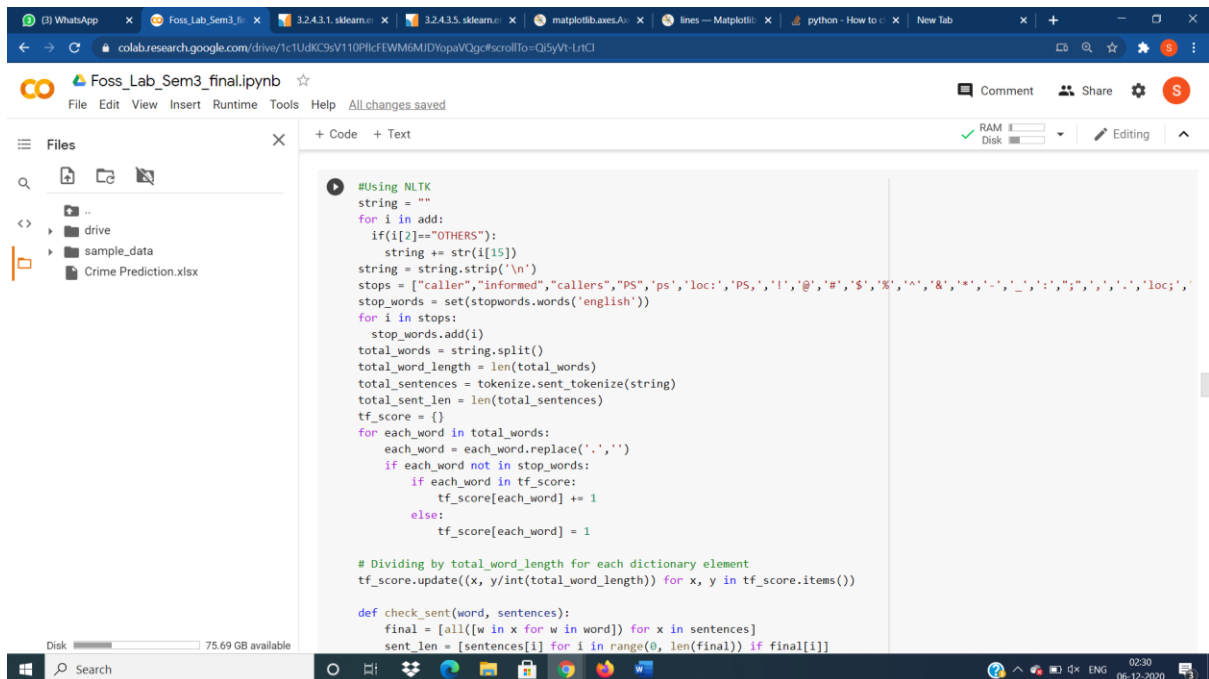




Crime rate in Different District in Word Cloud



Using NLTK we try to remove stop word



The screenshot shows a Google Colab notebook titled 'Foss_Lab_Sem3_final.ipynb'. The left sidebar displays a file explorer with folders 'drive' and 'sample_data', and a file 'Crime Prediction.xlsx'. The main code area contains the following Python code:

```
#Using NLTK
string = ""
for i in add:
    if(i[2]=="OTHERS"):
        string += str(i[15])
string = string.strip('\n')
stops = ["caller", "informed", "callers", "PS", "ps", "loc", "PS", "!", "@", "W", "S", "X", "H", "R", "K", "I", "J", "L", "O", "P", "Q", "U", "V", "Z", "loc", "loc"]
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))

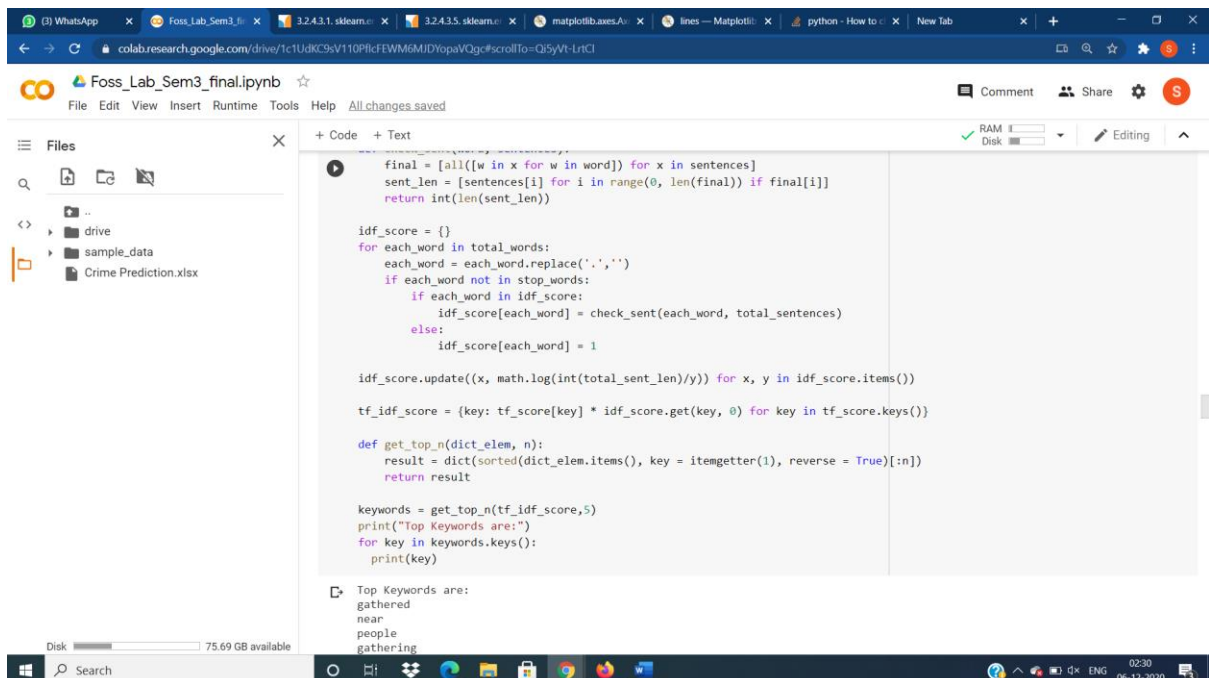
idf_score = {}
for each_word in total_words:
    each_word = each_word.replace('.', '')
    if each_word not in stop_words:
        if each_word in idf_score:
            idf_score[each_word] = check_sent(each_word, total_sentences)
        else:
            idf_score[each_word] = 1

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)
```



The screenshot shows the same Google Colab notebook, but with the final part of the code executed. The output of the script is displayed at the bottom:

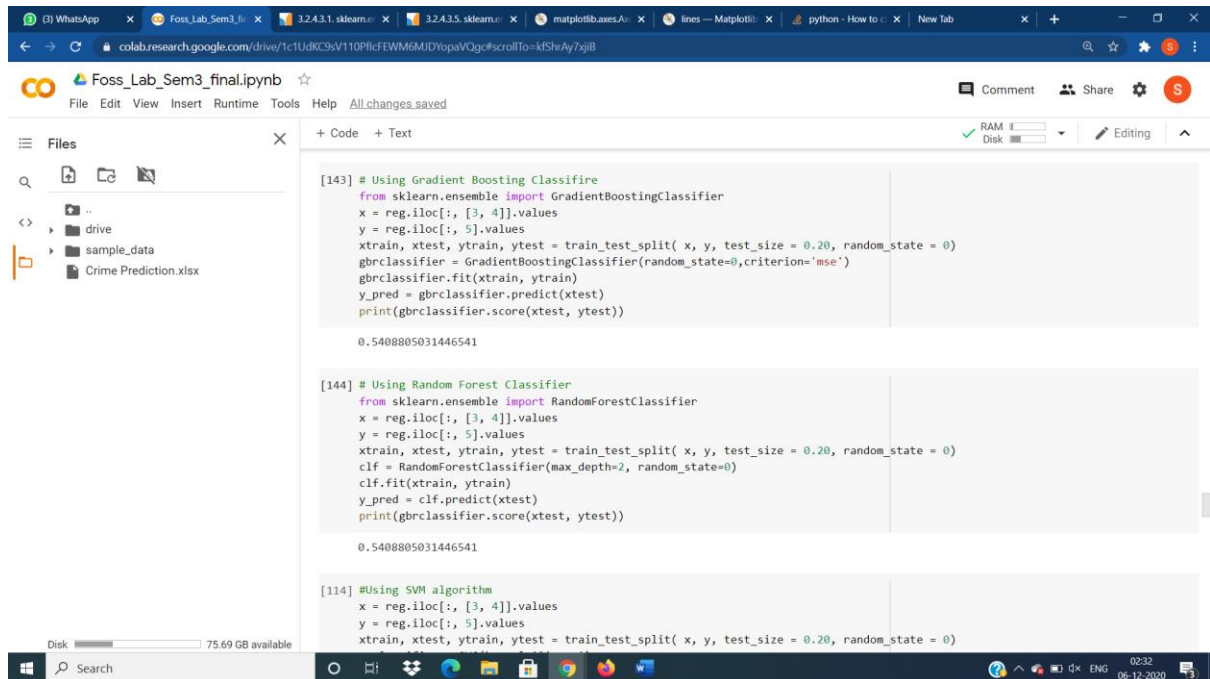
```
Top Keywords are:
gathered
near
people
gathering
```

Word Cloud Result of different event after using NLTK

Data set split into 80 : 20 ratio for training and testing purpose and in gradient boosting classifier parameter Random state 0 and criterion 'mse' we got accuracy 54%

In random forest classifier

Parameter: max_depth=2, random state=0 Accuracy = 54%



The screenshot shows a Jupyter Notebook interface with the following code cells:

```
[143] # Using Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
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)
gbrclassifier = GradientBoostingClassifier(random_state=0, criterion='mse')
gbrclassifier.fit(xtrain, ytrain)
y_pred = gbrclassifier.predict(xtest)
print(gbrclassifier.score(xtest, ytest))

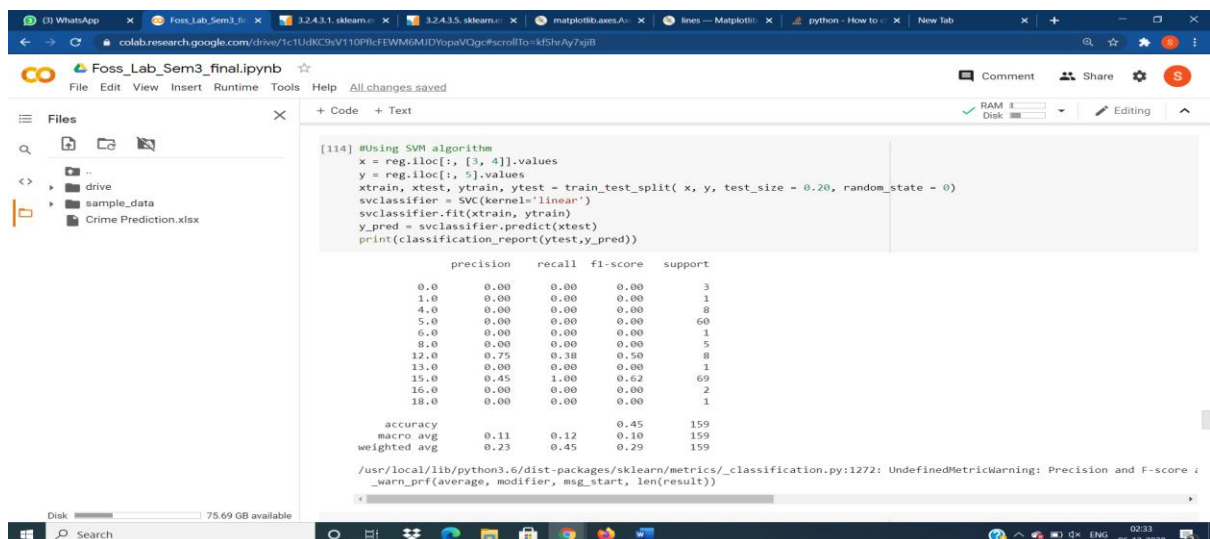
0.5408805031446541
```

```
[144] # Using Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
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)
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(xtrain, ytrain)
y_pred = clf.predict(xtest)
print(gbrclassifier.score(xtest, ytest))

0.5408805031446541
```

```
[114] #Using SVM algorithm
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)
```

Using SVM:



The screenshot shows a Jupyter Notebook interface with the following code cell:

```
[114] #Using SVM algorithm
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)
svclassifier = SVC(kernel='linear')
svclassifier.fit(xtrain, ytrain)
y_pred = svclassifier.predict(xtest)
print(classification_report(ytest, y_pred))
```

The output of the code is a classification report table:

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	3
1.0	0.00	0.00	0.00	1
4.0	0.00	0.00	0.00	8
5.0	0.00	0.00	0.00	60
6.0	0.00	0.00	0.00	1
8.0	0.00	0.00	0.00	5
12.0	0.75	0.38	0.50	8
13.0	0.00	0.00	0.00	1
15.0	0.45	1.00	0.62	69
16.0	0.00	0.00	0.00	2
18.0	0.00	0.00	0.00	1
accuracy			0.45	159
macro avg	0.11	0.12	0.10	159
weighted avg	0.23	0.45	0.29	159

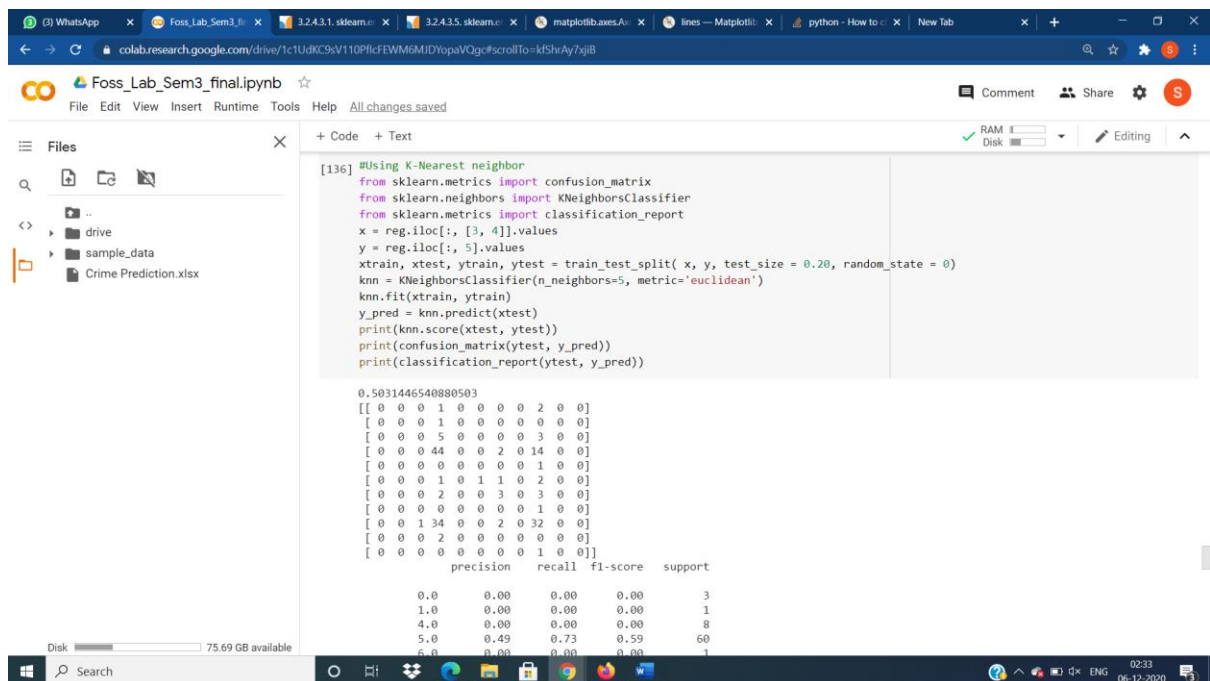
Below the table, a warning message is displayed:

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score :
_warn_prf(average, modifier, msg_start, len(result))
```

Accuracy 45%

Using KNN Classifier

Accuracy 50%



The screenshot shows a Google Colab notebook titled 'Foss_Lab_Sem3_final.ipynb'. The code cell [136] imports necessary libraries and performs a K-Nearest Neighbor classification. The output displays the accuracy score, a confusion matrix, and classification metrics.

```
[136] #Using K-Nearest neighbor
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
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)
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(xtrain, ytrain)
y_pred = knn.predict(xtest)
print(knn.score(xtest, ytest))
print(confusion_matrix(ytest, y_pred))
print(classification_report(ytest, y_pred))
```

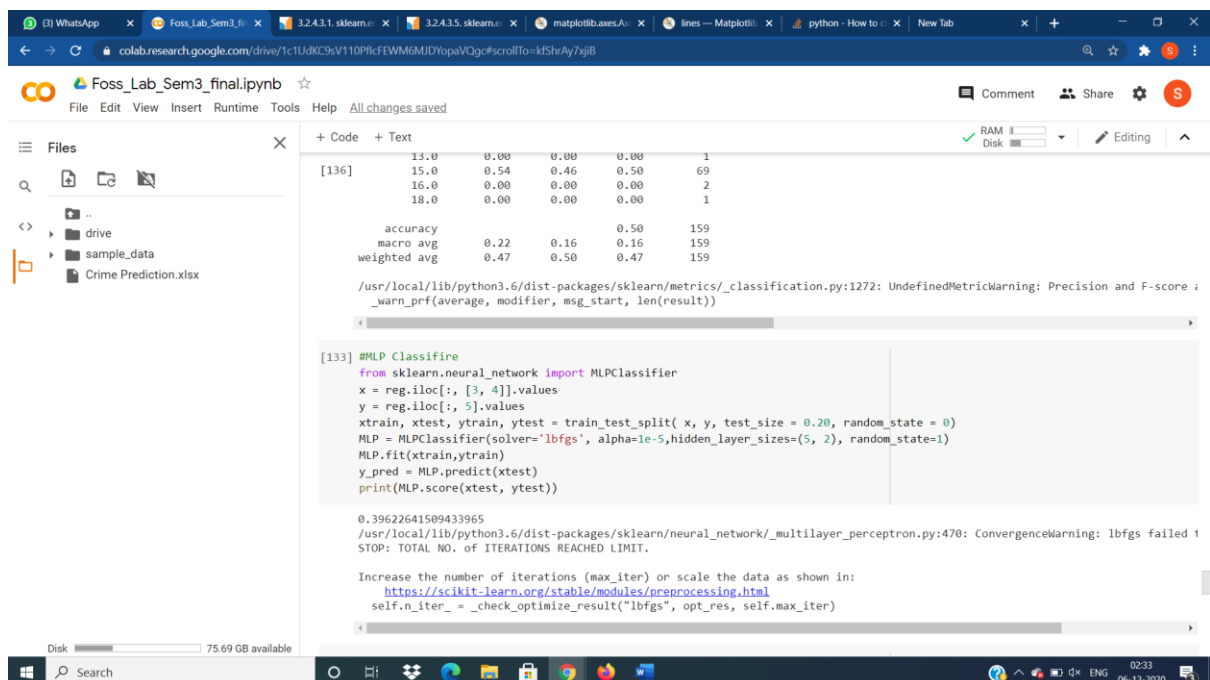
0.5031446540880503

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	3
1.0	0.00	0.00	0.00	1
4.0	0.00	0.00	0.00	8
5.0	0.49	0.73	0.59	60
6.0	0.00	0.00	0.00	1

Using MLP Classifier:

In Parameter: solver= 'lbfgs' , hidden layer size=(5,2),Random state=0

Accuracy 39%



The screenshot shows a Google Colab notebook titled 'Foss_Lab_Sem3_final.ipynb'. The code cell [133] imports the MLPClassifier and performs a training and testing process. The output shows the accuracy score and a convergence warning.

```
[133] #MLP Classifire
from sklearn.neural_network import MLPClassifier
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)
MLP = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=1)
MLP.fit(xtrain, ytrain)
y_pred = MLP.predict(xtest)
print(MLP.score(xtest, ytest))
```

0.39622641509433965

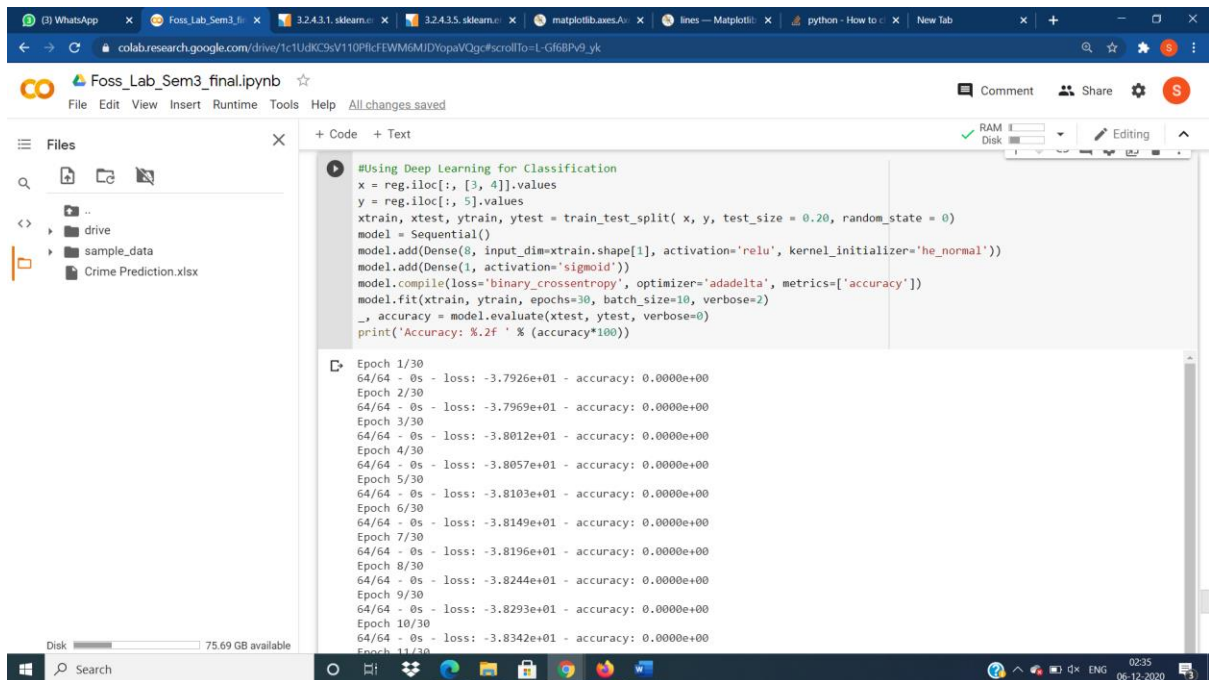
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:470: ConvergenceWarning: lbfgs failed to converge (5 iterations) after 100 iterations. Increase the number of iterations (max_iter) or scale the data as shown in: <https://scikit-learn.org/stable/modules/preprocessing.html>

Using Deep Learning :

Activation= 'relu' in dense layer 8 and Activation= 'sigmoid'

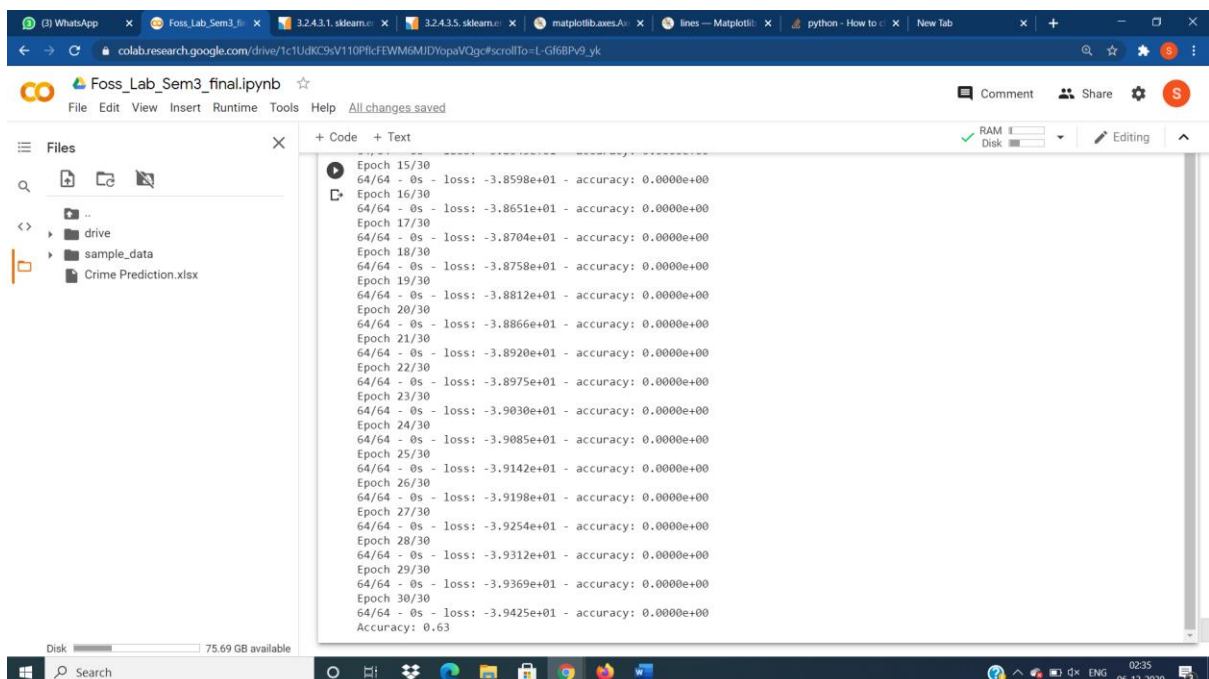
Model.compile loss= 'binary_crossentropy' and optimizer= 'adadelta'

Epoch=30 and batch size =10



```
#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 = 0)
model = Sequential()
model.add(Dense(8, input_dim=xtrain.shape[1], activation='relu', kernel_initializer='he_normal'))
model.add(Dense(5, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adadelta', metrics=['accuracy'])
model.fit(xtrain, ytrain, epochs=30, batch_size=10, verbose=2)
_, accuracy = model.evaluate(xtest, ytest, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

Epoch 1/30
64/64 - 0s - loss: -3.7926e+01 - accuracy: 0.0000e+00
Epoch 2/30
64/64 - 0s - loss: -3.7969e+01 - accuracy: 0.0000e+00
Epoch 3/30
64/64 - 0s - loss: -3.8012e+01 - accuracy: 0.0000e+00
Epoch 4/30
64/64 - 0s - loss: -3.8057e+01 - accuracy: 0.0000e+00
Epoch 5/30
64/64 - 0s - loss: -3.8103e+01 - accuracy: 0.0000e+00
Epoch 6/30
64/64 - 0s - loss: -3.8149e+01 - accuracy: 0.0000e+00
Epoch 7/30
64/64 - 0s - loss: -3.8196e+01 - accuracy: 0.0000e+00
Epoch 8/30
64/64 - 0s - loss: -3.8244e+01 - accuracy: 0.0000e+00
Epoch 9/30
64/64 - 0s - loss: -3.8293e+01 - accuracy: 0.0000e+00
Epoch 10/30
64/64 - 0s - loss: -3.8342e+01 - accuracy: 0.0000e+00
Epoch 11/30



```
Epoch 15/30  
64/64 - 0s - loss: -3.8598e+01 - accuracy: 0.0000e+00  
Epoch 16/30  
64/64 - 0s - loss: -3.8651e+01 - accuracy: 0.0000e+00  
Epoch 17/30  
64/64 - 0s - loss: -3.8704e+01 - accuracy: 0.0000e+00  
Epoch 18/30  
64/64 - 0s - loss: -3.8758e+01 - accuracy: 0.0000e+00  
Epoch 19/30  
64/64 - 0s - loss: -3.8812e+01 - accuracy: 0.0000e+00  
Epoch 20/30  
64/64 - 0s - loss: -3.8866e+01 - accuracy: 0.0000e+00  
Epoch 21/30  
64/64 - 0s - loss: -3.8920e+01 - accuracy: 0.0000e+00  
Epoch 22/30  
64/64 - 0s - loss: -3.8975e+01 - accuracy: 0.0000e+00  
Epoch 23/30  
64/64 - 0s - loss: -3.9030e+01 - accuracy: 0.0000e+00  
Epoch 24/30  
64/64 - 0s - loss: -3.9085e+01 - accuracy: 0.0000e+00  
Epoch 25/30  
64/64 - 0s - loss: -3.9142e+01 - accuracy: 0.0000e+00  
Epoch 26/30  
64/64 - 0s - loss: -3.9198e+01 - accuracy: 0.0000e+00  
Epoch 27/30  
64/64 - 0s - loss: -3.9254e+01 - accuracy: 0.0000e+00  
Epoch 28/30  
64/64 - 0s - loss: -3.9312e+01 - accuracy: 0.0000e+00  
Epoch 29/30  
64/64 - 0s - loss: -3.9369e+01 - accuracy: 0.0000e+00  
Epoch 30/30  
64/64 - 0s - loss: -3.9425e+01 - accuracy: 0.0000e+00  
Accuracy: 0.63
```

Accuracy 63%