Traffic Accidents Prediction Using Ensemble Machine Learning Approach

MSc Research Project

Programme Name

Monisha Lakshme Gowda

Student ID:x18195261

School of Computing

National College of Ireland

Supervisor: Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Monisha Lakshme Gowda
Student ID:	x18195261
Programmer:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Christian Horn
Submission Due Date:	17th August 2019
Project Title:	Configuration Manual
Word Count:	
Page Count:	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	

Assignments that are submitted to the programmer Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Monisha Lakshme Gowda
(x18195261)

1 Introduction

This design manual contains comprehensive guidance on applying the I.T approach implemented as result of both the Traffic Incident Forecasting study thesis utilizing Computer Analytics and maladaptive cognitive. The framework requires all measures being taken to apply the approach. The specifications for device setup would be as shown in:

- Processor used in the study: intel core i5 1.8Ghz DDR3
- RAM configuration: 8GB
- System: x64 processor

2 Environment Setup

Anaconda software can also be used to develop the project. The version of the python which is installed is 3.7. The complete software is available in the internet. The link is: Anaconda. As the compatibility and speed of Google Colab is better than Jupyter notebook, end to end development of the project is made in Google Colab. But the implemented code can be executed in Jupyter Notebook also. The Google Colab is online execution platform. The link for the Google colab is given here Google Colab.

3 Datasets

The dataset for the prediction of Traffic accidents is retrieved from the data.gov. As the historical data is necessary for the research traffic accidents dataset of UK which consist of 2012-2014 data entries.

4 Assessing the datasets

Initially required libraries are imported from the (libraries are pre-installed in the software). This step is a crucial step to libraries contains required package which we need to use during the implementation. The required libraries are mention bellow:

```
from dateutil.parser import parse
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import descartes
import geopandas as gpd
import pandas as pd
import os
import calendar
import datetime as dt
from shapely.geometry import Point, Polygon
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from sklearn.preprocessing import StandardScaler
from keras.optimizers import SGD
import datetime
import random
```

Next the dataset is loaded from the local device by setting the root directory. This directory can be changed by giving the dataset downloaded path. Then the dataset is loaded into the data frame.

```
[6] #Setting the root dicrectory
    os.chdir("/content/drive/My Drive")
```

```
#importing the dataset
df1 =pd.read_csv('/content/drive/My Drive/Traffic_Accidents.csv')
df2 = pd.read_csv('/content/drive/My Drive/Local_Authority_United_Kingdom.csv')
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (31) have mixed types.Specify dtype option or
  interactivity=interactivity, compiler=compiler, result=result)
```

5 Merging the Dataset

```
[19] #Changing the column name so that Authority data become key variable in both dataset
     df2['Local_Authority_(Highway)']= df2['LAD17CD']
```

```
[20] #Terminating the LAD17CD variable as it is renamed as Local_Authority_(Highway)
     df2.drop("LAD17CD", axis=1, inplace=True)
```

```
[21] #Concating the two dataset
     df = (df1.merge(df2[['Local_Authority_(Highway)', 'LAD17NM']], on='Local_Authority_(Highway)'))
```

```
[22] #Renaming the column LAD17NM
     df['District_Names'] = df['LAD17NM']
```

```
[23] #Terminating the column LAD17NM as it is renamed
     df.drop("LAD17NM", axis=1, inplace=True)
```

```
#Dropping the Juction_Detail as it doesn't contain any values

df.drop("Junction_Detail", axis=1, inplace=True)
```

After the concatenation the dataset consists of following variables.

```
 #   Column                                      Non-Null Count   Dtype
---  ------                                      --------------   -----
 0   Accident_Index                              293779 non-null  object
 1   Location_Easting_OSGR                       293779 non-null  int64
 2   Location_Northing_OSGR                      293779 non-null  int64
 3   Longitude                                   293779 non-null  float64
 4   Latitude                                    293779 non-null  float64
 5   Police_Force                                293779 non-null  int64
 6   Accident_Severity                           293779 non-null  int64
 7   Number_of_Vehicles                          293779 non-null  int64
 8   Number_of_Casualties                        293779 non-null  int64
 9   Date                                        293779 non-null  object
 10  Day_of_Week                                 293779 non-null  int64
 11  Time                                        293766 non-null  object
 12  Local_Authority_(District)                  293779 non-null  int64
 13  Local_Authority_(Highway)                   293779 non-null  object
 14  1st_Road_Class                              293779 non-null  int64
 15  1st_Road_Number                             293779 non-null  int64
 16  Road_Type                                   293779 non-null  object
 17  Speed_limit                                 293779 non-null  int64
 18  Junction_Detail                             0 non-null       float64
 19  Junction_Control                            193775 non-null  object
 20  2nd_Road_Class                              293779 non-null  int64
 21  2nd_Road_Number                             293779 non-null  int64
 22  Pedestrian_Crossing-Human_Control           293779 non-null  object
 23  Pedestrian_Crossing-Physical_Facilities     293779 non-null  object
 24  Light_Conditions                            293779 non-null  object
 25  Weather_Conditions                          293779 non-null  object
 26  Road_Surface_Conditions                     293570 non-null  object
 27  Special_Conditions_at_Site                  293777 non-null  object
 28  Carriageway_Hazards                         293776 non-null  object
 29  Urban_or_Rural_Area                         293779 non-null  int64
 30  Did_Police_Officer_Attend_Scene_of_Accident 293777 non-null  object
 31  LSOA_of_Accident_Location                   269293 non-null  object
 32  Year                                        293779 non-null  int64
 33  District_Names                              293779 non-null  object
dtypes: float64(3), int64(15), object(16)
```

6 Data Cleaning and Feature Engineering

The data may contain null values, duplicates values which may disturb the model accuracy. Hence, null and duplicate values are dropped. The format of date, time and year is changed according to the requirement

```
[28] #Dropping the Null values
     df.dropna(axis=0, how='all', inplace=True)
     #Dropping the Duplicate values
     df.drop_duplicates()
```

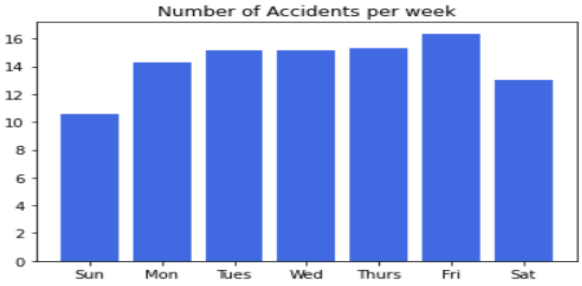
```
#Feature Engnieering
#Changing the Date format
df['Date'] = pd.to_datetime(df['Date'])
#Extracting the month and day from Date variable
df['month'] = df['Date'].dt.month
df['day']= df['Date'].dt.day
#Changeing the Date variable formatt
df['Time'] = pd.to_datetime(df['Time'], format = '%H:%M').dt.hour
df['dayofyear'] = df['Date'].dt.dayofyear
```

7 Exploratory Data Analysis

For better understanding of the data, EDA process is carried out using the variables of the dataset. This process explains the what kind of data is present in the dataset. Different types of graph are used to explore the different variables. Conducted EDA is given bellow

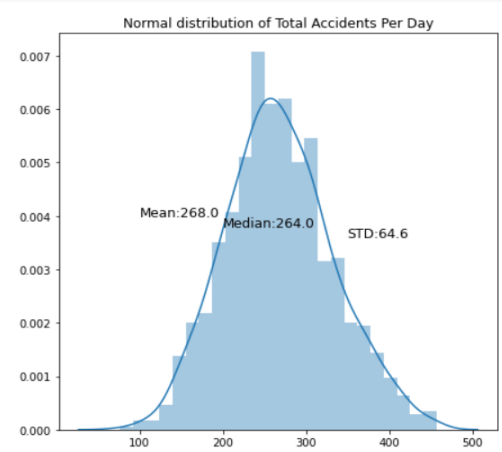
```
# Plotting the Bar chart for number of accidents per week

Day_of_Week_D = df["Day_of_Week"].value_counts().sort_index()/len(df)*100
Day_of_Week_D.index = ["Sun", "Mon", "Tues", "Wed", "Thurs", "Fri", "Sat"]
plt.bar(Day_of_Week_D.index, Day_of_Week_D.values, alpha=1, color="royalblue")
plt.title("Number of Accidents per week")
```

```
Text(0.5, 1.0, 'Number of Accidents per week')
```



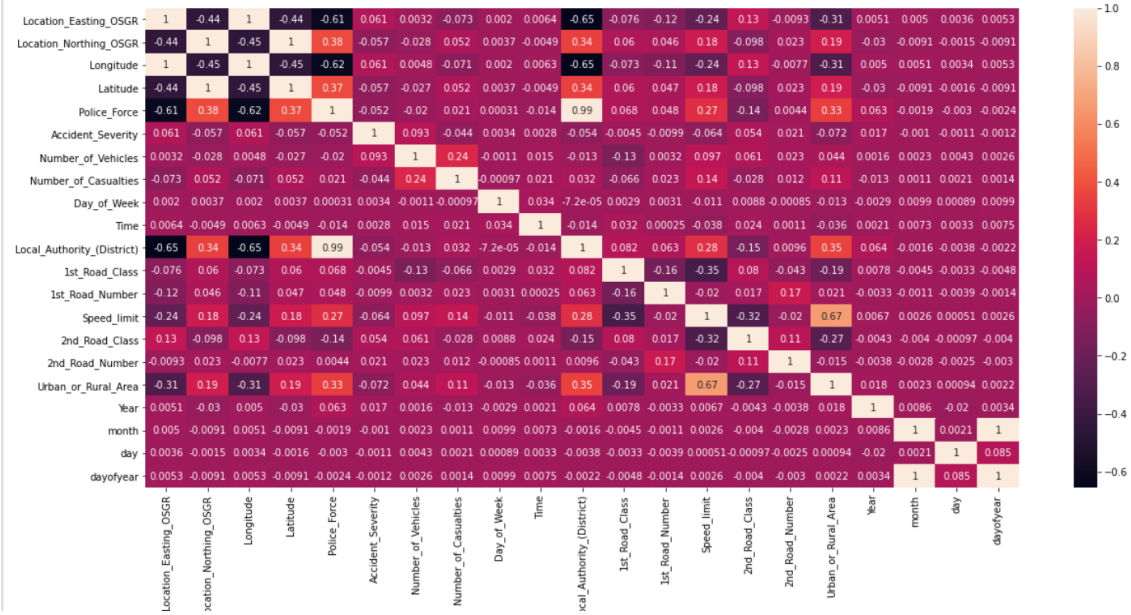

```
#Number of accidents per day
#grouping the data according to the date
Group_by_Date = pd.DataFrame()
Group_by_Date["Date"] = pd.to_datetime(df["Date"])
Group_by_Date["Year"] = df["Year"]
Group_by_Date["Day_of_Week"] = df["Day_of_Week"]
```

```
[59] #Plotting the chart to check the normality
     Total_number = Group_by_Date["Date"].value_counts()

     plt.figure(1, figsize=(7,7))
     sns.distplot(Total_number.values)
     plt.title("Normal distribution of Total Accidents Per Day")
     plt.text(100, 0.004,'Mean:'+ str( "{:.{}f}".format( Total_number.mean(), 1 )), fontsize=12)
     plt.text(200, 0.0038,'Median:'+ str( "{:.{}f}".format( Total_number.median(), 1 )), fontsize=12)
     plt.text(350, 0.0036,'STD:'+ str( "{:.{}f}".format( Total_number.std(), 1 )), fontsize=12)
     plt.show()
```




```
#Finding Correlation among the variables
corr =  df.corr()
plt.subplots(figsize=(20,9))
sns.heatmap(df.corr(), annot = True)
```

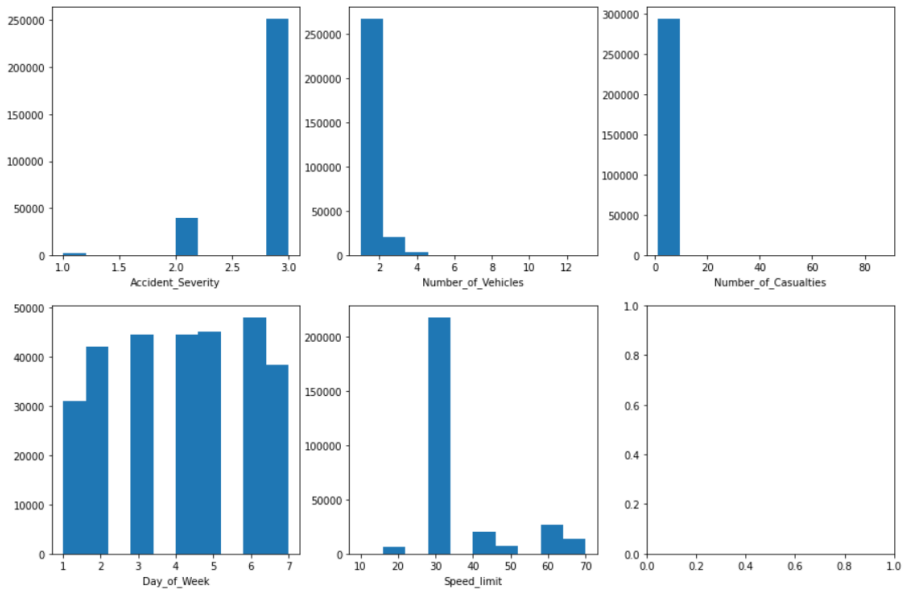
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f35d57e9898>
```



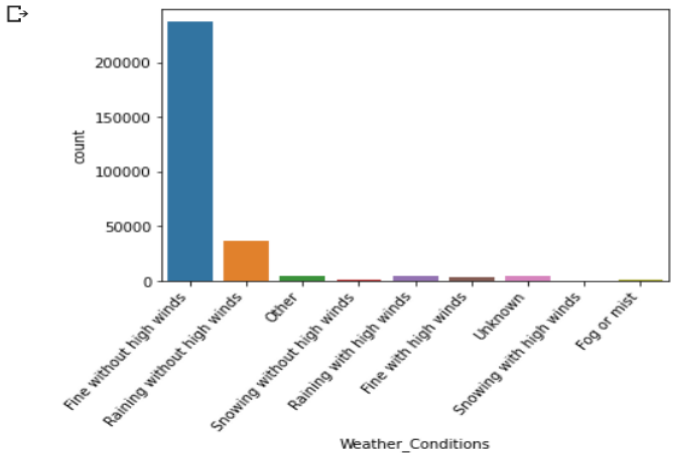

```
[42] #Analysing the accidents rates from 2009 to 2011

     accidents_year_wise = df.groupby(['Year'])['Number_of_Casualties'].sum()
     accidents_year_wise = accidents_year_wise.reset_index()
     accidents_year_wise
```

	Year	Number_of_Casualties
0	2012	156233
1	2013	112313
2	2014	118684

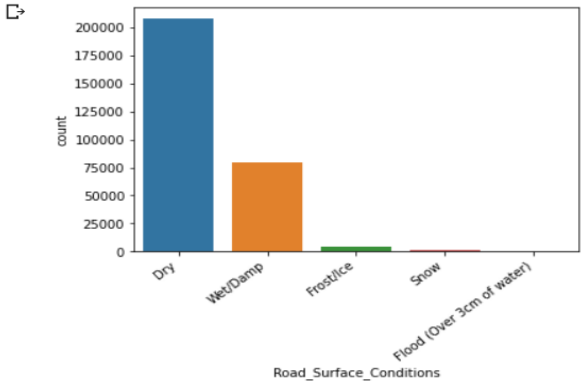
```
[43] #Histogram of numeric parameters
     cols = ["Accident_Severity","Number_of_Vehicles","Number_of_Casualties","Day_of_Week","Speed_limit"]

     f,a =  plt.subplots(2,3,figsize=(15,10))
     a = a.ravel()

     for idx, ax in  enumerate(a):
         if idx < 5:
             ax.hist(df[cols[idx]],bins=10)
             ax.set_xlabel(cols[idx])
```




```
[44] # Analysing the count of diffrent weather condition
     wplot = sns.countplot(data=df,x="Weather_Conditions")
     wplot.set_xticklabels(wplot.get_xticklabels(), rotation=50, ha="right")
     plt.show()
```




```
[45] # start with weather conditions first
     rplot = sns.countplot(data=df,x="Road_Surface_Conditions")
     rplot.set_xticklabels(rplot.get_xticklabels(), rotation=40, ha="right")
     plt.show()
```

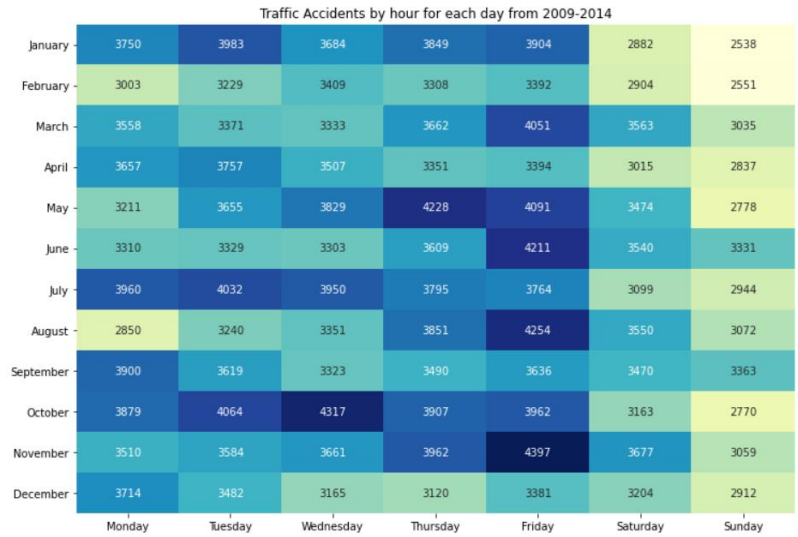


```
#Determine the number of accidents for each day of the week by month (combine years 2012-2014)

def accidents_by_month(df, month):
    month_list = []
    mon, tues, wed, thurs, fri, sat, sun = 0,0,0,0,0,0,0

    for idx, row in df.iterrows():
        if row["month_name"] == month and row["weekday"] == "Monday": mon +=1
        elif row["month_name"] == month and row["weekday"] == "Tuesday": tues +=1
        elif row["month_name"] == month and row["weekday"] == "Wednesday": wed +=1
        elif row["month_name"] == month and row["weekday"] == "Thursday": thurs +=1
        elif row["month_name"] == month and row["weekday"] == "Friday": fri +=1
        elif row["month_name"] == month and row["weekday"] == "Saturday": sat +=1
        elif row["month_name"] == month and row["weekday"] == "Sunday": sun +=1
        else: a=0

    month_list.append(mon)
    month_list.append(tues)
    month_list.append(wed)
    month_list.append(thurs)
    month_list.append(fri)
    month_list.append(sat)
    month_list.append(sun)

    total = mon+tues+wed+thurs+fri+sat+sun

    return(month_list)
```

```
[47] #Extract RTA data for each day by month
     January = accidents_by_month(df, "January")
     February = accidents_by_month(df, "February")
     March = accidents_by_month(df, "March")
     April = accidents_by_month(df, "April")
     May = accidents_by_month(df, "May")
     June = accidents_by_month(df, "June")
     July = accidents_by_month(df, "July")
     August = accidents_by_month(df, "August")
     September = accidents_by_month(df, "September")
     October = accidents_by_month(df, "October")
     November = accidents_by_month(df, "November")
     December = accidents_by_month(df, "December")
```

```
[49] import seaborn as sns
     import matplotlib.pyplot as plt
     fig = plt.figure(figsize=(12,12))
     ax = sns.heatmap(heatmap_df, annot=True, fmt="d", cmap='YlGnBu', cbar_kws={"orientation": "horizontal"} )
     ax.set_title("Traffic Accidents by hour for each day from 2009-2014")
```

```
Text(0.5, 1.0, 'Traffic Accidents by hour for each day from 2009-2014')
```



8 Feature Selection

The dataset consists of 33 variables along with the complete UK data entries, but for the research required variable and data is extracted. Here, data of 33 London Brough's is extract and concatenated data as per the Brough's names. Finally, the dataset consist of 83621 data entries.

```
###Extracting the Columns required for Clustering
City = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000001'] # Investigate City and Westminster boroughs
Barking_And_Dagenham = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000002']
Barnet = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000003']
Bexley = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000004']
Brent = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000005']
Bromley = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000006']
Camden = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000007']
Croydon = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000008']
Ealing = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000009']
Enfield = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000010']
Greenwich = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000011']
Hackney = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000012']
Hammersmith_and_Fulham = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000013']
Haringey = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000014']
Harrow = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000015']
Havering = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000016']
Hillingdon = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000017']
Hounslow = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000018']
Islington = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000019']
Kensington_and_Chelsea = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000020']
Kingston_upon_Thames = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000021']
Lambeth = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000022']
Lewisham = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000023']
Merton = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000024']
Newham = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000025']
Redbridge = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000026']
Richmond_Upon_Thames = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000027']
Southwark = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000028']
Sutton = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000029']
Tower_Hamlets = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000030']
Waltham_Forest = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000031']
Wandsworth = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000032']
Westminster = df_cluster[df_cluster['Local_Authority_(Highway)'] == 'E09000033']
```

```
#Concating the data as per the City Names
df_cluster = pd.concat([City,Barking_And_Dagenham,Barnet,Bexley,Brent,Bromley,Camden,Croydon,Ealing,Enfield,
                        Greenwich,Hackney,Hammersmith_and_Fulham,Haringey,Harrow,Havering,Hillingdon,Hounslow,
                        Islington,Kensington_and_Chelsea,Kingston_upon_Thames,Lambeth,Lewisham,Merton,Newham,Redbridge,
                        Richmond_Upon_Thames,Southwark,Sutton,Tower_Hamlets,Waltham_Forest,Wandsworth,Westminster], axis = 0)
```

```
[ ]  len(df_cluster)

     83621
```

9 k-NN Distance

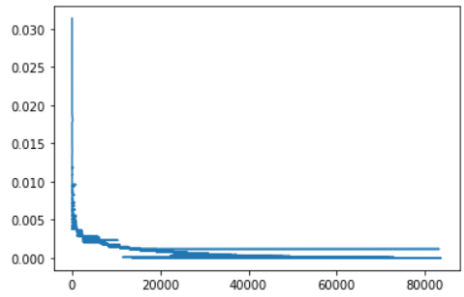
The k-NN distance is identified to calculate the distance between the accidents. So that the distance can be defined in the DBSCAN to form a cluster.

k-NN Distance

```
[ ]  data = df_cluster[["Longitude","Latitude"]]
```

```
[ ]  from sklearn.neighbors import NearestNeighbors

     ns = 4
     nbrs = NearestNeighbors(n_neighbors=ns).fit(data)
     distances, indices = nbrs.kneighbors(data)
     distanceDec = sorted(distances[:,ns-1], reverse=True)
     plt.plot(indices[:,0], distanceDec)
     distances.mean()
```

```
0.0003357308166863301
```



cluster distance is in terms of latitude degree. 0.001 degree in latitude is about 100 meters in metrics. Hence, 0.000335 is about 36 meters

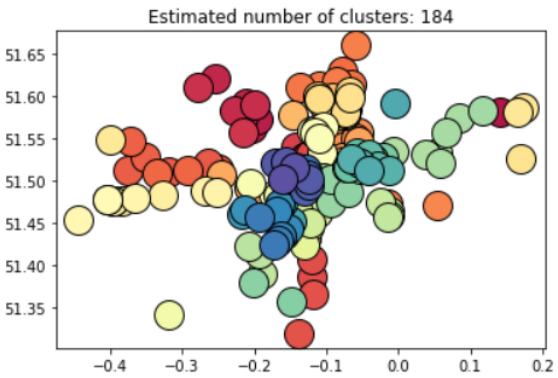
10 DBSCAN Clustering

Clustering of the accidents spots is made by using the k-nn distance obtained by finding the distance within the accident's points

```
[62] #Appling teh Clustering technique
     from sklearn.cluster import DBSCAN
     import numpy as np
     #eps = 36
     df_cluster_dbc = df_cluster
     # 0.001 degree in latitude means ~100 meters
     # cluster requires at least 20 accidents
     loc = df_cluster_dbc[['Latitude','Longitude']]

     db = DBSCAN(eps= 0.00033 , min_samples=20).fit(loc)

     labels = db.labels_
     n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
     print('Estimated number of clusters: %d' % n_clusters_)
     n_noise_ = list(labels).count(-1)
     print('Estimated number of noise points: %d' % n_noise_)

     core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
     core_samples_mask[db.core_sample_indices_] = True

     df_cluster_dbc['Cluster'] = labels
```

```
Estimated number of clusters: 184
Estimated number of noise points: 78497
```

```
[63] cluster_loc = []
     cluster_mean = {}

     unique_labels = set(labels)
     colors = [plt.cm.Spectral(each)
               for each in np.linspace(0, 1, len(unique_labels))]
     for k, col in zip(unique_labels, colors):
         if k != -1:
             class_member_mask = (labels == k)

             xy = loc[class_member_mask & core_samples_mask]
             plt.plot(xy.iloc[:, 1].mean(), xy.iloc[:, 0].mean(), 'o', markerfacecolor=tuple(col),
                      markeredgecolor='k', markersize=20)
             cluster_loc.append((xy.iloc[:, 1].mean(), xy.iloc[:, 0].mean(), xy.shape[1]))
             cluster_mean[k] = [xy.iloc[:, 1].mean(), xy.iloc[:, 0].mean()]

     plt.title('Estimated number of clusters: %d' % n_clusters_)
     plt.show()
```




```
[65] # drop the noise cluster
     df_cluster_dbc2.drop(df_cluster_dbc2[df_cluster_dbc2['Cluster'] == -1].index , inplace=True)
```

```
[66] #after droping the noise cluster the length of the cluster is 5124
     len(df_cluster_dbc2)
```

```
5124
```

```
[68] #Resetting the index of the data
     df_cluster_dbc2.reset_index(inplace=True)
```

```
[69] #Assiging the 1 value for positive accidents points
     df_cluster_dbc2['Accident'] = pd.Series([1 for x in range(len(df_cluster_dbc2.index))])
```

11 Negative Sampling

Generating the negative samples as non-accidents points which can be considered for classifying the accidents

```
#Positive data points are retreived from the dataste
df_nonacc_n_acc = pd.DataFrame(columns=['Longitude','Latitude','Cluster','Day_of_Week','day','month','Time','dayofyear','Number_of_Vehicles','Local_Auth

#for each of the each accidents, create 3 other data points
for index, row in df_cluster_dbc2.iterrows():
    #print(row['Accident_Index'], row['Cluster'])

    #add itself
    Longitude = row['Longitude']
    Latitude = row['Latitude']
    least = row['Location_Easting_OSGR']
    lnorth = row['Location_Northing_OSGR']
    nclust = row['Cluster']
    dweek = row['Day_of_Week']
    dday = row['day']
    mday = row['month']
    nhr = row['Time']
    nday = row['dayofyear']
    nvec = row['Number_of_Vehicles']
    lauthor = row['Local_Authority_(District)']
    #lhigh = row['Local_Authority_(Highway)']
    roadclass = row['1st_Road_Class']
    roadnum = row['1st_Road_Number']
    splimit = row['Speed_limit']
    accident = row['Accident']
    least = row['Location_Easting_OSGR']
    dfnew = pd.DataFrame([[Longitude,Latitude,least,lnorth,lauthor,nclust,dweek,dday,mday,nhr,nday,nvec,roadclass,roadnum,splimit,accident]], columns=['
    df_nonacc_n_acc = df_nonacc_n_acc.append(dfnew,ignore_index=True)
```

```
[70]  i=0
      while(i<3):
          dt_first = datetime.datetime(2010,1,1)
          nday = random.randint(1, 365)
          dtdelta = datetime.timedelta(days=nday)
          dt_targ = dt_first + dtdelta

          Longitude = row['Longitude']
          Latitude = row['Latitude']
          least = row['Location_Easting_OSGR']
          lnorth = row['Location_Northing_OSGR']
          nclust = row['Cluster']
          dweek = dt_targ.weekday()
          dday = row['day']
          mday = row['month']
          nhr = random.randint(0, 23)
          nvec = row['Number_of_Vehicles']
          lauthor = row['Local_Authority_(District)']
          #lhigh = row['Local_Authority_(Highway)']
          roadclass = row['1st_Road_Class']
          roadnum = row['1st_Road_Number']
          splimit = row['Speed_limit']
          accident = 0

          #whether such record exist first
          isacc = df_cluster_dbc2.loc[(df_cluster_dbc2['Cluster'] == nclust) & (df_cluster_dbc2['dayofyear'] == nday) & (df_cluster_dbc2['Time'] == nhr)]
          if isacc.empty:
              i += 1
              dfnew = pd.DataFrame([[Longitude,Latitude,least,lnorth,lauthor,nclust,dweek,dday,mday,nhr,nvec,nday,roadclass,roadnum,splimit,accident]], columns=['Lo
              df_nonacc_n_acc = df_nonacc_n_acc.append(dfnew,ignore_index=True)
          else:
              print([nclust,nday,nhr])
              print("Accident found!")
```

```
[70] Accident found!
     [106, 72, 8]
     Accident found!
     [115, 195, 23]
     Accident found!
     [118, 98, 15]
     Accident found!
     [121, 259, 13]
     Accident found!
     [119, 150, 18]
     Accident found!
     [133, 263, 17]
     Accident found!
```

12 Implementing the Model

12.1 Loading the Both the positive and negative dataset

```
[71] #Loading the generated Both Positive and Negative accident points
     df_points =pd.read_csv('/content/drive/My Drive/Accidents.csv')
```

```
[72] # using dictionary to convert specific columns
     convert_dict = {'Cluster': float,
                     'Day_of_Week': float,
                     'Time': float,
                     'day': float,
                     'month': float,
                     'dayofyear': float,
                     'Number_of_Vehicles': float,
                     '1st_Road_Class': float,
                     '1st_Road_Number': float,
                     'Speed_limit': float,
                     'Accident': float,
                     'Local_Authority_(District)':float

                    }

     df_points = df_points.astype(convert_dict)
     print(df_points.dtypes)
```

```
Unnamed: 0                    int64
Longitude                   float64
Latitude                    float64
Cluster                     float64
Day_of_Week                 float64
day                         float64
month                       float64
Time                        float64
dayofyear                   float64
Number_of_Vehicles          float64
Local_Authority_(District)  float64
1st_Road_Class              float64
1st_Road_Number             float64
Speed_limit                 float64
Accident                    float64
Location_Easting_OSGR       float64
Location_Northing_OSGR      float64
dtype: object
```

12.2 Dividing the dataset into training and testing

The dataset is divided in 70:30 ratio of training and testing respectively

```
[259] # Import train_test_split function
      from sklearn.model_selection import train_test_split

      X=df_points[['Cluster',
          'Day_of_Week',
          'Longitude',
          'Latitude',
          'Time',
          'month',
          'day',
          'dayofyear']]  # Features
      y=df_points['Accident']  # Labels

      # Split dataset into training set and test set
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
```

12.3 Applying the model:

Random Forest, AdaBoost Classifier, Ensemble Logistic Regression, XGBoost Classifier models is applied. The model is displayed along with the Accuracy.

AdaBoost Classifier

```
[262] #Importing the libraries
      from sklearn.ensemble import AdaBoostClassifier

      from sklearn.tree import DecisionTreeClassifier
```

```
[263] #Training the model
      ada = AdaBoostClassifier(DecisionTreeClassifier(),n_estimators=100, random_state=0)
      ada.fit(X_train,y_train)
      #Predicting the test dataset
      y_pred_ada = ada.predict(X_test)
```

```
[264] #Import scikit-learn metrics module for accuracy calculation
      from sklearn import metrics
      # Model Accuracy, how often is the classifier correct?
      import random
      random.seed(10)
      print("Accuracy:",metrics.accuracy_score(y_test, y_pred_ada))
```

```
Accuracy: 0.7419092535371605
```

Ensemble Logistic Regression

```
[265] from sklearn.ensemble import BaggingClassifier
      from sklearn.linear_model import LogisticRegression
      logbagClf = BaggingClassifier(LogisticRegression(random_state=0, solver='lbfgs'), n_estimators = 400, oob_score = True, random_state = 9
      logbagClf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
[266] #Predicting the test dataset
      preds_enlog = logbagClf.predict(X_test)

      print("Accuracy:",metrics.accuracy_score(y_test, preds_enlog))
```

```
Accuracy: 0.7588225727760611
```

XGBoost Classifer

```
[267] from xgboost import XGBClassifier
      from time import sleep
      import random
      xgb = XGBClassifier(n_estimators=100)
      #Training the data
      xgb.fit(X_train, y_train)
      #Predicting the test data
      preds_xgboost= xgb.predict(X_test)
```

```
[268] #Import scikit-learn metrics module for accuracy calculation
      from sklearn import metrics
      # Model Accuracy, how often is the classifier correct?
      import random
      random.seed(10)
      print("Accuracy:",metrics.accuracy_score(y_test, preds_xgboost))
```

```
Accuracy: 0.7845178077736217
```

Ensemble Models

(Decision Tree, SVM, Logistic Regression)

```
[269] from sklearn.ensemble import VotingClassifier
      from sklearn.svm import SVC
      from sklearn.linear_model import LogisticRegression
      from sklearn import tree
      from sklearn.tree import DecisionTreeClassifier

      #nbclf = GaussianNB()

      dfClf = DecisionTreeClassifier(random_state=0) # 500 trees.
      svmClf = SVC(probability=True, random_state=0) # probability calculation
      logClf = LogisticRegression(random_state=0)
      # constructing the ensemble classifier by mentioning the individual classifiers.
      clf2 = VotingClassifier(estimators = [('df',dfClf), ('svm',svmClf),('log',logClf)], voting='soft')

      # train the ensemble classifier
      clf2.fit(X_train, y_train)
```

```
VotingClassifier(estimators=[('df',
                              DecisionTreeClassifier(ccp_alpha=0.0,
                                                     class_weight=None,
                                                     criterion='gini',
                                                     max_depth=None,
                                                     max_features=None,
                                                     max_leaf_nodes=None
```

```
270] #Predicting teh test dataset
     preds_all = clf2.predict(X_test)

     print("Accuracy:",metrics.accuracy_score(y_test, preds_all))
```

```
Accuracy: 0.7451618149292568
```

13 Evaluation of the models

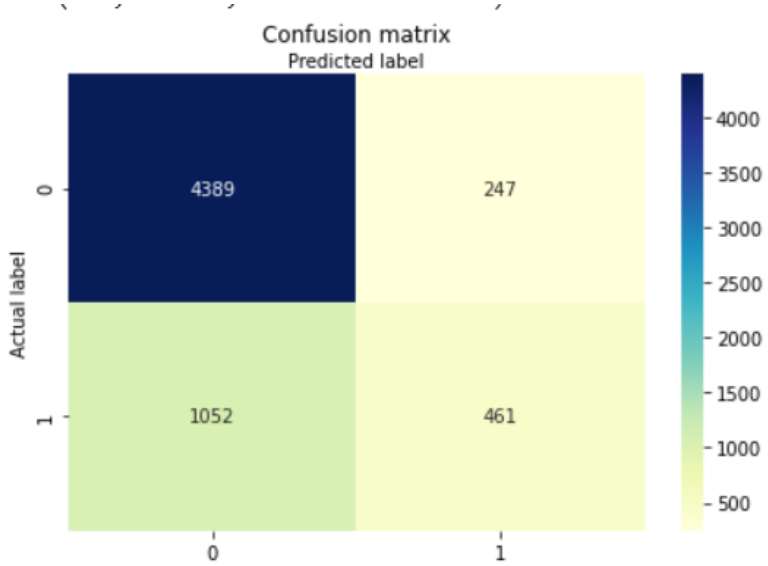
Evaluation of the model is done by use the evaluation metrics
1] Confusion Matrix
2] Precision
3] Recall
4] ROC

Importing the required libraries

```
[271] # import required modules
      import numpy as np
      import matplotlib.pyplot as plt
      import seaborn as sns
      %matplotlib inline
```

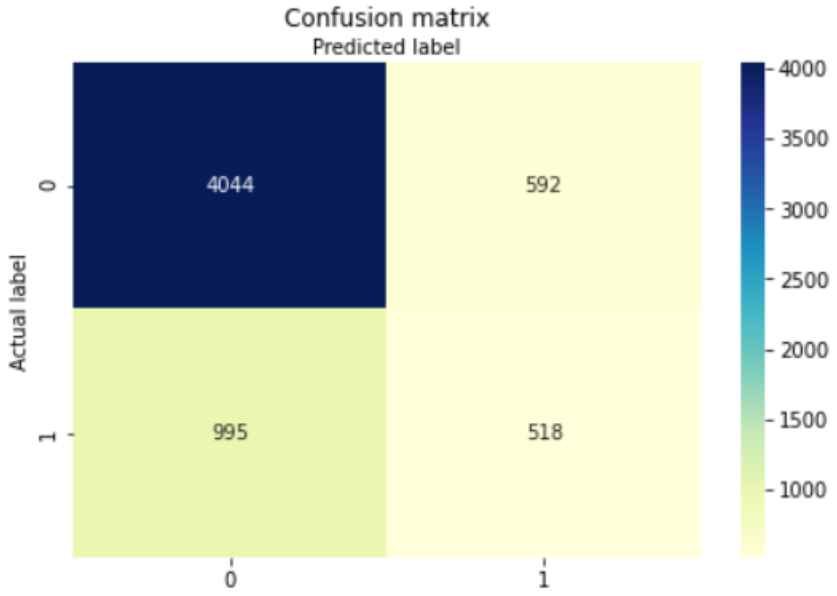
Confusion Matrix of Random Forest

```
# import the metrics class
from sklearn import metrics
cnf_matrix_ran= metrics.confusion_matrix(y_test, y_pred_ran)

class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix_ran), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```



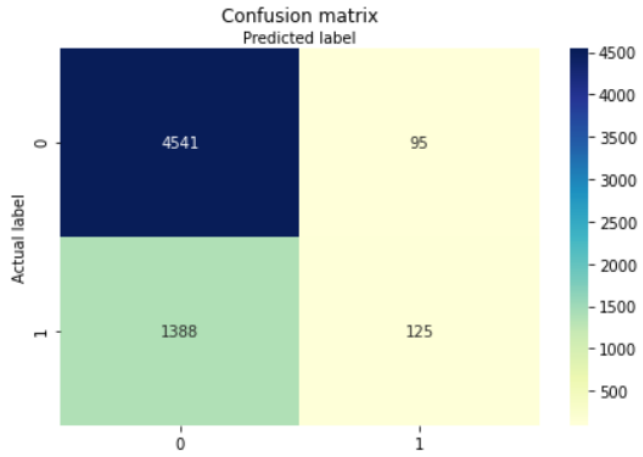
Confusion Matrix of AdaBoost Classifier

```
[273] # View confusion matrix for test data and predictions
      # import the metrics class
      from sklearn import metrics
      cnf_matrix_ada = metrics.confusion_matrix(y_test, y_pred_ada)

      class_names=[0,1] # name  of classes
      fig, ax = plt.subplots()
      tick_marks = np.arange(len(class_names))
      plt.xticks(tick_marks, class_names)
      plt.yticks(tick_marks, class_names)
      # create heatmap
      sns.heatmap(pd.DataFrame(cnf_matrix_ada), annot=True, cmap="YlGnBu" ,fmt='g')
      ax.xaxis.set_label_position("top")
      plt.tight_layout()
      plt.title('Confusion matrix', y=1.1)
      plt.ylabel('Actual label')
      plt.xlabel('Predicted label')
```



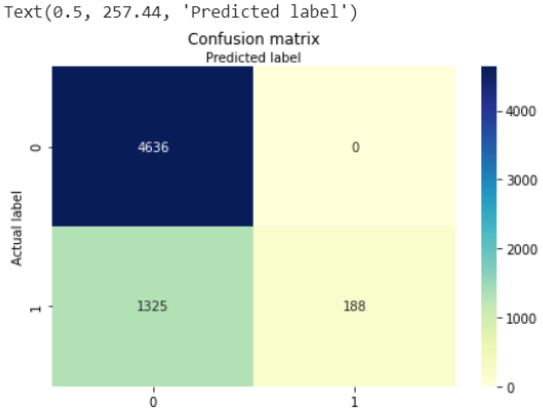
Confusion Matrix of Ensemble Logistic Regression

```
[274] # View confusion matrix for test data and predictions
      # import the metrics class
      from sklearn import metrics
      cnf_matrix_log = metrics.confusion_matrix(y_test, preds_enlog)

      class_names=[0,1] # name  of classes
      fig, ax = plt.subplots()
      tick_marks = np.arange(len(class_names))
      plt.xticks(tick_marks, class_names)
      plt.yticks(tick_marks, class_names)
      # create heatmap
      sns.heatmap(pd.DataFrame(cnf_matrix_log), annot=True, cmap="YlGnBu" ,fmt='g')
      ax.xaxis.set_label_position("top")
      plt.tight_layout()
      plt.title('Confusion matrix', y=1.1)
      plt.ylabel('Actual label')
      plt.xlabel('Predicted label')
```



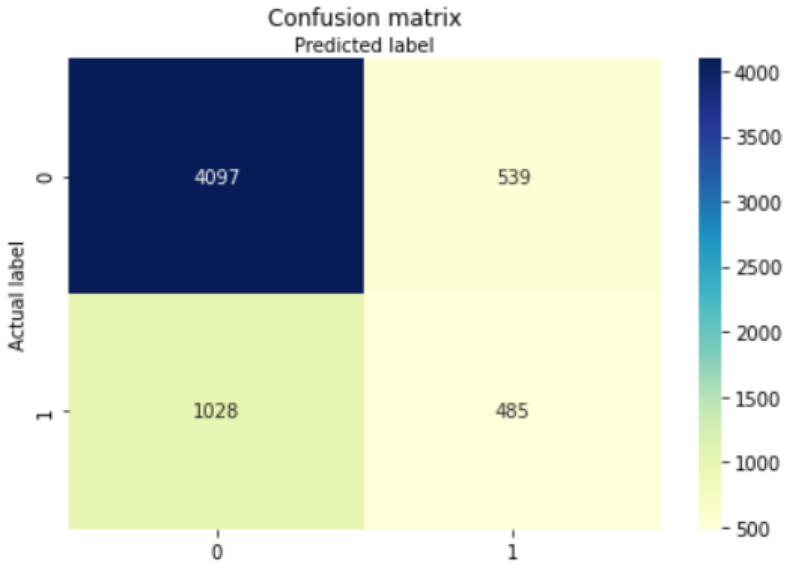
Confusion Matrix of XGBoost Classifier

```
[275] # View confusion matrix for test data and predictions
      # import the metrics class
      from sklearn import metrics
      cnf_matrix_xgboost= metrics.confusion_matrix(y_test, preds_xgboost)

      class_names=[0,1] # name  of classes
      fig, ax = plt.subplots()
      tick_marks = np.arange(len(class_names))
      plt.xticks(tick_marks, class_names)
      plt.yticks(tick_marks, class_names)
      # create heatmap
      sns.heatmap(pd.DataFrame(cnf_matrix_xgboost), annot=True, cmap="YlGnBu" ,fmt='g')
      ax.xaxis.set_label_position("top")
      plt.tight_layout()
      plt.title('Confusion matrix', y=1.1)
      plt.ylabel('Actual label')
      plt.xlabel('Predicted label')
```



Confusion Matrix of Ensemble model(Decision Tree, SVM, Logistic Regression)

```
[276] # View confusion matrix for test data and predictions
      # import the metrics class
      from sklearn import metrics
      cnf_matrix_all= metrics.confusion_matrix(y_test, preds_all)

      class_names=[0,1] # name  of classes
      fig, ax = plt.subplots()
      tick_marks = np.arange(len(class_names))
      plt.xticks(tick_marks, class_names)
      plt.yticks(tick_marks, class_names)
      # create heatmap
      sns.heatmap(pd.DataFrame(cnf_matrix_all), annot=True, cmap="YlGnBu" ,fmt='g')
      ax.xaxis.set_label_position("top")
      plt.tight_layout()
      plt.title('Confusion matrix', y=1.1)
      plt.ylabel('Actual label')
      plt.xlabel('Predicted label')
```



Precision of the Models

```
277] #Dispaying the Precision of the model

     #Precision of Random Forest
     print("Precision of Random Forest:",metrics.precision_score(y_test, y_pred_ran))
     #Precision of AdaBoost
     print("Precision of AdaBoost:",metrics.precision_score(y_test, y_pred_ada))
     #Precision of Ensemble Logistic Regression
     print("Precision of Ensemble Logistic Regression:",metrics.precision_score(y_test, preds_enlog))
     #Precision of XGBoost
     print("Precision of XGBoost:",metrics.precision_score(y_test, preds_xgboost))
     #Precision of Ensemble(log,svm,Dt)
     print("Precision of Ensemble(log,svm,Dt):",metrics.precision_score(y_test, preds_all))
```

```
Precision of Random Forest: 0.6511299435028248
Precision of AdaBoost: 0.4666666666666667
Precision of Ensemble Logistic Regression: 0.5681818181818182
Precision of XGBoost: 1.0
Precision of Ensemble(log,svm,Dt): 0.4736328125
```

Recall of The Models

```
[278] #Dispaying the Recall of the model

     #Recall of Random Forest
     print("Recall of Random Forest:",metrics.recall_score(y_test, y_pred_ran))
     #Recall of AdaBoost
     print("Recall of AdaBoost:",metrics.recall_score(y_test, y_pred_ada))
     #Recall of Ensemble Logistic Regression
     print("Recall of Ensemble Logistic Regression:",metrics.recall_score(y_test, preds_enlog))
     #Recall of XGBoost
     print("Recall of XGBoost:",metrics.recall_score(y_test, preds_xgboost))
     #Recall of Ensemble(log,svm,Dt)
     print("Recall of Ensemble(log,svm,Dt):",metrics.recall_score(y_test, preds_all))
```

```
Recall of Random Forest: 0.30469266358228686
Recall of AdaBoost: 0.34236615994712494
Recall of Ensemble Logistic Regression: 0.08261731658955718
Recall of XGBoost: 0.12425644415069398
Recall of Ensemble(log,svm,Dt): 0.3205551883674818
```

F-1 Score Of The Models

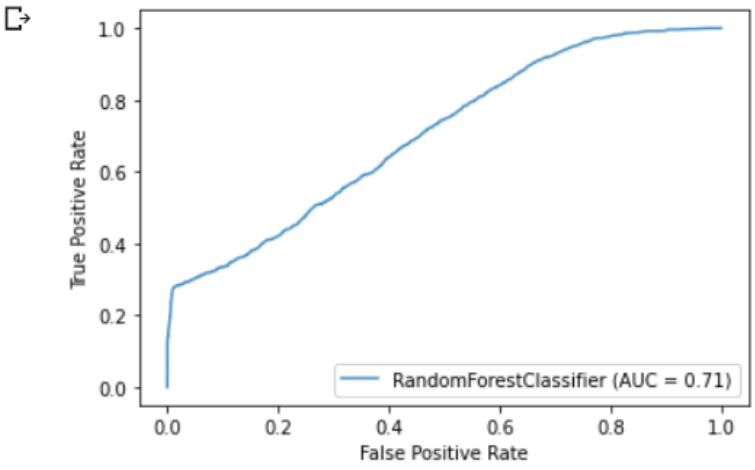
```
[279] #Dispaying the F-1 Score of the model

      #F-1 Score of Random Forest
      print("F-1 Score of Random Forest:",metrics.f1_score(y_test, y_pred_ran))
      #F-1 Score of AdaBoost
      print("F-1 Score of AdaBoost:",metrics.f1_score(y_test, y_pred_ada))
      #F-1 Score of Ensemble Logistic Regression
      print("F-1 Score of Ensemble Logistic Regression:",metrics.f1_score(y_test, preds_enlog))
      #F-1 Score of XGBoost
      print("F-1 Score of XGBoost:",metrics.f1_score(y_test, preds_xgboost))
      #F-1 Score of Ensemble(log,svm,Dt)
      print("F-1 Score of Ensemble(log,svm,Dt):",metrics.f1_score(y_test, preds_all))
```

```
F-1 Score of Random Forest: 0.415128320576317
F-1 Score of AdaBoost: 0.3949675943576058
F-1 Score of Ensemble Logistic Regression: 0.14425851125216388
F-1 Score of XGBoost: 0.2210464432686655
F-1 Score of Ensemble(log,svm,Dt): 0.38234134804887665
```

ROC Of The Models

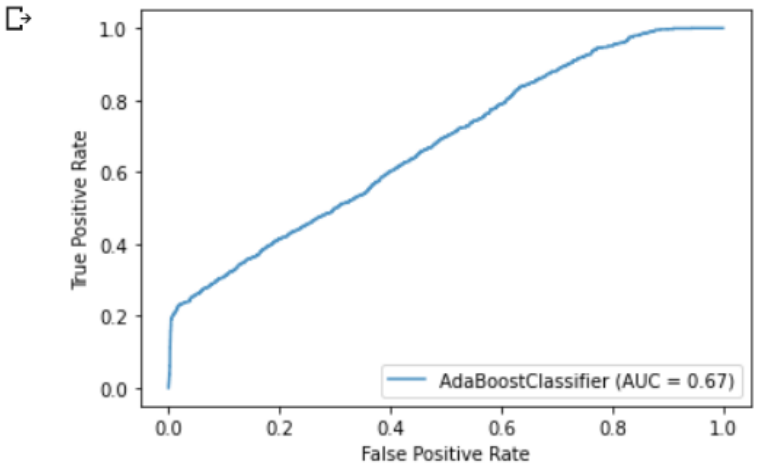
ROC of Random Forest

```
[280] from sklearn.metrics import plot_roc_curve
      ax = plt.gca()
      clf_disp = plot_roc_curve(clf, X_test, y_test, ax=ax, alpha=0.8)
      plt.show()
```



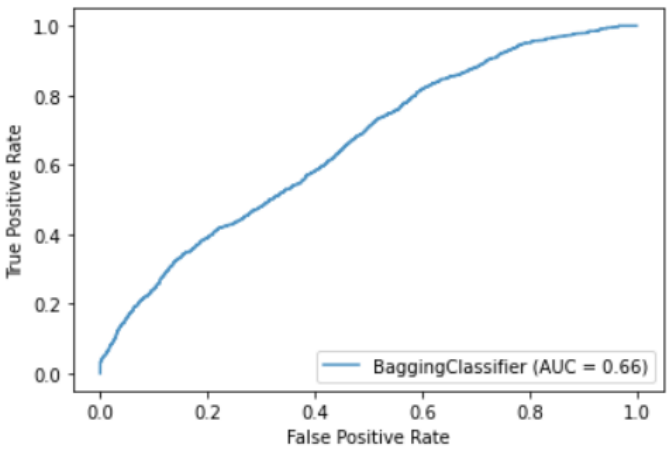
ROC of AdaBoost Classifier

```
[281] from sklearn.metrics import plot_roc_curve
      ax = plt.gca()
      ada_disp = plot_roc_curve(ada, X_test, y_test, ax=ax, alpha=0.8)
      plt.show()
```



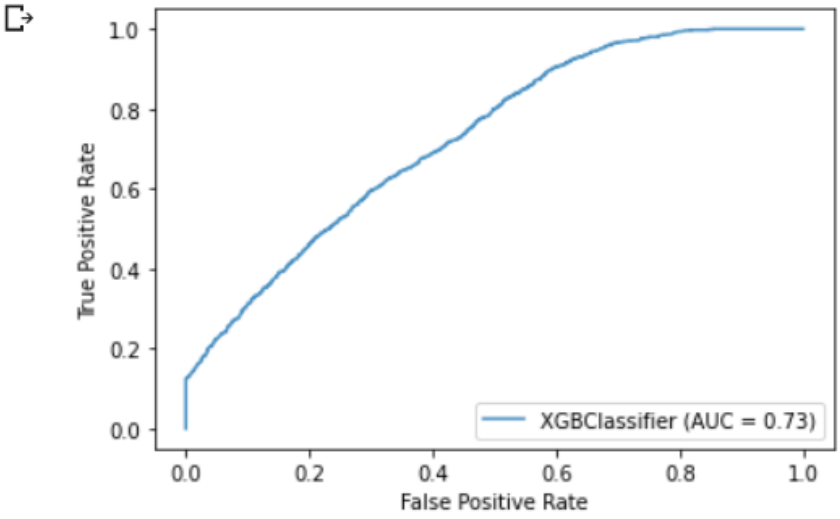
ROC of Ensemble Logistics Regression

```
[282] from sklearn.metrics import plot_roc_curve
      ax = plt.gca()
      logbagClf_disp = plot_roc_curve(logbagClf, X_test, y_test, ax=ax, alpha=0.8)
      plt.show()
```



ROC of XGBoost

```
[283] from sklearn.metrics import plot_roc_curve
      ax = plt.gca()
      xgb_disp = plot_roc_curve(xgb, X_test, y_test, ax=ax, alpha=0.8)
      plt.show()
```



ROC of Ensemble(Decision tree, SVM, Logistic Regression)

```
[284] from sklearn.metrics import plot_roc_curve
      ax = plt.gca()
      clf2_disp = plot_roc_curve(clf2, X_test, y_test, ax=ax, alpha=0.8)
      plt.show()
```