```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/
```

```
path = "/content/drive/MyDrive/datasets/Chicago_Crimes_2012_to_2017.csv"
```

```
!pip install category_encoders
!pip install eli5
!pip install -U pandas-profiling
```

```
Requirement already satisfied, skipping upgrade: ipywidgets>=7.5.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied, skipping upgrade: attrs>=19.3.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied, skipping upgrade: requests>=2.24.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied, skipping upgrade: seaborn>=0.10.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied, skipping upgrade: confuse>=1.0.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied, skipping upgrade: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied, skipping upgrade: missingno>=0.4.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied, skipping upgrade: joblib in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied, skipping upgrade: htmlmin>=0.1.12 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied, skipping upgrade: phik>=0.10.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied, skipping upgrade: visions[type_image_path]==0.6.0 in /usr/local/lib/p
Requirement already satisfied, skipping upgrade: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied, skipping upgrade: matplotlib>=3.2.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied, skipping upgrade: tqdm>=4.48.2 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied, skipping upgrade: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/d
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied, skipping upgrade: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist
Requirement already satisfied, skipping upgrade: widgetsnbextension~=3.5.0 in /usr/local/lib/python
Requirement already satisfied, skipping upgrade: nbformat>=4.2.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied, skipping upgrade: ipykernel>=4.5.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied, skipping upgrade: ipython>=4.0.0; python_version >= "3.3" in /usr/loca
Requirement already satisfied, skipping upgrade: traitlets>=4.3.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied, skipping upgrade: jupyterlab-widgets>=1.0.0; python_version >= "3.6" i
Requirement already satisfied, skipping upgrade: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-
Requirement already satisfied, skipping upgrade: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied, skipping upgrade: chardet<5,>=3.0.2 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied, skipping upgrade: pyyaml in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied, skipping upgrade: networkx>=2.4 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied, skipping upgrade: imagehash; extra == "type_image_path" in /usr/loca
Requirement already satisfied, skipping upgrade: Pillow; extra == "type_image_path" in /usr/local/lib
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/l
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied, skipping upgrade: six>=1.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied, skipping upgrade: notebook>=4.4.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied, skipping upgrade: jupyter-core in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied, skipping upgrade: jsonschema!=2.5.0,>=2.4 in /usr/local/lib/python3.7
Requirement already satisfied, skipping upgrade: ipython-genutils in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied, skipping upgrade: jupyter-client in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied, skipping upgrade: tornado>=4.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied, skipping upgrade: pickleshare in /usr/local/lib/python3.7/dist-packag
```

```
Requirement already satisfied, skipping upgrade: picklesha
Requirement already satisfied, skipping upgrade: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python
Requirement already satisfied, skipping upgrade: setuptools>=18.5 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied, skipping upgrade: simplegeneric>0.8 in /usr/local/lib/python3.7/dist-
Requirement already satisfied, skipping upgrade: pygments in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied, skipping upgrade: decorator in /usr/local/lib/python3.7/dist-package
Requirement already satisfied, skipping upgrade: pexpect; sys_platform != "win32" in /usr/local/lib/
Requirement already satisfied, skipping upgrade: PyWavelets in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied, skipping upgrade: terminado>=0.8.1 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied, skipping upgrade: Send2Trash in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied, skipping upgrade: nbconvert in /usr/local/lib/python3.7/dist-package
Requirement already satisfied, skipping upgrade: pyzmq>=13 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied, skipping upgrade: wcwidth in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied, skipping upgrade: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied, skipping upgrade: defusedxml in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied, skipping upgrade: bleach in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied, skipping upgrade: testpath in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied, skipping upgrade: pandocfilters>=1.4.1 in /usr/local/lib/python3.7/di
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_auc_score, roc_cu
from sklearn.utils.multiclass import unique_labels
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import eli5
from eli5.sklearn import PermutationImportance
import category_encoders as ce
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

```
df= pd.read_csv(path, sep=",")
```

```
print(df.shape)
```

```
(361741, 24)
```

```
df.head()
```

	CrimeID	Case Number	Date	Month	Day	Block	IUCR	Primary Ty
0	10029021	HY218730	4/12/2015	December	Friday	019XX W 21ST ST	1025	ARSC
1	10109149	HY297697	6/10/2015	October	Tuesday	034XX N OSCEOLA AVE	1792	KIDNAPPIN
2	10026582	HY216050	4/09/2015	September	Friday	022XX N LAWLER AVE	910	MOTC VEHICI THEI
3	10019181	HY208405	4/02/2015	February	Wednesday	087XX S SAGINAW AVE	041A	BATTEI
4	10029325	HY219256	4/11/2015	November	Wednesday	0000X S HALSTED	860	THEI

df.describe()

	CrimeID	Beat	District	Ward	Community Area	X Co
count	3.617410e+05	361741.000000	361740.000000	361737.000000	361731.000000	3.608
mean	9.154340e+06	1153.135536	11.270728	22.847265	37.595995	1.164
std	4.408294e+05	692.830525	6.912999	13.753419	21.464206	1.957
min	2.085900e+04	111.000000	1.000000	1.000000	0.000000	0.000
25%	8.785609e+06	613.000000	6.000000	10.000000	23.000000	1.152
50%	9.147912e+06	1024.000000	10.000000	23.000000	32.000000	1.165
75%	9.521796e+06	1712.000000	17.000000	34.000000	57.000000	1.176
max	1.055035e+07	2535.000000	31.000000	50.000000	77.000000	1.205

df.describe(exclude='number').T.sort_values(by='unique')

	count	unique	top	freq
Arrest	361741	2	False	259594
Domestic	361741	2	False	308580
Day	361741	7	Tuesday	52887
Month	361741	12	January	35501
FBI Code	361741	26	6	81970
Primary Type	361741	32	THEFT	81970

```
#Check for null values
df.isnull().sum()
```

```
CrimeID                0
Case Number            0
Date                   0
Month                  0
Day                    0
Block                  0
IUCR                   0
Primary Type           0
Description            0
Location Description   215
Arrest                 0
Domestic               0
Beat                   0
District               1
Ward                   4
Community Area         10
FBI Code               0
X Coordinate           873
Y Coordinate           873
Year                   0
Updated On             0
Latitude               873
Longitude              873
Location               873
dtype: int64
```

```
print('Just', (873/361741)*100,'% of the data are null values. \nFor this reason We will drop all these rows')
```

```
Just 0.24133288734204858 % of the data are null values.
For this reason We will drop all these rows
```

```
#Droping null values
df= df.dropna()
```

```
#size after dropping nan values
df.shape
```

```
(360646, 24)
```

```
df.dtypes
```

```
CrimeID                int64
Case Number            object
Date                   object
Month                  object
Day                    object
Block                  object
IUCR                   object
Primary Type           object
Description            object
Location Description   object
Arrest                 bool
Domestic               bool
Beat                   int64
District               float64
Ward                   float64
Community Area         float64
FBI Code               object
X Coordinate           float64
Y Coordinate           float64
Year                   int64
Updated On             object
Latitude               float64
Longitude              float64
Location               object
dtype: object
```

```
#Drop duplicate rows
df=df.drop_duplicates()
```

```
df['Year'].value_counts()
```

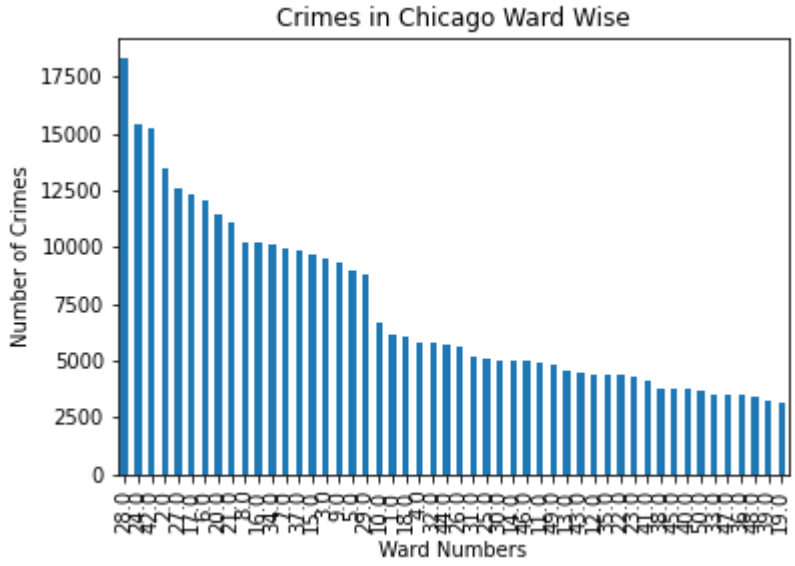
```
2012    132576
2013    121091
2014    105427
2016      1400
2015       152
Name: Year, dtype: int64
```

```
df['Arrest'].value_counts().plot(kind='pie')
plt.title("Distribution of Arrest in Chicago")
```

Text(0.5, 1.0, 'Distribution of Arrest in Chicago')

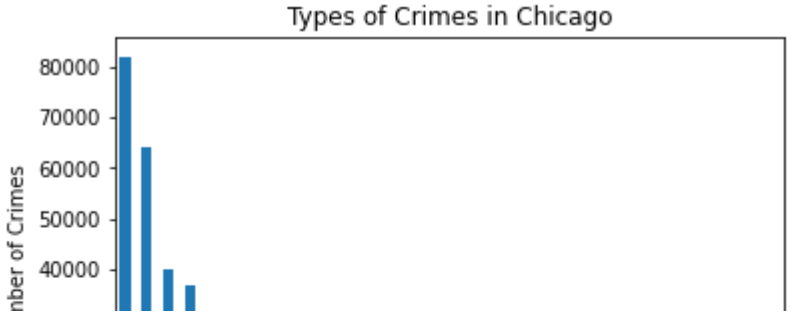

```
df['Ward'].value_counts().plot(kind='bar')
plt.ylabel("Number of Crimes")
plt.xlabel("Ward Numbers")
plt.title("Crimes in Chicago Ward Wise")
```

Text(0.5, 1.0, 'Crimes in Chicago Ward Wise')

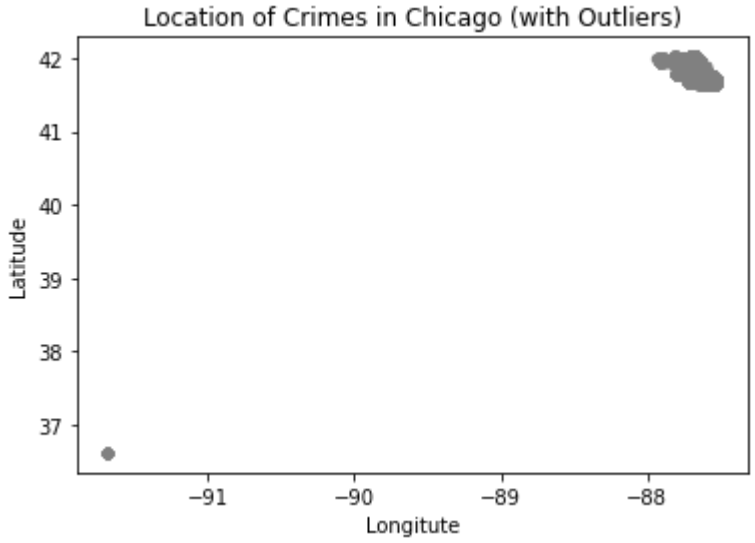



```
df['Primary Type'].value_counts().plot(kind='bar')
plt.ylabel("Number of Crimes")
plt.xlabel("Location of Crime")
plt.title("Types of Crimes in Chicago")
```

Text(0.5, 1.0, 'Types of Crimes in Chicago')




```
#Visualization of the Longitude and Latitude.
plt.scatter('Longitude', 'Latitude', c='gray', data=df, s=20)
plt.ylabel("Latitude")
plt.xlabel("Longitute")
plt.title("Location of Crimes in Chicago (with Outliers)")
plt.show();
```



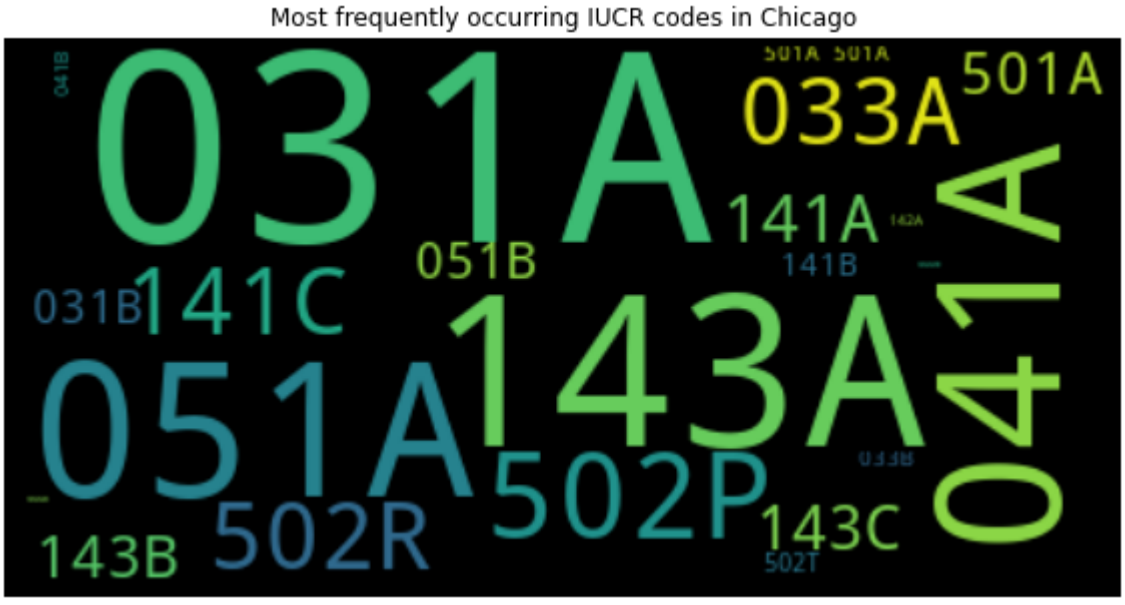

```
outliers = ( df['Location'] == '(36.619446395, -91.686565684)') # Outliers


df = df[-outliers] # Removing Outliers


plt.scatter('Longitude', 'Latitude', c='gray', data=df, s=20)
plt.ylabel("Latitude")
plt.xlabel("Longitute")
plt.title("Location of Crimes in Chicago (without Outliers)")
plt.show() #After removing Outliers
```


```
a = df.groupby(['Arrest', 'Domestic'])
a.count()
```

		CrimeID	Case Number	Date	Month	Day	Block	IUCR	Primary Type	D
Arrest	**Domestic**									
False	**False**	216324	216324	216324	216324	216324	216324	216324	216324	
	True	42497	42497	42497	42497	42497	42497	42497	42497	
True	**False**	91204	91204	91204	91204	91204	91204	91204	91204	
	True	10593	10593	10593	10593	10593	10593	10593	10593	

```
#change true and false by number to plot the data in a map with different colors
df= df.replace({False: 0, True: 1})


df.to_csv("Cleaned_Data", index=False)


fig = px.scatter_mapbox(df, lat='Latitude', lon='Longitude', color='Arrest',hover_name='Primary Type', opaci
fig.update_layout(mapbox_style='open-street-map')
fig.show()
```


```
# Join together a single string of IUCR codes
df.index = pd.DatetimeIndex(df.Date)
crime_types_code = " ".join(crime for crime in df ["IUCR"])

# Create and generate a word cloud image
crime_code_wordcloud = WordCloud().generate(crime_types_code)

# Plot wordcloud image
plt.figure (figsize=[10, 10])
plt.imshow (crime_code_wordcloud, interpolation="bilinear")
plt.axis ("off")
plt.title("Most frequently occurring IUCR codes in Chicago")
plt.show()
```




```
# Drop NaN Values in ["Block"] column
df_wordcloud = df.copy ()
df_wordcloud.dropna (axis=0, subset=["Block"], inplace=True)

# Create string of Block codes and USA flag image mask
crime_types_location = " ".join(crime for crime in df_wordcloud["Block"])
```

```
mask = np.array(Image.open("/content/drive/MyDrive/datasets/US.jpeg"))

#Create and generate a word cloud image
crime_location_wordcloud = WordCloud(background_color="white", mode="RGBA", max_words=1000, mask=mc

#Create coloring from image and plot word cloud
image_colors = ImageColorGenerator(mask)
plt.figure (figsize=[10,10])
plt.imshow (crime_location_wordcloud.recolor(color_func=image_colors), interpolation="bilinear")
plt.axis ("off")
plt.title("Locations of high Crime Rate in Chicago")
plt.show()
```


```
df['Details'] = df['Primary Type'] + "," + df ['Description']

# Group by crime details and show top 10 crimes with highest arrest rates
top_crimes = df.groupby (['Details'])['Arrest'].count()
top_crimes = pd. DataFrame(top_crimes).nlargest (10, 'Arrest').reset_index()
top_crimes = list(top_crimes ['Details'])
top_crimes
```

```
['THEFT,$500 AND UNDER',
 'BATTERY,DOMESTIC BATTERY SIMPLE',
 'BATTERY,SIMPLE',
 'NARCOTICS,POSS: CANNABIS 30GMS OR LESS',
 'THEFT,OVER $500',
 'CRIMINAL DAMAGE,TO PROPERTY',
 'CRIMINAL DAMAGE,TO VEHICLE',
 'ASSAULT,SIMPLE',
 'BURGLARY,FORCIBLE ENTRY',
 'MOTOR VEHICLE THEFT,AUTOMOBILE']
```

```
#Spliting Data into Train, Validate andTest

target = 'Arrest'
features = df.columns.drop([target,'CrimeID'])

X = df[features]
y = df[target]

X.shape,y.shape
```

```
((360618, 23), (360618,))
```

```
X_trainval, X_test, y_trainval, y_test = train_test_split(
   X, y, train_size=0.80, test_size=0.20, random_state=42)
X_trainval.shape, X_test.shape, y_trainval.shape, y_test.shape
```

```
((288494, 23), (72124, 23), (288494,), (72124,))
```

```
X_train, X_val, y_train, y_val = train_test_split(
   X_trainval, y_trainval, test_size=0.2, random_state=42)

print('X_train shape', X_train.shape)
print('y_train shape', y_train.shape)
print('X_val shape', X_val.shape)
print('y_val shape', y_val.shape)
print('X_test shape', X_test.shape)
print('y_test shape', y_test.shape)
```

```
X_train shape (230795, 23)
y_train shape (230795,)
X_val shape (57699, 23)
y_val shape (57699,)
X_test shape (72124, 23)
y_test shape (72124,)
```

```
y_train.value_counts(normalize=True)
```

```
0    0.718122
1    0.281878
Name: Arrest, dtype: float64
```

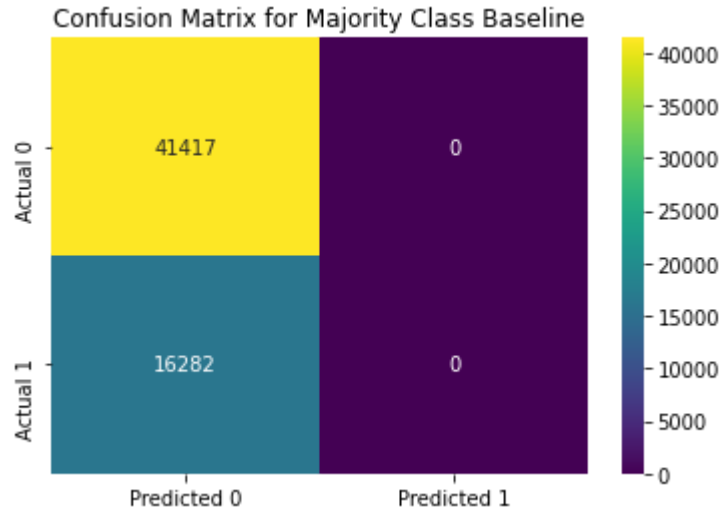
```
majority_class = y_train.mode()[0]
y_pred = np.full_like(y_val, fill_value=majority_class)
accuracy_score(y_val, y_pred)
```

```
0.7178114005442036
```

```
def plot_confusion_matrix(y_true, y_pred):
```

```
    labels = unique_labels(y_true)
    columns = [f'Predicted {label}' for label in labels]
    index = [f'Actual {label}' for label in labels]
    table = pd.DataFrame(confusion_matrix(y_true, y_pred),
                         columns=columns, index=index)
    return sns.heatmap(table, annot=True, fmt='d', cmap='viridis')


plot_confusion_matrix(y_val, y_pred);
plt.title("Confusion Matrix for Majority Class Baseline")
```

```
Text(0.5, 1.0, 'Confusion Matrix for Majority Class Baseline')
```




```
confusion_matrix(y_val, y_pred)
```

```
array([[41417,     0],
       [16282,     0]])
```

```
conf_matrix=pd.DataFrame(confusion_matrix(y_val, y_pred))
conf_matrix.index = ['Actual False','Actual True']
conf_matrix.columns = ['Predicted False','Predicted True']
conf_matrix
```

	Predicted False	Predicted True
Actual False	41417	0
Actual True	16282	0

```
#Get precision & recall for majority class baseline
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      1.00      0.84     41417
           1       0.00      0.00      0.00     16282
```

```
              accuracy                           0.72     57699
             macro avg       0.36      0.50      0.42     57699
          weighted avg       0.52      0.72      0.60     57699

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWar

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero
```

```
y_pred_proba = np.full_like(y_val, fill_value=1.00)
roc_auc_score(y_val, y_pred_proba)
```

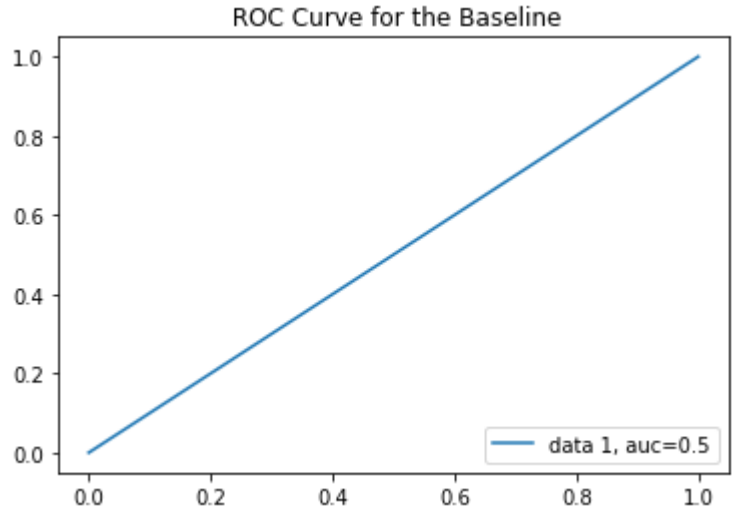
```
0.5
```

```
y_pred_proba = np.full_like(y_val, fill_value=0)
auc=roc_auc_score(y_val, y_pred_proba)
auc
```

```
0.5
```

```
 fpr,tpr,thresholds=roc_curve(y_val, y_pred_proba)
```

```
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.title(" ROC Curve for the Baseline")
plt.show()
```




```
#Logistic Regression
lr = make_pipeline(
    ce.OrdinalEncoder(),
    SimpleImputer(strategy='mean'),
    LogisticRegression(n_jobs=-1,solver='lbfgs', random_state=42, penalty='l2'))
```

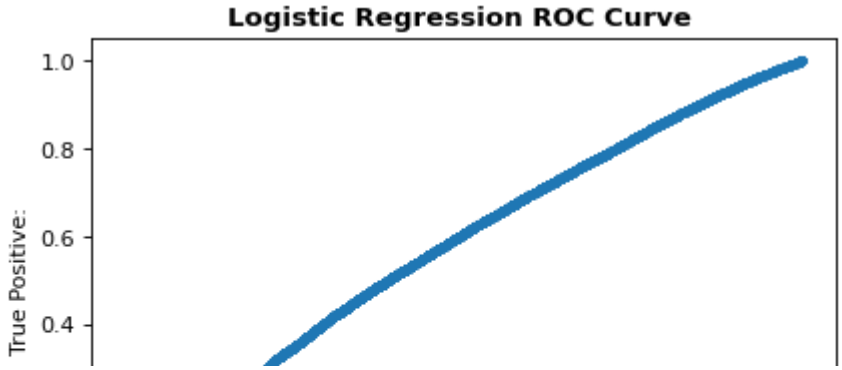
```
lr.fit(X_train, y_train)
```

```
        Pipeline(memory=None,
                 steps=[('ordinalencoder',
                         OrdinalEncoder(cols=['Case Number', 'Date', 'Month', 'Day',
                                              'Block', 'IUCR', 'Primary Type',
                                              'Description', 'Location Description',
                                              'FBI Code', 'Updated On', 'Location',
                                              'Details'],
                                        drop_invariant=False, handle_missing='value',
                                        handle_unknown='value',
                                        mapping=[{'col': 'Case Number',
                                                  'data_type': dtype('O'),
                                                  'mapping': HV281376        1
        HV...
                         SimpleImputer(add_indicator=False, copy=True, fill_value=None,
                                       missing_values=nan, strategy='mean',
                                       verbose=0)),
                        ('logisticregression',
                         LogisticRegression(C=1.0, class_weight=None, dual=False,
                                            fit_intercept=True, intercept_scaling=1,
                                            l1_ratio=None, max_iter=100,
                                            multi_class='auto', n_jobs=-1, penalty='l2',
                                            random_state=42, solver='lbfgs', tol=0.0001,
                                            verbose=0, warm_start=False))],
                 verbose=False)
```

```
y_pred_val = lr.predict(X_val)
y_pred_test = lr.predict(X_test)
print('Accuracy for the validation data= ',accuracy_score(y_val, y_pred_val))
print('Accuracy for the test data= ',accuracy_score(y_test, y_pred_test))
```

```
        Accuracy for the validation data=  0.7178114005442036
        Accuracy for the test data=  0.7163357550884588
```

```
lr_fp,lr_tp,thresholds = roc_curve(y_val,lr.predict_proba(X_val)[:,1])

auc_lr=roc_auc_score(y_val, lr.predict_proba(X_val)[:,1])

fig = plt.figure(dpi=80)
plt.scatter(lr_fp,lr_tp, s=6,label="AUC="+str(auc_lr))
plt.title('Logistic Regression ROC Curve ', fontweight = 'bold', fontsize = 12)
plt.legend(loc=4)
plt.xlabel('False Positive:', fontsize = 10)
plt.ylabel('True Positive:', fontsize = 10);
```



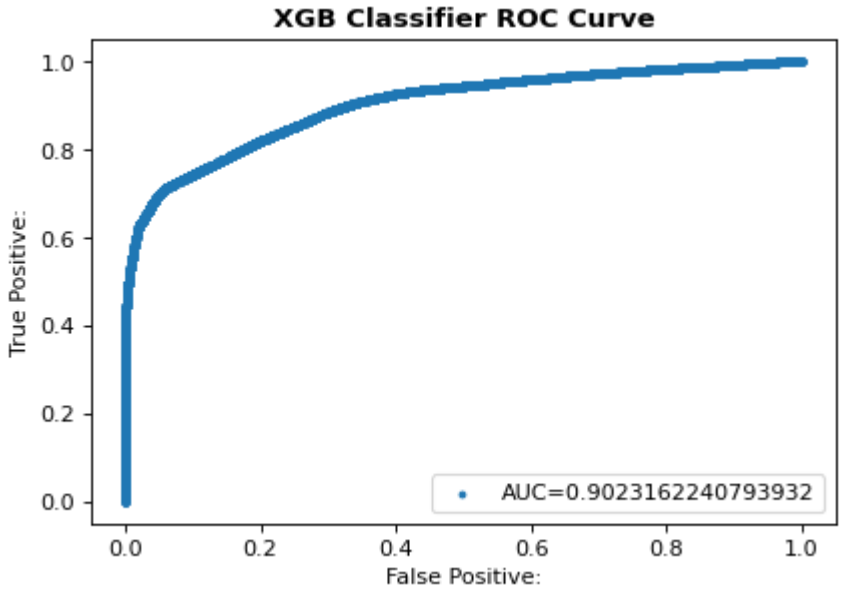

```
xgb = make_pipeline(
   ce.OrdinalEncoder(),
   SimpleImputer(strategy='mean'),
   XGBClassifier(n_estimators=100, random_state=42, n_jobs=-1)
)

xgb.fit(X_train, y_train)
```

```
Pipeline(memory=None,
         steps=[('ordinalencoder',
                 OrdinalEncoder(cols=['Case Number', 'Date', 'Month', 'Day',
                                      'Block', 'IUCR', 'Primary Type',
                                      'Description', 'Location Description',
                                      'FBI Code', 'Updated On', 'Location',
                                      'Details'],
                                drop_invariant=False, handle_missing='value',
                                handle_unknown='value',
                                mapping=[{'col': 'Case Number',
                                          'data_type': dtype('O'),
                                          'mapping': HV281376        1
HV...
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, gamma=0, learning_rate=0.1,
                               max_delta_step=0, max_depth=3,
                               min_child_weight=1, missing=None,
                               n_estimators=100, n_jobs=-1, nthread=None,
                               objective='binary:logistic', random_state=42,
                               reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                               seed=None, silent=None, subsample=1,
                               verbosity=1))],
         verbose=False)
```

```
# XGBClassifier
y_pred_val = xgb.predict(X_val)
y_pred_test = xgb.predict(X_test)
print('Accuracy for the validation data= ',accuracy_score(y_val, y_pred_val))
print('Accuracy for the test data= ',accuracy_score(y_test, y_pred_test))
```

```
Accuracy for the validation data=  0.8782647879512643
Accuracy for the test data=  0.8784177250291165
```

```
xgb_fp,xgb_tp,thresholds = roc_curve(y_val,xgb.predict_proba(X_val)[:,1])

auc_xgb=roc_auc_score(y_val, xgb.predict_proba(X_val)[:,1])

fig = plt.figure(dpi=80)
plt.scatter(xgb_fp,xgb_tp, s=6,label="AUC="+str(auc_xgb))
plt.title('XGB Classifier ROC Curve', fontweight = 'bold', fontsize = 12)
plt.legend(loc=4)
plt.xlabel('False Positive:', fontsize = 10)
plt.ylabel('True Positive:', fontsize = 10);
```

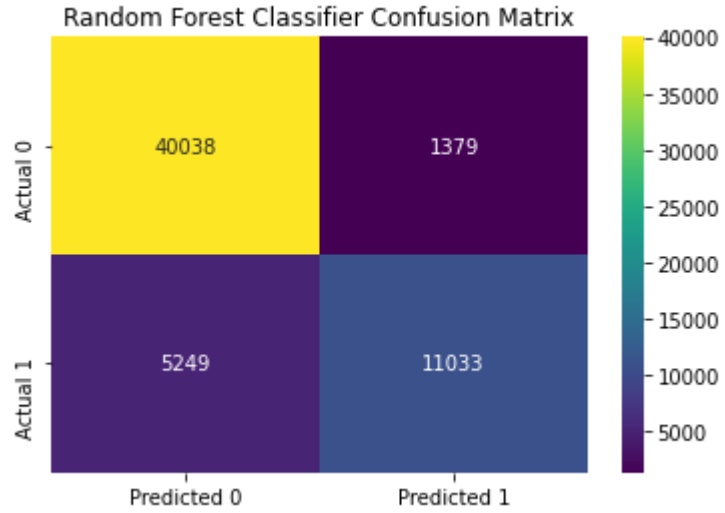



```
rf = make_pipeline(
    ce.OrdinalEncoder(),
    SimpleImputer(strategy='mean'),
    RandomForestClassifier(n_estimators=100, n_jobs=-1, random_state=42)
)

rf.fit(X_train, y_train);


#Random Forest Classifier
y_pred_val = rf.predict(X_val)
y_pred_test = rf.predict(X_test)
print('Accuracy for the validation data= ',accuracy_score(y_val, y_pred_val))
print('Accuracy for the test data= ',accuracy_score(y_test, y_pred_test))
```

```
Accuracy for the validation data=  0.8851279918196155
Accuracy for the test data=  0.8849481448616272
```

```
#Confusion Matrix
plot_confusion_matrix(y_val, y_pred_val);
plt.title("Random Forest Classifier Confusion Matrix")
```

Text(0.5, 1.0, 'Random Forest Classifier Confusion Matrix')



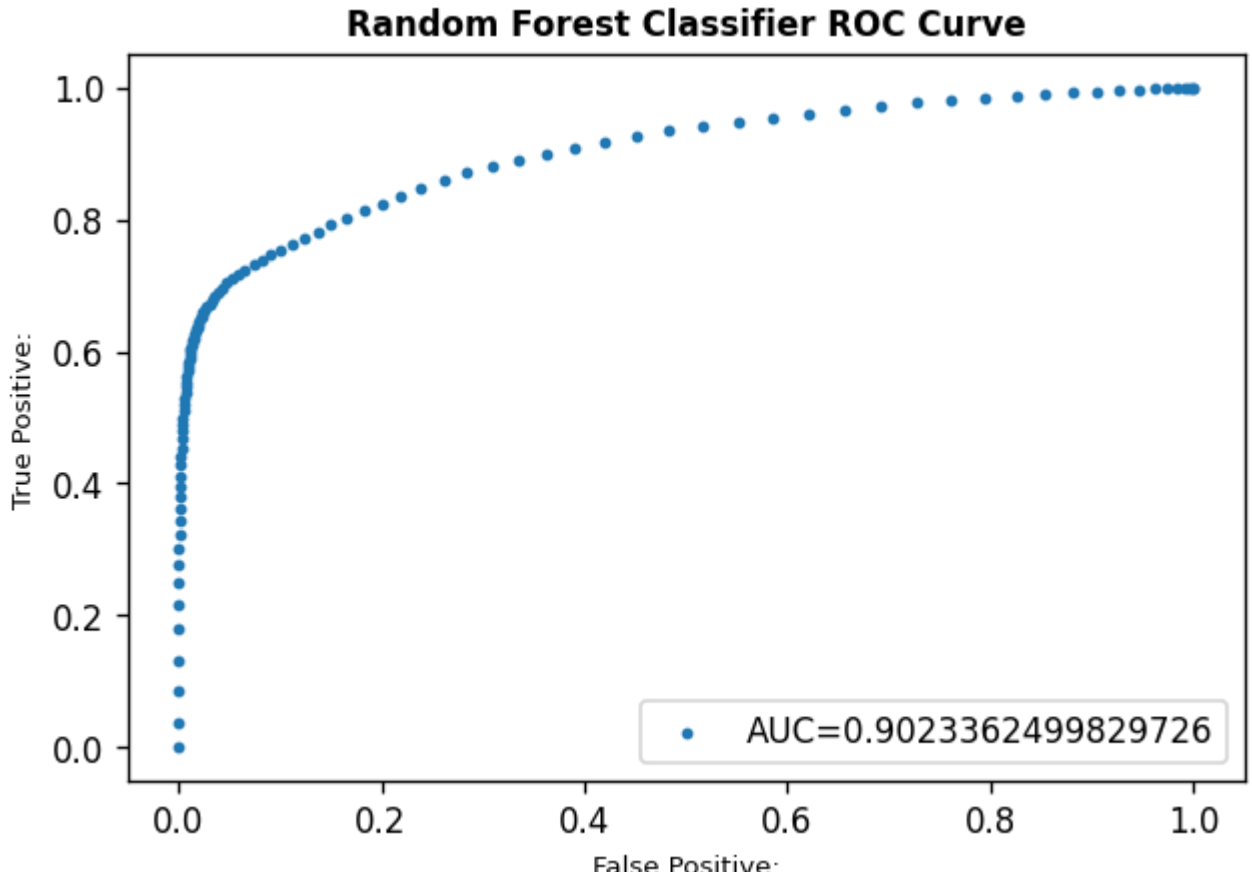

```
conf_matrix=pd.DataFrame(confusion_matrix(y_val, y_pred_val))
conf_matrix.index = ['Actual False','Actual True']
conf_matrix.columns = ['Predicted False','Predicted True']
conf_matrix
```

	Predicted False	Predicted True
Actual False	40038	1379
Actual True	5249	11033

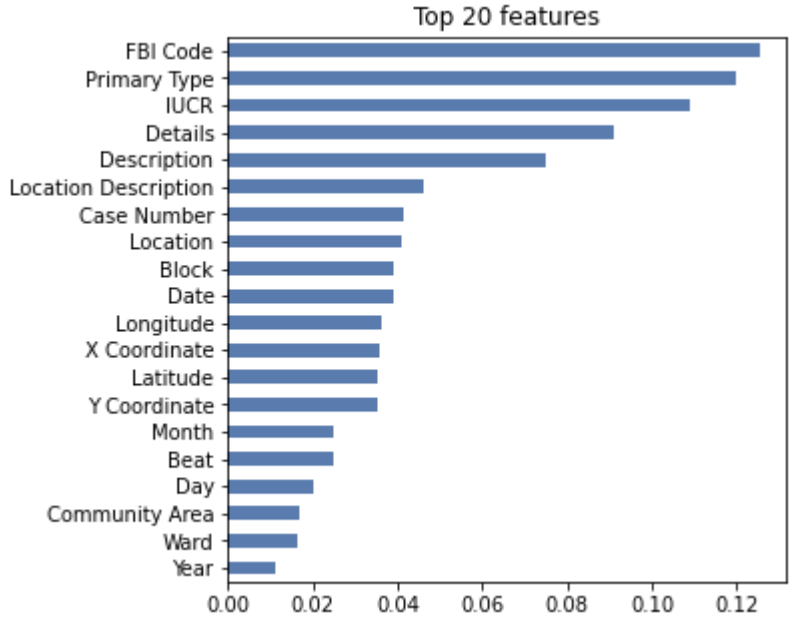
```
print(classification_report(y_val, y_pred_val))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92     41417
           1       0.89      0.68      0.77     16282

    accuracy                           0.89     57699
   macro avg       0.89      0.82      0.85     57699
weighted avg       0.89      0.89      0.88     57699
```

```
rf_fp,rf_tp,thresholds = roc_curve(y_val,rf.predict_proba(X_val)[:,1])


auc_rf=roc_auc_score(y_val, rf.predict_proba(X_val)[:,1])

fig = plt.figure(dpi=120)
plt.scatter(rf_fp,rf_tp,s=6,label="AUC="+str(auc_rf))
plt.title('Random Forest Classifier ROC Curve', fontweight = 'bold', fontsize = 10)
plt.legend(loc=4)
plt.xlabel('False Positive:', fontsize = 8)
plt.ylabel('True Positive:', fontsize = 8);
```




```
#Feature importances
rf_steps = rf.named_steps['randomforestclassifier']
importances = pd.Series(rf_steps.feature_importances_, X_train.columns)

n = 20
plt.figure(figsize=(5,5))
plt.title(f'Top {n} features')
importances.sort_values()[-n:].plot.barh(color='#5a7bad');
```



```
column = 'FBI Code'

#Fitting without FBI Code
rf_without = make_pipeline(
    ce.OrdinalEncoder(),
    SimpleImputer(strategy='median'),
    RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
)
rf_without.fit(X_train.drop(columns=column), y_train)
score_without = rf_without.score(X_val.drop(columns=column), y_val)
print(f'Validation Accuracy without {column}: {score_without}')

#Fitting with FBI Code
rf_with = make_pipeline(
    ce.OrdinalEncoder(),
    SimpleImputer(strategy='median'),
    RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
)
rf_with.fit(X_train, y_train)
score_with = rf_with.score(X_val, y_val)
print(f'Validation Accuracy with {column}: {score_with}')
print(f'Drop-Column Importance for {column}: {score_with - score_without}')
```

```
        Validation Accuracy without FBI Code: 0.8846947087471186
        Validation Accuracy with FBI Code: 0.8851279918196155
        Drop-Column Importance for FBI Code: 0.0004332830724969039
```

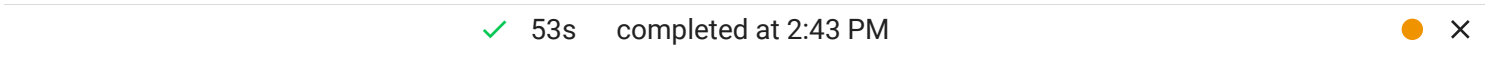
```
#Using Permutation Importance ELI5 Library
transformers = make_pipeline(
    ce.OrdinalEncoder(),
    SimpleImputer(strategy='median')
)

X_train_transformed = transformers.fit_transform(X_train)
X_val_transformed = transformers.transform(X_val)

randomforest = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
randomforest.fit(X_train_transformed, y_train)
```

```
        RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                               criterion='gini', max_depth=None, max_features='auto',
                               max_leaf_nodes=None, max_samples=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=100,
                               n_jobs=-1, oob_score=False, random_state=42, verbose=0,
                               warm_start=False)
```

```
#permute features
permuter = PermutationImportance(
    randomforest,
    scoring='accuracy'
```

```
    scoring= accuracy ,
    n_iter=2,
    random_state=42
)

permuter.fit(X_val_transformed, y_val)
feature_names = X_val.columns.tolist()

eli5.show_weights(
    permuter,
    feature_names=feature_names
)
```

Weight	Feature
0.0546 ± 0.0010	Primary Type
0.0341 ± 0.0012	FBI Code
0.0252 ± 0.0001	IUCR
0.0223 ± 0.0003	Description
0.0121 ± 0.0000	Details
0.0058 ± 0.0012	Location Description
0.0053 ± 0.0001	Domestic
0.0018 ± 0.0007	Location
0.0016 ± 0.0002	Longitude
0.0016 ± 0.0006	X Coordinate
0.0011 ± 0.0004	Latitude
0.0007 ± 0.0000	Y Coordinate
0.0005 ± 0.0002	Community Area
0.0004 ± 0.0004	Block
0.0000 ± 0.0006	Month
0.0000 ± 0.0000	Updated On
0 ± 0.0000	Case Number
-0.0000 ± 0.0005	Day
-0.0001 ± 0.0001	Date
-0.0003 ± 0.0001	Ward
… 3 more …	


53s
completed at 2:43 PM