Supervised Machine Learning Model Integration Using Flask

Project # 4

By: Mubashira Qari

Data Source:

- Data records from the NYPD Stop, Question, and Frisk database are available for download from the links provided below. Data is made available in CSV format.
- Publications, Reports - NYPD

Problem Worth Solving, Analyzing, and Visualizing:

- ‘Stop, Question, and Frisk’ database has multiple features which are important in affecting the outcome. The aim of this machine learning project is to predict, whether the summons is issued or not for the suspect.
- Also to find out which features, mostly contribute to the arrest of the suspect.

Implementation:

The project implementation is done using Scikit-learn library in machine learning, along with the following.

- Python Pandas
- Flask Web Framework
- Python Matplotlib
- HTML/CSS/Bootstrap
- JavaScript D3.js
- SQL Database
- Tableau

Host application using Heroku or amazon cloud for deployment

Exploratory Data Analysis (EDA)

- The 2017, 2018 and 2019 ‘Stop, Question and Frisk’ datasets are merged together to perform the preprocessing steps and prepare for the training dataset.
- The 2020 ‘Stop, Question and Frisk’ dataset is retrieved to perform the preprocessing steps and prepare for the testing dataset.

```
# Import our dependencies
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import datetime
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from pathlib import Path
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, OneHotEncoder
```

```
# Loading the dataset from the resources folder

sqf_2020_df = pd.read_csv(Path('Resources/sqf-2020.csv'))
sqf_2019_df = pd.read_csv(Path('Resources/sqf-2019.csv'))
sqf_2018_df = pd.read_csv(Path('Resources/sqf-2018.csv'))
sqf_2017_df = pd.read_csv(Path('Resources/sqf-2017.csv'))

dataFrame_list = [sqf_2019_df, sqf_2018_df, sqf_2017_df]
merge_df = pd.concat(dataFrame_list)
merge_df
```

	STOP_ID_ANONY	STOP_FRISK_DATE	STOP_FRISK_TIME	YEAR2	MONTH2	DAY2	STOP_WAS_INITIATED	RECORD_STATUS_CODE	ISSUING_OFFI
0	1.0	1/2/2019	14:30:00	2019.0	January	Wednesday	Based on C/W on Scene	APP	
1	2.0	1/8/2019	2:30:00	2019.0	January	Tuesday	Based on Self Initiated	APP	
2	3.0	1/12/2019	16:54:00	2019.0	January	Saturday	Based on Radio Run	APP	
3	4.0	1/14/2019	21:21:00	2019.0	January	Monday	Based on Radio Run	APP	
4	5.0	1/15/2019	18:50:00	2019.0	January	Tuesday	Based on Radio Run	APP	

Data Preprocessing (ETL):

- Import our dependencies
- Loading the dataset from the resources folder
- Merging 2017,18, &19 for training data and keeping 2020 for testing
- Removing unwanted text from columns
- Find null values
- Function checking for missing values
- Generate our categorical variable lists
- Check the number of unique values in each column

Data Preprocessing (ETL):

```
# Removing unwanted text from columns
merge_df = merge_df.replace('PM','', regex=True)
merge_df = merge_df.replace('AM','', regex=True)
sqf_2020_df = sqf_2020_df.replace('AM','', regex=True)
sqf_2020_df = sqf_2020_df.replace('PM','', regex=True)
merge_df
sqf_2020_df
```

	STOP_ID	STOP_FRISK_DATE	STOP_FRISK_TIME	YEAR2	MONTH2	DAY2	STOP_WAS_INITIATED	RECORD_STATUS_CODE
0	1	1/1/2020	1:12:00	2020	January	Wednesday	Based on Radio Run	APP
1	2	1/1/2020	1:11:00	2020	January	Wednesday	Based on Radio Run	APP
2	3	1/1/2020	1:11:00	2020	January	Wednesday	Based on Radio Run	APP
3	4	1/1/2020	10:18:00	2020	January	Wednesday	Based on Radio Run	APP

Data Preprocessing (ETL):

```
# Find null values
for column in merge_df.columns:
    print(f"Column {column} has {merge_df[column].isnull().sum()} null values")
```

```
Column STOP_ID_ANONY has 22637 null values
Column STOP_FRISK_DATE has 0 null values
Column STOP_FRISK_TIME has 11013 null values
Column YEAR2 has 0 null values
Column MONTH2 has 0 null values
Column DAY2 has 0 null values
```

Data Preprocessing (ETL):

```
# Function checking for missing values
def missing_values_table(df):
    mis_val = df.isnull().sum()
    mis_val_percent = 100 * df.isnull().sum() / len(df)
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
    mis_val_table_ren_columns = mis_val_table.rename(
    columns = {0 : 'Missing Values', 1 : '% of Total Values'})
    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
    '% of Total Values', ascending=False).round(1)
    print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
        "There are " + str(mis_val_table_ren_columns.shape[0]) +
            " columns that have missing values.")
    return mis_val_table_ren_columns
```

```
missing_values_table(merge_df)
```

```
Your selected dataframe has 87 columns.
There are 17 columns that have missing values.
```

	Missing Values	% of Total Values
Stop Frisk Time	25090	69.5
STOP_ID_ANONY	22637	62.7

Data Preprocessing (ETL):

```
# Generate our categorical variable lists
float_columns = merge_df.dtypes[merge_df.dtypes == "float"].index.tolist()
len(float_columns)
```

```
5
```

```
# Check the number of unique values in each column
merge_df[float_columns]
```

	STOP_ID_ANONY	YEAR2	OBSERVED_DURATION_MINUTES	STOP_DURATION_MINUTES	STOP_FRISK_ID
0	1.0	2019.0	1.0	10.0	NaN
1	2.0	2019.0	1.0	10.0	NaN
2	3.0	2019.0	1.0	4.0	NaN
3	4.0	2019.0	0.0	5.0	NaN
4	5.0	2019.0	1.0	5.0	NaN

Data Preprocessing (ETL):

```
# Finding the value_counts of each column
for c in merge_df.columns:
    print("---- %s ---" % c)
    print(merge_df[c].value_counts())
```

```
---- STOP_ID_ANONY ---
11783.0    1
6684.0     1
8095.0     1
11717.0    1
9860.0     1
          ..
10094.0    1
2491.0     1
5629.0     1
11521.0    1
1.0        1
Name: STOP_ID_ANONY, Length: 13459, dtype: int64
---- STOP_FRISK_DATE ---
3/31/2018     106
4/3/2019       84
2017-10-31     81
5/10/2019      74
3/23/2019      72
              ...
```

Data Preprocessing (ETL):

```
# Drop the null columns where mostly values are null
sqf_2020_df = sqf_2020_df.dropna(axis='columns', how='all')
merge_df = merge_df.dropna(axis='columns', how='all')
merge_df
sqf_2020_df
```

	STOP_ID	STOP_FRISK_TIME	YEAR2	MONTH2	DAY2	STOP_WAS_INITIATED	RECORD_STATUS_CODE	I
0	1	1:12:00	2020	January	Wednesday	Based on Radio Run	APP	
1	2	1:11:00	2020	January	Wednesday	Based on Radio Run	APP	
2	3	1:11:00	2020	January	Wednesday	Based on Radio Run	APP	
3	4	10:18:00	2020	January	Wednesday	Based on Radio Run	APP	
4	5	8:45:00	2020	January	Wednesday	Based on Radio Run	APP	

Data Preprocessing (ETL):

- Convert time into seconds to have integer values for machine learning
- Replacing the text strings with zeros in the integer columns
- Replacing two different category names with one category
- Removing special characters from the values
- Rename the columns
- Dropping columns which are not playing any role in the feature importances of the datasets
- Converting to correct data type
- Transform 'Summons Issued' outcome column into binary
- Binning of categories in columns is performed
- Files are saved as a csv files for further applications of data

Data Preprocessing (ETL):

```
# Convert time into seconds to have integer values for machine learning
sqf_2020_df["TIME"] = sqf_2020_df["STOP_FRISK_TIME"].astype(str)
merge_df["TIME"] = merge_df["STOP_FRISK_TIME"].astype(str) + merge_df["Stop Frisk Time"].astype(str)
merge_df["TIME"]
```

```
0        14:30:00nan
1         2:30:00nan
2        16:54:00nan
3        21:21:00nan
4        18:50:00nan
```

Data Preprocessing (ETL):

```
# Removing special characters from the values
merge_df.SUSPECT_WEIGHT = merge_df.SUSPECT_WEIGHT.str.replace('[^\d]+', '')
merge_df.SUSPECT_HEIGHT = merge_df.SUSPECT_HEIGHT.str.replace('[^\d]+', '')
merge_df.SUSPECT_REPORTED_AGE = merge_df.SUSPECT_REPORTED_AGE.str.replace('[^\d]+', '')

sqf_2020_df.SUSPECT_WEIGHT = sqf_2020_df.SUSPECT_WEIGHT.str.replace('[^\d]+', '')
sqf_2020_df.SUSPECT_HEIGHT = sqf_2020_df.SUSPECT_HEIGHT.str.replace('[^\d]+', '')
sqf_2020_df.SUSPECT_REPORTED_AGE = sqf_2020_df.SUSPECT_REPORTED_AGE.str.replace('[^\d]+', '')
sqf_2020_df.SUSPECT_REPORTED_AGE
```

```
0       18
1       18
2       17
3       33
```

Data Preprocessing (ETL):

```
# Converting to correct data type
merge_df['SUSPECT_REPORTED_AGE'] = merge_df['SUSPECT_REPORTED_AGE'].apply(pd.to_numeric)
merge_df['SUSPECT_HEIGHT'] = merge_df['SUSPECT_HEIGHT'].apply(pd.to_numeric)
merge_df['SUSPECT_WEIGHT'] = merge_df['SUSPECT_WEIGHT'].apply(pd.to_numeric)
merge_df['STOP_LOCATION_X'] = merge_df['STOP_LOCATION_X'].apply(pd.to_numeric)
merge_df['STOP_LOCATION_Y'] = merge_df['STOP_LOCATION_Y'].apply(pd.to_numeric)

# Converting to correct data type
sqf_2020_df['SUSPECT_REPORTED_AGE'] = sqf_2020_df['SUSPECT_REPORTED_AGE'].apply(pd.to_numeric)
sqf_2020_df['SUSPECT_HEIGHT'] = sqf_2020_df['SUSPECT_HEIGHT'].apply(pd.to_numeric)
sqf_2020_df['SUSPECT_WEIGHT'] = sqf_2020_df['SUSPECT_WEIGHT'].apply(pd.to_numeric)
sqf_2020_df['STOP_LOCATION_X'] = sqf_2020_df['STOP_LOCATION_X'].apply(pd.to_numeric)
sqf_2020_df['STOP_LOCATION_Y'] = sqf_2020_df['STOP_LOCATION_Y'].apply(pd.to_numeric)
```

Data Preprocessing (ETL):

Binning of Column Values is Performed:

```
merge_df['ISSUING_OFFICER_RANK'].value_counts
```

```
<bound method IndexOpsMixin.value_counts of 0         POM
1         POM
2         POM
3         POM
4         POM
         ...
11624     POM
11625     POM
11626     POM
11627     POM
11628     POM
Name: ISSUING_OFFICER_RANK, Length: 36096, dtype: object>
```

```
merge_df['ISSUING_OFFICER_RANK'].nunique()
sqf_2020_df['ISSUING_OFFICER_RANK'].nunique()
```

```
14
```

```
# Look at 'STOP_WAS_INITIATED' value counts for binning
report_value_count_merge_data = merge_df['STOP_WAS_INITIATED'].value_counts()
# a_list[0]
report_value_count_merge_data.index

# Look at 'STOP_WAS_INITIATED' value counts for binning
report_value_count_2020_data = sqf_2020_df['STOP_WAS_INITIATED'].value_counts()
# a_list[0]
report_value_count_2020_data.index
```

Data Preprocessing (ETL):

```
# Binning 'STOP_WAS_INITIATED'
# Getting the values that need to be binned in 'STOP_WAS_INITIATED'
report_value_count_merge_data = report_value_count_merge_data.loc[~report_value_count_merge_data.index.isin(['Based on Radio Run
report_value_count_merge_data.index
report_to_replace_merge_data = report_value_count_merge_data.index.tolist()
report_to_replace_merge_data

# Getting the values that need to be binned in 'STOP_WAS_INITIATED'
report_value_count_2020_data = report_value_count_2020_data.loc[~report_value_count_2020_data.index.isin(['Based on Radio Run',
report_value_count_2020_data.index
report_to_replace_2020_data = report_value_count_2020_data.index.tolist()
report_to_replace_2020_data

# Replace in dataframe
for report in report_to_replace_merge_data:
    merge_df['STOP_WAS_INITIATED'] = merge_df['STOP_WAS_INITIATED'].replace(report,"Based on C/W on Scene")
# Check to make sure binning was successful
merge_df['STOP_WAS_INITIATED'].value_counts()

# Replace in dataframe
for report in report_to_replace_2020_data:
    sqf_2020_df['STOP_WAS_INITIATED'] = sqf_2020_df['STOP_WAS_INITIATED'].replace(report,"Based on C/W on Scene")
# Check to make sure binning was successful
sqf_2020_df['STOP_WAS_INITIATED'].value_counts()
```

```
Based on Radio Run         6424
Based on Self Initiated    1781
Based on C/W on Scene      1339
Name: STOP_WAS_INITIATED, dtype: int64
```

Data Preprocessing (ETL):

```
# Save as a csv for further applications of data
# Note to avoid any issues later, use encoding="utf-8"
sqf_2020_df.to_csv("posgres_2020_df.csv", encoding="utf-8", index=False)
```

```
# Save as a csv
# Note to avoid any issues later, use encoding="utf-8"
merge_df.to_csv("posgres_merged_df.csv", encoding="utf-8", index=False)
```

Data Preprocessing (ETL):

- Converting categorical data to numeric using Label Encoder for unhirarchial data value columns
- Convert categorical data to numeric and separate target feature for training data using get_dummies encoding method for the entire dataframe
- Scaling the data by using the StandardScaler() function
- Assigning first the outcome as: SUMMONS_ISSUED_FLAG
- Logistic Model fitting
- Random Forest Classifier fitting
- Fitting Different Models on Imbalance Data for Optimization
- Import an Extremely Random Trees classifier
- Import an Adaptive Boosting classifier
- Applying K-nearest neighbors

Data Preprocessing (ETL):

Converting Categorical Data to Numeric:

```
columns_to_encode = ['OFFICER_EXPLAINED_STOP_FLAG',
                     'SUSPECT_ARRESTED_FLAG',
                     'SUMMONS_ISSUED_FLAG',
                     'OFFICER_IN_UNIFORM_FLAG',
                     'FRISKED_FLAG',
                     'SEARCHED_FLAG',
                     'WEAPON_FOUND_FLAG',
                     'MONTH',
                     'DAY']
for column in columns_to_encode:
    # label encoding max categories columns
    merge_df[column] = LabelEncoder().fit_transform(merge_df[column])

    # label encoding max categories columns
    sqf_2020_df[column] = LabelEncoder().fit_transform(sqf_2020_df[column])
sqf_2020_df
```

	YEAR	MONTH	DAY	STOP_WAS_INITIATED	ISSUING_OFFICER_RANK	OBSERVED_DURATION_MINUTES	SUSPECTED_CRIME_DESCRIPTION
0	2020	4	6	Based on Radio Run	POM	1	Other
1	2020	4	6	Based on Radio Run	POM	1	Other
2	2020	4	6	Based on Radio Run	POM	1	Other

Data Preprocessing (ETL):

```
# Convert categorical data to numeric and separate target feature for training data
# using get_dummies encoding method for the entire dataframe


merge_dummies = pd.get_dummies(merge_df)

merge_dummies

# sqf_2020_df
data2020_dummies = pd.get_dummies(sqf_2020_df)



merge_dummies
```

	YEAR	MONTH	DAY	OBSERVED_DURATION_MINUTES	STOP_DURATION_MINUTES	OFFICER_EXPLAINED_STOP_FLAG
0	2019	4	6	1.0	10.0	1
1	2019	4	5	1.0	10.0	1
2	2019	4	2	1.0	4.0	1
3	2019	4	1	0.0	5.0	1
4	2019	4	5	1.0	5.0	1

Data Preprocessing (ETL):

Scaling the Data by Standard Scaler Function:

```
# Scaling the data by using the StandardScaler() function
scaler = StandardScaler()

# Fitting the scaler
X_scaler = scaler.fit(X_train)

# Scaling data
X_train_scaled = X_scaler.transform(X_train)
X_train_scaled
```

```
array([[ 1.14078912, -0.43713456,  1.4501575 , ...,  1.69106746,
        -0.4630476 , -0.23122975],
       [ 1.14078912, -0.43713456,  0.95543239, ...,  1.69106746,
        -0.4630476 , -0.23122975],
       [ 1.14078912, -0.43713456, -0.52874294, ...,  1.69106746,
        -0.4630476 , -0.23122975],
```

Logistic Regression Model:

- Fit (train) our model by using the training data.
- Validate the model by using the test data

Logistic Regression Model:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier
```

```
LogisticRegression()
```

```
# Fit (train) our model by using the training data
classifier.fit(X_train_scaled, y_train)
```

```
LogisticRegression()
```

```
# Validate the model by using the test data
print(f"Training Data Score: {classifier.score(X_train_scaled, y_train)}")
print(f"Testing Data Score: {classifier.score(X_test_scaled, y_test)}")
```

```
Training Data Score: 0.9711602393617021
Testing Data Score: 0.9724434199497066
```

Random Forest Classifier:

Random Forest Classifier:

```
# Import a Random Forests classifier
from sklearn.ensemble import RandomForestClassifier
```

```
# Fit a model, and then print a classification report
clf = RandomForestClassifier(random_state=1).fit(X_train_scaled, y_train)
y_pred = clf.predict(X_test_scaled)
print(classification_report(y_test, y_pred))
print(f'Training Score: {clf.score(X_train_scaled, y_train)}')
print(f'Testing Score: {clf.score(X_test_scaled, y_test)}')
```

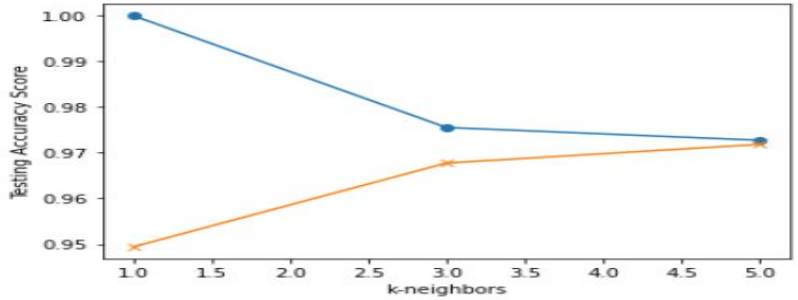
```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99      9281
           1       0.00      0.00      0.00       263

    accuracy                           0.97      9544
   macro avg       0.49      0.50      0.49      9544
weighted avg       0.95      0.97      0.96      9544

Training Score: 0.9999445921985816
Testing Score: 0.9724434199497066
```

K-nearest neighbors:

K-nearest neighbors:

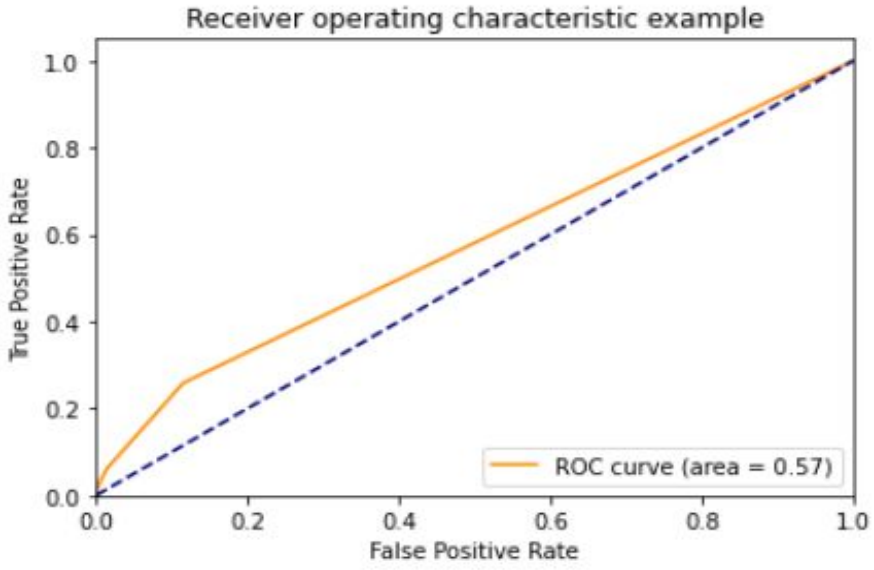
```
from sklearn.neighbors import KNeighborsClassifier
# Loop through different k values to find which has the highest accuracy.
# Note: We use only odd numbers because we don't want any ties.
train_scores = []
test_scores = []
for k in range(1, 6, 2):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    train_score = knn.score(X_train_scaled, y_train)
    test_score = knn.score(X_test_scaled, y_test)
    train_scores.append(train_score)
    test_scores.append(test_score)
    print(f"k: {k}, Train/Test Score: {train_score:.3f}/{test_score:.3f}")


plt.plot(range(1, 6, 2), train_scores, marker='o')
plt.plot(range(1, 6, 2), test_scores, marker="x")
plt.xlabel("k-neighbors")
plt.ylabel("Testing Accuracy Score")
plt.show()
```

```
k: 1, Train/Test Score: 1.000/0.949
k: 3, Train/Test Score: 0.976/0.968
k: 5, Train/Test Score: 0.973/0.972
```



Receiver Operating Characteristic Curve (ROC):

The receiver operating characteristic (ROC) curve helps us visualize this tradeoff. The false positive rate and the true positive rate are calculated for several thresholds, and we plot them against each other.



Resampling Techniques:

Resampling Techniques for Imbalance Data: Over-sampling: SMOTE

```
sqf_2020_df['SUMMONS_ISSUED_FLAG'].value_counts()
```

```
0    9281
1     263
Name: SUMMONS_ISSUED_FLAG, dtype: int64
```

```
merge_df['SUMMONS_ISSUED_FLAG'].value_counts()[0]
```

```
35056
```

```
merge_df['SUMMONS_ISSUED_FLAG'].value_counts()[1]
```

```
1040
```

Resampling Techniques:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy='minority')
X_sm, y_sm = smote.fit_resample(X_train, y_train)

plt.scatter(X_sm['SEARCHED_FLAG'], y_sm, c='r')
```

```
<matplotlib.collections.PathCollection at 0x1bd4635c9b0>
```

Refitting the Model:

Fitting the Logistic Model on Balance data

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier
```

```
LogisticRegression()
```

```
# from sklearn.linear_model import LogisticRegression
# classifier = LogisticRegression(class_weight='balanced')
```

```
# Fit (train) our model by using the training data
classifier.fit(X_sm, y_sm)
```

```
LogisticRegression()
```

```
# Validate the model by using the test data
print(f"Training Data Score: {classifier.score(X_sm, y_sm)}")
print(f"Testing Data Score: {classifier.score(X_test_scaled, y_test)}")
```

```
Training Data Score: 0.5587203331811958
Testing Data Score: 0.759010896898575
```

```
from sklearn.metrics import confusion_matrix

y_true = y_test
y_pred = classifier.predict(X_test_scaled)
confusion_matrix(y_true, y_pred)
```

```
array([[7161, 2120],
       [ 180,   83]], dtype=int64)
```

Resampling Techniques:

Random under-sampling Technique:

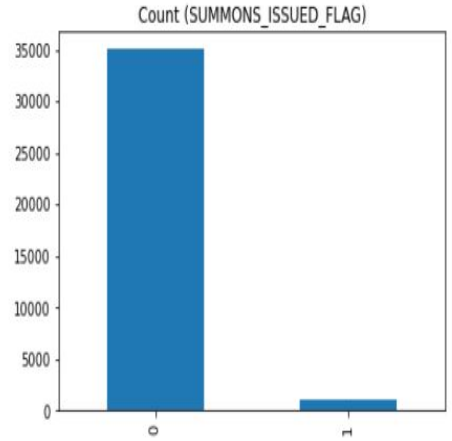
```
y_train[y_train==0]
```

```
0         0
1         0
2         0
3         0
4         0
         ..
11624     0
11625     0
11626     0
11627     0
11628     0
Name: SUMMONS_ISSUED_FLAG, Length: 35056, dtype: int64
```

Resampling Techniques:

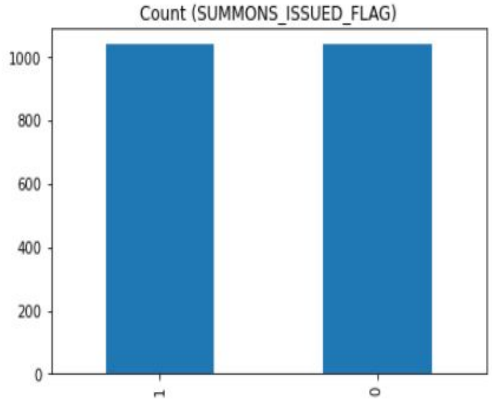
```
# Before resampling
print('Before Random under-sampling:')
print(merge_df.SUMMONS_ISSUED_FLAG.value_counts())

merge_df.SUMMONS_ISSUED_FLAG.value_counts().plot(kind='bar', ti
```

```
Before Random under-sampling:
0    35056
1     1040
Name: SUMMONS_ISSUED_FLAG, dtype: int64
```




```
# After sampling
df_class_0_under = df_class_0.sample(count_class_1)
df_test_under = pd.concat([df_class_0_under, df_class_1], axis=0)

print('Random under-sampling:')
print(df_test_under.SUMMONS_ISSUED_FLAG.value_counts())

df_test_under.SUMMONS_ISSUED_FLAG.value_counts().plot(kind='bar', title='Count (SUMMONS_ISSUED_FLAG)');
```

```
Random under-sampling:
1    1040
0    1040
Name: SUMMONS_ISSUED_FLAG, dtype: int64
```



Resampling Techniques:

Random under-sampling and over-sampling with imbalanced-learn

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler()
X_rus, y_rus = rus.fit_resample(X_train, y_train)

X_rus
```

	YEAR	MONTH	DAY	OBSERVED_DURATION_MINUTES	STOP_DURATION_MINUTES	OFFICER_EXPLAINED_STOP_FLAG
0	2017	10	0	1.0	10.0	1
1	2017	10	1	1.0	3.0	1
2	2019	8	2	1.0	5.0	1
3	2017	0	0	0.0	22.0	1
4	2018	5	5	1.0	3.0	1

Refitting the Model:

Fitting the Models Again:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier
```

```
LogisticRegression()
```

```
# Fit (train) our model by using the training data
classifier.fit(X_rus, y_rus)
```

```
LogisticRegression()
```

```
# Validate the model by using the test data
print(f"Training Data Score: {classifier.score(X_rus, y_rus)}")
print(f"Testing Data Score: {classifier.score(X_test_scaled, y_test)}")
```

```
Training Data Score: 0.55625
Testing Data Score: 0.8510058675607711
```

```
from sklearn.metrics import confusion_matrix

y_true = y_test
y_pred = classifier.predict(X_test_scaled)
confusion_matrix(y_true, y_pred)
```

```
array([[8072, 1209],
       [ 213,   50]], dtype=int64)
```

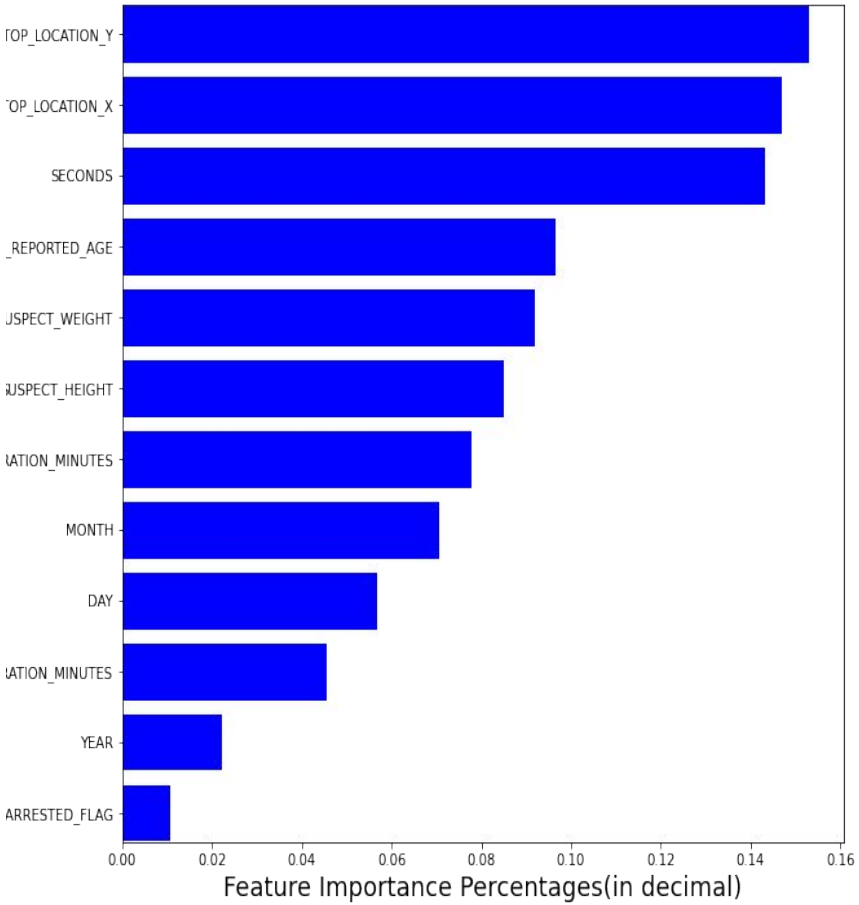
```
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

          NO       0.97      0.87      0.92      9281
         YES       0.04      0.19      0.07       263
```

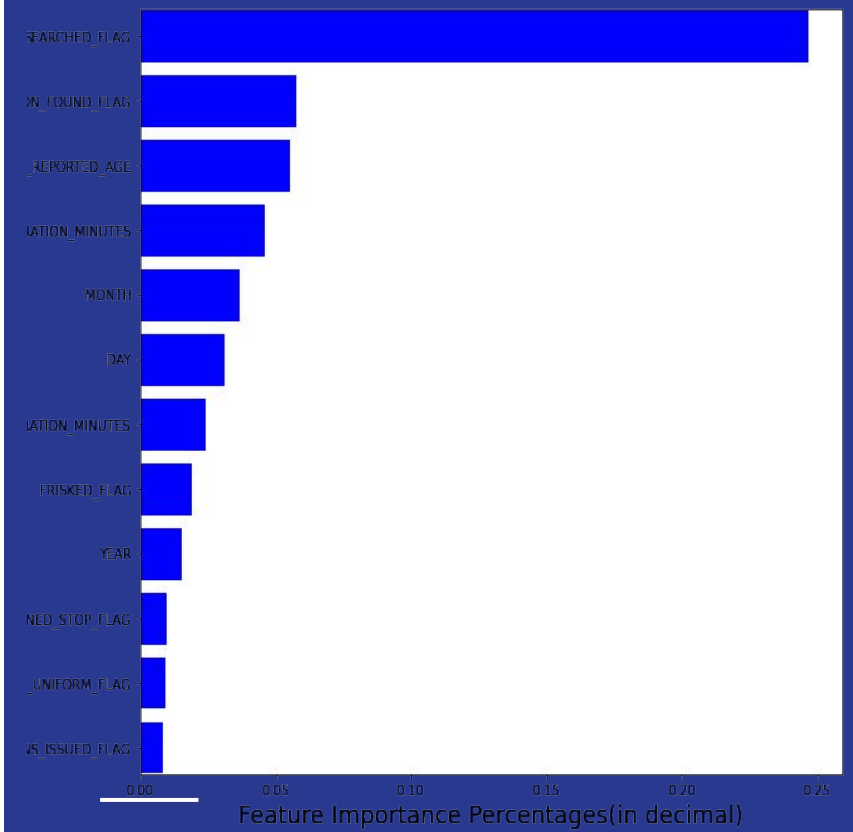
- Assigning Another outcome : 'SUSPECT_ARRESTED_FLAG' and repeat the all the steps

Finding the Important Features

Important Fearures for Summon Issued Outcome



Important Fearures for Arrest Issued Outcome



Flask Application:

Visual Studio Code

main.py

```python
def visual_tableau():
    return render_template('tableau.html')


@app.route('/summon_predictions')
def predictions():
    return render_template('summon_predictions.html')


@app.route('/api/generate_summon_prediction', methods=['POST'])
def generate_prediction():
    user_inputs=request.json
    predict_df=pd.DataFrame({
    'SUSPECT_ARRESTED_FLAG':[int(user_inputs['SUSPECT_ARRESTED_FLAG'])],
    'YEAR':[int(user_inputs['YEAR'])],
    'OBSERVED_DURATION_MINUTES':[int(user_inputs['OBSERVED_DURATION_MINUTES'])],
    'DAY':[int(user_inputs['DAY'])],
    'MONTH':[int(user_inputs['MONTH'])],
    'STOP_DURATION_MINUTES':[int(user_inputs['STOP_DURATION_MINUTES'])],
    'SUSPECT_HEIGHT':[int(user_inputs['SUSPECT_HEIGHT'])],
    'SUSPECT_WEIGHT':[int(user_inputs['SUSPECT_WEIGHT'])],
    'SUSPECT_REPORTED_AGE':[int(user_inputs['SUSPECT_REPORTED_AGE'])],
    'SECONDS':[int(user_inputs['SECONDS'])],
    'STOP_LOCATION_X':[float(user_inputs['STOP_LOCATION_X'])],
    'STOP_LOCATION_Y':[float(user_inputs['STOP_LOCATION_Y'])]
    })
    prediction=str(trained_machine_learning_model.predict(predict_df)[0])
    return jsonify([prediction])


@app.route('/api/feature_names')
def feature_names():
    return jsonify(reduced_column_names)

@app.route('/arrest_predictions')
```

summon_predictions.js - outcome-prediction-using-machine-learning -

EXPLORER

OUTCOME-PREDICTION-USING-MA...

- __pycache__
- static
  - css
  - data_processing
  - images
  - js
    - analysis.js
    - arrest_predictions.js
    - base.js
    - home.js
    - summon_predictions.js
    - tableau.js
  - templates
    - 404.html
    - analysis.html
    - arrest_predictions.html
    - base.html
    - home.html
    - summon_predictions.html
    - tableau.html
  - .gcloudignore
  - .gitignore
  - app.yaml
  - data.py
  - final_project_proposal.pptx

OPEN EDITORS

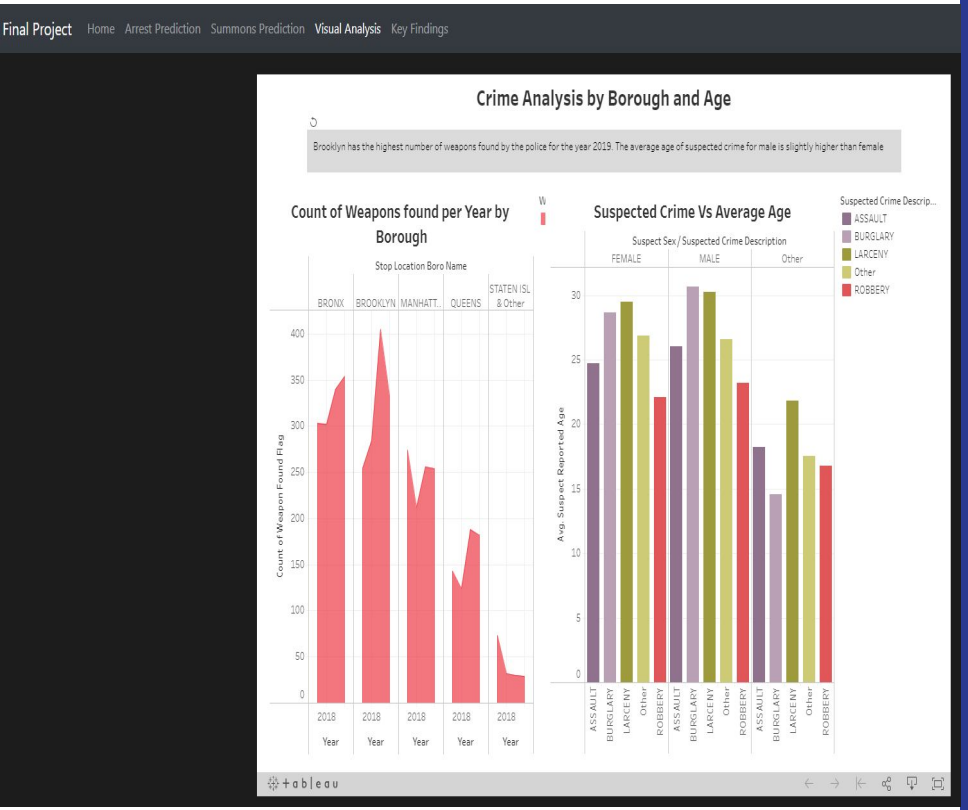
- tableau.html templates
- home.html templates
- summon_predictions.ht...
- base.html templates

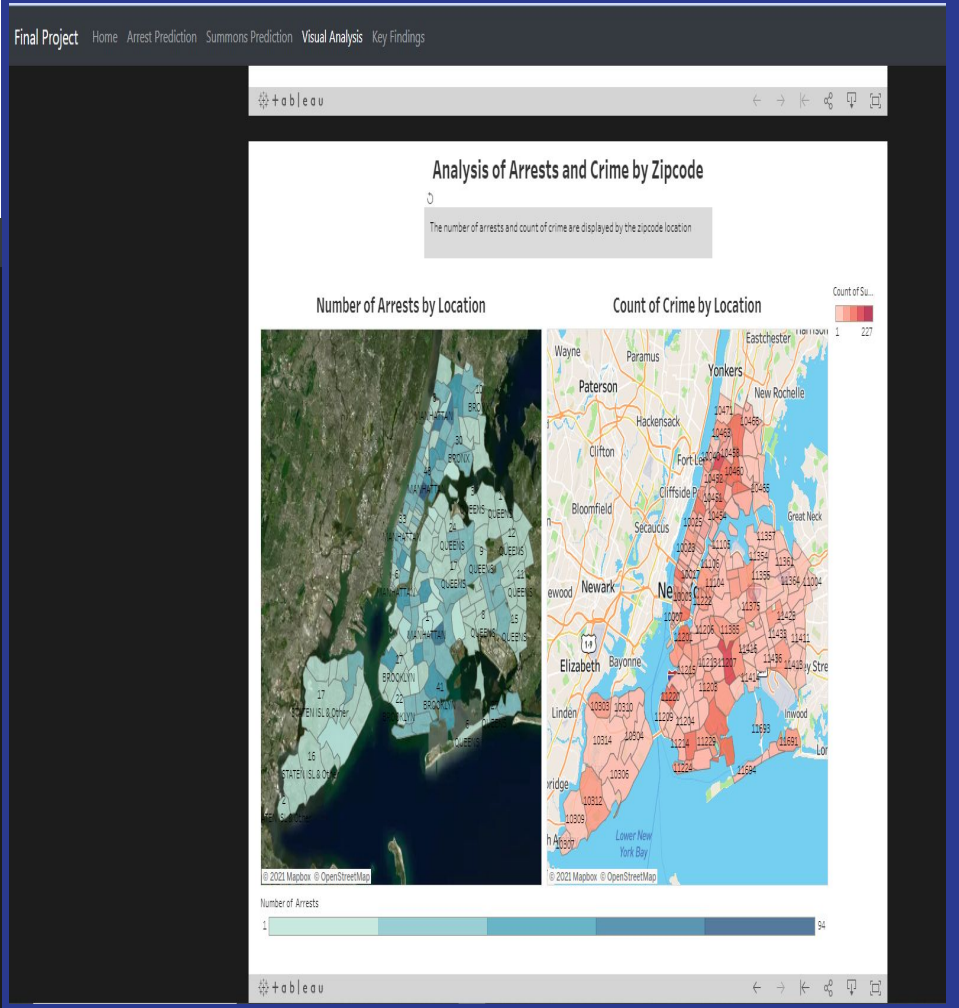
tableau.html | home.html | summon_predictions.html | base.html | summon_predictions.js

static > js > summon_predictions.js > submit_button_pressed

```javascript
var reduced_column_names = []
fetch('/api/feature_names')
  .then(response => response.json())
  .then(feature_names => {
    console.log(feature_names);
    var user_input_html = ''
    feature_names.forEach(feature_name => {
      reduced_column_names.push(feature_name)
      user_input_html += `
        <input id = '${feature_name}' placeholder = '${feature_name}' style = 'width: 400px; text-
        <br>
        `
    });
    $('#user-inputs').html(user_input_html)
  });


function submit_button_pressed() {

  var typed_inputs = {}
  reduced_column_names.forEach(reduced_column_name => {
    var user_input = d3.select(`#${reduced_column_name}`).property('value')
    typed_inputs[reduced_column_name] = user_input
  })

  fetch('/api/generate_summon_prediction', {
    method: 'POST', // or 'PUT'
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(typed_inputs),
  })
    .then(response => response.json())
    .then(data => {
```

Tableau Visual Analytics





PostgreSQL Database