Final_Project_Part3_Model 072220

July 24, 2020

1 File Information

Name: Amie Davis

Course: DSC630 - Predictive Analytics

Assignment Number: Final Project Part 3

Purpose: Build model(s)

Usage: Python 3.7.6

Developed using Jupter Notebook 6.0.3

2 Data Source

Uniform Crime Reporting Program Data: National Incident-Based Reporting System, [United States], 2016; United States Federal Bureau of Investigation; Inter-university Consortium for Political and Social Research (ICPSR), University of Michigan; https://www.icpsr.umich.edu/icpsrweb/NACJD/NIBRS/

Geodetic Data for US Cities: https://simplemaps.com/data/us-cities

3 Part 3

In Part 3, I will build a decision tree classification model to predict the type of offenses committed, given location information.

3.1 Import required packages

[1]:
```
# Suppress Warnings
#import warnings
#warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

4 Prepare Data

[2]:
```
# Load data into dataframe
data_file = "Data\crime_offenses_top6.csv"    # Data from Top 6 States
#data_file = "Data\crime_offenses_all.csv"    # Data from All States
df = pd.read_csv(data_file)
```

```
C:\Users\amomu\Anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3063: DtypeWarning: Columns
(7,11,14,15,16,17,18,19,41,42,44,46,51,52,53,54,56) have mixed types.Specify
dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

4.1 Eliminate features

[3]:
```
print(df.columns)
```

```
Index(['Unnamed: 0', 'X1', 'ORI', 'INC_NUM', 'VIC_INC_DATE', 'VICTIM_TYPE',
       'ACT_TYPE_OFFC', 'ASSG_TYPE_OFFC', 'AGE_OF_VICTIM', 'SEX_OF_VICTIM',
       'RACE_OF_VICTIM', 'ETHNIC_OF_VIC', 'VIC_RESIDENT', 'ASSAULT_CIRC1',
       'ASSAULT_CIRC2', 'JUST_HOM_CIRC', 'INJURY_TYPE1', 'INJURY_TYPE2',
       'INJURY_TYPE3', 'INJURY_TYPE4', 'INJURY_TYPE5', 'NUM_RECS_PER_VICTIM',
       'VIC_INC_YEAR', 'VIC_INC_MONTH', 'VIC_INC_DAY', 'VIC_INC_DOW',
       'NUM_STATE_CODE', 'CITY', 'STATE', 'POP_GROUP', 'CTRY_DIVISION',
       'CTRY_REGION', 'AGENCY_IND', 'CORE_CITY', 'FBI_OFFICE', 'JUDICIAL_DIST',
       'CURRENT_POP1', 'UCR_COUNTY_CD1', 'MSA_CD1', 'LAST_POP1',
       'FIPS_COUNTY1', 'city_ascii', 'state_name', 'county_fips',
       'county_name', 'county_fips_all', 'county_name_all', 'lat', 'lng',
       'population', 'density', 'source', 'military', 'incorporated',
       'timezone', 'ranking', 'zips', 'id', 'OFF_CODE'],
      dtype='object')
```

[4]:
```
# Remove irrelevant and redundant fields

# Drop unneeded columns
df.drop(['X1','id', 'county_fips', 'county_fips_all', 'Unnamed: 0',
         'ORI', 'INC_NUM', 'NUM_RECS_PER_VICTIM', 'VIC_INC_DATE',␣
↪'VIC_INC_YEAR', 'VIC_INC_DAY',
         'ASSAULT_CIRC1', 'ASSAULT_CIRC2', 'JUST_HOM_CIRC',
         'INJURY_TYPE1', 'INJURY_TYPE2', 'INJURY_TYPE3',
         'INJURY_TYPE4', 'INJURY_TYPE5', 'NUM_RECS_PER_VICTIM',
         'FBI_OFFICE', 'JUDICIAL_DIST', 'FIPS_COUNTY1',
         'LAST_POP1', 'UCR_COUNTY_CD1', 'MSA_CD1', 'city_ascii', 'CITY',␣
↪'STATE',
         'state_name', 'county_name', 'county_name_all', 'population', 'zips',
         'source'],
        axis=1, inplace = True)
```

```
# Keep AGENCY_IND

# Also removing victim demograohics since they are not relevant to predict␣
↪offenses and locations
#df.drop(['VICTIM_TYPE','RACE_OF_VICTIM', 'AGE_OF_VICTIM',
#          'SEX_OF_VICTIM', 'ETHNIC_OF_VIC', 'VIC_RESIDENT'],
#         axis=1, inplace = True)

# Verify Change
#print(df.columns)
```

[5]:
```
# Replace NA values with UNKNOWN
#df.replace('NA', 'U')

# Drop columns with mostly NULL data
df.drop(['ACT_TYPE_OFFC','ASSG_TYPE_OFFC','CORE_CITY'],
        axis=1, inplace = True)

# Drop records with remaining null values
df.dropna(axis=0, inplace=True)
```

[6]:
```
# Find null records
#count_nan_in_df = df.isnull().sum()
#print (count_nan_in_df)
```

4.2 Encoding

[7]:
```
# Change DOW to numeric value
def f_dow(df):
    if df['VIC_INC_DOW'] == 'Sunday':
        val = 1
    elif df['VIC_INC_DOW'] == 'Monday':
        val = 2
    elif df['VIC_INC_DOW'] == 'Tuesday':
        val = 3
    elif df['VIC_INC_DOW'] == 'Wednesday':
        val = 4
    elif df['VIC_INC_DOW'] == 'Thursday':
        val = 5
    elif df['VIC_INC_DOW'] == 'Friday':
        val = 6
    elif df['VIC_INC_DOW'] == 'Saturday':
        val = 7
    else:
        val=0
```

```
    return val

# Change Timezone to numeric value
def f_tz(df):
    if df['timezone'] == 'America/New_York':
        val = 1
    elif df['timezone'] == 'America/Detroit':
        val = 2
    elif df['timezone'] == 'America/Chicago':
        val = 2
    elif df['timezone'] == 'America/Denver':
        val = 3
    elif df['timezone'] == 'America/Los_Angeles':
        val = 4
    else:
        val=0
    return val

# Convert simple categorical features to numeric to limit dummy features
df['VIC_INC_DOW'] = df.apply(f_dow, axis=1)
df['timezone'] = df.apply(f_tz, axis=1)

df['SEX_OF_VICTIM'] = df['SEX_OF_VICTIM'].map({'M': 2, 'F': 1, 'U': 0})
df['ETHNIC_OF_VIC'] = df['ETHNIC_OF_VIC'].map({'H': 2, 'N': 1, 'U': 0})
df['VIC_RESIDENT'] = df['VIC_RESIDENT'].map({'R': 2, 'N': 1, 'U': 0})
#df['CORE_CITY'] = df['CORE_CITY'].map({'Y': 1, 'N': 0})
df['military'] = df['military'].map({True: 1, False: 0})
df['incorporated'] = df['incorporated'].map({True: 1, False: 0})

# Change target feature to easily translated numeric values
def f_off(df):
    if df['OFF_CODE'] == '09A':
        val = 91
    elif df['OFF_CODE'] == '09B':
        val = 92
    elif df['OFF_CODE'] == '09C':
        val = 93
    elif df['OFF_CODE'] == '100':
        val = 100
    elif df['OFF_CODE'] == '11A':
        val = 111
    elif df['OFF_CODE'] == '11B':
        val = 112
    elif df['OFF_CODE'] == '11C':
        val = 113
    elif df['OFF_CODE'] == '11D':
        val = 114
```

```
    elif df['OFF_CODE'] == '120':
        val = 120
    elif df['OFF_CODE'] == '13A':
        val = 131
    elif df['OFF_CODE'] == '13B':
        val = 132
    elif df['OFF_CODE'] == '13C':
        val = 133
    elif df['OFF_CODE'] == '200':
        val = 200
    elif df['OFF_CODE'] == '210':
        val = 210
    elif df['OFF_CODE'] == '220':
        val = 220
    elif df['OFF_CODE'] == '23A':
        val = 231
    elif df['OFF_CODE'] == '23B':
        val = 232
    elif df['OFF_CODE'] == '23C':
        val = 233
    elif df['OFF_CODE'] == '23D':
        val = 234
    elif df['OFF_CODE'] == '23E':
        val = 235
    elif df['OFF_CODE'] == '23F':
        val = 236
    elif df['OFF_CODE'] == '23G':
        val = 237
    elif df['OFF_CODE'] == '23H':
        val = 238
    elif df['OFF_CODE'] == '240':
        val = 240
    elif df['OFF_CODE'] == '250':
        val = 250
    elif df['OFF_CODE'] == '26A':
        val = 261
    elif df['OFF_CODE'] == '26B':
        val = 262
    elif df['OFF_CODE'] == '26C':
        val = 263
    elif df['OFF_CODE'] == '26D':
        val = 264
    elif df['OFF_CODE'] == '26E':
        val = 265
    elif df['OFF_CODE'] == '26F':
        val = 266
    elif df['OFF_CODE'] == '26G':
```

```
        val = 267
    elif df['OFF_CODE'] == '270':
        val = 270
    elif df['OFF_CODE'] == '280':
        val = 280
    elif df['OFF_CODE'] == '290':
        val = 290
    elif df['OFF_CODE'] == '35A':
        val = 351
    elif df['OFF_CODE'] == '35B':
        val = 352
    elif df['OFF_CODE'] == '36A':
        val = 361
    elif df['OFF_CODE'] == '36B':
        val = 362
    elif df['OFF_CODE'] == '370':
        val = 370
    elif df['OFF_CODE'] == '39A':
        val = 391
    elif df['OFF_CODE'] == '39B':
        val = 392
    elif df['OFF_CODE'] == '39C':
        val = 393
    elif df['OFF_CODE'] == '39D':
        val = 394
    elif df['OFF_CODE'] == '40A':
        val = 401
    elif df['OFF_CODE'] == '40B':
        val = 402
    elif df['OFF_CODE'] == '40C':
        val = 403
    elif df['OFF_CODE'] == '510':
        val = 510
    elif df['OFF_CODE'] == '520':
        val = 520
    elif df['OFF_CODE'] == '64A':
        val = 641
    elif df['OFF_CODE'] == '64B':
        val = 642
    elif df['OFF_CODE'] == '720':
        val = 720
    else:
        val=0
    return val
df['OFF_CODE'] = df.apply(f_off, axis=1)

# Convert population group to easily translated numeric values
```

```python
def f_pop(df):
    if df['POP_GROUP'] == '1A':
        val = 11
    elif df['POP_GROUP'] == '1B':
        val = 12
    elif df['POP_GROUP'] == '1C':
        val = 13
    elif df['POP_GROUP'] == '8A':
        val = 81
    elif df['POP_GROUP'] == '8B':
        val = 82
    elif df['POP_GROUP'] == '8C':
        val = 83
    elif df['POP_GROUP'] == '8D':
        val = 84
    elif df['POP_GROUP'] == '8E':
        val = 85
    elif df['POP_GROUP'] == '9A':
        val = 91
    elif df['POP_GROUP'] == '9B':
        val = 92
    elif df['POP_GROUP'] == '9C':
        val = 93
    elif df['POP_GROUP'] == '9D':
        val = 94
    elif df['POP_GROUP'] == '9E':
        val = 95
    else:
        val=df['POP_GROUP']
    return val
df['POP_GROUP'] = df.apply(f_pop, axis=1)

# Convert victim type
def f_vt(df):
    if df['VICTIM_TYPE'] == 'I':
        val = 1
    if df['VICTIM_TYPE'] == 'B':
        val = 2
    if df['VICTIM_TYPE'] == 'F':
        val = 3
    if df['VICTIM_TYPE'] == 'G':
        val = 4
    if df['VICTIM_TYPE'] == 'L':
        val = 5
    if df['VICTIM_TYPE'] == 'R':
        val = 6
    if df['VICTIM_TYPE'] == 'S':
```

```
        val = 7
    if df['VICTIM_TYPE'] == 'O':
        val = 8
    else:
        val = 0
    return val
df['VICTIM_TYPE'] = df.apply(f_vt, axis=1)

# Convert race
def f_race(df):
    if df['RACE_OF_VICTIM'] == 'W':
        val = 1
    if df['RACE_OF_VICTIM'] == 'B':
        val = 2
    if df['RACE_OF_VICTIM'] == 'I':
        val = 3
    if df['RACE_OF_VICTIM'] == 'A':
        val = 4
    if df['RACE_OF_VICTIM'] == 'P':
        val = 5
    else:
        val = 0
    return val
df['RACE_OF_VICTIM'] = df.apply(f_race, axis=1)

df.head()
```

[7]:

```
    VICTIM_TYPE  AGE_OF_VICTIM  SEX_OF_VICTIM  RACE_OF_VICTIM  ETHNIC_OF_VIC  \
0             0             27              1               0              0
1             0             25              1               0              1
2             0             51              2               0              1
4             0             50              2               0              0
21            0             30              1               0              2

    VIC_RESIDENT  VIC_INC_MONTH  VIC_INC_DOW  NUM_STATE_CODE  POP_GROUP  …  \
0              1              1            6              20        4.0  …
1              2              1            6              20        4.0  …
2              2              1            6              20        4.0  …
4              1              1            7              20        4.0  …
21             2              1            6              20        4.0  …

    AGENCY_IND  CURRENT_POP1      lat       lng  density  military  \
0            1       43974.0  41.6722 -70.3599    284.0         0
1            1       43974.0  41.6722 -70.3599    284.0         0
2            1       43974.0  41.6722 -70.3599    284.0         0
4            1       43974.0  41.6722 -70.3599    284.0         0
21           1       43974.0  41.6722 -70.3599    284.0         0
```

```
    incorporated  timezone  ranking  OFF_CODE
0              1         1      2.0       261
1              1         1      2.0       237
2              1         1      2.0       133
4              1         1      2.0       290
21             1         1      2.0       114

[5 rows x 22 columns]
```

4.3 Determine Target Variable

Use offense code as the target variable. The goal is to be able to predict offense code based on varying features.

For the first model, I will select Justifiable Homicide as the target variable. A second model will predict Suspicious Activity.

4.4 Split Datasets

[8]:
```python
# Split data into two sets: Training and Testing.

# Split out target variable
data_model_y = df.OFF_CODE

# Remove target variable from feature list
data_model_X = df.drop(['OFF_CODE'], axis=1, inplace = False)

# Split the data into training and validation datasets
# Save 30% for validation
X_train, X_val, y_train, y_val = train_test_split(data_model_X, data_model_y,␣
 ↪test_size =0.3, random_state=7)

# Check details of the datasets
print("No. of samples in original set1: ", data_model_X.shape[0])
print("No. of samples in training set1: ", X_train.shape[0])
print("No. of samples in validation set1: ", X_val.shape[0])
print("No. of features: ", X_train.shape[1])
```

```
No. of samples in original set1:  1126927
No. of samples in training set1:  788848
No. of samples in validation set1:  338079
No. of features:  21
```

[9]:
```python
#data_model_y
data_model_X
```

[9]:

```
         VICTIM_TYPE  AGE_OF_VICTIM  SEX_OF_VICTIM  RACE_OF_VICTIM  \
0                  0             27              1               0
1                  0             25              1               0
2                  0             51              2               0
4                  0             50              2               0
21                 0             30              1               0
…                …              …              …               …
3945578            0             31              1               0
3945579            0             23              2               0
3945583            0             58              2               0
3945584            0             59              1               0
3945593            0             51              1               0

         ETHNIC_OF_VIC  VIC_RESIDENT  VIC_INC_MONTH  VIC_INC_DOW  \
0                    0             1              1            6
1                    1             2              1            6
2                    1             2              1            6
4                    0             1              1            7
21                   2             2              1            6
…                  …             …              …          …
3945578              1             2             12            2
3945579              1             2             11            4
3945583              1             2              5            5
3945584              1             2              5            5
3945593              2             2             10            4

         NUM_STATE_CODE  POP_GROUP  …  CTRY_REGION  AGENCY_IND  \
0                    20        4.0  …            1           1
1                    20        4.0  …            1           1
2                    20        4.0  …            1           1
4                    20        4.0  …            1           1
21                   20        4.0  …            1           1
…                   …        …  …          …         …
3945578              46        3.0  …            4           1
3945579              46        3.0  …            4           1
3945583              46        5.0  …            4           1
3945584              46        5.0  …            4           1
3945593              46        3.0  …            4           1

         CURRENT_POP1      lat       lng  density  military  incorporated  \
0             43974.0  41.6722  -70.3599    284.0         0             1
1             43974.0  41.6722  -70.3599    284.0         0             1
2             43974.0  41.6722  -70.3599    284.0         0             1
4             43974.0  41.6722  -70.3599    284.0         0             1
21            43974.0  41.6722  -70.3599    284.0         0             1
…                 …      …       …      …       …           …
3945578       86005.0  48.7543 -122.4687   1241.0         0             1
```

```
3945579       86005.0  48.7543 -122.4687   1241.0          0             1
3945583       13338.0  48.8525 -122.5893    775.0          0             1
3945584       13338.0  48.8525 -122.5893    775.0          0             1
3945593       94111.0  46.5923 -120.5496   1302.0          0             1

         timezone  ranking
0               1      2.0
1               1      2.0
2               1      2.0
4               1      2.0
21              1      2.0
…             …      …
3945578         4      2.0
3945579         4      2.0
3945583         4      3.0
3945584         4      3.0
3945593         4      2.0

[1126927 rows x 21 columns]
```

5 Model Evaluation and Selection

5.1 1) Decision Tree Model

5.1.1 Build Model

[10]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 788848 entries, 1816890 to 2238547
Data columns (total 21 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   VICTIM_TYPE     788848 non-null  int64
 1   AGE_OF_VICTIM   788848 non-null  int64
 2   SEX_OF_VICTIM   788848 non-null  int64
 3   RACE_OF_VICTIM  788848 non-null  int64
 4   ETHNIC_OF_VIC   788848 non-null  int64
 5   VIC_RESIDENT    788848 non-null  int64
 6   VIC_INC_MONTH   788848 non-null  int64
 7   VIC_INC_DOW     788848 non-null  int64
 8   NUM_STATE_CODE  788848 non-null  int64
 9   POP_GROUP       788848 non-null  float64
 10  CTRY_DIVISION   788848 non-null  int64
 11  CTRY_REGION     788848 non-null  int64
 12  AGENCY_IND      788848 non-null  int64
 13  CURRENT_POP1    788848 non-null  float64
```

```
 14  lat             788848 non-null  float64
 15  lng             788848 non-null  float64
 16  density         788848 non-null  float64
 17  military        788848 non-null  int64
 18  incorporated    788848 non-null  int64
 19  timezone        788848 non-null  int64
 20  ranking         788848 non-null  float64
dtypes: float64(6), int64(15)
memory usage: 132.4 MB
```

[11]:
```
# Create decision tree classifer object
from sklearn.tree import DecisionTreeClassifier
#decisiontree = DecisionTreeClassifier(random_state=0, class_weight="balanced")
decisiontree = DecisionTreeClassifier(random_state=0)

# Train model
tree = decisiontree.fit(X_train, y_train)
```

5.1.2 Model Evaluation

[12]:
```
# Predict values
y_pred_tree = tree.predict(X_val)
```

[13]:
```
# Create classification report
print(classification_report(y_val, y_pred_tree))
```

```
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

          91       0.16      0.21      0.18       262
          92       0.09      0.09      0.09        11
          93       0.08      0.10      0.09        10
         100       0.19      0.29      0.23      1322
         111       0.13      0.14      0.13      1843
         112       0.41      0.42      0.41       505
         113       0.14      0.15      0.15       133
         114       0.40      0.37      0.38      2104
         120       0.41      0.50      0.45      7490
```

131	0.41	0.51	0.45	23283
132	0.45	0.48	0.46	59003
133	0.39	0.41	0.40	18873
200	0.44	0.54	0.48	965
210	0.10	0.10	0.10	153
220	0.37	0.40	0.38	32217
231	0.13	0.12	0.12	433
232	0.11	0.10	0.10	372
233	0.19	0.20	0.19	838
234	0.24	0.22	0.23	13036
235	0.17	0.23	0.19	43
236	0.46	0.48	0.47	37920
237	0.10	0.09	0.09	3776
238	0.39	0.36	0.37	37307
240	0.17	0.15	0.16	11164
250	0.32	0.34	0.33	4031
261	0.29	0.28	0.29	6937
262	0.34	0.32	0.33	6765
263	0.38	0.35	0.36	5825
264	0.11	0.17	0.14	23
265	0.13	0.12	0.12	355
266	0.29	0.25	0.27	2840
267	0.33	0.21	0.26	28
270	0.75	0.73	0.74	742
280	0.50	0.52	0.51	7260
290	0.34	0.30	0.32	44837
351	0.00	0.00	0.00	2127
352	0.00	0.00	0.00	988
361	0.16	0.23	0.19	26
362	0.25	0.21	0.23	236
370	0.00	0.00	0.00	62
393	0.00	0.00	0.00	3
401	0.00	0.00	0.00	17
402	0.00	0.00	0.00	6
403	0.00	0.00	0.00	1
510	0.00	0.00	0.00	7
520	0.01	0.00	0.00	1863
641	0.56	0.82	0.67	33
642	0.00	0.00	0.00	0
720	0.00	0.00	0.00	4
accuracy			0.38	338079
macro avg	0.22	0.23	0.23	338079
weighted avg	0.37	0.38	0.38	338079

5.1.3 Confusion Matrix

[14]:
```python
# Use Confusion Matrix to evaluate the model
from sklearn.metrics import confusion_matrix
tree_cm = confusion_matrix(y_val, y_pred_tree)

# Output confusion matrix array
# Note that print options need to be modified to accomodate large array
with np.printoptions(threshold=np.inf):
    print(tree_cm)
```

```
[[  54    0    0    4    3    0    0    0   19   52   26    4
     6    0   16    0    0    0    5    0   22    1    9    7
     0    2    3    4    0    0    1    0    0    3   17    1
     0    0    0    0    0    0    0    0    0    3    0    0
     0]
 [   0    1    0    0    0    0    0    0    1    1    5    0
     0    0    0    0    0    0    0    0    1    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    2
     0    0    0    0    0    0    0    0    0    0    0    0
     0]
 [   0    0    1    0    0    0    0    0    0    2    1    0
     0    0    1    0    0    0    0    0    3    0    0    1
     0    0    1    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0]
 [   2    0    0  387   49    3    3   23  135  183  231   55
     3    0   79    1    1    2   15    0   19    1   43   14
     0    3    4    2    0    1    3    0    0    7   38    6
     0    0    0    1    0    0    1    0    0    6    1    0
     0]
 [   2    0    0   60  250   33    7   95   42  115  438  136
     0    0  109    6    2    4   51    0  108    9  155   34
     8   12   15   14    0    1    2    0    0   10  104    1
     1    2    8    4    0    1    1    0    0    2    1    0
     0]
 [   0    0    0    5   33  214    3   23    6   21   88   16
     4    1   16    0    0    0   10    0   13    1   22    3
     2    2    2    2    0    0    2    0    0    0   11    1
     1    1    1    0    0    1    0    0    0    0    0    0
     0]
 [   0    0    0    4   12    3   20    9    2   11   26    4
     3    0    5    0    0    0    5    0    9    1    6    2
     1    2    0    0    0    0    1    0    0    0    4    0
     0    1    1    0    0    0    0    0    0    1    0    0
     0]
 [   1    0    0   27   95   24    8  777   23  152  440  124
     6    3   86    1    2    1   40    0   59    1   83   23
```

```
      5     6    16     3     0     1     1     0     1     8    50     3
      1     7    21     3     0     0     1     0     0     1     0     0
      0]
 [   10     1     1   178    25     9     3    22  3721   430   707   172
     11     5   479     3     6    15   118     1   344    41   309   184
     31    63    50    39     0     5    22     0     2    39   353    30
      6     0     1     0     0     1     0     0     0    53     0     0
      0]
 [   61     0     1   234    97    16     4    86   448 11779  3162   704
     49    12  1204    17     3    33   374     0   835   124   891   368
     54   154   121   111     1     7    36     0     3   125  1786   120
     25     1    11     1     2     0     0     0     0   222     0     1
      0]
 [   43     1     4   385   446   105    38   395   981  4077 28172  2806
    109    15  3342    67    55   198  1589     9  3180   410  3669  1251
    205   598   533   401     4    48   203     1    18   219  4925   260
     78     8    47     8     0     2     0     0     0    90     6     1
      1]
 [   10     0     1   125   129    21     6    94   252   953  3117  7660
     29    11  1088    16    22    49   514     1   875   124  1153   341
     67   217   161   135     3     9    72     1     5   100  1353    62
     27     2    22     9     0     0     0     0     0    35     1     1
      0]
 [    1     0     0     3     6     1     0     4     7    43    92    22
    517     1    38     0     2     0    19     0    54     3    35    13
      5    11     3     5     0     0     3     0     2    10    63     2
      0     0     0     0     0     0     0     0     0     0     0     0
      0]
 [    1     0     0     3     0     0     0     1     2    10    23    15
      0    16    13     0     0     0     5     0     8     3    18     3
      2     2     1     3     0     0     3     0     0     1    14     2
      0     0     1     3     0     0     0     0     0     0     0     0
      0]
 [   20     1     0   155    87    15     8    66   536  1578  3465  1028
     89     7 12801    20    29    31  1205     6  1798   236  2309   828
    127   396   292   231     3    28   124     4    17   264  4298    58
     18     1     3     0     0     1     0     0     0    34     0     0
      0]
 [    0     0     0     3     3     1     1     2     8    33    77    26
      0     1    47    51     1     4    12     0    26     4    38    12
      2    13     9     6     0     1     3     0     1     6    39     0
      0     0     1     0     0     2     0     0     0     0     0     0
      0]
 [    0     0     0     2     6     1     0     3     4    16    62    21
      0     0    24     3    38     2    20     0    36     3    48    14
      6     7    11     5     0     0     1     0     0     5    32     1
      0     0     1     0     0     0     0     0     0     0     0     0
      0]
```

```
[    0     0     0     2     0     0     0     0    21    76   235    48
     2     0    21     1     0   166    15     0    29     3    54    10
     5    14     6    35     0     0     5     0     0    34    48     6
     1     0     0     0     0     0     0     0     0     1     0     0
     0]
[   10     1     0    36    61     4     6    56   214   618  1981   595
    35     7  1589    24    22    21  2853     2   933   153   904   368
   154   308   337   143     3     9    67     0    12   122  1311    41
    10     0     6     1     0     0     0     0     0    17     2     0
     0]
[    0     0     0     0     0     0     0     0     1     2     2     3
     1     0     3     0     0     2     3    10     3     0     3     1
     0     3     2     0     0     0     0     0     0     0     4     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0]
[   18     2     1    64   127     7     4    43   458  1130  3443  1077
    56    16  2191    36    26    48   907     0 18046   437  2616  1028
   187   414   515   262     2    38   138     1    17   372  4089    60
    13     2     3     0     1     0     0     0     0    25     0     0
     0]
[    3     0     0     5    15     1     0     6    65   239   559   156
    12     5   324     8     4     8   172     0   534   322   325   220
    35    77    67    52     0     1    43     0     3    54   450     8
     1     0     0     0     0     0     0     0     0     2     0     0
     0]
[   19     0     1    91   152    19     7    88   498  1419  4520  1400
    62    12  3158    49    43    46  1016     7  3252   354 13276  1134
   482   641   689   532     8    40   194     2    30   611  3325    65
    20     2     3     0     0     1     0     0     0    36     3     0
     0]
[   16     2     1    47    44    11     3    19   218   615  1608   396
    26    11  1265    12    14    29   399     2  1389   209  1166  1627
    97   180   149   125     1     9    50     0     4   176  1174    43
    12     0     0     0     0     0     0     0     0    13     2     0
     0]
[    3     0     0     3     3     2     0     5    32    65   204    76
     2     1   165     3     4    12   161     0   179    23   487    63
  1376   389    96    90     0     6   119     0    16   277   140    19
     7     0     0     0     0     0     0     0     0     3     0     0
     0]
[    4     1     0     8    17     2     3    14    92   268   722   244
    11    10   467     7    11    24   300     2   482    80   644   166
   358  1954   137   140     1    11   146     1    11   103   448    30
     7     0     0     1     0     0     0     0     0    10     0     0
     0]
[    5     0     0     8    15     4     0     9    74   216   625   215
     4     3   408    16    12     8   370     2   597    59   702   146
   108   166  2181   142     1     8   123     2     7   123   388     7
```

```
     2     0     1     0     0     0     0     0     0     8     0     0
     0]
 [   3     0     0    12    14     1     3     8    57   179   520   158
     7     3   342    10     4    43   174     2   390    47   522   130
   136   174   210  2011     2    10    42     0    12   137   368    60
    21     0     5     0     0     0     0     0     0     8     0     0
     0]
 [   0     0     0     0     0     0     0     0     0     1     0     2
     0     0     0     0     0     0     1     0     1     0     0     1
     0     6     0     0     4     0     0     0     0     3     3     1
     0     0     0     0     0     0     0     0     0     0     0     0
     0]
 [   0     0     0     0     1     0     0     0     7    17    49    22
     1     0    26     0     0     0    16     0    38     8    43     3
     6    15    13    16     0    42     8     0     0     4    20     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0]
 [   3     0     0     2     8     2     0     2    25    69   272    77
     7     2   166     3     3     5    79     0   180    22   227    69
   263   134   170    56     0    18   712     0     3    84   171     3
     0     0     0     0     0     0     0     0     0     2     1     0
     0]
 [   0     0     0     0     0     0     0     0     0     2     2     2
     0     0     0     0     0     0     0     0     5     1     5     0
     0     0     1     1     0     0     0     6     0     0     3     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0]
 [   0     0     0     1     0     0     0     0     2     4    18     6
     0     0    16     0     0     0     9     0    13     4    23     4
    42    17     5    18     0     1     1     0   540     3    14     1
     0     0     0     0     0     0     0     0     0     0     0     0
     0]
 [   3     0     0    10    12     2     1     2    70   147   240    88
     6     0   308     3     6    53   117     0   483    40   482   161
   298    89   112   172     0     1   110     0     2  3765   164   198
    50     1     0     0     0     0     0     0     0    64     0     0
     0]
 [  29     0     1   135   117    18     7    84   682  2907  6837  1814
    99    16  4894    47    41    59  1253    16  5270   465  3449  1110
   232   515   433   326     0    29   174     0    17   279 13292    61
    23     2     3     0     0     1     0     0     0   100     0     0
     0]
 [   2     1     0    15    11     2     0     5    65   273   453   123
     3     0   123     1     3    18    43     0   121    12   150    50
    29    53    17    75     2     1    12     0     0   272   107     4
    53     0     1     0     0     1     0     0     0    26     0     0
     0]
 [   1     0     0     6     3     0     0     0    22    98   189    77
```

```
     3    0   51    0    0    6   22    0   45    8   66   26
    27   28    1   51    0    0    3    0    0  151   28   67
     0    0    0    0    0    0    0    0    0    9    0    0
     0]
[    0    0    0    0    1    1    1    0    0    2    5    1
     0    0    1    0    0    0    0    0    1    0    1    0
     0    0    0    0    0    0    0    0    0    0    1    0
     0    6    3    1    0    0    0    0    0    0    1    0
     0]
[    0    0    0    0   17    3    1   15    2   25   55   21
     0    0    4    0    0    0    5    0    5    0   20    0
     0    2    1    1    0    0    0    0    0    0    3    0
     1    1   49    3    0    0    1    0    0    0    1    0
     0]
[    0    0    0    1    8    1    1    2    0    4   11   16
     0    4    0    0    0    0    0    0    1    0    4    0
     0    1    0    0    0    0    0    0    0    1    1    0
     0    1    5    0    0    0    0    0    0    0    0    0
     0]
[    0    0    0    0    0    0    0    0    0    1    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    2    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0]
[    0    0    0    1    2    0    0    0    4    1    4    0
     0    0    0    0    0    0    0    0    1    0    0    0
     1    0    0    0    0    0    0    0    0    1    0    0
     0    0    1    0    0    0    0    0    0    1    0    0
     0]
[    0    0    0    0    1    0    0    2    0    1    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    2    0
     0]
[    0    0    0    1    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0]
[    0    0    0    0    0    0    0    0    2    2    1    0
     0    0    0    0    0    1    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    1    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0]
[    8    0    0   36    6    1    0    4  184  732  187   81
     5    0   94    0    3    2   11    0   57    5   52   22
     7   11    4   22    0    0    1    0    0  125  170   24
     4    0    1    0    0    0    0    0    0    4    0    0
```

```
      0]
 [    0     0     0     0     0     1     0     1     0     0     1     0
      0     0     0     0     0     0     0     0     2     0     1     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0    27     0
      0]
 [    0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0]
 [    0     0     0     0     0     0     0     0     0     0     2     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     2     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0]]
```

5.2 2) Keras Neural Network Model

5.2.1 Build Model

[17]:
```
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Dense
from keras import metrics
import keras.backend as K
from sklearn.preprocessing import StandardScaler

# Create standardizer
standardizer = StandardScaler()

# Standardize features
feat_std = standardizer.fit_transform(X_train)

# This function returns a compiled neural network

num_features = feat_std.shape[1]

def build_network():

    # Define the keras model
    nn = Sequential()
    nn.add(Dense(400, activation='relu', input_dim=num_features))
    nn.add(Dense(400, activation='relu'))
    nn.add(Dense(400, activation='relu'))
    nn.add(Dense(400, activation='relu'))
    nn.add(Dense(400, activation='relu'))
```

```
    # Use number of output classes in output layer
    nn.add(Dense(49, activation='softmax'))

    # Compile the keras model
    nn.compile(loss='categorical_crossentropy', optimizer='rmsprop',␣
↪metrics=['accuracy'])

    return nn

# Compile Keras neural network
clf = KerasClassifier(build_fn=build_network, epochs=7, batch_size=1000,␣
↪verbose=0)

# Fit the keras model on the dataset
history = clf.fit(feat_std,y_train)
```

```
WARNING:tensorflow:From C:\Users\amomu\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.
```

5.2.2 Model Evaluation

[18]:
```
# Predict values
val_std = standardizer.fit_transform(X_val)
y_pred_nn = clf.predict(val_std)
```

[19]:
```
# Create classification report
print(classification_report(y_val, y_pred_nn))
```

```
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

          91       0.00      0.00      0.00       262
          92       0.00      0.00      0.00        11
          93       0.00      0.00      0.00        10
         100       0.53      0.05      0.10      1322
         111       0.37      0.01      0.03      1843
         112       0.84      0.23      0.36       505
         113       0.00      0.00      0.00       133
         114       0.36      0.22      0.27      2104
         120       0.35      0.06      0.11      7490
         131       0.32      0.20      0.24     23283
```

132	0.27	0.70	0.39	59003
133	0.37	0.19	0.25	18873
200	0.59	0.24	0.34	965
210	0.00	0.00	0.00	153
220	0.23	0.16	0.19	32217
231	0.32	0.03	0.05	433
232	0.00	0.00	0.00	372
233	0.40	0.09	0.15	838
234	0.34	0.06	0.10	13036
235	0.00	0.00	0.00	43
236	0.39	0.36	0.37	37920
237	0.20	0.01	0.02	3776
238	0.36	0.32	0.34	37307
240	0.22	0.03	0.05	11164
250	0.54	0.15	0.23	4031
261	0.35	0.15	0.21	6937
262	0.54	0.16	0.25	6765
263	0.57	0.23	0.33	5825
264	0.00	0.00	0.00	23
265	0.00	0.00	0.00	355
266	0.26	0.33	0.29	2840
267	0.00	0.00	0.00	28
270	0.88	0.61	0.72	742
280	0.60	0.41	0.48	7260
290	0.27	0.27	0.27	44837
351	0.05	0.00	0.00	2127
352	0.00	0.00	0.00	988
361	0.00	0.00	0.00	26
362	0.44	0.05	0.08	236
370	0.00	0.00	0.00	62
393	0.00	0.00	0.00	3
401	0.00	0.00	0.00	17
402	0.00	0.00	0.00	6
403	0.00	0.00	0.00	1
510	0.00	0.00	0.00	7
520	0.07	0.00	0.00	1863
641	0.86	0.73	0.79	33
720	0.00	0.00	0.00	4
accuracy			0.31	338079
macro avg	0.25	0.13	0.15	338079
weighted avg	0.32	0.31	0.28	338079

5.3 3) Random Forest Classifer

5.3.1 Build Model

[46]:
```
# Create random forest classifer object
from sklearn.ensemble import RandomForestClassifier
randomforest = RandomForestClassifier(random_state=0, n_estimators=100,␣
↪n_jobs=-1, max_depth=25, bootstrap=False)
#randomforest = RandomForestClassifier(random_state=0, n_estimators=5,␣
↪n_jobs=-1, max_depth=25, bootstrap=False)

# Train model
forest = randomforest.fit(X_train, y_train)
```

5.3.2 Model Evaluation

[47]:
```
# Predict values
y_pred_forest = forest.predict(X_val)
```

[48]:
```
# Create classification report
print(classification_report(y_val, y_pred_forest))
```

```
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

          91       0.23      0.16      0.19       262
          92       0.17      0.09      0.12        11
          93       0.09      0.10      0.10        10
         100       0.30      0.23      0.26      1322
         111       0.24      0.13      0.17      1843
         112       0.62      0.42      0.50       505
         113       0.29      0.14      0.18       133
         114       0.54      0.36      0.43      2104
         120       0.51      0.45      0.48      7490
         131       0.48      0.47      0.47     23283
         132       0.42      0.57      0.48     59003
         133       0.45      0.41      0.43     18873
         200       0.64      0.50      0.56       965
         210       0.14      0.08      0.10       153
```

```
         220       0.39      0.39      0.39     32217
         231       0.22      0.11      0.15       433
         232       0.20      0.11      0.15       372
         233       0.25      0.18      0.21       838
         234       0.28      0.21      0.24     13036
         235       0.26      0.23      0.24        43
         236       0.48      0.48      0.48     37920
         237       0.16      0.08      0.11      3776
         238       0.40      0.38      0.39     37307
         240       0.20      0.15      0.17     11164
         250       0.38      0.32      0.35      4031
         261       0.34      0.28      0.31      6937
         262       0.41      0.32      0.36      6765
         263       0.45      0.35      0.39      5825
         264       0.14      0.13      0.13        23
         265       0.21      0.10      0.14       355
         266       0.32      0.28      0.30      2840
         267       0.33      0.21      0.26        28
         270       0.82      0.73      0.77       742
         280       0.55      0.53      0.54      7260
         290       0.32      0.34      0.33     44837
         351       0.01      0.00      0.00      2127
         352       0.00      0.00      0.00       988
         361       0.30      0.23      0.26        26
         362       0.36      0.25      0.29       236
         370       0.00      0.00      0.00        62
         393       0.00      0.00      0.00         3
         401       0.00      0.00      0.00        17
         402       0.00      0.00      0.00         6
         403       0.00      0.00      0.00         1
         510       0.00      0.00      0.00         7
         520       0.01      0.00      0.00      1863
         641       0.84      0.82      0.83        33
         642       0.00      0.00      0.00         0
         720       0.00      0.00      0.00         4

    accuracy                           0.40    338079
   macro avg       0.28      0.23      0.25    338079
weighted avg       0.39      0.40      0.39    338079
```

5.4 4) Bagging

[36]:
```python
# https://machinelearningmastery.com/bagging-ensemble-with-python/
from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier

# define the model
model = BaggingClassifier()
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, data_model_X, data_model_y,␣
↪scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:667: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:667: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:667: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
```

```
Accuracy: 0.396 (0.001)
```

5.5 5) Boosting

[37]:
```
# https://towardsdatascience.com/
↪machine-learning-part-18-boosting-algorithms-gradient-boosting-in-python-ef5ae6965be4
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

regressor = GradientBoostingRegressor(
    max_depth=10,    #leaves
    n_estimators=10,    #trees
    learning_rate=1.0    # scales the contribution of each tree. If you set it␣
↪to a low value, you will need more trees in the ensemble to fit the training␣
↪set, but the overall variance will be lower.
)
regressor.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred) for y_pred in regressor.
↪staged_predict(X_val)]
best_n_estimators = np.argmin(errors)
```

```
best_regressor = GradientBoostingRegressor(
    max_depth=2,
    n_estimators=best_n_estimators,
    learning_rate=1.0
)
best_regressor.fit(X_train, y_train)

y_pred_boost = best_regressor.predict(X_val)
mean_absolute_error(y_val, y_pred_boost)
```

[37]: 49.737743430882

6 Model Selection Conclusion

The Boosting model resulted in better accuracy than the other Python models, resulting in 50% accuracy.
As expected, it performed better than the bagging ensemble model, which performed better than the random forest ensemble. After adjusting hyperparameters for random forests, I was able to improve accuracy to 40%.

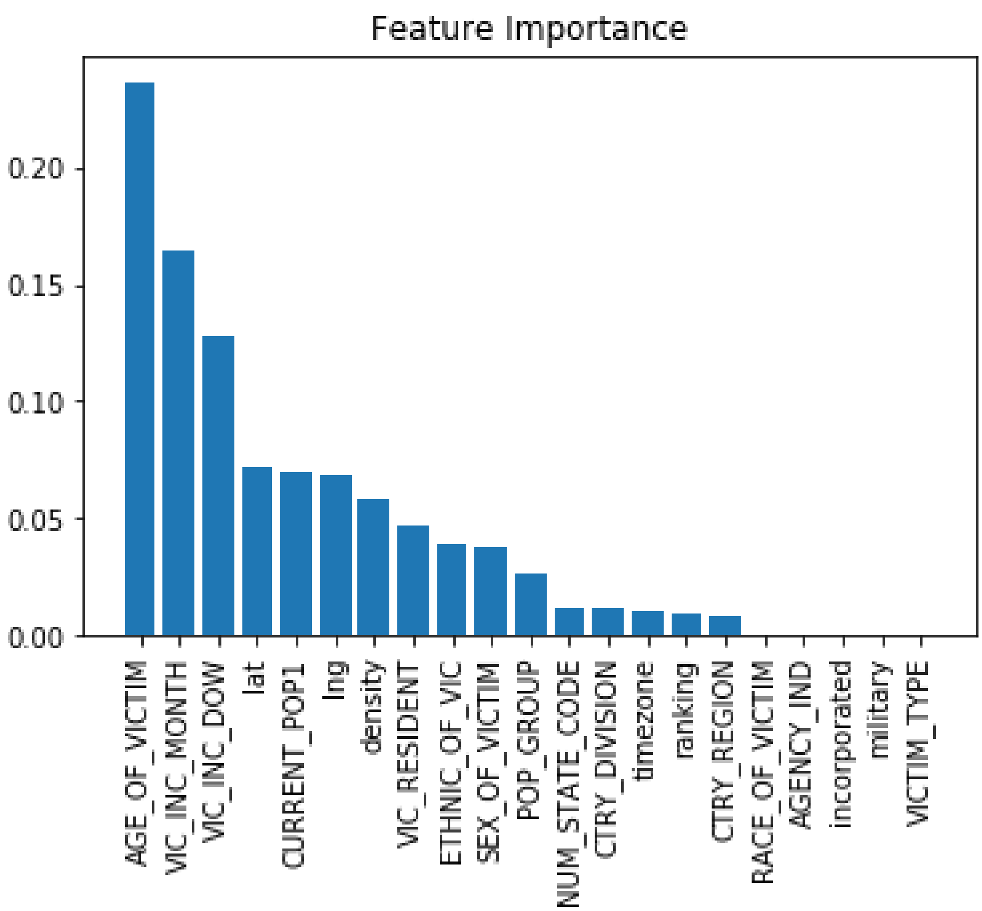
I increased the neural network accuracy by adjusting hyperparameters and adding hidden layers, up to 31%.

Unexpectedly, the decision tree model provided better accuracy than the random forest ensemble model. After adjusting hyperparameters for decision trees and random forests, I was able to improve to 28%.

7 Model Visualization

[38]:
```
# Find most important features in Random Forest model
import matplotlib.pyplot as plt

# Calculate feature importances
importances = forest.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [X_train.columns[i] for i in indices]

# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")
```

```python
# Add bars
plt.bar(range(X_train.shape[1]), importances[indices])

# Add feature names as x-axis labels
plt.xticks(range(X_train.shape[1]), names, rotation=90)

# Show plot
plt.show()
```



[39]:
```python
# Re-run random forest, limiting to important features

# Drop features w/ little importance
data_model_X.drop(['RACE_OF_VICTIM', 'AGENCY_IND', 'military', 'incorporated',␣
↪'VICTIM_TYPE'],
                  axis=1, inplace = True)

# Split the data into training and validation datasets
```

```
# Save 30% for validation
X_train, X_val, y_train, y_val = train_test_split(data_model_X, data_model_y,␣
↪test_size =0.3, random_state=7)
```

[49]:
```
# Rerun random forest classifer object
from sklearn.ensemble import RandomForestClassifier
randomforest = RandomForestClassifier(random_state=0, n_estimators=100,␣
↪n_jobs=-1, max_depth=25, bootstrap=False)
#randomforest = RandomForestClassifier(random_state=0, n_estimators=284,␣
↪n_jobs=-1, max_depth=17, bootstrap=False)

# Train model
forest = randomforest.fit(X_train, y_train)

# Predict values
y_pred_forest = forest.predict(X_val)

# Create classification report
print(classification_report(y_val, y_pred_tree))
```

```
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\amomu\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1272: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

          91       0.16      0.21      0.18       262
          92       0.09      0.09      0.09        11
          93       0.08      0.10      0.09        10
         100       0.19      0.29      0.23      1322
         111       0.13      0.14      0.13      1843
         112       0.41      0.42      0.41       505
         113       0.14      0.15      0.15       133
         114       0.40      0.37      0.38      2104
         120       0.41      0.50      0.45      7490
         131       0.41      0.51      0.45     23283
         132       0.45      0.48      0.46     59003
         133       0.39      0.41      0.40     18873
         200       0.44      0.54      0.48       965
         210       0.10      0.10      0.10       153
         220       0.37      0.40      0.38     32217
```

```
         231       0.13      0.12      0.12       433
         232       0.11      0.10      0.10       372
         233       0.19      0.20      0.19       838
         234       0.24      0.22      0.23     13036
         235       0.17      0.23      0.19        43
         236       0.46      0.48      0.47     37920
         237       0.10      0.09      0.09      3776
         238       0.39      0.36      0.37     37307
         240       0.17      0.15      0.16     11164
         250       0.32      0.34      0.33      4031
         261       0.29      0.28      0.29      6937
         262       0.34      0.32      0.33      6765
         263       0.38      0.35      0.36      5825
         264       0.11      0.17      0.14        23
         265       0.13      0.12      0.12       355
         266       0.29      0.25      0.27      2840
         267       0.33      0.21      0.26        28
         270       0.75      0.73      0.74       742
         280       0.50      0.52      0.51      7260
         290       0.34      0.30      0.32     44837
         351       0.00      0.00      0.00      2127
         352       0.00      0.00      0.00       988
         361       0.16      0.23      0.19        26
         362       0.25      0.21      0.23       236
         370       0.00      0.00      0.00        62
         393       0.00      0.00      0.00         3
         401       0.00      0.00      0.00        17
         402       0.00      0.00      0.00         6
         403       0.00      0.00      0.00         1
         510       0.00      0.00      0.00         7
         520       0.01      0.00      0.00      1863
         641       0.56      0.82      0.67        33
         642       0.00      0.00      0.00         0
         720       0.00      0.00      0.00         4

    accuracy                           0.38    338079
   macro avg       0.22      0.23      0.23    338079
weighted avg       0.37      0.38      0.38    338079
```

[ ]: