Final Project: New York City Case Study using CRSIP-DM

Report 4: Predictive modelling

Juan J. Holguin

The Southern Alberta Institute of Technology

DATA 475: Advanced Concepts in Data Analytics

Can we predict if a person is armed using the other variables?

Data Preparation

For data preparation, a list of all the attributes prior to a Police SQF are going to be selected and processed as follows:

1st: merge all weapon use related columns.

2nd: new dataset with all attributes related

3rd: set a target variable

4th: split the data

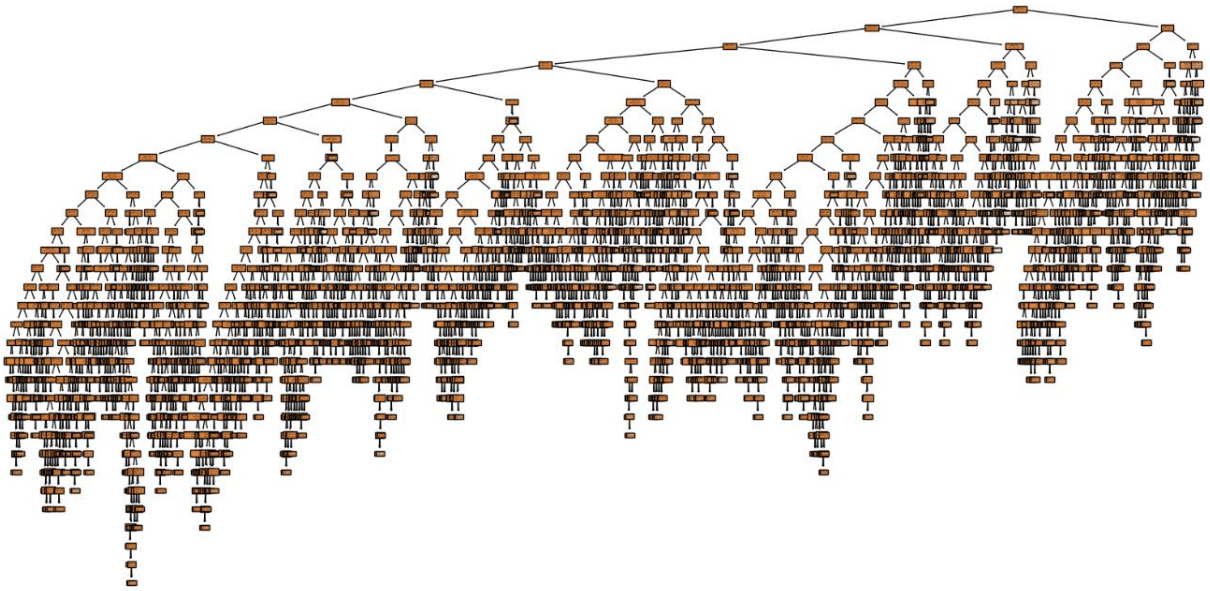
```
Run Cell | Run Above | Debug Cell | Go to [188]
#%% Can we predict if a person is armed using the other variables?
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

# Create a new column 'new_column' that is True if any of the dummy variables are True
crimes['armed'] = crimes[['pistol', 'riflshot', 'asltweap', 'machgun', 'othrweap']].apply(lambda x: 'Y' in x.values, axis=1)
crimes = crimes.assign(armed=crimes['armed'].fillna(0))
crimes['armed'] = crimes['armed'].astype(int)

fire_arms = crimes[['armed', 'pct', 'hour', 'month', 'perobs', 'crimsusp_grouped', 'ac_rept', 'ac_inves', 'rf_vcrim', 'rf_othsw',
                    'ac_proxm', 'rf_attir', 'cs_objcs', 'cs_descr', 'cs_casng', 'cs_lkout', 'rf_vcact', 'cs_cloth', 'cs_drgtr',
                    'ac_evasv', 'ac_assoc', 'cs_furtv', 'rf_rfcmp', 'ac_cgdir', 'rf_verbl', 'cs_vcrim', 'cs_bulge', 'cs_other',
                    'ac_incid', 'ac_time', 'rf_knowl', 'ac_stsnd', 'ac_other', 'sex', 'race', 'ht_inches', 'weight', 'city']]

# Encode categorical variables
categorical_cols = fire_arms.select_dtypes(include=['object']).columns
encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_cols = pd.DataFrame(encoder.fit_transform(fire_arms[categorical_cols]), columns=encoder.get_feature_names_out(categorical_cols))

# Drop original categorical columns and concatenate encoded columns
fire_arms = fire_arms.drop(columns=categorical_cols).reset_index(drop=True)
fire_arms = pd.concat([fire_arms, encoded_cols], axis=1)

# Define the feature set and the target variable
X = fire_arms.drop(columns=['armed'])
y = fire_arms['armed']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Output the shapes of the training and testing sets
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
```

Data modelling

The following models are run to search for the best accuracy:

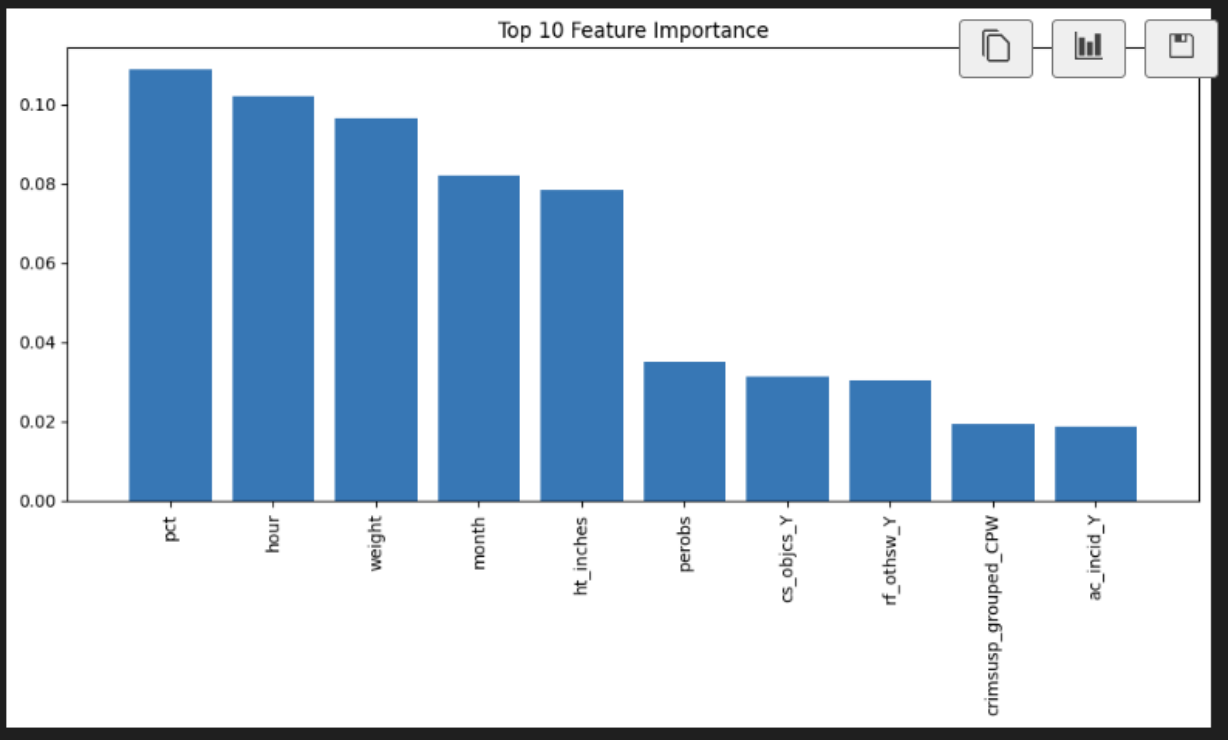
Decision tree:

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 10, 'min_samples_split': 2}
accuracy: 0.9959460476315889
```



Even though the accuracy is pretty high, a random forest with the Best parameters found is the following step:

```
Random Forest accuracy: 0.9960871364611159
```



The previous chart shows the most important features to address in order to mitigate the risk of an event with a weapon.

Some additional models analyzed:

Logistic regression

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# Scale the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize LogisticRegression model
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test_scaled)
```

```
Logistic Regression accuracy: 0.9960965423830844
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    105900
           1       1.00      0.00      0.00       416

    accuracy                           1.00    106316
   macro avg       1.00      0.50      0.50    106316
weighted avg       1.00      1.00      0.99    106316

[[105900      0]
 [   415      1]]
```

MultinomailNB

```python
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(X_train, y_train)

# metrics
from sklearn.metrics import accuracy_score
y_pred = nb.predict(X_test)
print(f"accuracy: {accuracy_score(y_test, y_pred)}")
```

```
✓ from sklearn.naive_bayes import MultinomialNB ···

accuracy: 0.9954851574551338
```

Evaluation

Every single model has a high accuracy. It can be due to a high amount of information and attributes for each event. The top features plot will be decision-makers a good insight for proactive resource management.

Can we predict if an arrest will be made?

Data Preparation

For data preparation, a list of all the attributes prior to an arrest are going to be selected and processed as follows:

1st: new dataset with all attributes related

2nd: set a target variable

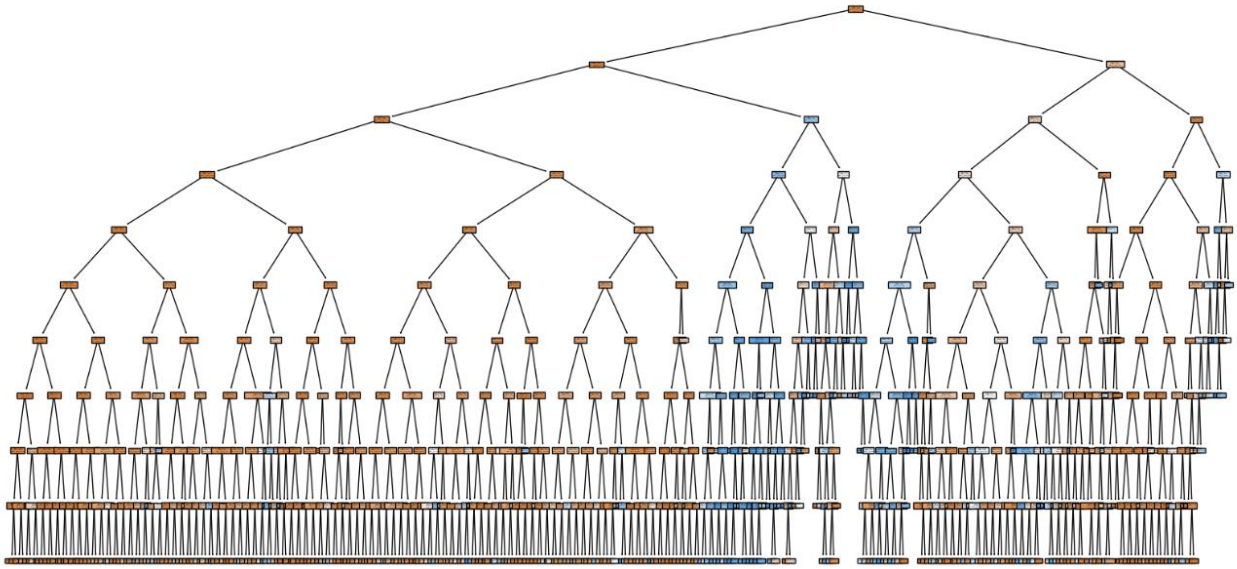
3rd: split the data

```
Run Cell | Run Above | Debug Cell | Go to [204]
#%% Can we predict if an arrest will be made?

arrests = crimes[['armed', 'pct', 'hour', 'month', 'perobs', 'crimsusp_grouped', 'ac_rept', 'ac_inves', 'rf_vcrim', 'rf_othsw',
                  'ac_proxm', 'rf_attir', 'cs_objcs', 'cs_descr', 'cs_casng', 'cs_lkout', 'rf_vcact', 'cs_cloth', 'cs_drgtr',
                  'ac_evasv', 'ac_assoc', 'cs_furtv', 'rf_rfcmp', 'ac_cgdir', 'rf_verbl', 'cs_vcrim', 'cs_bulge', 'cs_other',
                  'ac_incid', 'ac_time', 'rf_knowl', 'ac_stsnd', 'ac_other', 'sex', 'race', 'ht_inches', 'weight', 'city', 'arstmade']]

# Convert 'arstmade' to numerical values
arrests['arstmade'] = arrests['arstmade'].map({'N': 0, 'Y': 1})

# Encode categorical variables
categorical_cols = arrests.select_dtypes(include=['object']).columns
encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_cols = pd.DataFrame(encoder.fit_transform(arrests[categorical_cols]), columns=encoder.get_feature_names_out(categorical_cols))

# Drop original categorical columns and concatenate encoded columns
arrests = arrests.drop(columns=categorical_cols).reset_index(drop=True)
arrests = pd.concat([arrests, encoded_cols], axis=1)

# Ensure 'arstmade' is still in the DataFrame
if 'arstmade' not in arrests.columns:
    raise KeyError("'arstmade' column is missing from the DataFrame")

# Define the feature set and the target variable
X = arrests.drop(columns=['arstmade'])
y = arrests['arstmade']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Data modelling

The following models are run to search for the best accuracy:

Decision tree:



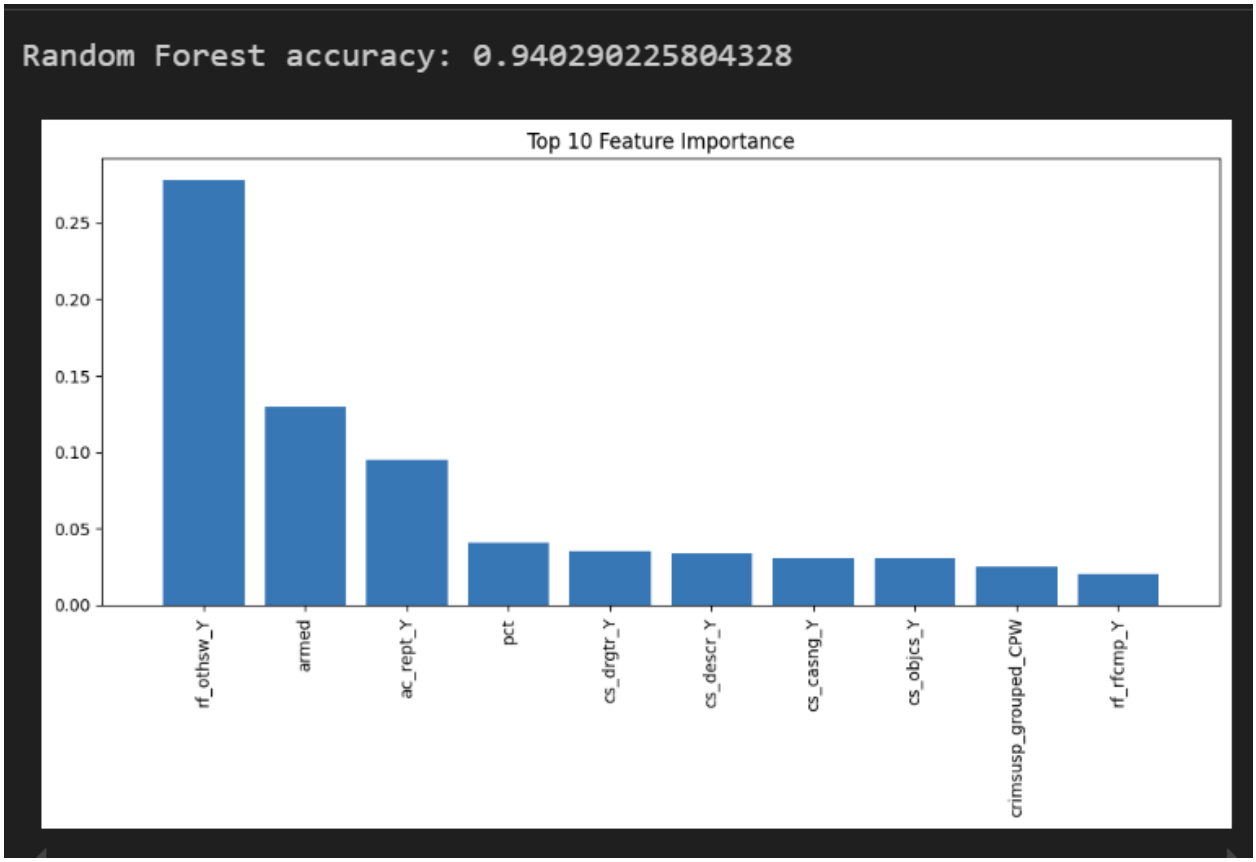
```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 20}
accuracy: 0.9424263516309869
```

Kneighbors Clasiffier:

```
knn = KNeighborsClassifier(n_neighbors=2)

knn.fit(X_train, y_train)

# metrics
from sklearn.metrics import accuracy_score
y_pred = knn.predict(X_test)
print(f"accuracy: {accuracy_score(y_test, y_pred)}")
```

```
✓ knn = KNeighborsClassifier(n_neighbors=2) ···

accuracy: 0.9372342827043907
```

Random forest:



Evaluation

Every single model has a high accuracy. It can be due to a high amount of information and attributes for each event, as well as in the previous question.

Can we predict what force will be used by an officer given the other variables?

Data Preparation

Since there are several columns related to police force: “pf_xxx “ , a multi output classifier is going to be used. The following code prepared the data with all related columns:

```
Run Cell | Run Above | Debug Cell | Go to [27]
# %% Can we predict what force will be used by an officer given the other variables?

pf_force = crimes[['armed', 'pct', 'hour', 'month', 'perobs', 'crimsusp_grouped', 'ac_rept', 'ac_inves', 'rf_vcrim', 'rf_othsw',
                   'ac_proxm', 'rf_attir', 'cs_objcs', 'cs_descr', 'cs_casng', 'cs_lkout', 'rf_vcact', 'cs_cloth', 'cs_drgtr',
                   'ac_evasv', 'ac_assoc', 'cs_furtv', 'rf_rfcmp', 'ac_cgdir', 'rf_verbl', 'cs_vcrim', 'cs_bulge', 'cs_other',
                   'ac_incid', 'ac_time', 'rf_knowl', 'ac_stsnd', 'ac_other', 'sex', 'race', 'ht_inches', 'weight', 'city',
                   'pf_hands', 'pf_wall', 'pf_grnd', 'pf_drwep', 'pf_ptwep', 'pf_baton', 'pf_hcuff', 'pf_pepsp', 'pf_other']]

# Convert 'pf_xx' to numerical values
pf_force['pf_hands'] = pf_force['pf_hands'].map({'N': 0, 'Y': 1})
pf_force['pf_wall'] = pf_force['pf_wall'].map({'N': 0, 'Y': 1})
pf_force['pf_grnd'] = pf_force['pf_grnd'].map({'N': 0, 'Y': 1})
pf_force['pf_drwep'] = pf_force['pf_drwep'].map({'N': 0, 'Y': 1})
pf_force['pf_ptwep'] = pf_force['pf_ptwep'].map({'N': 0, 'Y': 1})
pf_force['pf_baton'] = pf_force['pf_baton'].map({'N': 0, 'Y': 1})
pf_force['pf_hcuff'] = pf_force['pf_hcuff'].map({'N': 0, 'Y': 1})
pf_force['pf_pepsp'] = pf_force['pf_pepsp'].map({'N': 0, 'Y': 1})
pf_force['pf_other'] = pf_force['pf_other'].map({'N': 0, 'Y': 1})

# Encode categorical variables
categorical_cols = pf_force.select_dtypes(include=['object']).columns
encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_cols = pd.DataFrame(encoder.fit_transform(pf_force[categorical_cols]), columns=encoder.get_feature_names_out(categorical_cols

# Drop original categorical columns and concatenate encoded columns
pf_force = pf_force.drop(columns=categorical_cols).reset_index(drop=True)
pf_force = pd.concat([pf_force, encoded_cols], axis=1)

# Define the feature set and the target variable
X = pf_force.drop(columns=['pf_hands', 'pf_wall', 'pf_grnd', 'pf_drwep', 'pf_ptwep', 'pf_baton', 'pf_hcuff', 'pf_pepsp', 'pf_other'])
y = pf_force[['pf_hands', 'pf_wall', 'pf_grnd', 'pf_drwep', 'pf_ptwep', 'pf_baton', 'pf_hcuff', 'pf_pepsp', 'pf_other']]
```

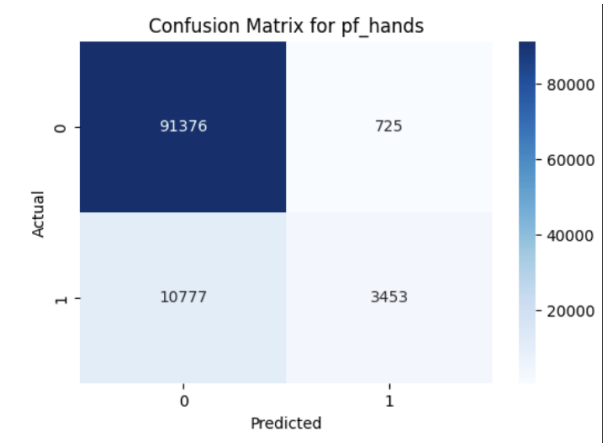
Data modelling

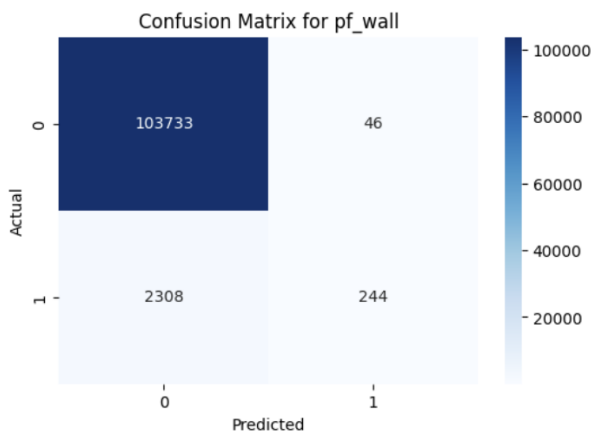
First model to run:

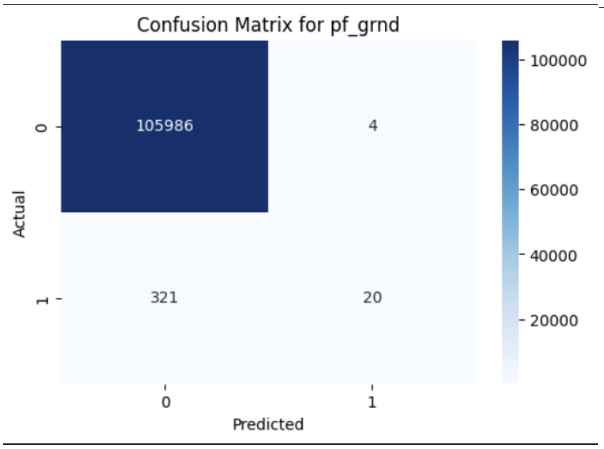
Random forest

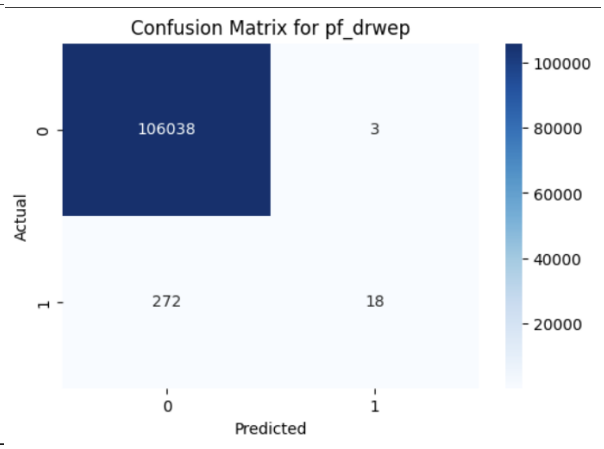
```
# Initialize the base model
base_model = RandomForestClassifier(random_state=42)

# Wrap it with MultiOutputClassifier
model = MultiOutputClassifier(base_model, n_jobs=-1)
```

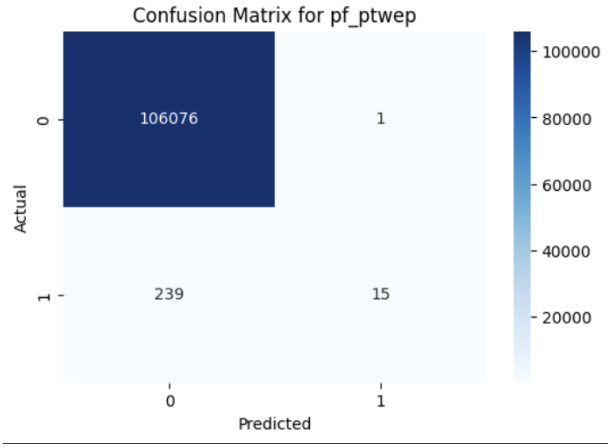
With the following results for each variable, accuracy and statistical values can be found in the Jupiter file attached.

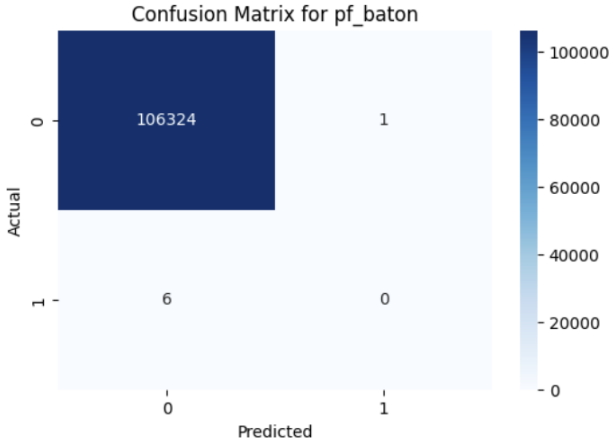


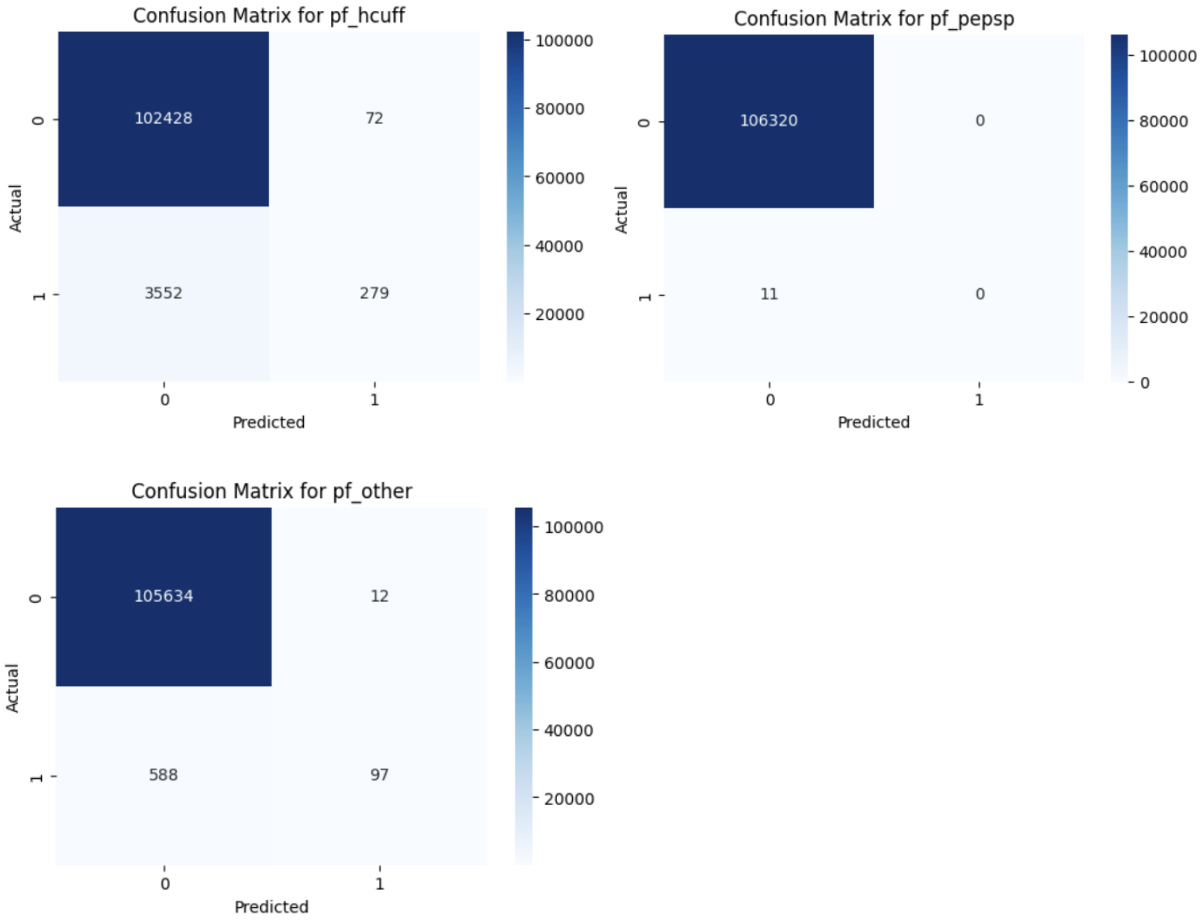












Second model to run:

KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multioutput import MultiOutputClassifier

# Initialize the base model
base_model = KNeighborsClassifier()

# Wrap it with MultiOutputClassifier
model = MultiOutputClassifier(base_model, n_jobs=1)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

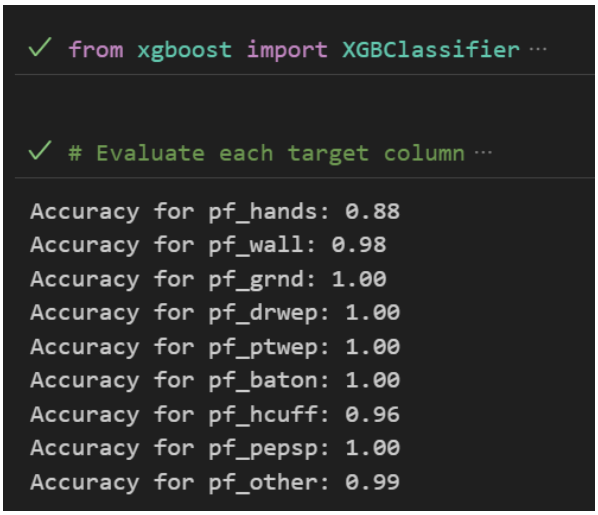
```
✓ from sklearn.metrics import accuracy_score ···

Accuracy for pf_hands: 0.86
ns...  curacy for pf_wall: 0.98
Accuracy for pf_grnd: 1.00
Accuracy for pf_drwep: 1.00
Accuracy for pf_ptwep: 1.00
Accuracy for pf_baton: 1.00
Accuracy for pf_hcuff: 0.96
Accuracy for pf_pepsp: 1.00
Accuracy for pf_other: 0.99
```

Based on this accuracy, the model is overfitted. The random forest model is a better option.

3rd model run:

XGBClassifier

```
from xgboost import XGBClassifier
from sklearn.multioutput import MultiOutputClassifier

# Initialize the base model
base_model = XGBClassifier(random_state=42)

# Wrap it with MultiOutputClassifier
model = MultiOutputClassifier(base_model, n_jobs=1)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```



Same accuracy results due to overfitting.

Conclusions & Recommendations

This study applied the Apriori algorithm, clustering techniques, and predictive modeling to analyze NYC SQF crime statistics. Each method provided unique insights and demonstrated the potential for enhancing crime prevention strategies.

Apriori Model: The Apriori algorithm effectively identified frequent itemsets and association rules within the crime data. This method revealed significant co-occurrences of crime types, locations, and times, offering valuable insights for resource allocation and strategic planning. However, the model's reliance on predefined thresholds for support and confidence can limit its flexibility and may overlook less frequent but still important associations.

Clustering: Clustering techniques successfully segmented the crime data into meaningful groups. These clusters highlighted patterns and hotspots of criminal activity, which are crucial for targeted policing. The primary challenge with clustering lies in selecting the optimal number of clusters, attributes and interpreting the results, which can be subjective and require domain expertise.

Predictive Modeling: The Random Forest model demonstrated high accuracy in predicting crimes, police force or chances of a suspect to be armed based on historical data. Its ability to handle large datasets and complex interactions between variables makes it a robust tool for predictive policing.