1.Business Understanding

A Terry stop is a police procedure that permits law enforcement officers to briefly detain an individual based on reasonable suspicion of criminal activity. Terry Stops are controversial because they give police a wider scope of authority or freedom to make decisions which may lead to wrongful arrests. If most stops don't lead to arrests, it raises questions about whether they are fair or effective, a concern to policy makers and civil rights organizations.

Stakeholder

The primary stakeholder is the Seattle Police Department (SPD) leadership and the City Council, who oversee policing practices. They aim to ensure stops are efficient, fair, and resource-effective amid public scrutiny on racial bias and over-policing.

Business Problem

Terry Stops consume significant officer time and resources. Predicting whether a stop will lead to an arrest (`Arrest Flag: Y/N`) can help SPD prioritize high-risk stops, allocate resources efficiently, and identify patterns for officer training to reduce low-yield stops. This is a binary classification problem where the target variable is `Arrest Flag` (Y = positive class, N = negative class)

Objectives

Main Objectives

To develop a machine learning model that predicts whether a Terry Stop conducted by the Seattle Police Department will result in an arrest, optimizing for precision to minimize unnecessary stops while maintaining acceptable recall to identify high-risk stops, thereby supporting efficient resource allocation and equitable policing practices.

Specific Objectives

1. To preprocess and explore the Terry Stops dataset to identify key features influencing arrest outcomes, such as call type, precinct, and subject demographics.
2. To build and compare multiple classification models (logistic regression and decision trees) to determine the most effective model for predicting arrests.
3. To tune the selected model to achieve a balance between precision and recall, prioritizing the reduction of false positives to enhance public trust.
4. To interpret model results to provide actionable insights for SPD leadership, such as prioritizing high-risk stops and addressing potential biases in policing practices.\n",

Research Questions

1. Which features (e.g., `Initial Call Type`, `Precinct`, `Subject Perceived Race`) are the strongest predictors of whether a Terry Stop results in an arrest?
2. How effectively can a machine learning model predict arrests while minimizing false positives (i.e., achieving high precision)?
3. How do logistic regression and decision tree models compare in terms of precision, recall, and interpretability for this classification task?
4. What actionable recommendations can be derived from the model's predictions to improve SPD's resource allocation and training on equitable policing?"

Success Metrics

- **Precision**: Proportion of predicted arrests that are correct (minimize false positives to avoid wrongful arrests and public scrutiny).
- **Recall**: Proportion of actual arrests correctly predicted (ensure high-risk stops are not missed).
- **Baseline**: A dummy classifier predicting the majority class (no arrest) achieves ~85% accuracy due to class imbalance (15% arrests). We aim to improve precision and recall over this baseline.

2. Data Understanding

The dataset to be used in this project is from Seattle Government. Each row is a unique record of a Terry stop, as reported by the officer conducting the stop.

Rows 64.8K Columns 23 Each row is a A unique record of a Terry Stop, as reported by the officer conducting the stop.

Columns

1. **Subject Age Group** - Subject Age Group (10 year increments) as reported by the officer.

2. **Subject ID** - Key(Unique Identifier)

3. **GO / SC Num** - General Offense or Street Check number, relating the Terry Stop to the parent report. This field may have a one to many relationship in the data.

4. **Terry Stop ID** - Key identifying unique Terry Stop reports.

5. **Stop Resolution** - Resolution of the stop as reported by the officer.

6. **Weapon Type** - Type of weapon, if any, identified during a search or frisk of the subject. Indicates "None" if no weapons was found.

7. **Officer ID** - Key identifying unique officers in the dataset.

8. **Officer YOB** - Year of birth, as reported by the officer.

9. **Officer Gender** - Gender of the officer, as reported by the officer.

10. **Officer Race** - Race of the officer, as reported by the officer.

11. **Subject Perceived Race** - Perceived race of the subject, as reported by the officer.
12. **Subject Perceived Gender** - Perceived gender of the subject, as reported by the officer.
13. **Reported Date** - Date the report was filed in the Records Management System (RMS). Not necessarily the date the stop occurred but generally within 1 day.
14. **Reported Time** - Time the stop was reported in the Records Management System (RMS). Not the time the stop occurred but generally within 10 hours.
15. **Initial Call Type** - Initial classification of the call as assigned by 911.
16. **Final Call Type** - Final classification of the call as assigned by the primary officer closing the event.
17. **Call Type** - How the call was received by the communication center.
18. **Officer Squad** - Functional squad assignment (not budget) of the officer as reported by the Data Analytics Platform (DAP).
19. **Arrest Flag** - Indicator of whether a "physical arrest" was made, of the subject, during the Terry Stop. Does not necessarily reflect a report of an arrest in the Records Management System (RMS).
20. **Frisk Flag** - Indicator of whether a "frisk" was conducted, by the officer, of the subject, during the Terry Stop.
21. **Precinct** - Precinct of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.
22. **Sector** - Sector of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.
23. **Beat** - Beat of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.

In [77]:
```
# Importing the necessecary libraries
import pandas as pd # Data manipulation
import numpy as np # Mathematics
import matplotlib.pyplot as plt # Visualization
# This makes our graphs show up in the notebook
%matplotlib inline

import seaborn as sns # Advanved visualization
# Let's set a style for our graphs to make them look nicer
sns.set_style("whitegrid")
```

In [78]:
```
# Setting the maximum display of columns to 30
# This is to get a look at all the columns
pd.options.display.max_columns = 30
```

In [79]:
```
# We read the CSV file into a Pandas DataFrame, which is like a super-powered Excel

df = pd.read_csv(r"C:\Users\Jeremy\Downloads\Terry_Stops_20250908.csv")

# Let's see what we're working with!
```

```
print("Dataset Shape:", df.shape) # Tells us (number of rows, number of columns)
df.head() # Shows the first 5 rows
```

Dataset Shape: (64699, 23)

Out[79]:

	Subject Age Group	Subject ID	GO / SC Num	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Office Gende
0	46 - 55	-1	20180000275629	481899	Field Contact	NaN	8544	1993	Femal
1	36 - 45	53986235598	20240000029589	53986202139	Field Contact	-	8723	1994	Mal
2	26 - 35	-1	20170000036835	234548	Offense Report	NaN	4852	1953	Mal
3	18 - 25	-1	20180000271087	445585	Offense Report	NaN	8588	1986	Femal
4	18 - 25	-1	20150000002928	54115	Field Contact	NaN	7745	1988	Femal

In [80]:

```
# Getting the information of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64699 entries, 0 to 64698
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Subject Age Group         64699 non-null  object
 1   Subject ID                64699 non-null  int64 
 2   GO / SC Num               64699 non-null  int64 
 3   Terry Stop ID             64699 non-null  int64 
 4   Stop Resolution           64699 non-null  object
 5   Weapon Type               32134 non-null  object
 6   Officer ID                64699 non-null  object
 7   Officer YOB               64699 non-null  int64 
 8   Officer Gender            64699 non-null  object
 9   Officer Race              64699 non-null  object
 10  Subject Perceived Race    64699 non-null  object
 11  Subject Perceived Gender  64699 non-null  object
 12  Reported Date             64699 non-null  object
 13  Reported Time             64699 non-null  object
 14  Initial Call Type         64699 non-null  object
 15  Final Call Type           64699 non-null  object
```

```
 16  Call Type                64699 non-null  object
 17  Officer Squad            64133 non-null  object
 18  Arrest Flag              64699 non-null  object
 19  Frisk Flag               64699 non-null  object
 20  Precinct                 64699 non-null  object
 21  Sector                   64699 non-null  object
 22  Beat                     64699 non-null  object
dtypes: int64(4), object(19)
memory usage: 11.4+ MB
```

In [81]:

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

Out[81]:

```
Subject Age Group               0
Subject ID                      0
GO / SC Num                     0
Terry Stop ID                   0
Stop Resolution                 0
Weapon Type                 32565
Officer ID                      0
Officer YOB                     0
Officer Gender                  0
Officer Race                    0
Subject Perceived Race          0
Subject Perceived Gender        0
Reported Date                   0
Reported Time                   0
Initial Call Type               0
Final Call Type                 0
Call Type                       0
Officer Squad                 566
Arrest Flag                     0
Frisk Flag                      0
Precinct                        0
Sector                          0
Beat                            0
dtype: int64
```

Observations

- There are a total of 23 columns and 64699 rows in this dataset
- From this information, we can use `Arrest Flag` as the target variable
- There are several null values from my data
- There's a mixture of both categorical data and numeric data
- Some of the predictor variables include `weapon_type` , `Frisk Flag` , `reported_time` among others.

3. Data Preparation

This is where data cleaning, preprocessing, analysis is done.

3.1 Data cleaning

Data from the real world is often messy. We need to clean it up before we can use it.

This is where the following is done:

- Dealing with missing values
- Checking for duplicates
- Dealing with outliers among others.

In [82]:
```
# Check for duplicates
df.duplicated().sum()
```

Out[82]:
```
np.int64(0)
```

No duplicates.

In [83]:
```
# Check for missing values
print("Missing Values in Each Column:")
print(df.isnull().sum()) # This counts true 'NaN' values
```

```
Missing Values in Each Column:
Subject Age Group               0
Subject ID                      0
GO / SC Num                     0
Terry Stop ID                   0
Stop Resolution                 0
Weapon Type                 32565
Officer ID                      0
Officer YOB                     0
Officer Gender                  0
Officer Race                    0
Subject Perceived Race          0
Subject Perceived Gender        0
Reported Date                   0
Reported Time                   0
Initial Call Type               0
Final Call Type                 0
Call Type                       0
Officer Squad                 566
Arrest Flag                     0
Frisk Flag                      0
Precinct                        0
Sector                          0
Beat                            0
dtype: int64
```

In [84]:
```
# Creating a function to replace '-' with 'Unknown'
def replace_func(data):
    data = data.replace(to_replace= '-', value= 'Unknown', inplace= True)
    return data
```

In [85]:
```
# Reolacing NaN with Unknown from weapon type column
df['Weapon Type'] = df['Weapon Type'].replace(to_replace= np.nan, value= 'Unknown')

# Replacing '-' with 'Unknown'
replace_func(df['Weapon Type'])
```

The dash(-) in this case has been used as a placeholder, so we'll have to add it to the Unknown value. Also the null values will be added to the Unknown value. We can't use mode in this case to fill in the missing values as the values in this column are very sensitive so its best if we just add them to the None category.

In [86]:
```
# mapping dictionary for merging categories

weapon_map = {
    # Knives
    "Knife/Cutting/Stabbing Instrument": "Knife/Cutting",
    "Lethal Cutting Instrument": "Knife/Cutting",

    # Firearms (general/other)
    "Firearm": "Firearm",
    "Firearm Other": "Firearm",
    "Other Firearm": "Firearm",
```

```
    "Firearm (unk type)": "Firearm",
    "Rifle": "Firearm",
    "Shotgun": "Firearm",
    "Handgun": "Firearm",
    "Automatic Handgun": "Firearm",

    # Blunt objects
    "Blunt Object/Striking Implement": "Blunt Object",
    "Club": "Blunt Object",
    "Blackjack": "Blunt Object",
    "Brass Knuckles": "Blunt Object",
    "Club, Blackjack, Brass Knuckles": "Blunt Object",

    # Chemicals
    "Mace/Pepper Spray": "Chemical",
    "Poison": "Chemical",

    # Other weapons
    "Taser/Stun Gun": "Other",
    "Fire/Incendiary Device": "Other",
    "Personal Weapons (hands, feet, etc.)": "Other",

    # None
    "None/Not Applicable": "None"

}

# apply mapping
df['Weapon Type'] = df['Weapon Type'].replace(weapon_map)

# check results
df['Weapon Type'].value_counts()
```

Out[86]:

```
Weapon Type
Unknown          60566
Knife/Cutting     2973
Firearm            780
Blunt Object       259
Chemical            65
Other               35
None                21
Name: count, dtype: int64
```

In [87]:

```
# Dropping the remaining null values as they are little and may not impact the datas
df.dropna(inplace = True)
df.isna().sum() # Checking if the changes have been made
```

Out[87]:

```
Subject Age Group           0
Subject ID                  0
GO / SC Num                 0
Terry Stop ID               0
Stop Resolution             0
Weapon Type                 0
Officer ID                  0
Officer YOB                 0
Officer Gender              0
Officer Race                0
Subject Perceived Race      0
Subject Perceived Gender    0
Reported Date               0
Reported Time               0
Initial Call Type           0
Final Call Type             0
Call Type                   0
Officer Squad               0
Arrest Flag                 0
```

```
Frisk Flag              0
Precinct                0
Sector                  0
Beat                    0
dtype: int64
```

In [88]:
```
# Matches the indexing with the current number of rows after dropping the null rows
df = df.reset_index(drop = True)
```

In [89]:
```
# Checking if the indexing has worked by looking at the total entries
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64133 entries, 0 to 64132
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Subject Age Group         64133 non-null  object
 1   Subject ID                64133 non-null  int64 
 2   GO / SC Num               64133 non-null  int64 
 3   Terry Stop ID             64133 non-null  int64 
 4   Stop Resolution           64133 non-null  object
 5   Weapon Type               64133 non-null  object
 6   Officer ID                64133 non-null  object
 7   Officer YOB               64133 non-null  int64 
 8   Officer Gender            64133 non-null  object
 9   Officer Race              64133 non-null  object
 10  Subject Perceived Race    64133 non-null  object
 11  Subject Perceived Gender  64133 non-null  object
 12  Reported Date             64133 non-null  object
 13  Reported Time             64133 non-null  object
 14  Initial Call Type         64133 non-null  object
 15  Final Call Type           64133 non-null  object
 16  Call Type                 64133 non-null  object
 17  Officer Squad             64133 non-null  object
 18  Arrest Flag               64133 non-null  object
 19  Frisk Flag                64133 non-null  object
 20  Precinct                  64133 non-null  object
 21  Sector                    64133 non-null  object
 22  Beat                      64133 non-null  object
dtypes: int64(4), object(19)
memory usage: 11.3+ MB
```

In [90]:
```
# Standardizing formats
df['Reported Date'] = pd.to_datetime(df['Reported Date'])
df['Reported Date']
```

Out[90]:
```
0       2018-07-30
1       2024-02-01
2       2017-01-30
3       2018-07-23
4       2015-06-17
           ...    
64128   2015-09-29
64129   2021-12-19
64130   2016-05-23
64131   2023-03-03
64132   2022-07-05
Name: Reported Date, Length: 64133, dtype: datetime64[ns]
```

In [91]:
```
df['Reported Time'] = pd.to_datetime(df['Reported Time'])
```

```
C:\Users\Jeremy\AppData\Local\Temp\ipykernel_23716\2153869983.py:1: UserWarning: Coul
d not infer format, so each element will be parsed individually, falling back to `dat
eutil`. To ensure parsing is consistent and as-expected, please specify a format.
  df['Reported Time'] = pd.to_datetime(df['Reported Time'])
```

In [92]:
```
# Change the dash placeholder with Unknown
replace_func(df['Call Type'])
```

In [93]:
```
# Get stop year so that I can compute the officer's age
df["stop_year"] = df["Reported Date"].dt.year

# Compute officer age at time of stop
df["Officer Age"] = df["stop_year"] - df["Officer YOB"]
```

In [94]:
```
df["Officer Age"] = df["Officer Age"].astype(int)
```

In [95]:
```
type(df["Officer Age"])
```

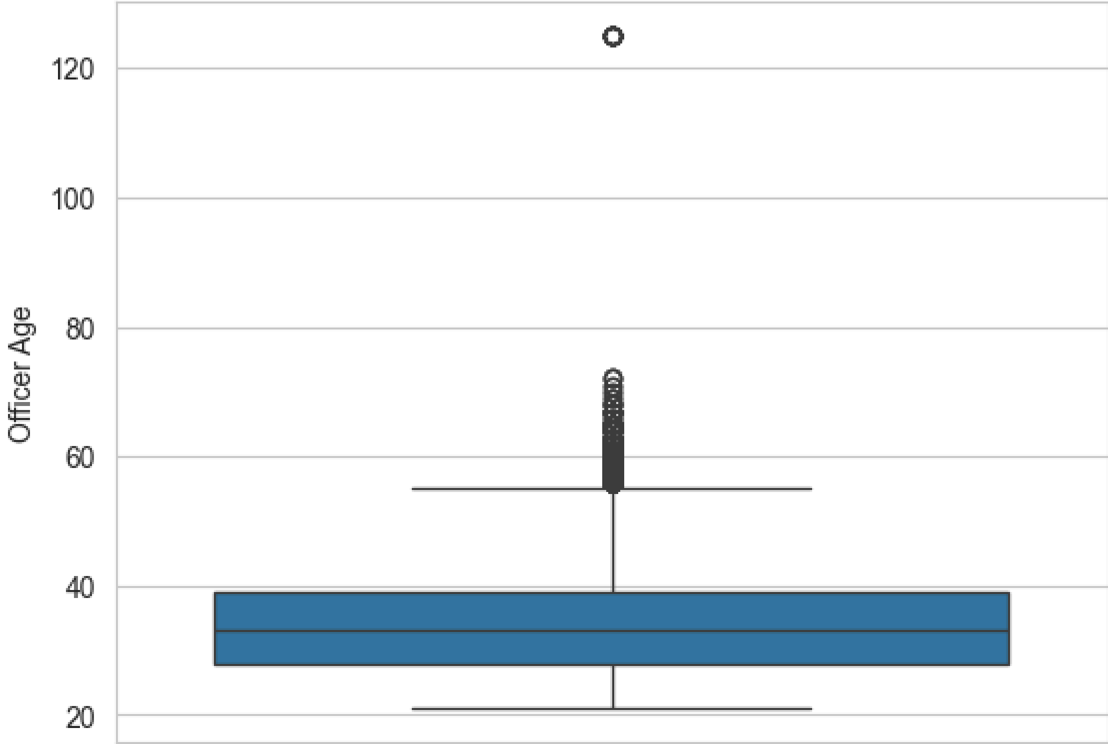
Out[95]:
```
pandas.core.series.Series
```

In [96]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64133 entries, 0 to 64132
Data columns (total 25 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Subject Age Group         64133 non-null  object
 1   Subject ID                64133 non-null  int64
 2   GO / SC Num               64133 non-null  int64
 3   Terry Stop ID             64133 non-null  int64
 4   Stop Resolution           64133 non-null  object
 5   Weapon Type               64133 non-null  object
 6   Officer ID                64133 non-null  object
 7   Officer YOB               64133 non-null  int64
 8   Officer Gender            64133 non-null  object
 9   Officer Race              64133 non-null  object
 10  Subject Perceived Race    64133 non-null  object
 11  Subject Perceived Gender  64133 non-null  object
 12  Reported Date             64133 non-null  datetime64[ns]
 13  Reported Time             64133 non-null  datetime64[ns]
 14  Initial Call Type         64133 non-null  object
 15  Final Call Type           64133 non-null  object
 16  Call Type                 64133 non-null  object
 17  Officer Squad             64133 non-null  object
 18  Arrest Flag               64133 non-null  object
 19  Frisk Flag                64133 non-null  object
 20  Precinct                  64133 non-null  object
 21  Sector                    64133 non-null  object
 22  Beat                      64133 non-null  object
 23  stop_year                 64133 non-null  int32
 24  Officer Age               64133 non-null  int64
dtypes: datetime64[ns](2), int32(1), int64(5), object(17)
memory usage: 12.0+ MB
```

In [97]:
```
# Creating a boxplot to look for outliers
sns.boxplot(data= df["Officer Age"])
```

Out[97]:
```
<Axes: ylabel='Officer Age'>
```



In [98]:

```
# Removing outliers
Q1 = df["Officer Age"].quantile(0.25)
Q3 = df["Officer Age"].quantile(0.75)

# Calculating the IQR( IQR= Q3- Q1)
IQR = Q3 - Q1

# Detecting the outliers
lower_fence = Q1 - (1.5*IQR)
upper_fence = Q3 + (1.5*IQR)

# Removing the Outliers
df["Officer Age"] = (df["Officer Age"] >= lower_fence) & (df["Officer Age"] <= upper
```

In [99]:

```
# Check for the values in the subject age category
df['Subject Age Group'].value_counts(normalize= True)
```

Out[99]:

```
Subject Age Group
26 - 35         0.333588
36 - 45         0.227309
18 - 25         0.186254
46 - 55         0.126565
56 and Above    0.052890
1 - 17          0.036861
-               0.036533
Name: proportion, dtype: float64
```

In [100...

```
# Since the dash placeholder contains 3% of the data in that column and I have no op
# We've opted to drop the rows with the dash placeholder in that specific column
df = df[df['Subject Age Group'] != '-']
df = df.reset_index(drop = True) # Make the indexing correct after manipulating the
```

In [101...

```
# Check for the values in the Frisk Flag category
df['Frisk Flag'].value_counts(normalize= True)
```

Out[101...

```
Frisk Flag
N    0.750607
Y    0.243098
```

```
-    0.006296
Name: proportion, dtype: float64
```

In [102…

```
# Since the dash placeholder contains 0.6% of the data in that column and I have no
# I've opted to drop the rows with the dash placeholder in that specific column
df = df[df['Frisk Flag'] != '-']
df = df.reset_index(drop = True)
```

In [103…

```
df['Arrest Flag']= df['Arrest Flag'].map({'N': 0, 'Y': 1}) # To reduce bias in the m
```

In [104…

```
# For simplicity, we might drop columns with too many missing values or that are har
# We'll also drop columns that are just IDs or exact times for now.
columns_to_drop = ['Subject ID', 'GO / SC Num', 'Terry Stop ID', 'Officer ID', 'Repo
df_clean = df.drop(columns=columns_to_drop, errors='ignore')

# Let's also just use a few key features for our first model to keep it simple.
# We can add more later to see if it improves performance!
selected_features = ['Subject Perceived Race', 'Subject Perceived Gender', 'Weapon T
df_model = df_clean[selected_features + ['Arrest Flag']].copy()

# Drop any rows where our selected features are still missing
df_model.dropna(inplace=True)
print("New shape of our modeling dataset:", df_model.shape)
```

```
New shape of our modeling dataset: (61401, 7)
```

4. Exploratory Data Analysis (EDA)

Now let's make some graphs to understand our data better.

1. Arrests by Subject's Perceived Race

The goal is to se if there is disparity in arrest rates across different racial groups

In [105…

```
# Getting a table of subject's race and arrest flag
df.groupby(['Subject Perceived Race', 'Arrest Flag']).size().unstack(fill_value= 0)
```

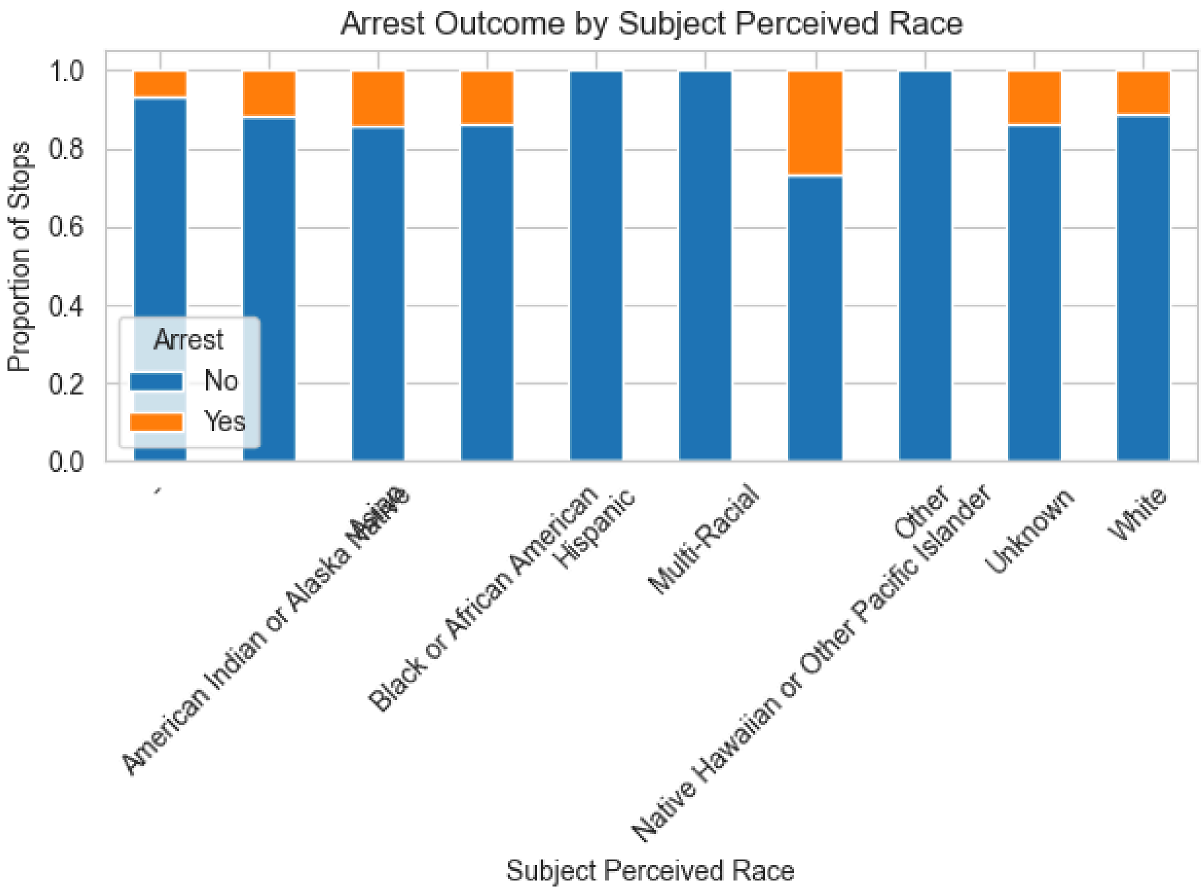
Out[105…

Arrest Flag	0	1
Subject Perceived Race		
-	1318	97
American Indian or Alaska Native	1490	195
Asian	1842	303
Black or African American	15995	2534
Hispanic	1634	0
Multi-Racial	781	0
Native Hawaiian or Other Pacific Islander	130	47
Other	146	0
Unknown	3822	619
White	26984	3464

In [106…

```
plt.figure(figsize=(12, 6))
# Use a crosstab to count arrests vs race
arrest_by_race = pd.crosstab(df_model['Subject Perceived Race'], df_model['Arrest Fl
arrest_by_race.plot(kind='bar', stacked=True)
plt.title('Arrest Outcome by Subject Perceived Race')
plt.xlabel('Subject Perceived Race')
plt.ylabel('Proportion of Stops')
plt.legend(title='Arrest', labels=['No', 'Yes'])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



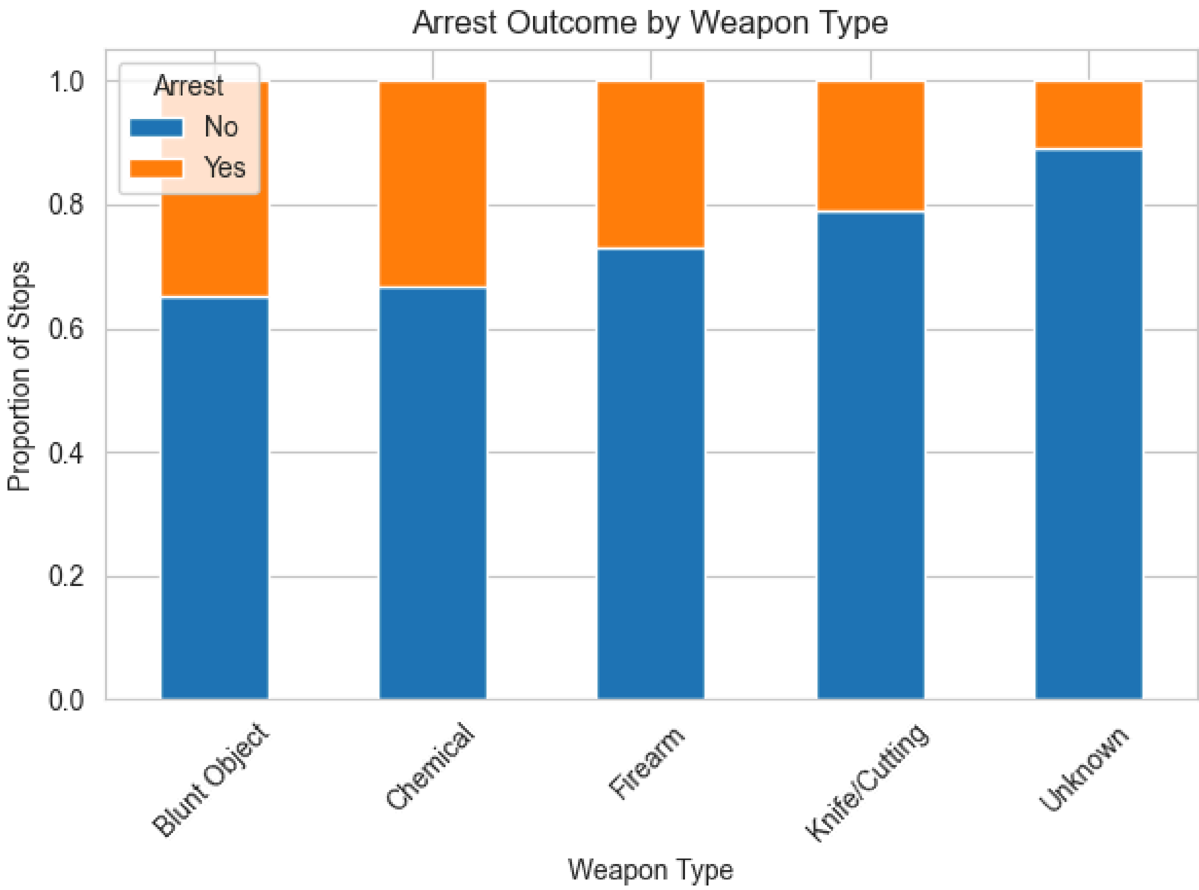
2. Arrests by Weapon Type

The goal is to see if weapon presence is a strong predictor of arrest.

In [107…

```
top_weapons = df_model['Weapon Type'].value_counts().nlargest(5).index
df_top_weapons = df_model[df_model['Weapon Type'].isin(top_weapons)]

plt.figure(figsize=(10, 5))
arrest_by_weapon = pd.crosstab(df_top_weapons['Weapon Type'], df_top_weapons['Arrest
arrest_by_weapon.plot(kind='bar', stacked=True)
plt.title('Arrest Outcome by Weapon Type')
plt.xlabel('Weapon Type')
plt.ylabel('Proportion of Stops')
plt.legend(title='Arrest', labels=['No', 'Yes'])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```

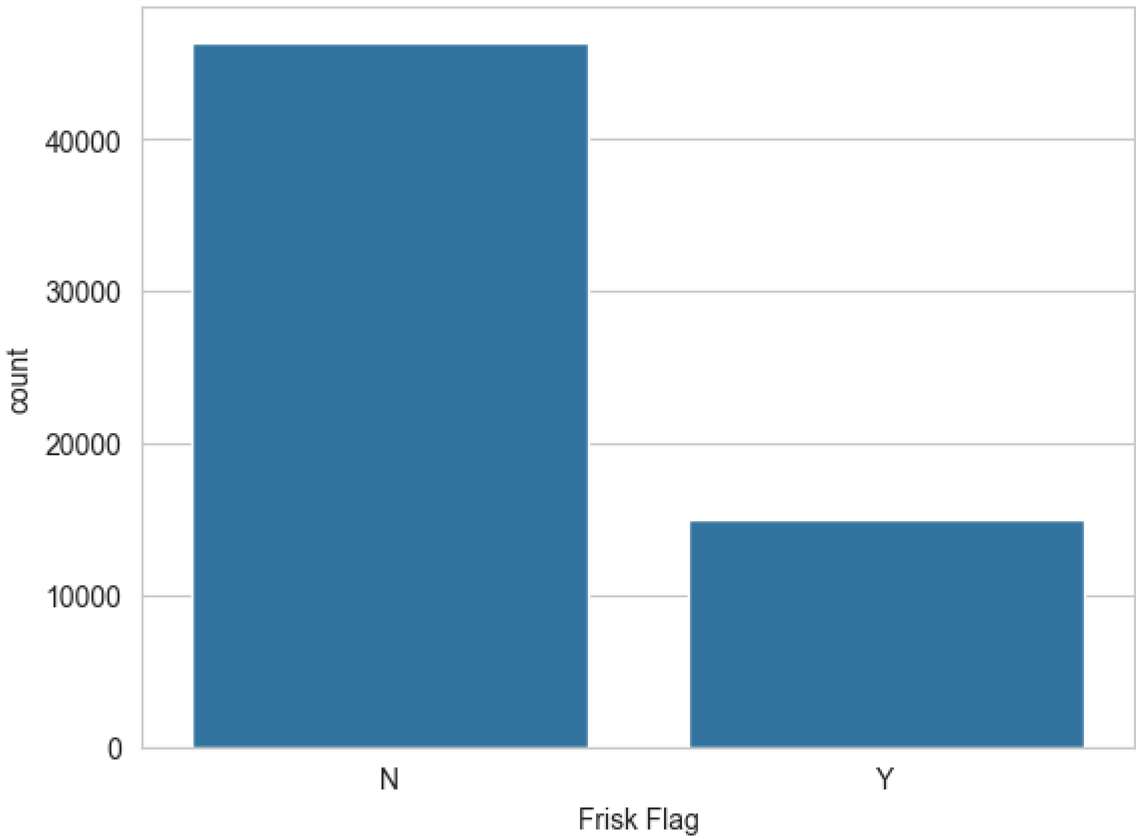


In [108…

```
# countplot for the Frisk Flag variable
sns.countplot(x='Frisk Flag', data=df)
```

Out[108…

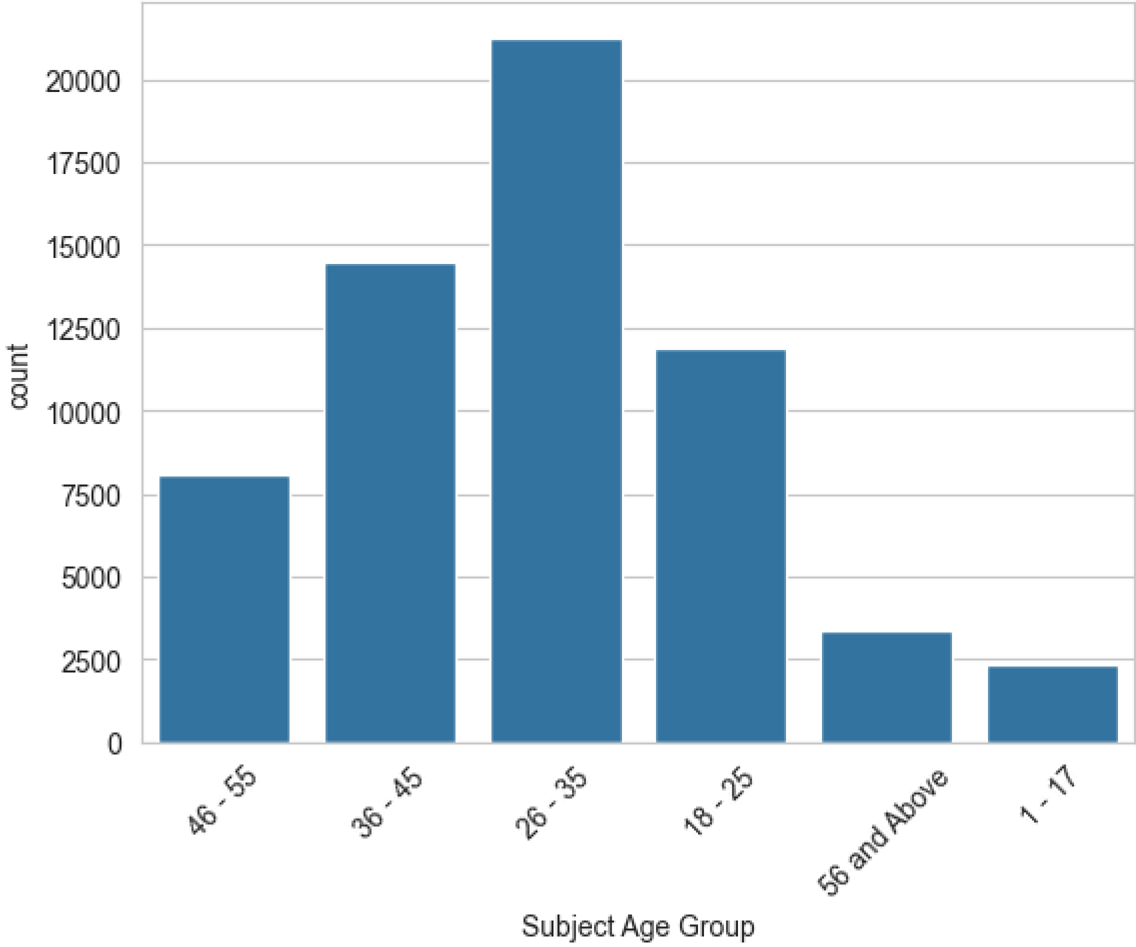
```
<Axes: xlabel='Frisk Flag', ylabel='count'>
```



In [109…

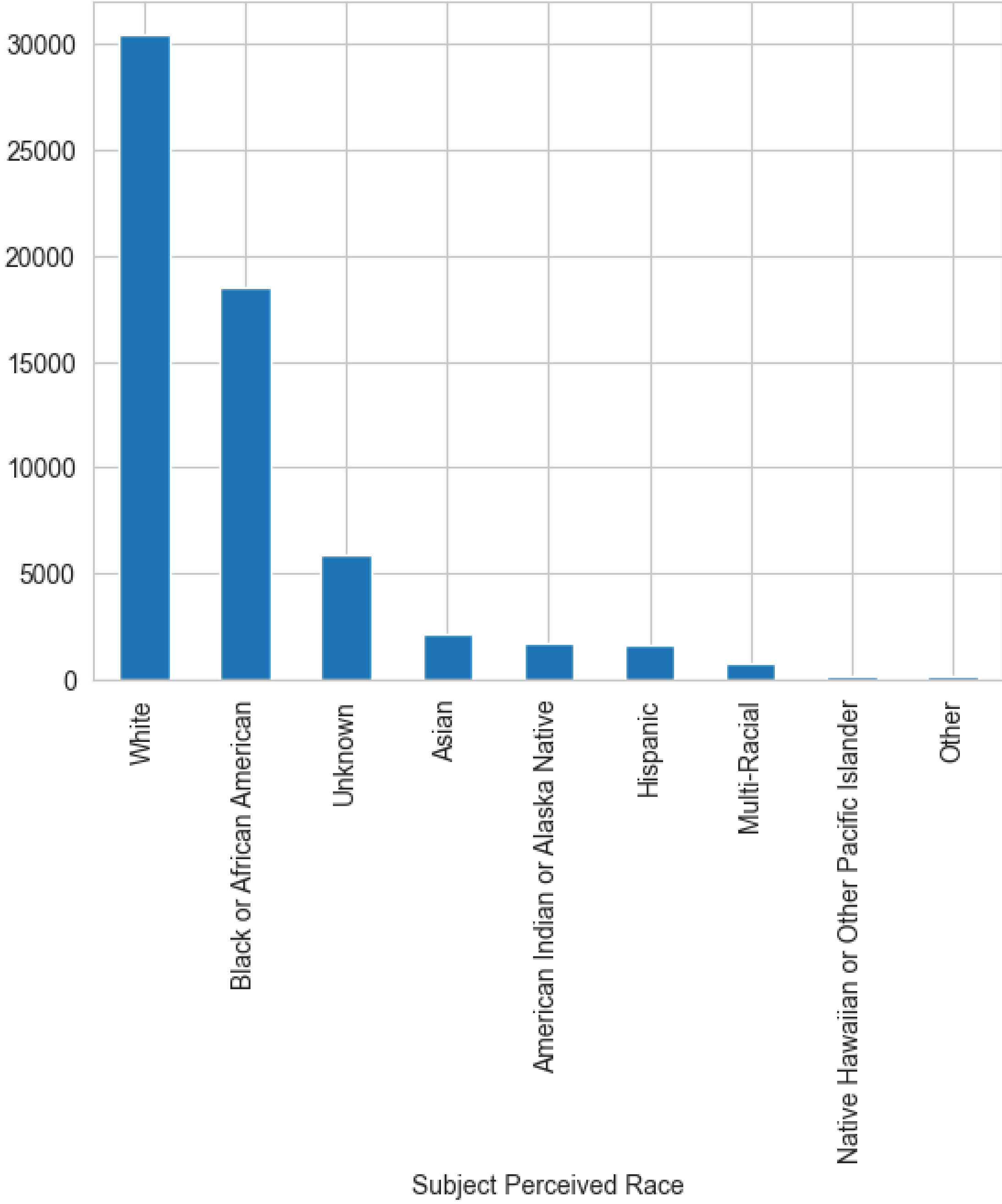
```
sns.countplot(x='Subject Age Group', data=df)
plt.xticks(rotation=45)
```

Out[109…
```
([0, 1, 2, 3, 4, 5],
 [Text(0, 0, '46 - 55'),
  Text(1, 0, '36 - 45'),
  Text(2, 0, '26 - 35'),
  Text(3, 0, '18 - 25'),
  Text(4, 0, '56 and Above'),
  Text(5, 0, '1 - 17')])
```



In [110…
```
replace_func(df['Subject Perceived Race'])
# Getting a visual of the subject race
df['Subject Perceived Race'].value_counts().plot(kind='bar')
```

Out[110…
```
<Axes: xlabel='Subject Perceived Race'>
```

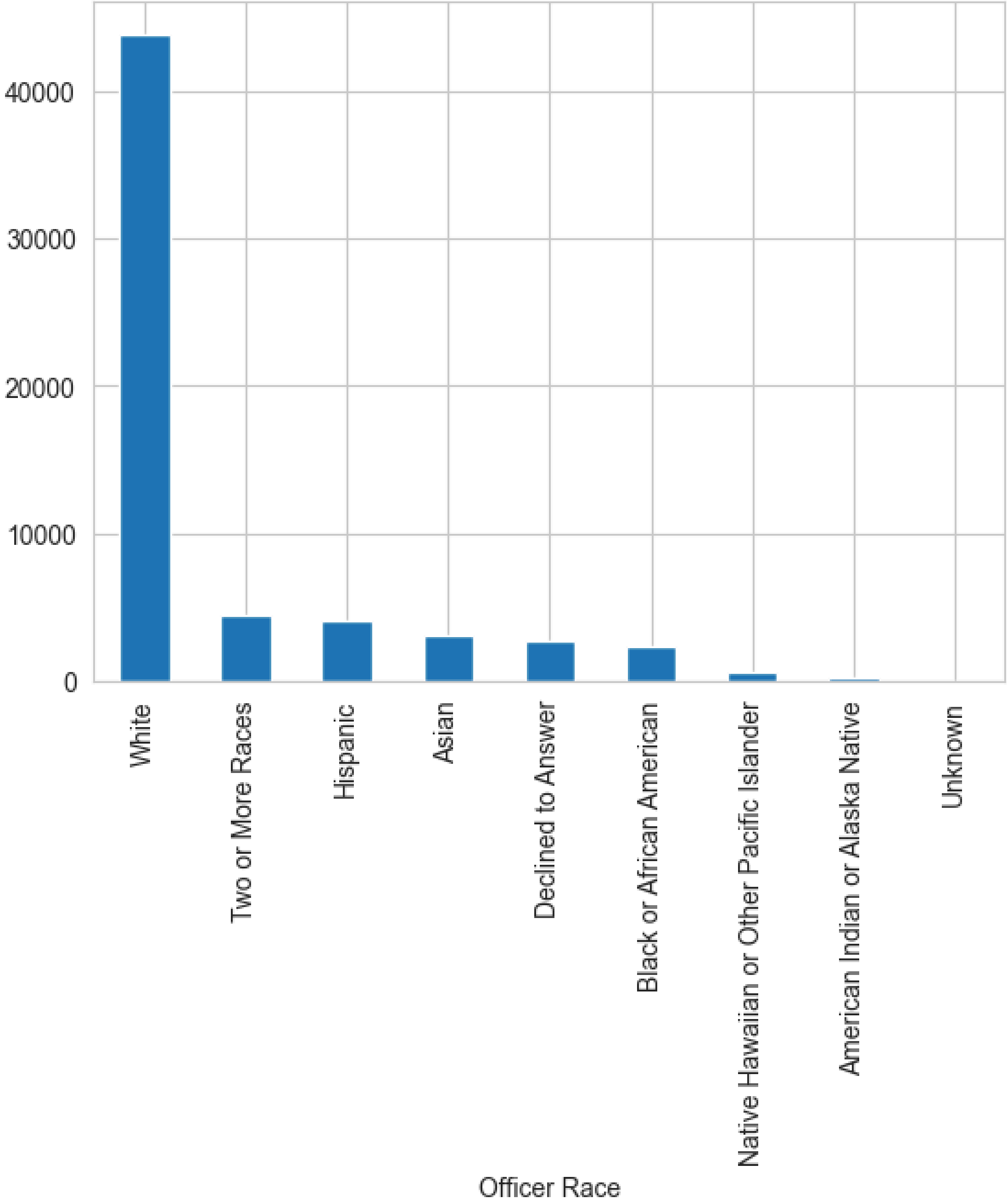


In [111…

```
df['Officer Race'].value_counts().plot(kind='bar')
```

Out[111…

```
<Axes: xlabel='Officer Race'>
```



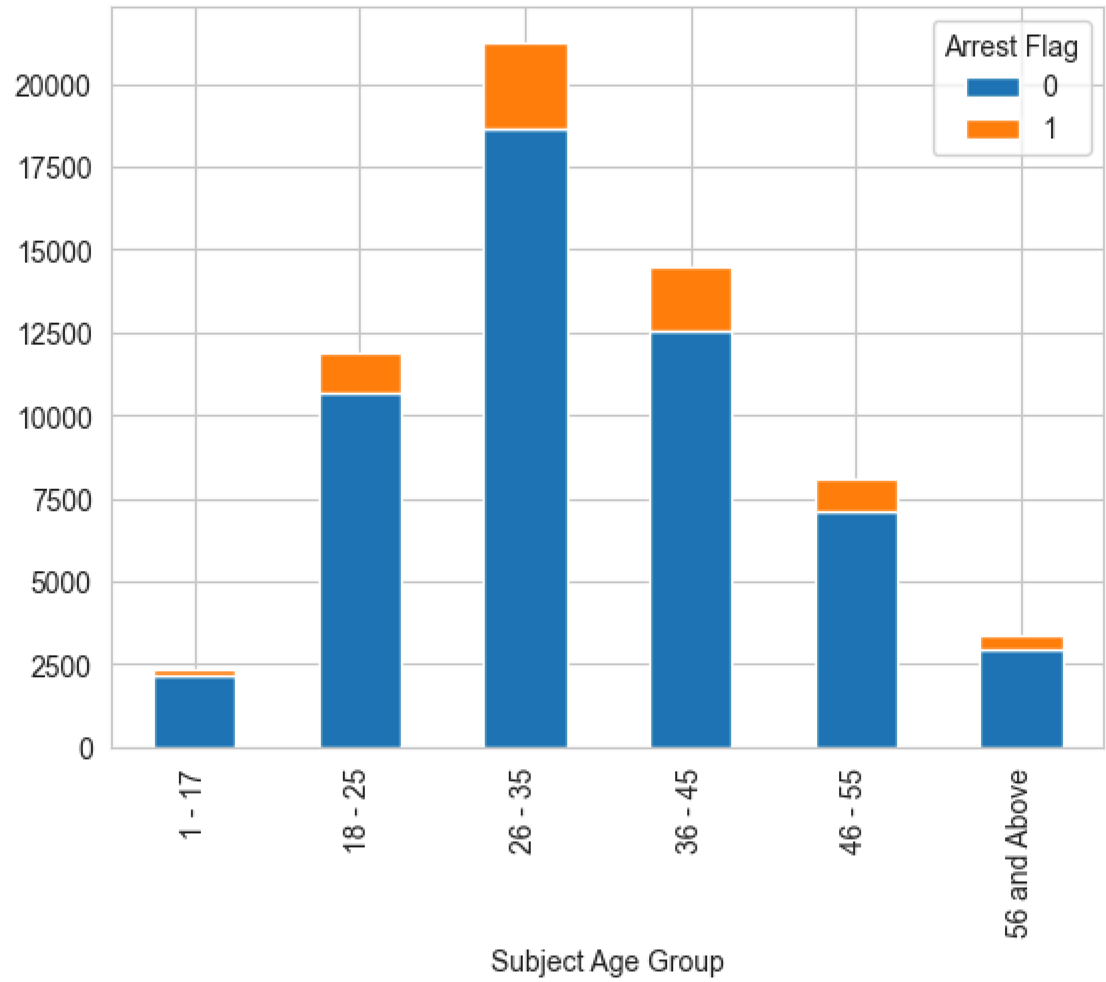
In [112...
```
df.groupby(['Subject Age Group', 'Arrest Flag']).size().unstack()
```

Out[112...

Arrest Flag	0	1
Subject Age Group		
1 - 17	2186	160
18 - 25	10670	1188
26 - 35	18638	2626
36 - 45	12548	1954
46 - 55	7132	926
56 and Above	2968	405

In [113...
```
df.groupby(['Subject Age Group', 'Arrest Flag']).size().unstack().plot(kind='bar', s
```

Out[113...
```
<Axes: xlabel='Subject Age Group'>
```



In [114...

```
# Getting a table to show subject's gender and the stop resolution
df.groupby('Subject Perceived Gender')['Stop Resolution'].value_counts().unstack(fil
```

Out[114...

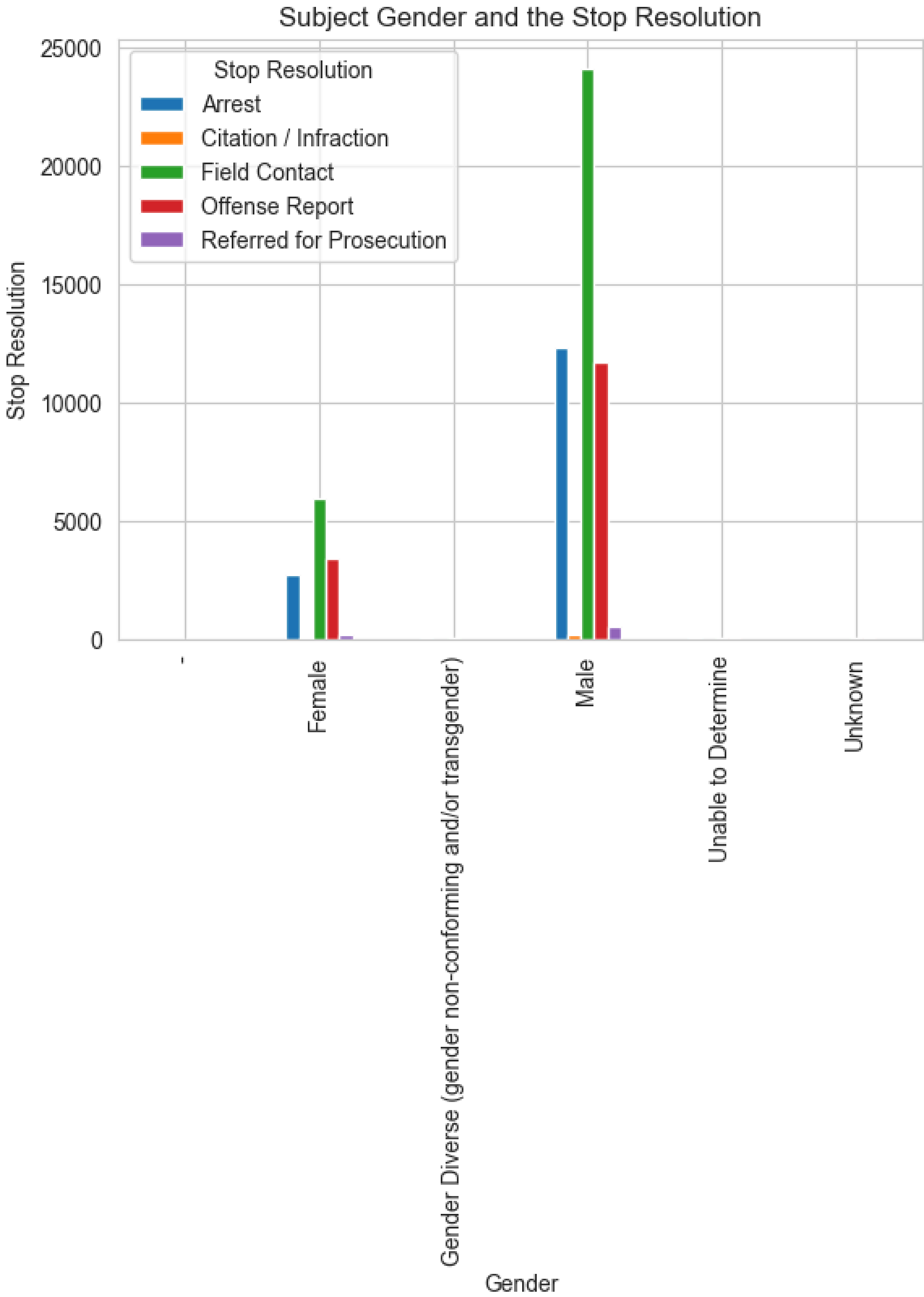
Stop Resolution / Subject Perceived Gender	Arrest	Citation / Infraction	Field Contact	Offense Report	Referred for Prosecution
-	4	0	6	4	0
Female	2746	41	5954	3416	175
Gender Diverse (gender non-conforming and/or transgender)	15	0	41	3	0
Male	12336	175	24120	11671	524
Unable to Determine	14	1	53	37	0
Unknown	6	0	49	10	0

In [115...

```
# Creating a bar grapg to show subject's gender and the stop resolution
df.groupby('Subject Perceived Gender')['Stop Resolution'].value_counts().unstack(fil
```

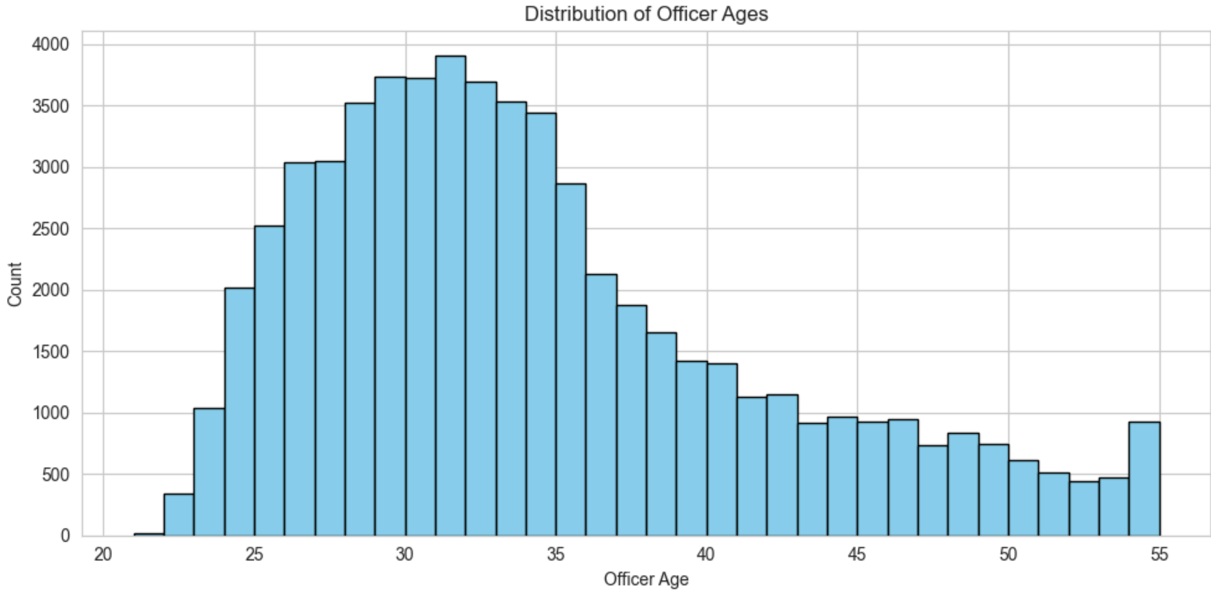
Out[115...

```
<Axes: title={'center': 'Subject Gender and the Stop Resolution'}, xlabel='Gender', y
label='Stop Resolution'>
```



In [116…

```
# Recalculate numeric Officer Age for plotting
officer_age_numeric = df["stop_year"] - df["Officer YOB"]
plt.figure(figsize=(10, 5))
plt.hist(officer_age_numeric, bins=range(officer_age_numeric.min(), int(upper_fence)
plt.title("Distribution of Officer Ages")
plt.xlabel("Officer Age")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```



In [ ]:

In [ ]:

In [117…

```
# Chossing the relevant columns to use in achieving my objective
relevant_columns = [
    'Subject Age Group', 'Stop Resolution', 'Weapon Type', 'Officer Age', 'Officer G
    'Call Type', 'Arrest Flag', 'Frisk Flag', 'Precinct'
]
```

In [118…

```
# Check for placeholders in the values of each column
for col in relevant_columns:
    print(f'{col}: {df[col].unique()}')
```

```
Subject Age Group: ['46 - 55' '36 - 45' '26 - 35' '18 - 25' '56 and Above' '1 - 17']
Stop Resolution: ['Field Contact' 'Offense Report' 'Arrest' 'Citation / Infraction'
 'Referred for Prosecution']
Weapon Type: ['Unknown' 'Knife/Cutting' 'Blunt Object' 'Firearm' 'Chemical' 'None'
 'Other']
Officer Age: [ True False]
Officer Gender: ['Female' 'Male']
Officer Race: ['Hispanic' 'White' 'Asian' 'Declined to Answer'
 'Black or African American' 'Two or More Races'
 'Native Hawaiian or Other Pacific Islander'
 'American Indian or Alaska Native' 'Unknown']
Subject Perceived Race: ['White' 'Black or African American'
 'Native Hawaiian or Other Pacific Islander'
 'American Indian or Alaska Native' 'Unknown' 'Hispanic' 'Multi-Racial'
 'Other' 'Asian']
Subject Perceived Gender: ['Male' 'Female' 'Unable to Determine'
 'Gender Diverse (gender non-conforming and/or transgender)' 'Unknown' '-']
Call Type: ['Unknown' 'ONVIEW' '911' 'TELEPHONE OTHER, NOT 911'
 'ALARM CALL (NOT POLICE ALARM)' 'SCHEDULED EVENT (RECURRING)'
 'TEXT MESSAGE' 'HISTORY CALL (RETRO)']
Arrest Flag: [0 1]
Frisk Flag: ['N' 'Y']
Precinct: ['West' 'North' '-' 'South' 'East' 'Southwest' 'OOJ' 'Unknown' 'FK ERROR']
```

In [119…

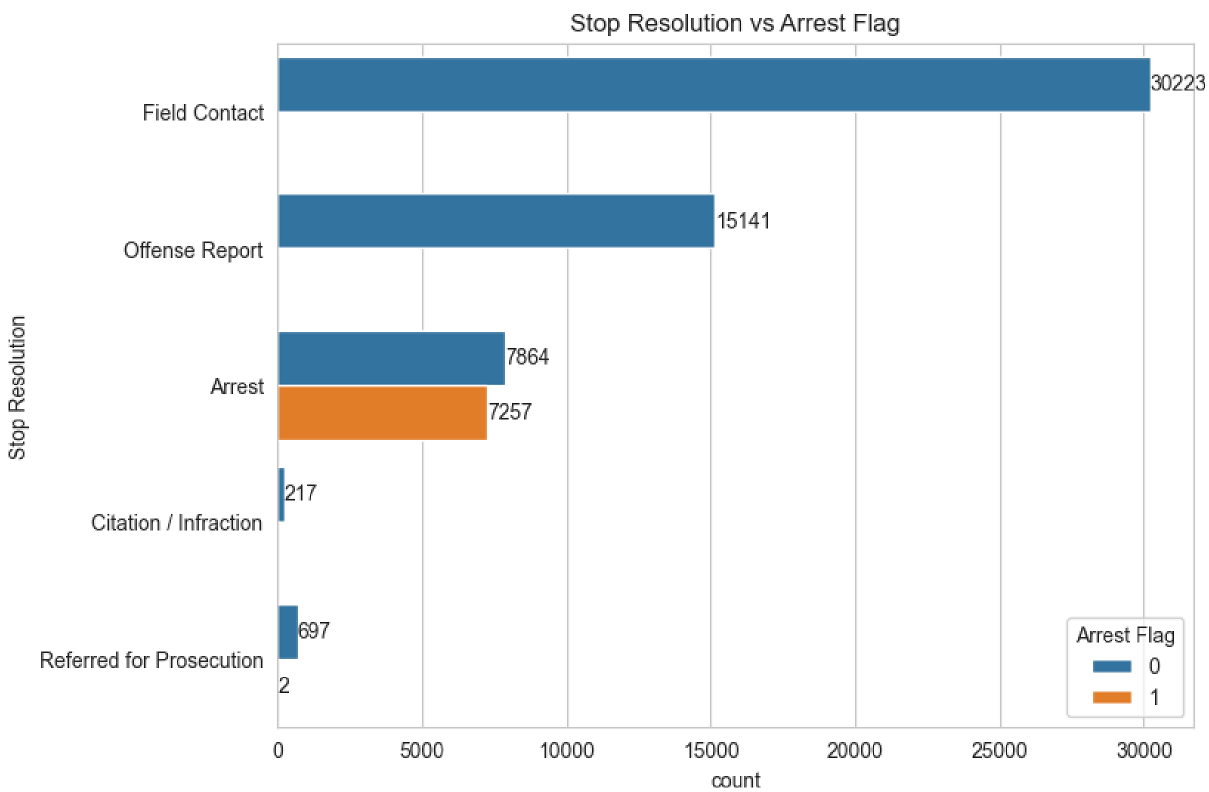
```
# Compiling the relevant columns
df_2 = df.loc[:,relevant_columns]
df_2.head() # View top 5 entries
```

Out[119…

	Subject Age Group	Stop Resolution	Weapon Type	Officer Age	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Call Type	Arrest Flag
0	46 - 55	Field Contact	Unknown	True	Female	Hispanic	White	Male	Unknown	0
1	36 - 45	Field Contact	Unknown	True	Male	White	Black or African American	Male	ONVIEW	0
2	26 - 35	Offense Report	Unknown	False	Male	Asian	White	Male	911	0
3	18 - 25	Offense Report	Unknown	True	Female	White	Black or African American	Male	911	0
4	18 - 25	Field Contact	Unknown	True	Female	Declined to Answer	Black or African American	Female	Unknown	0

In [120…

```
figure, ax = plt.subplots(figsize = (8, 6))
# Plot the countplot
sns.countplot(data = df_2, y = 'Stop Resolution', hue = df['Arrest Flag'].astype(str
ax.set_title('Stop Resolution vs Arrest Flag') # Set title
# Add labels to each bar
for container in ax.containers:
    ax.bar_label(container, label_type='edge');
```



In [121…

```
# Import the necessary libraries
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.metrics import accuracy_score, f1_score, classification_report, roc_cur
from sklearn.linear_model import LogisticRegression
```

In [122...

```
# Define the variables
X = df_2.drop('Arrest Flag', axis = 1) # Independent/features
y = df_2['Arrest Flag'] # Dependent/target
```

In [123...

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

In [124...

```
# Change the categories using one-hot encoder
ohe = OneHotEncoder(drop = 'first', sparse_output = False, handle_unknown = 'ignore'

X_train_categorical = X_train.select_dtypes('object').copy() # Defining the categori
X_train_categorical
X_test_categorical = X_test.select_dtypes('object').copy() # Defining the categorica
X_test_categorical
```

Out[124...

	Subject Age Group	Stop Resolution	Weapon Type	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Call Type	Frisk Flag
28641	26 - 35	Field Contact	Unknown	Male	Hispanic	Black or African American	Male	Unknown	N
34913	36 - 45	Arrest	Knife/Cutting	Male	Hispanic	White	Male	911	Y
9785	46 - 55	Field Contact	Unknown	Male	White	Black or African American	Male	911	Y
53339	36 - 45	Arrest	Unknown	Male	White	Black or African American	Male	TELEPHONE OTHER, NOT 911	Y
57504	18 - 25	Offense Report	Unknown	Male	White	White	Male	911	N
...	...	...	...	...	...	...	...	...	...
47784	36 - 45	Offense Report	Unknown	Male	White	American Indian or Alaska Native	Male	911	Y
54214	18 - 25	Arrest	Unknown	Male	White	Black or African American	Male	911	N
50781	46 - 55	Field Contact	Unknown	Male	White	White	Male	911	N
39331	36 - 45	Field Contact	Unknown	Male	White	Unknown	Male	ONVIEW	N
20083	18 - 25	Arrest	Unknown	Male	Two or More Races	White	Female	TELEPHONE OTHER, NOT 911	N

12281 rows × 10 columns

In [125...

```
ohe.fit(X_train_categorical) # Fit the data to the onehotencoder
X_train_ohe = pd.DataFrame( # Change it to a dataframe
    ohe.transform(X_train_categorical),
    index = X_train_categorical.index,
```

```
    columns=ohe.get_feature_names_out(X_train_categorical.columns) # Get column name
)
X_train_ohe
```

Out[125…

	Subject Age Group_18 - 25	Subject Age Group_26 - 35	Subject Age Group_36 - 45	Subject Age Group_46 - 55	Subject Age Group_56 and Above	Stop Resolution_Citation / Infraction	Stop Resolution_Field Contact	F
29827	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
59938	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
29555	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
51597	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
30547	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
54343	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
38158	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
860	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
15795	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
56422	0.0	1.0	0.0	0.0	0.0	0.0	1.0	

49120 rows × 52 columns

In [126…

```
# Transform the test data using the encoder
X_test_ohe = pd.DataFrame(
    ohe.transform(X_test_categorical),
    index=X_test_categorical.index,
    columns=ohe.get_feature_names_out(X_test_categorical.columns)
)
X_test_ohe
```

```
c:\Users\Jeremy\anaconda3\envs\env\Lib\site-packages\sklearn\preprocessing\_encoders.
py:246: UserWarning: Found unknown categories in columns [7] during transform. These
unknown categories will be encoded as all zeros
  warnings.warn(
```

Out[126…

	Subject Age Group_18 - 25	Subject Age Group_26 - 35	Subject Age Group_36 - 45	Subject Age Group_46 - 55	Subject Age Group_56 and Above	Stop Resolution_Citation / Infraction	Stop Resolution_Field Contact	F
28641	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
34913	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
9785	0.0	0.0	0.0	1.0	0.0	0.0	1.0	
53339	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
57504	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
47784	0.0	0.0	1.0	0.0	0.0	0.0	0.0	

	Subject Age Group_18 - 25	Subject Age Group_26 - 35	Subject Age Group_36 - 45	Subject Age Group_46 - 55	Subject Age Group_56 and Above	Stop Resolution_Citation / Infraction	Stop Resolution_Field Contact	F
54214	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
50781	0.0	0.0	0.0	1.0	0.0	0.0	1.0	
39331	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
20083	1.0	0.0	0.0	0.0	0.0	0.0	0.0	

12281 rows × 52 columns

In [127…

```
# Scale the officer's age using minmaxscaler
scaler = MinMaxScaler()
numeric_features = ['Officer Age']
X_train_numeric = X_train[numeric_features].copy() # Defining the numeric values in
X_train_numeric
X_test_numeric = X_test[numeric_features].copy() # Defining the numeric values in th
X_test_numeric
```

Out[127…

	Officer Age
28641	True
34913	True
9785	True
53339	True
57504	True
...	...
47784	True
54214	True
50781	True
39331	True
20083	True

12281 rows × 1 columns

In [128…

```
# Fit the scaler
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train_numeric),
                    index = X_train_numeric.index,
                    columns = X_train_numeric.columns)
X_train_scaled
```

Out[128…

	Officer Age
29827	1.0
59938	1.0
29555	1.0
51597	1.0

	Officer Age
30547	1.0
...	...
54343	1.0
38158	1.0
860	1.0
15795	1.0
56422	1.0

49120 rows × 1 columns

In [129…

```
X_test_scaled = pd.DataFrame(scaler.transform(X_test_numeric),
                            index = X_test_numeric.index,
                            columns = X_test_numeric.columns)
X_test_scaled
```

Out[129…

	Officer Age
28641	1.0
34913	1.0
9785	1.0
53339	1.0
57504	1.0
...	...
47784	1.0
54214	1.0
50781	1.0
39331	1.0
20083	1.0

12281 rows × 1 columns

In [130…

```
# Add back all the transformed numerical X features
X_train_full = pd.concat([X_train_ohe.reset_index(drop = True), X_train_scaled.reset
X_train_full
```

Out[130…

	Subject Age Group_18 - 25	Subject Age Group_26 - 35	Subject Age Group_36 - 45	Subject Age Group_46 - 55	Subject Age Group_56 and Above	Stop Resolution_Citation / Infraction	Stop Resolution_Field Contact	F
0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
1	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
2	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	

	Subject Age Group_18 - 25	Subject Age Group_26 - 35	Subject Age Group_36 - 45	Subject Age Group_46 - 55	Subject Age Group_56 and Above	Stop Resolution_Citation / Infraction	Stop Resolution_Field Contact
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
49115	0.0	0.0	1.0	0.0	0.0	0.0	0.0
49116	0.0	0.0	0.0	0.0	1.0	0.0	0.0
49117	1.0	0.0	0.0	0.0	0.0	0.0	0.0
49118	0.0	0.0	1.0	0.0	0.0	0.0	1.0
49119	0.0	1.0	0.0	0.0	0.0	0.0	1.0

49120 rows × 53 columns

In [131…

```
# Add back all the transformed numerical X features
X_test_full = pd.concat([X_test_ohe.reset_index(drop = True), X_test_scaled.reset_in
X_test_full
```

Out[131…

	Subject Age Group_18 - 25	Subject Age Group_26 - 35	Subject Age Group_36 - 45	Subject Age Group_46 - 55	Subject Age Group_56 and Above	Stop Resolution_Citation / Infraction	Stop Resolution_Field Contact
0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0	0.0	1.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
12276	0.0	0.0	1.0	0.0	0.0	0.0	0.0
12277	1.0	0.0	0.0	0.0	0.0	0.0	0.0
12278	0.0	0.0	0.0	1.0	0.0	0.0	1.0
12279	0.0	0.0	1.0	0.0	0.0	0.0	1.0
12280	1.0	0.0	0.0	0.0	0.0	0.0	0.0

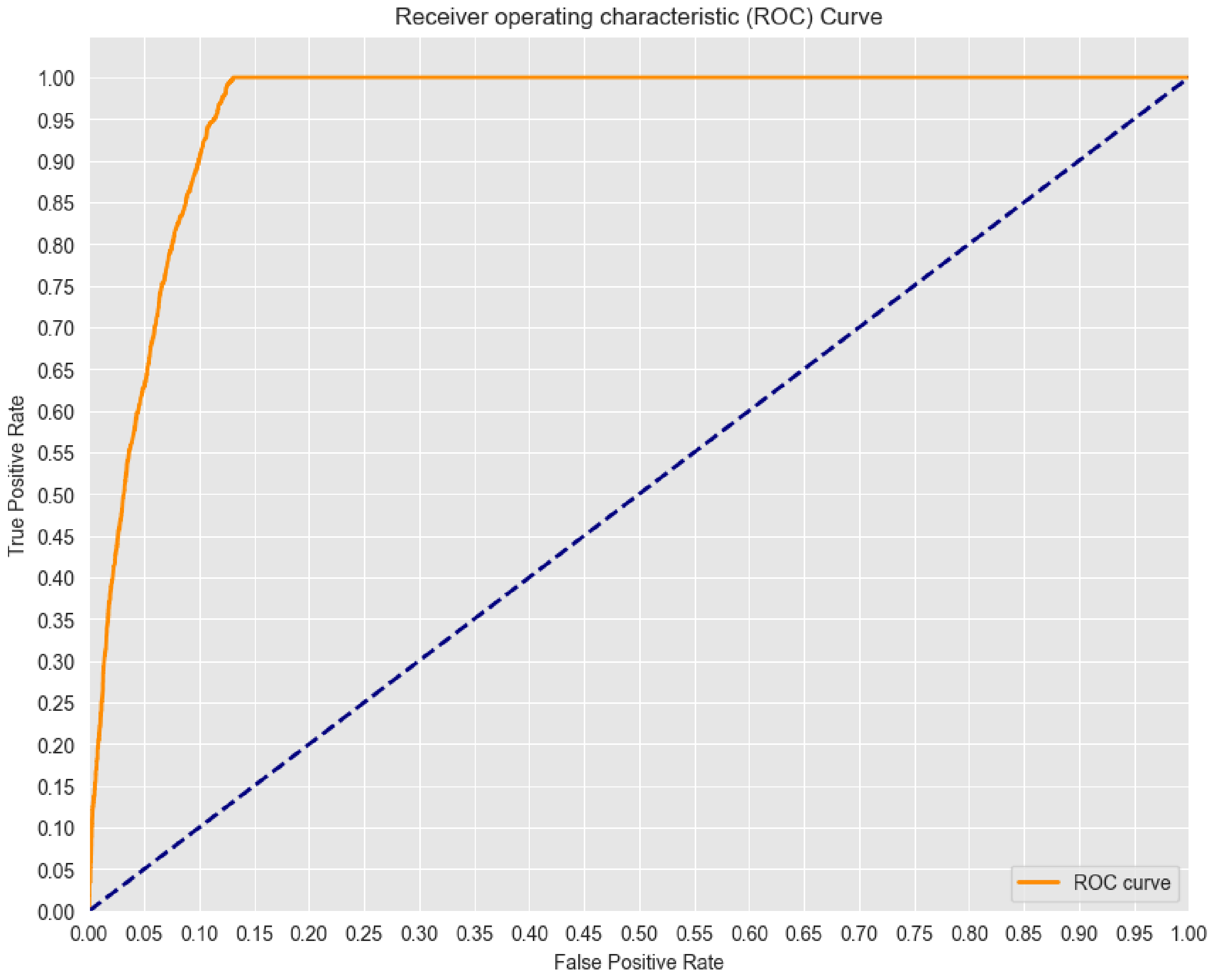
12281 rows × 53 columns

4 & 5. Modelling & Evaluation

In [132…

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state = 42) # Using SMOTE to solve class imbalance
X1_train, y1_train = smote.fit_resample(X_train_full, y_train)
```

In [133…
```
y1_train.value_counts() # To check if the classes have balanced data
```
Out[133…
```
Arrest Flag
0    43367
1    43367
Name: count, dtype: int64
```
In [134…
```
# Fit the training data to the model
# Saga works well for large datasets, the minor class is used 3x
logreg = LogisticRegression(fit_intercept = False, solver = 'saga', C = 1.0, class_w
model = logreg.fit(X1_train, y1_train)
model
```
Out[134…
```
▾ LogisticRegression
▶ Parameters
```
In [135…
```
# Evaluating the model using precision, accuracy, recall and f1 score metrics
y_pred_lr = model.predict(X_test_full)
print(classification_report(y_test, y_pred_lr))
```
```
              precision    recall  f1-score   support

           0       1.00      0.87      0.93     10775
           1       0.52      1.00      0.68      1506

    accuracy                           0.88     12281
   macro avg       0.76      0.93      0.80     12281
weighted avg       0.94      0.88      0.90     12281
```
In [136…
```
y_score = logreg.fit(X_train_full, y_train).decision_function(X_test_full)

fpr, tpr, thresholds = roc_curve(y_test, y_score)
```
In [137…
```
print('AUC: {}'.format(auc(fpr, tpr))) # AUC score of the model
```
```
AUC: 0.9581891767808889
```
In [138…
```
# Seaborn's beautiful styling
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

print('AUC: {}'.format(auc(fpr, tpr)))
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```
```
AUC: 0.9581891767808889
```



In [139…

```
# Import the Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
tree_no_tune= DecisionTreeClassifier()# the minor class is used 3x

tree_no_tune.fit(X1_train, y1_train)# Fit the training data to the model
```

Out[139…

```
▾ DecisionTreeClassifier

▶ Parameters
```

In [140…

```
y_pred_dr = tree_no_tune.predict(X_test_full) # Form predictions using the test set

print(accuracy_score(y_test, y_pred_dr)) # Accuracy of the model
print(classification_report(y_test, y_pred_dr)) # A classification report that has a
```

```
0.8940640013028255
              precision    recall  f1-score   support

           0       0.96      0.91      0.94     10775
           1       0.55      0.76      0.64      1506

    accuracy                           0.89     12281
   macro avg       0.76      0.84      0.79     12281
weighted avg       0.91      0.89      0.90     12281
```

In [141…

```
# Import the Decision Tree tuning using entropy
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
tree = DecisionTreeClassifier(criterion='entropy', class_weight = {0:1, 1:3})# the m
```

```
tree.fit(X1_train, y1_train)# Fit the training data to the model
```

Out[141…

```
▾ DecisionTreeClassifier

▶ Parameters
```

In [142…

```
y_pred_dr = tree.predict(X_test_full) # Form predictions using the test set
```

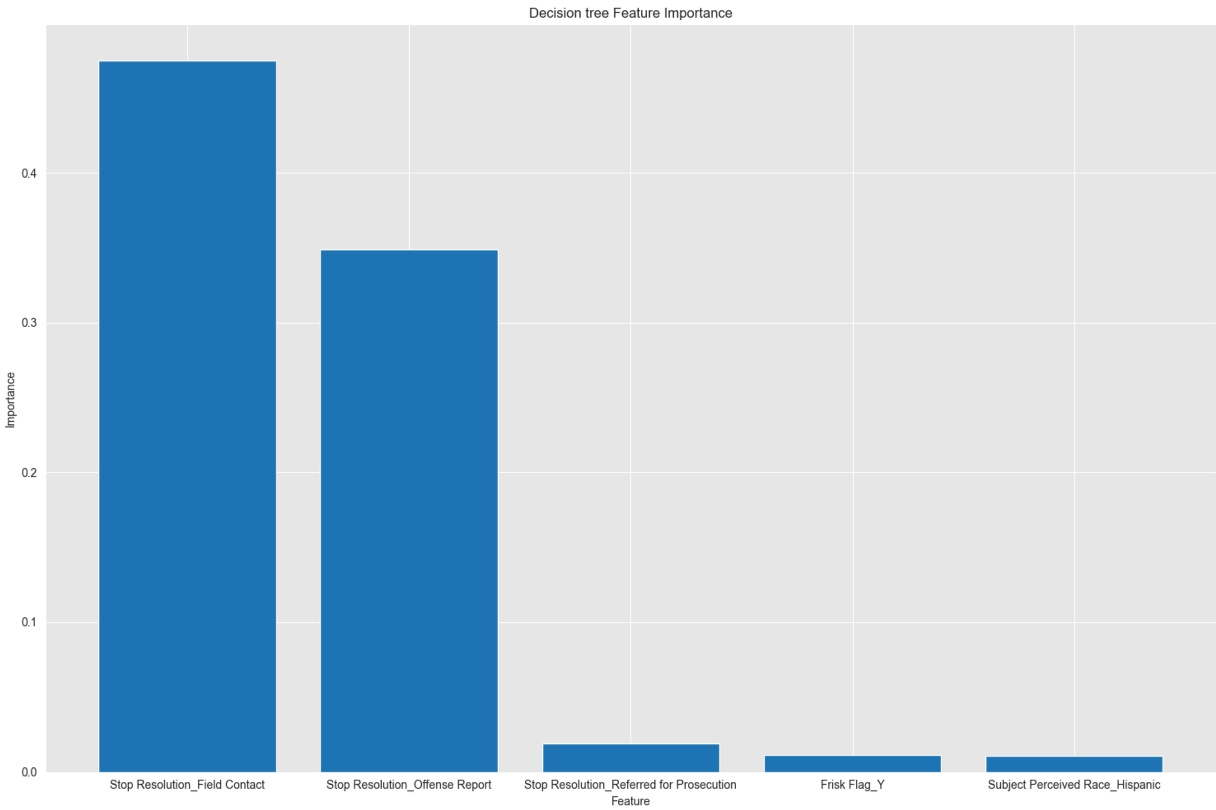
In [143…

```
print(accuracy_score(y_test, y_pred_dr)) # Accuracy of the model
print(classification_report(y_test, y_pred_dr)) # A classification report that has a
```

```
0.8945525608663789
              precision    recall  f1-score   support

           0       0.98      0.90      0.94     10775
           1       0.55      0.84      0.66      1506

    accuracy                           0.89     12281
   macro avg       0.76      0.87      0.80     12281
weighted avg       0.92      0.89      0.90     12281
```

In [144…

```
# Feature Importance from Random Forest
feature_importance = pd.DataFrame({
    'feature': X_train_full.columns,
    'importance': tree.feature_importances_
}).sort_values('importance', ascending=False).nlargest(5,'importance' )

plt.figure(figsize=(15, 10))
plt.bar(feature_importance['feature'], feature_importance['importance'])
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Decision tree Feature Importance')
plt.tight_layout()
plt.show()
```



In [145…

```
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rand = RandomForestClassifier(random_state = 42, class_weight = 'balanced', n_estima
rand.fit(X1_train, y1_train)
```

Out[145…

```
▾ RandomForestClassifier

▶ Parameters
```

In [146…

```
y_pred_rf = rand.predict(X_test_full) # Form the predictions using test set
```

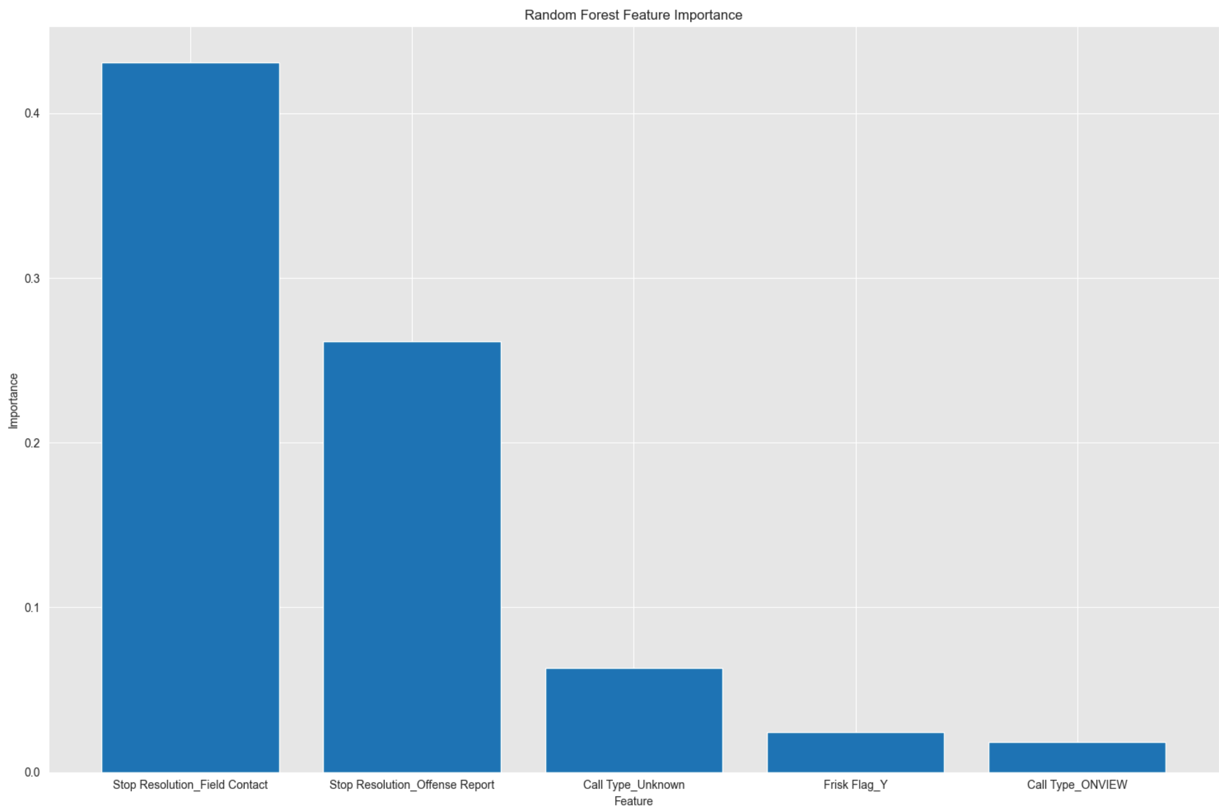
In [147…

```
print(classification_report(y_test, y_pred_rf)) # See the classification report whic
```

```
              precision    recall  f1-score   support

           0       0.97      0.91      0.94     10775
           1       0.56      0.82      0.66      1506

    accuracy                           0.90     12281
   macro avg       0.77      0.86      0.80     12281
weighted avg       0.92      0.90      0.91     12281
```

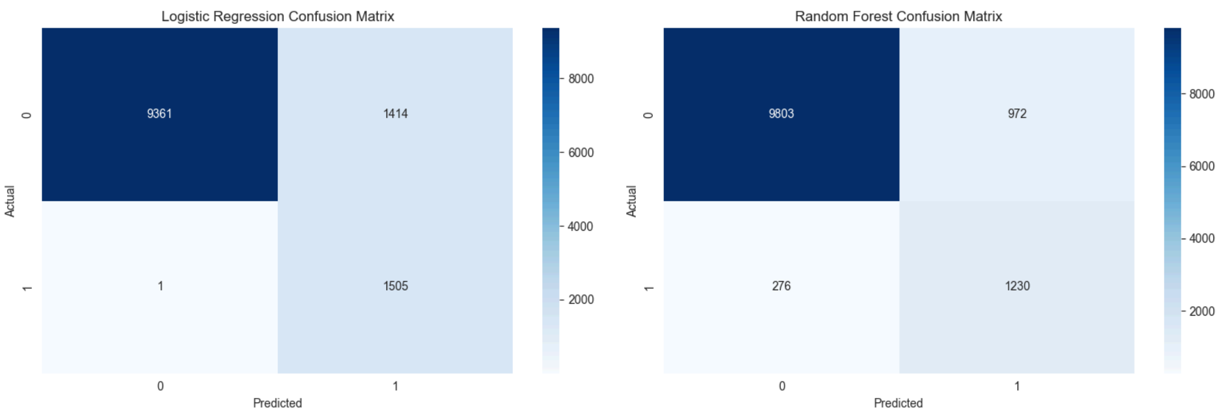
In [148…

```
# Feature Importance from Random Forest
feature_importance = pd.DataFrame({
    'feature': X_train_full.columns,
    'importance': rand.feature_importances_
}).sort_values('importance', ascending=False).nlargest(5,'importance' )

plt.figure(figsize=(15, 10))
plt.bar(feature_importance['feature'], feature_importance['importance'])
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()
```



In [149…

```python
# 10. Confusion Matrix Visualization
from sklearn.metrics import confusion_matrix


fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# Logistic Regression Confusion Matrix
cm_lr = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues', ax=ax1)
ax1.set_title('Logistic Regression Confusion Matrix')
ax1.set_xlabel('Predicted')
ax1.set_ylabel('Actual')

# Random Forest Confusion Matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', ax=ax2)
ax2.set_title('Random Forest Confusion Matrix')
ax2.set_xlabel('Predicted')
ax2.set_ylabel('Actual')

plt.tight_layout()
plt.show()
```



8. Analysis of Key Findings

In [150…

```python
# Weapon Type analysis
weapon_effect = df_2.groupby('Weapon Type')['Arrest Flag'].agg(['mean', 'count']).so
print("\nWeapon Types with Highest Arrest Rates:")
print(weapon_effect.head(10))
```

```
Weapon Types with Highest Arrest Rates:
                   mean  count
Weapon Type
Other          0.500000     34
None           0.350000     20
Blunt Object   0.347826    253
Chemical       0.333333     63
Firearm        0.268358    749
Knife/Cutting  0.211184   2879
Unknown        0.110047  57403
```

In [151…

```python
# Frisk analysis
frisk_effect = df_2.groupby('Frisk Flag')['Arrest Flag'].agg(['mean', 'count'])
print("\nArrest Rates by Frisk Status:")
print(frisk_effect)
```

```
Arrest Rates by Frisk Status:
                mean  count
Frisk Flag
```

```
N           0.098081  46380
Y           0.180414  15021
```

In [152…

```
# Demographic analysis
race_effect = df_model.groupby('Subject Perceived Race')['Arrest Flag'].agg(['mean',
print("\nArrest Rates by Race:")
print(race_effect)
```

```
Arrest Rates by Race:
                                               mean  count
Subject Perceived Race
Native Hawaiian or Other Pacific Islander  0.265537    177
Asian                                      0.141259   2145
Unknown                                    0.139383   4441
Black or African American                  0.136759  18529
American Indian or Alaska Native           0.115727   1685
White                                      0.113768  30448
-                                          0.068551   1415
Hispanic                                   0.000000   1634
Multi-Racial                               0.000000    781
Other                                      0.000000    146
```

9. Busines Recommendations

1. **Weapon Presence is Key**: The type of weapon involved is the strongest predictor of arrest outcomes. Officers should receive continued training on proper assessment and response to different weapon types.

2. **Frisk Procedures**: The data shows that frisks are associated with different arrest rates. Review frisk procedures to ensure they are conducted appropriately and consistently.

3. **Demographic Disparities**: Analyze any demographic patterns in arrest rates to ensure fair and equitable policing practices.

4. **Call Type Patterns**: Certain call types lead to higher arrest rates. Use this information for better resource allocation and officer preparedness.

5. **Precinct-level Analysis**: Investigate why arrest rates vary by precinct to identify best practices and ensure consistency across districts.

6. **Ongoing Monitoring**: Implement regular review of these patterns to identify changes over time and address any emerging issues.

In [ ]:

In [ ]:

In [ ]:

In [ ]: