1 Business Understanding

1.1 Overview

Terry Stops are a critical part of day-to-day policing, allowing officers to briefly detain individuals based on reasonable suspicion. While intended as a proactive safety measure, these stops have also raised questions about effectiveness, proportionality, and community trust.

For law enforcement agencies, examining historical data on Terry Stops can shed light on how and why stops occur, what outcomes they produce, and whether certain groups or areas are more affected than others. This type of analysis supports the larger goal of creating policing strategies that are both effective and equitable.

1.2 Background

Modern policing operates in a complex environment where community safety, officer decision-making, and public accountability intersect. Agencies must balance several challenges, including:

- Allocating officers efficiently across districts and shifts
- Ensuring consistency and fairness in stop decisions
- Training officers to reduce bias and improve judgment under uncertainty
- Evaluating whether stops lead to meaningful public safety outcomes
- Maintaining trust and transparency with the public

By digging into Terry Stop data, agencies can better understand patterns in stops and outcomes, such as frisks, searches, and arrests, as well as identify areas where practice may need to evolve.

1.3 Objectives

This project aims to:

1. Detect the main drivers behind different Terry Stop outcomes (e.g., whether a frisk, search, or arrest occurs)
2. Assess whether demographic or situational disparities are present
3. Provide evidence-based recommendations for officer deployment, training, and policy adjustments
4. Develop clear visualizations and models that support decision-makers and promote transparency

1.4 Problem Statement

Police leadership wants to address key questions such as:

- Which factors most strongly predict whether a stop escalates into a frisk, search, or arrest?
- Do outcomes vary significantly across race, gender, or age groups?
- How do officer experience, squad assignment, or unit placement influence stop patterns?
- Are there hotspots in terms of location or time of day where stops are concentrated?
- What adjustments could make stops both more effective and more equitable?

The central challenge is to use past Terry Stop records to uncover reliable insights that can guide operational improvements and policy decisions.

1.5 Success Metrics

We will consider the project successful if we are able to:

- Isolate the most influential features connected to stop outcomes
- Develop predictive models that achieve an F1-score of at least 0.70, while maintaining overall accuracy above 80%
- Translate findings into specific, actionable steps for training, deployment, and oversight
- Document the process and share results in accessible formats for multiple audiences

1.6 Tools and Methods

Our analysis will rely on a range of technologies and practices, including:

- Python which includes, pandas, NumPy, Scikit-learn, Seaborn, Matplotlib, for data cleaning, analysis, and modeling
- Jupyter Notebooks for exploratory work and reproducibility
- GitHub for version control and team collaboration
- Google Workspace i.e. Docs or Slides, for reporting and presentations

1.7 Stakeholders

The primary audience for this analysis is the police department's command staff, who are responsible for operational decisions and officer guidance.

Secondary stakeholders include:

- City oversight boards and auditors
- Community advocacy groups and civil rights organizations
- Local policymakers who shape police accountability standards
- Residents who seek clarity and fairness in public safety practices

2 Data Understanding

To uncover the factors that influence the outcomes of Terry Stops, we must begin with a clear and comprehensive understanding of the data available to us. This section examines the structure, scope, and reliability of the datasets we are working with.

Our data includes publicly available Terry Stop records with information on:

- **Stop details**: date, time, location, reason for the stop, and resolution
- **Officer information**: age, squad, years of service, and assignment
- **Subject information**: demographic details such as race, gender, and age
- **Contextual details**: call type, presence of weapons, and associated case numbers

The dataset was obtained from:

- Terry Traffic Stops (https://data.seattle.gov/Public-Safety/Terry-Stops/28ny-9ts8/about_data)

The data is provided primarily in CSV format and contains multiple attributes describing both the officers and the subjects involved in Terry Stops.

By carefully reviewing these attributes and exploring preliminary patterns, we aim to:

- Determine which variables are most relevant to our business goals and clarify their data types
- Identify missing, duplicated, or inconsistent records that may affect analysis
- Extract early insights into temporal, demographic, or situational trends that could guide later modeling

This foundational understanding of the data will inform the cleaning process, drive feature engineering decisions, and set the stage for deeper analysis and predictive modeling in the subsequent phases of the project.

Before diving into data exploration, we need to import the key Python libraries that will support our data analysis, visualization, and modeling tasks.

In [1]:
```
#import the libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import re

#import sklearn libraries
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from scipy.stats import randint, uniform
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_curve
```

2.1 Initial Data Exploration of the dataset

In this section, we begin exploring the raw Terry Stops dataset to understand its structure, completeness, and key variables. This includes checking for missing values, data types, duplicates, and overall distribution of records.

In [2]:
```
#load and read the dataset
df = pd.read_csv("Terry_Stops_20250909.csv")

#check the first five rows of the dataset
df.head()
```

Out[2]:

	Subject Age Group	Subject ID	GO / SC Num	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	...	Reportec
0	26 - 35	-1	20170000036835	234548	Offense Report	None	4852	1953	Male	Asian	...	18:36:00.0C
1	46 - 55	-1	20180000275629	481899	Field Contact	None	8544	1993	Female	Hispanic	...	13:47:00.0C
2	36 - 45	49326761681	20230000118635	49327076666	Field Contact	Knife/Cutting/Stabbing Instrument	7766	1984	Male	White	...	07:25:58.0C
3	36 - 45	53986235598	20240000029589	53986202139	Field Contact	-	8723	1994	Male	White	...	02:50:52.0C
4	18 - 25	-1	20150000002928	54115	Field Contact	None	7745	1988	Female	Declined to Answer	...	00:22:00.0C

5 rows × 23 columns

In [3]:
```
#check the last rows of the dataset
df.tail()
```

Out[3]:

	Subject Age Group	Subject ID	GO / SC Num	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	...	Reported Time	Initial Cal Typ
64732	18 - 25	-1	20170000325937	354197	Offense Report	None	7755	1971	Male	White	...	22:58:00.0000000	
64733	26 - 35	-1	20190000122094	549491	Arrest	None	7489	1985	Female	White	...	13:01:00.0000000	SUSPICIOUS STOP OFFICEF INITIATED ONVIEV
64734	36 - 45	-1	20180000467077	512455	Arrest	None	7758	1987	Male	White	...	04:35:00.0000000	OBS - BURC - IP/JO COMM BURC (INCLUDES SCHOOLS
64735	26 - 35	-1	20190000059044	532813	Offense Report	None	7649	1986	Male	White	...	14:35:00.0000000	ASLT CRITICAI (NC SHOOTINGS
64736	18 - 25	-1	20180000001424	412377	Field Contact	None	8558	1993	Male	White	...	14:10:00.0000000	

5 rows × 23 columns

In [4]:
```
#check the shape of the dataset
df.shape
```

Out[4]:
```
(64737, 23)
```

In [5]:
```
#check the data types of each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64737 entries, 0 to 64736
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Subject Age Group         64737 non-null  object
 1   Subject ID                64737 non-null  int64 
 2   GO / SC Num               64737 non-null  int64 
 3   Terry Stop ID             64737 non-null  int64 
 4   Stop Resolution           64737 non-null  object
 5   Weapon Type               64737 non-null  object
 6   Officer ID                64737 non-null  object
 7   Officer YOB               64737 non-null  int64 
 8   Officer Gender            64737 non-null  object
 9   Officer Race              64737 non-null  object
 10  Subject Perceived Race    64737 non-null  object
 11  Subject Perceived Gender  64737 non-null  object
 12  Reported Date             64737 non-null  object
 13  Reported Time             64737 non-null  object
 14  Initial Call Type         64737 non-null  object
 15  Final Call Type           64737 non-null  object
 16  Call Type                 64737 non-null  object
 17  Officer Squad             64168 non-null  object
 18  Arrest Flag               64737 non-null  object
 19  Frisk Flag                64737 non-null  object
 20  Precinct                  64737 non-null  object
 21  Sector                    64737 non-null  object
 22  Beat                      64737 non-null  object
dtypes: int64(4), object(19)
memory usage: 11.4+ MB
```

```
In [6]: #check statistical summary for numerical columns
        df.describe()
```

Out[6]:

	Subject ID	GO / SC Num	Terry Stop ID	Officer YOB
count	6.473700e+04	6.473700e+04	6.473700e+04	64737.000000
mean	8.993283e+09	2.019008e+13	1.507724e+10	1984.467090
std	1.538334e+10	8.439261e+10	2.069983e+10	9.189523
min	-8.000000e+00	-1.000000e+00	2.802000e+04	1900.000000
25%	-1.000000e+00	2.017000e+13	2.547370e+05	1980.000000
50%	-1.000000e+00	2.019000e+13	5.347130e+05	1986.000000
75%	7.803391e+09	2.021000e+13	2.667422e+10	1991.000000
max	6.581126e+10	2.025000e+13	6.581120e+10	2003.000000

```
In [7]: #check statistical summary for categorical columns
        df.describe(include="object").T
```

Subject Perceived Race	64737	11	White	31649
Subject Perceived Gender	64737	7	Male	51093
Reported Date	64737	3829	2015-10-01T00:00:00	101
Reported Time	64737	26254	02:56:00.0000000	52
Initial Call Type	64737	186	-	13602
Final Call Type	64737	203	-	13602
Call Type	64737	8	911	30545
Officer Squad	64168	277	TRAINING - FIELD TRAINING SQUAD	6998
Arrest Flag	64737	2	N	57400
Frisk Flag	64737	3	N	48527
Precinct	64737	9	West	18416
Sector	64737	20	-	10973
Beat	64737	55	-	10967

In [8]:
```
#check unique values
for coln in df:
    uni_value = df[coln].unique()
    print(f" {coln}\n, {uni_value}\n")
```

```
 Subject Age Group
, ['26 - 35' '46 - 55' '36 - 45' '18 - 25' '-' '56 and Above' '1 - 17']

 Subject ID
, [          -1 49326761681 53986235598 ... 12282720243 15947861935
  7751138304]

 GO / SC Num
, [20170000036835 20180000275629 20230000118635 ... 20180000467077
 20190000059044 20180000001424]

 Terry Stop ID
, [     234548      481899 49327076666 ...      512455      532813
      412377]

 Stop Resolution
, ['Offense Report' 'Field Contact' 'Arrest' 'Referred for Prosecution'
 'Citation / Infraction']
```

In [9]:
```
#check value counts for each column
for coln in df:
    value_count = df[coln].value_counts()
    print(f"{coln}:\n{value_count}\n")
```

```
Subject Age Group:
26 - 35         21576
36 - 45         14705
18 - 25         12081
46 - 55          8189
56 and Above     3426
1 - 17           2392
-                2368
Name: Subject Age Group, dtype: int64

Subject ID:
-1              35221
 7753260438        30
 7774286580        22
 21375848115       22
 7726918259        22
                ...  
 7727594916         1
 7731121570         1
```

In [10]:
```
#check for null values
df.isna().sum()
```

Out[10]:
```
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                569
Arrest Flag                    0
Frisk Flag                     0
Precinct                       0
Sector                         0
Beat                           0
dtype: int64
```

2.2 Observations

The dataframe comprises of **64737 rows** and **23 columns**.

The dataset is uniform from top to bottom on inspection of the first 5 rows and last 5 rows.

2.2.1 Columns

The dataset contains 23 columns with the following information:

- `Subject Age Group` : Subject Age Group (10 year increments) as reported by the officer.

- `Subject ID` : Key, generated daily, identifying unique subjects in the dataset using a character to character match of first name and last name. "Null" values indicate an "anonymous" or "unidentified" subject. Subjects of a Terry Stop are not required to present identification.
- `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.
- `Terry Stop ID` : Key identifying unique Terry Stop reports.
- `Stop Resolution` : Resolution of the stop as reported by the officer.
- `Weapon Type` : Type of weapon, if any, identified during a search or frisk of the subject. Indicates "None" if no weapons was found.
- `Officer ID` : Key identifying unique officers in the dataset.
- `Officer YOB` : Year of birth, as reported by the officer.
- `Officer Gender` : Gender of the officer, as reported by the officer.
- `Officer Race` : Race of the officer, as reported by the officer.
- `Subject Perceived Race` : Perceived race of the subject, as reported by the officer.
- `Subject Perceived Gender` : Perceived gender of the subject, as reported by the officer.
- `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.
- `Reported Time` : Time the stop was reported in the Records Management System (RMS). Not the time the stop occurred but generally within 10 hours.
- `Initial Call Type` : Initial classification of the call as assigned by 911.
- `Final Call Type` : Final classification of the call as assigned by the primary officer closing the event.
- `Call Type` : How the call was received by the communication center.
- `Officer Squad` : Functional squad assignment (not budget) of the officer as reported by the Data Analytics Platform (DAP).
- `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).
- `Frisk Flag` : Indicator of whether a "frisk" was conducted, by the officer, of the subject, during the Terry Stop.
- `Precinct` : Precinct of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.
- `Sector` : Sector of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.
- `Beat` : Beat of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.

2.2.2 Data Types

The summary information of the data frame reveals two datatypes, that is, categorical data and numerical data, and we see that out of the 23 columns, only 4 columns are numerical and the rest are categorical. The numerical columns are `Subject ID`, `GO / SC Num`, `Terry Stop ID` and `Officer YOB`.

We note that there will be need for data cleaning as follows:

- Change the `Reported Year` and `Reported Time` columns from categorical datatype(object) to datetime

2.2.3 Missing Data

According to the dataframe info, the `Officer Squad` is the only column that has missing values. But if you check the unique value counts in each column, you will notice that there are columns with the '-' value which basically tells us that these values are missing. We

3 Data Preparation

3.0.1 Overview

Following our Data Understanding phase, we now transition into the Data Preparation stage of the CRISP-DM methodology. This phase is crucial in transforming raw data into a clean and structured format that can be used effectively in analysis and modeling.

The tasks in this section include:

- Selecting relevant tables and columns.
- Handling missing or inconsistent data.
- Converting data types (e.g. dates, floats, integers).
- Creating derived attributes where necessary (e.g. year of release from a full date).
- Filtering and sampling records for exploratory and predictive tasks.

3.0.2 Goals

- Ensure the dataset is clean, consistent, and analysis-ready.
- Retain only data relevant to our business objectives.
- Prepare a consolidated and structured dataset for feature engineering and modeling in subsequent phases.

3.1 Data Cleaning

Before we start data cleaning, we are going to make a copy of our original dataset so that we can work on our copy and not lose the original data, in case we need it.

In [11]:
```
#Make a copy of the original dataset
df1 = df.copy(deep=True)
df1.head()
```

Out[11]:

	Subject Age Group	Subject ID	GO / SC Num	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	...	Repo
0	26 - 35	-1	20170000036835	234548	Offense Report	None	4852	1953	Male	Asian	...	18:36:0
1	46 - 55	-1	20180000275629	481899	Field Contact	None	8544	1993	Female	Hispanic	...	13:47:0
					Field	Knife/Cutting/Stabbing						

In [12]:
```
#Check the column names
df1.columns
```

Out[12]:
```
Index(['Subject Age Group', 'Subject ID', 'GO / SC Num', 'Terry Stop ID',
       'Stop Resolution', 'Weapon Type', 'Officer ID', 'Officer YOB',
       'Officer Gender', 'Officer Race', 'Subject Perceived Race',
       'Subject Perceived Gender', 'Reported Date', 'Reported Time',
       'Initial Call Type', 'Final Call Type', 'Call Type', 'Officer Squad',
       'Arrest Flag', 'Frisk Flag', 'Precinct', 'Sector', 'Beat'],
      dtype='object')
```

In [13]:
```
#Rename the columns to remove whitespaces and convert the column names to lowercase
df1.columns = df1.columns.str.lower().str.strip().str.replace(" ", "_")
df1.columns
```

Out[13]:
```
Index(['subject_age_group', 'subject_id', 'go_/_sc_num', 'terry_stop_id',
       'stop_resolution', 'weapon_type', 'officer_id', 'officer_yob',
       'officer_gender', 'officer_race', 'subject_perceived_race',
       'subject_perceived_gender', 'reported_date', 'reported_time',
       'initial_call_type', 'final_call_type', 'call_type', 'officer_squad',
       'arrest_flag', 'frisk_flag', 'precinct', 'sector', 'beat'],
      dtype='object')
```

3.1.1 Interpretation:

The initial column names have whitespaces in between the names and to clean that, we have standardized them by converting them to lowercase, removing extra spaces, and replacing the whitespaces with underscores, as this will help to make them cleaner and easier to work with in code.

In [14]:
```
#Strip spaces from all string columns
df1 = df1.apply(lambda col: col.str.strip() if col.dtype == "object" else col)
```

3.1.2 Interpretation:

We have stripped off trailing spaces in all the categorical columns so as to make them cleaner and to prevent any errors that may arise later on in the next stages like feature engineering.

```
In [15]:
#Replace all the '-' values in the dataset with 'Unknown'
def replace_dash_with_unknown(df1):
    """
    Replaces dash-like values ('-', '–', '—')
    across all object columns in the DataFrame.
    """

    # Replace dash-like values with 'Unknown'
    df1 = df1.replace(r"^[-–—]$", "Unknown", regex=True)

    return df1

df1 = replace_dash_with_unknown(df1)
```

3.1.3 Interpretation:

We noticed that some columns had null values in them in the form of '-', and to clean that we have replaced all those dash values with 'Unknown' cause they are all from the categorical columns. We did not use the mode in this case because most of the columns contain a lot of those missing values and replacing them with mode will enhance bias and lead to misinterpretation, which should generally be avoided.

In [16]:
```
#Merge the values in the 'weapon_type' column that are separated but are in the same category
df1["weapon_type"] = df1["weapon_type"].replace(
    {
        "Firearm": "Other Firearm",
        "Firearm Other": "Other Firearm",
        "Firearm (unk type)": "Other Firearm",
        "Lethal Cutting Instrument": "Knife/Cutting/Stabbing Instrument",
        "None/Not Applicable": "None",
        "Automatic Handgun":"Handgun",
        "Club": "Blunt Object/Striking Implement",
        "Blackjack": "Blunt Object/Striking Implement",
        "Brass Knuckles": "Blunt Object/Striking Implement",
        "Club, Blackjack, Brass Knuckles": "Blunt Object/Striking Implement"
    }
)
df1["weapon_type"].value_counts()
```

Out[16]:
```
None                                  32586
Unknown                               28035
Knife/Cutting/Stabbing Instrument      2976
Handgun                                 433
Other Firearm                           331
Blunt Object/Striking Implement         259
Mace/Pepper Spray                        64
Taser/Stun Gun                           20
Fire/Incendiary Device                   13
Rifle                                    11
Shotgun                                   6
Personal Weapons (hands, feet, etc.)      2
Poison                                    1
Name: weapon_type, dtype: int64
```

In [17]:
```
#Replace values in the 'officer_race' column
df1["officer_race"] = df1["officer_race"].replace("Declined to Answer", "Unknown")
df1["officer_race"].value_counts()
```

Out[17]:
```
White                                        46063
Two or More Races                             4679
Hispanic                                      4304
Asian                                         3320
Unknown                                       2943
Black or African American                     2582
Native Hawaiian or Other Pacific Islander      599
American Indian or Alaska Native               247
Name: officer_race, dtype: int64
```

In [18]:
```
#Replace values in the 'subject_perceived_race' column
df1["subject_perceived_race"] = df1["subject_perceived_race"].replace("MULTIPLE SUBJECTS", "Other")
df1["subject_perceived_race"].value_counts()
```

Out[18]:
```
White                                        31649
Black or African American                    19516
Unknown                                       6763
Asian                                         2231
American Indian or Alaska Native              1746
Hispanic                                      1684
Multi-Racial                                   809
Native Hawaiian or Other Pacific Islander      186
Other                                          153
Name: subject_perceived_race, dtype: int64
```

In [19]:
```
#Replace values in the 'subject_perceived_gender' column
df1["subject_perceived_gender"] = df1["subject_perceived_gender"].replace(
    {
    "Unable to Determine": "Unknown",
     "MULTIPLE SUBJECTS": "Unknown"
    }
)
df1["subject_perceived_gender"].value_counts()
```

Out[19]:
```
Male                                                    51093
Female                                                  12890
Unknown                                                   689
Gender Diverse (gender non-conforming and/or transgender)      65
Name: subject_perceived_gender, dtype: int64
```

In [20]:
```
#Replace values in the 'precinct' column
df1["precinct"] = df1["precinct"].replace("FK ERROR", "Unknown")
df1["precinct"].value_counts()
```

Out[20]:
```
West         18416
North        13464
Unknown      11044
East          8871
South         7833
Southwest     4989
OOJ            120
Name: precinct, dtype: int64
```

3.1.4 Interpretation:

We noticed that there are values in some columns, e.g. the `weapon_type` column that belonged to the same category but were named differently or rather the name was separated into small categories instead of one final category. To clean this, we merged this values together and added the smaller categories to the broad categories so as to make analysis easier. We did the same for the other columns and merged all values that belonged to the same category by using the replacing method.

In [21]:
```
#check for duplicates
df1.duplicated().sum()
```

Out[21]:
```
1
```

In [22]:
```
#Dropping the duplicated rows
df1.drop_duplicates(inplace=True)
df1.duplicated().sum()
```

Out[22]:
```
0
```

3.1.5 Interpretation:

We checked for the duplicates and found that it was only one row that was duplicated, and to clean that we dropped that row so as to to ensure data integrity and prevent bias or redundancy in our analysis.

In [23]:
```
#check for null values
df1.isna().sum()
```

Out[23]:
```
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             569
arrest_flag                 0
frisk_flag                  0
precinct                    0
sector                      0
beat                        0
dtype: int64
```

In [24]:
```
#Fill the null values
df1["officer_squad"] = df1["officer_squad"].fillna("Unknown")

#Check for any remaining null values after filling the null
df1.isna().sum()
```

Out[24]:
```
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
```

3.1.6 Interpretation:

Since we had null values in the `officer_squad` column, we filled them with 'Unknown' to prevent bias that could arise from dropping rows and filling the nulls with mode. This will also preserve the records without losing data, and clearly indicate that the squad information was not provided.

3.2 Feature Engineering

We are going to start feature engineering, where we will transform and create features from our dataset to make it more suitable for analysis and to improve the performance of our machine learning models

In [25]:
```
#Extract Stop Year from GO/SC Num
df1["stop_year"] = df1["go_/_sc_num"].astype(str).str[:4]
df1.head()
```

Out[25]:

	subject_age_group	subject_id	go_/_sc_num	terry_stop_id	stop_resolution	weapon_type	officer_id	officer_yob	officer_ge
0	26 - 35	-1	20170000036835	234548	Offense Report	None	4852	1953	
1	46 - 55	-1	20180000275629	481899	Field Contact	None	8544	1993	Fe
2	36 - 45	49326761681	20230000118635	49327076666	Field Contact	Knife/Cutting/Stabbing Instrument	7766	1984	
3	36 - 45	53986235598	20240000029589	53986202139	Field Contact	Unknown	8723	1994	
4	18 - 25	-1	20150000002928	54115	Field Contact	None	7745	1988	Fe

5 rows × 24 columns

3.2.1 Interpretation:

While examining the `go_/_sc_num` column, we observed that the first four digits follow the format of a year. To make this information more useful for analysis, we extracted those digits and created a new feature representing the specific year in which the Terry Stop occurred. We named the new column `stop_year` , and we might use it to analyze trends over time, such as changes in stop frequency across different years or to identify patterns linked to specific periods.

```
In [26]: #Function to extract and create new time-based features from the 'reported_date' column
def add_date_features(df1):
    """
    Extracts features from Reported Date.
    """
    # Convert to datetime
    df1["reported_date"] = pd.to_datetime(df1["reported_date"], errors="coerce")

    # Extract parts of the date
    df1["reported_year"] = df1["reported_date"].dt.year
    df1["reported_month"] = df1["reported_date"].dt.month_name()
    df1["day_of_week"] = df1["reported_date"].dt.dayofweek  # 0=Monday, 6=Sunday
    df1["weekday_name"] = df1["reported_date"].dt.day_name()

    # Bin into seasons
    def get_season(month):
        if month in [12, 1, 2]:
            return "Winter"
        elif month in [3, 4, 5]:
            return "Spring"
        elif month in [6, 7, 8]:
            return "Summer"
        else:
            return "Fall"

    df1["season"] = df1["reported_month"].apply(get_season)

    # Weekday vs Weekend
    df1["weekend"] = df1["day_of_week"].apply(lambda x: "Weekend" if x >= 5 else "Weekday")

    return df1

add_date_features(df1)
```

Out[26]:

	subject_age_group	subject_id	go_/_sc_num	terry_stop_id	stop_resolution	weapon_type	officer_id	officer_yob	o
0	26 - 35	-1	20170000036835	234548	Offense Report	None	4852	1953	
1	46 - 55	-1	20180000275629	481899	Field Contact	None	8544	1993	
2	36 - 45	49326761681	20230000118635	49327076666	Field Contact	Knife/Cutting/Stabbing Instrument	7766	1984	
3	36 - 45	53986235598	20240000029589	53986202139	Field Contact	Unknown	8723	1994	
4	18 - 25	-1	20150000002928	54115	Field Contact	None	7745	1988	
...	...	...	...	...	...	...	...	...	
64732	18 - 25	-1	20170000325937	354197	Offense Report	None	7755	1971	
64733	26 - 35	-1	20190000122094	549491	Arrest	None	7489	1985	
64734	36 - 45	-1	20180000467077	512455	Arrest	None	7758	1987	
64735	26 - 35	-1	20190000059044	532813	Offense Report	None	7649	1986	

3.2.2 Interpretation:

In the `reported_date` column, we transformed the raw datetime information into several new features that can help us uncover time-based patterns in the data. Specifically, we extracted the year, month, day of the week, and weekday name to analyze variations across different time frames. We also created broader seasonal categories i.e. Winter, Spring, Summer and Fall, to capture seasonal effects, and a simplified weekend/weekday feature to see whether incidents are more likely to occur during workdays or weekends. These engineered features will allow us to analyze trends over time, detect seasonal behaviors, and explore weekly patterns that might otherwise remain hidden.

In [27]:
```
#Function to extract and create new time-based features from the 'reported_time' column
def add_time_features(df1):
    """
    Extracts features from Reported Time.
    """
    # Convert Reported Time into datetime (only time part matters)
    df1["reported_time"] = pd.to_datetime(df1["reported_time"], errors="coerce")

    # Extract the hour directly
    df1["Hour"] = df1["reported_time"].dt.hour

    # Bin into parts of the day
    def get_time_bin(hour):
        if 5 <= hour < 12:
            return "Morning"
        elif 12 <= hour < 17:
            return "Afternoon"
        elif 17 <= hour < 21:
            return "Evening"
        else:
            return "Night"

    df1["time_of_day"] = df1["Hour"].apply(get_time_bin)

    return df1

add_time_features(df1)
```

Out[27]:

	subject_age_group	subject_id	go_/_sc_num	terry_stop_id	stop_resolution	weapon_type	officer_id	officer_yob	office
0	26 - 35	-1	20170000036835	234548	Offense Report	None	4852	1953	
1	46 - 55	-1	20180000275629	481899	Field Contact	None	8544	1993	
2	36 - 45	49326761681	20230000118635	49327076666	Field Contact	Knife/Cutting/Stabbing Instrument	7766	1984	
3	36 - 45	53986235598	20240000029589	53986202139	Field Contact	Unknown	8723	1994	
4	18 - 25	-1	20150000002928	54115	Field Contact	None	7745	1988	
...	...	...	...	...	...	...	...	...	
64732	18 - 25	-1	20170000325937	354197	Offense Report	None	7755	1971	
64733	26 - 35	-1	20190000122094	549491	Arrest	None	7489	1985	
64734	36 - 45	-1	20180000467077	512455	Arrest	None	7758	1987	
64735	26 - 35	-1	20190000059044	532813	Offense Report	None	7649	1986	
64736	18 - 25	-1	20180000001424	412377	Field Contact	None	8558	1993	

64736 rows × 32 columns

3.2.3 Interpretation:

From the `reported_time` column, we created new features to capture patterns based on the time of day. We first extracted the exact hour from each reported time to allow for detailed hourly analysis. To make the information more interpretable, we also grouped the hours into broader categories such as Morning, Afternoon, Evening, and Night. These engineered features will help us identify whether incidents are more common at certain times of the day, compare activity levels across different time periods, and detect daily trends that may influence behaviors or outcomes.

In [28]:
```
#Comparing the 'stop_year' column and the 'reported_year' column
match_rate = (df1["stop_year"] == df1["reported_year"].astype(str)).mean()
print(f"Match rate between Stop Year and Reported Year: {match_rate:.2%}")
```

```
Match rate between Stop Year and Reported Year: 99.42%
```

3.2.4 Interpretation:

In this step, we compared the `stop_year` column with the `reported_year` column that we had both feature engineered, to check how closely they match. By calculating the match rate, we can see whether both columns provide essentially the same information. If the overlap is very high, it would indicate redundancy, and we may choose to drop one of the columns to simplify the dataset while avoiding unnecessary duplication. Since the match rate is 99%, we will drop one of the column.

In [29]:
```
#Calculate officer age at the time of stop
df1["OfficerAge"] = df1["reported_year"] - df1["officer_yob"]

#Bin officer age into categories
def categorize_age(age):
    if 20 <= age <= 30:
        return "Young"
    elif 31 <= age <= 45:
        return "Mid-career"
    elif age >= 46:
        return "Senior"
    else:
        return "Other"

df1["officer_age_group"] = df1["OfficerAge"].apply(categorize_age)

#Quick check of results
df1[["reported_year", "OfficerAge", "officer_age_group"]].head()
```

Out[29]:

	reported_year	OfficerAge	officer_age_group
0	2017	64	Senior
1	2018	25	Young
2	2023	39	Mid-career
3	2024	30	Young
4	2015	27	Young

3.2.5 Interpretation:

We calculated each officer's age at the time of the stop by subtracting their year of birth from the reported year. To make this information more meaningful, we grouped the ages into categories: 'Young', 'Mid-career', and 'Senior', with any values outside these ranges labeled as 'Other'. This categorization allows us to analyze patterns and behaviors across different stages of an officer's career and check whether age group has any influence on outcomes.

In [30]:
```
#Map weapon types into broader groups
weapon_map = {
    "Handgun": "Firearm",
    "Other Firearm": "Firearm",
    "Rifle": "Firearm",
    "Shotgun": "Firearm",

    "Knife/Cutting/Stabbing Instrument": "Knife/Sharp",

    "Blunt Object/Striking Implement": "Blunt Object",
    "Mace/Pepper Spray": "Blunt Object",
    "Taser/Stun Gun": "Blunt Object",

    "Fire/Incendiary Device": "Other",
    "Personal Weapons (hands, feet, etc.)": "Other",
    "Poison": "Other",

    "None": "None",
    "Unknown": "Unknown"
}

df1["weapon_group"] = df1["weapon_type"].map(weapon_map)

df1["weapon_group"].value_counts()
```

Out[30]:
```
None            32586
Unknown         28035
Knife/Sharp      2976
Firearm           780
Blunt Object      343
Other              16
Name: weapon_group, dtype: int64
```

3.2.6 Interpretation:

We grouped the detailed `weapon_type` categories into broader and more meaningful groups such as 'Firearm', 'Knife/Sharp', 'Blunt Object', 'Other', 'None', and 'Unknown'. This mapping reduces the number of unique categories, making the data easier to analyze and interpret. By creating the new `weapon_group` column, we can more effectively compare how different classes of weapons are distributed and used, rather than working with many highly specific categories.

In [31]:
```
#Function to extract and create new features from the 'weapon_present' column
def weapon_present(value):
    if value == "None":
        return "No"
    elif value == "Unknown":
        return np.nan
    else:
        return "Yes"

df1["weapon_present"] = df1["weapon_type"].apply(weapon_present)

df1["weapon_present"].value_counts(dropna=False)
```

Out[31]:
```
No      32586
NaN     28035
Yes      4115
Name: weapon_present, dtype: int64
```

3.3 Interpretation:

We simplified the `weapon_type` column into a new binary-style feature called `weapon_present` . If the value was 'None', it was recoded as 'No', indicating no weapon was involved, and if the value was 'Unknown', it was treated as missing (NaN) to avoid making assumptions. All other weapon types were grouped under 'Yes', indicating that a weapon was present. This transformation helps us more easily analyze the role of weapons in stops by focusing on a clear distinction between weapon presence versus absence, while also acknowledging uncertainty in the data.

In [32]:
```
#Extract the first two words
df1["officer_squad_group"] = df1["officer_squad"].str.extract(r"^([A-Za-z]+(?:\s+[A-Za-z]+)?)")

#Check the grouping
df1["officer_squad_group"].value_counts()
```

Out[32]:
```
WEST PCT                 18800
NORTH PCT                14081
EAST PCT                  9151
SOUTH PCT                 7859
TRAINING                  7005
SOUTHWEST PCT             5494
CRG                        879
Unknown                    569
GUN VIOLENCE               141
TRAF                       103
CRISIS RESPONSE            100
HR                          93
ALTERNATIVE RESPONSE        86
CANINE                      60
CCI                         41
SWAT                        33
CRISIS INTERVENTION         21
HARBOR                      19
SAU SQUAD                   18
MAJOR CRIMES                18
SPECIAL OPS                 18
ROBBERY SQUAD               18
COP                         16
BURG                        15
HUMAN TRAFFICKING           14
NARC                        13
JOINT ENFORCEMENT           10
SVU SQUAD                    9
DV SQUAD                     9
COMMUNITY OUTREACH           9
AUTO THEFT                   7
OPS BUREAU                   4
ICAC                         3
PAWN DETAIL                  3
FORCE INVESTIGATIONS         3
PUBLIC AFFAIRS               2
GUILD PRESIDENT              2
HOMICIDE SQUAD               2
FORENSICS                    1
RECORDS                      1
ZOLD CRIME                   1
CSI SQUAD                    1
METRO                        1
```

```
ARSON                       1
FORCE REVIEW                1
COMM                        1
Name: officer_squad_group, dtype: int64
```

3.3.1 Interpretation:

From the `officer_squad` column, we extracted the first two words to create a new feature called `officer_squad_group` . This allows us to group squads into broader categories, therefore reducing the complexity of the original text data.

In [33]:
```
#Group rare categories
def group_rare(df1, column, min_count=200):
    value_counts = df1[column].value_counts()
    rare_values = value_counts[value_counts < min_count].index
    df1[column] = df1[column].replace(rare_values, "Other")
    return df1

#Apply to Final Call Type
df1 = group_rare(df1, "final_call_type", min_count=200)
df1["final_call_type"].value_counts()
```

Out[33]:
```
Unknown                                          13602
SUSPICIOUS CIRCUM. - SUSPICIOUS PERSON            6167
PROWLER - TRESPASS                                4572
DISTURBANCE - OTHER                               3893
Other                                             3878
ASSAULTS, OTHER                                   3236
WARRANT SERVICES - FELONY                         2252
SUSPICIOUS CIRCUM. - SUSPICIOUS VEHICLE           1950
NARCOTICS - OTHER                                 1881
THEFT - SHOPLIFT                                  1803
DV - ARGUMENTS, DISTURBANCE (NO ARREST)           1764
DV - DOMESTIC VIOL/ASLT (ARREST MANDATORY)        1636
ASSAULTS - HARASSMENT, THREATS                    1594
WARRANT SERVICES - MISDEMEANOR                    1373
CRISIS COMPLAINT - GENERAL                        1213
AUTOMOBILES - RECOVERY (THEFT)                    1093
THEFT - ALL OTHER                                 1073
PROPERTY DEST (DAMG)                               973
BURGLARY - NON RESIDENTIAL/COMMERCIAL              903
ROBBERY - STRONG ARM                               849
TRAFFIC - D.U.I.                                   834
WEAPON, PERSON WITH - GUN                          757
ROBBERY - ARMED                                    733
DISTURBANCE - FIGHT                                670
THEFT - CAR PROWL                                  630
WEAPON,PERSON WITH - OTHER WEAPON                  618
NARCOTICS - DRUG TRAFFIC LOITERING                 572
BURGLARY - RESIDENTIAL OCCUPIED                    558
TRAFFIC - MOVING VIOLATION                         544
ASSAULTS - FIREARM INVOLVED                        493
DV - ENFORCE COURT ORDER (ARREST MANDATED)         346
BURGLARY - RESIDENTIAL, UNOCCUPIED                 317
AUTOMOBILES - AUTO THEFT & RECOVERY                288
TRAFFIC - MV COLLISION INVESTIGATION               279
DV - DOMESTIC VIOLENCE (ARREST DISCRETIONARY)      268
ARSON, BOMBS, EXPLO - RECKLESS BURNING             253
SEX OFFENSES (NON-RAPE) - LEWD CONDUCT             230
MISCHIEF OR NUISANCE - GENERAL                     226
PROPERTY DEST (DAMG) - GRAFFITI (INCLUDES GANG)    208
ASSIST PUBLIC - OTHER (NON-SPECIFIED)              207
Name: final_call_type, dtype: int64
```

3.3.2 Interpretation:

We grouped together rare categories in the `final_call_type` column by replacing any category with fewer than 200 occurrences with the label 'Other'. This reduces the number of very small or unique categories, which can otherwise add noise, make the data harder to analyze, and weaken model performance. By consolidating rare categories, we keep the most frequent and meaningful call types while still retaining information about less common cases under a single group.

In [34]:
```
#List of columns you want to drop
cols_to_drop = [
    "sector",
    "beat",
    "day_of_week",
    "Hour",
    "stop_year",
    "stop_resolution",
    "initial_call_type",
    "officer_yob",
    "weapon_type",
    "reported_date",
    "reported_time",
    "officer_squad",
    "OfficerAge",
    "subject_id",
    "officer_id",
    "terry_stop_id",
    "go_/_sc_num"
]

#Drop them from the dataframe
df1 = df1.drop(columns=cols_to_drop)

#Confirm that the columns have been dropped
df1.columns
```

Out[34]:
```
Index(['subject_age_group', 'officer_gender', 'officer_race',
       'subject_perceived_race', 'subject_perceived_gender', 'final_call_type',
       'call_type', 'arrest_flag', 'frisk_flag', 'precinct', 'reported_year',
       'reported_month', 'weekday_name', 'season', 'weekend', 'time_of_day',
       'officer_age_group', 'weapon_group', 'weapon_present',
       'officer_squad_group'],
      dtype='object')
```

3.3.3 Interpretation:

We dropped all the unnecessary columns that are not relevant to our analysis in order to reduce noise, simplify the dataset, and focus only on the features that add value to our insights and modeling.

Reasons as to why we dropped each column:

- `go_/_sc_num` : Excluded after we extracted the year information from it, as the raw column itself was no longer needed.
- `subject_id` , `officer_id` , and `terry_stop_id` : Dropped because they are unique identifiers that do not provide meaningful information for analysis or modeling.
- `OfficerAge` : Excluded because it was replaced with the categorized version `officer_age_group` , which is more interpretable.
- `officer_squad` : Dropped because it was simplified into the new officer_squad_group feature.
- `reported_date` and `reported_time` : Removed after deriving more useful features from them, including year, month, season, day of week, and time of day.
- `weapon_type` : Excluded because it had already been transformed into more meaningful features such as weapon_group and weapon_present.
- `officer_yob` : Dropped since it contains sensitive personal information and was already used to compute the officer's age.
- `initial_call_type` : Removed because the `final_call_type` column provides a more accurate and reliable representation of the type of call.
- `stop_resolution` : Excluded because it is closely related to the outcome of the stop and could introduce data leakage if used in modeling, more so because 'Arrest' was included as one of the values.
- `stop_year` : Dropped because it contained the same information as `reported_year` , and keeping both would only create duplication.
- `day_of_week` and `Hour` : Removed since we had already extracted broader and more useful time-based features such as weekday vs. weekend, time of day, and season, making these columns redundant.
- `sector` and `beat` : Dropped because they provide very detailed location information that results in too many categories, making the data difficult to analyze and less meaningful at that level of granularity.

```
In [35]: #Check for any duplicates after feature engineering
         df1.duplicated().sum()

Out[35]: 2798
```

```
In [36]: #Drop the duplicated rows
         df1 = df1.drop_duplicates()

         #Check for any remaining duplicates after dropping
         df1.duplicated().sum()

Out[36]: 0
```

3.3.4 Interpretation:

After dropping all the unnecessary columns in our dataset after feature engineering, we checked for the duplicates and found that there were 2798 duplicated rows. To clean that we dropped that row so as to to ensure data integrity and prevent bias or redundancy in our analysis.

3.4 Exploratory Data Analysis

Exploratory Data Analysis (EDA) plays a central role in understanding the information contained in our Terry Stops dataset before moving into advanced modeling or decision-making. It allows us to carefully examine the structure of the data, identify potential quality issues, and uncover meaningful trends that can inform both policy and practice.

The Terry Stops dataset provides detailed information about stop-and-frisk incidents, including officer demographics, subject details, stop outcomes, reported times and locations, and whether weapons were involved. This rich dataset gives us an opportunity to analyze not only individual incidents but also broader patterns and trends across time and groups.

By using summary statistics and visualizations, our goal is to generate insights that can guide both operational decisions and policy considerations. Specifically, we aim to:

- Benchmark stop outcomes and identify potential disparities across different demographics.
- Detect temporal and spatial patterns, such as which times of day or locations are associated with higher escalation or weapon presence.
- Evaluate the role of officer experience and assignment in shaping stop outcomes.
- Establish baselines that can inform strategies for accountability, training, and risk reduction.

This analysis will form the foundation for building predictive models and developing evidence-based recommendations on Terry Stops, ensuring that future decisions are both data-driven and aligned with fairness and public safety objectives.

In [37]:
```
#Saving the cleaned dataset
df1.to_csv("cleaned_Terry_Stops_20250909.csv")
df2 = pd.read_csv("cleaned_Terry_Stops_20250909.csv", index_col=0)
df2.head()
```

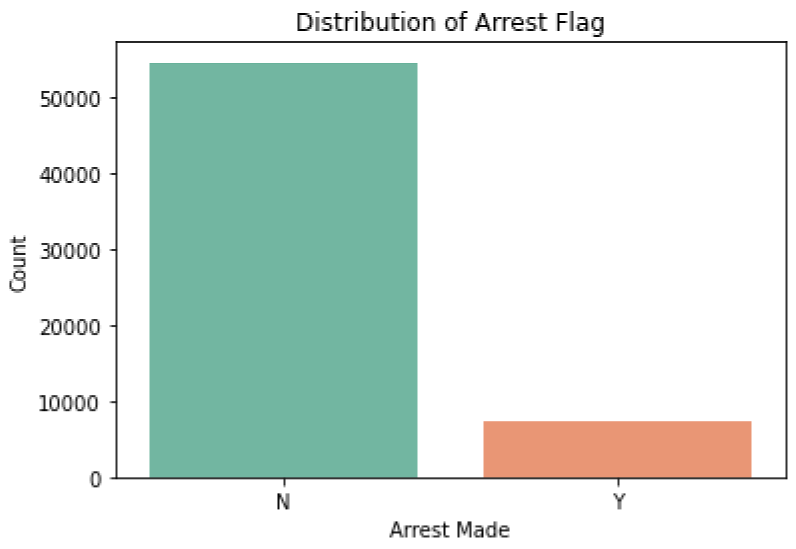
Out[37]:

	subject_age_group	officer_gender	officer_race	subject_perceived_race	subject_perceived_gender	final_call_type	call_type	arrest_flag
0	26 - 35	Male	Asian	White	Male	THEFT - SHOPLIFT	911	N
1	46 - 55	Female	Hispanic	White	Male	Unknown	Unknown	N
2	36 - 45	Male	White	Native Hawaiian or Other Pacific Islander	Male	SUSPICIOUS CIRCUM. - SUSPICIOUS PERSON	ONVIEW	N
3	36 - 45	Male	White	Black or African American	Male	DISTURBANCE - OTHER	ONVIEW	N
4	18 - 25	Female	Unknown	Black or African American	Female	Unknown	Unknown	N

3.4.1 Univariate Analysis

First of all, we are going to do Univariate Analysis which is basically examining one variable at a time, in order to understand the distribution of the variable.

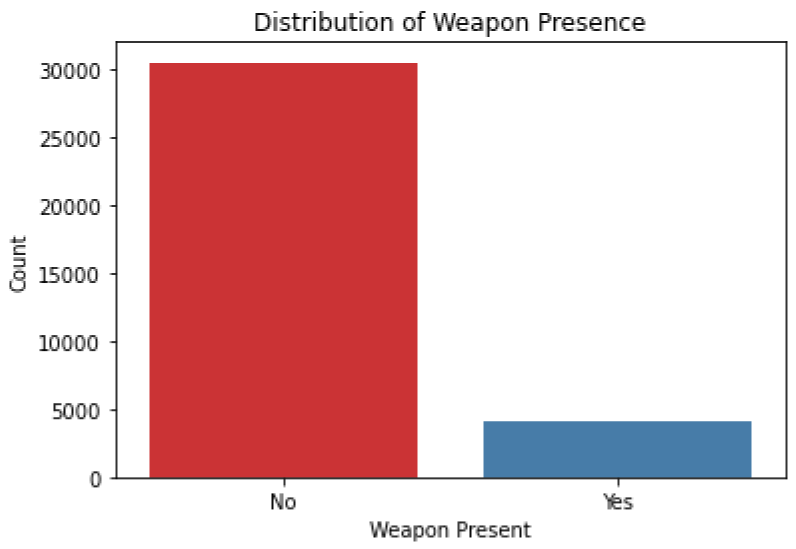
In [38]:
```
#Plotting the Distribution of Arrest Flag
plt.figure(figsize=(6,4))
sns.countplot(data=df2, x='arrest_flag', palette='Set2')
plt.title('Distribution of Arrest Flag')
plt.xlabel('Arrest Made')
plt.ylabel('Count')
plt.show()
```



3.4.1.1 Observation:

In most Terry Stops, no arrest was made, and only a smaller number of stops led to an arrest. This could be because many stops are carried out as precautionary measures or for questioning, without always finding enough reason to arrest someone. The takeaway is that while arrests are less common, understanding the situations where arrests do happen can give important insight into patterns of enforcement and decision-making.

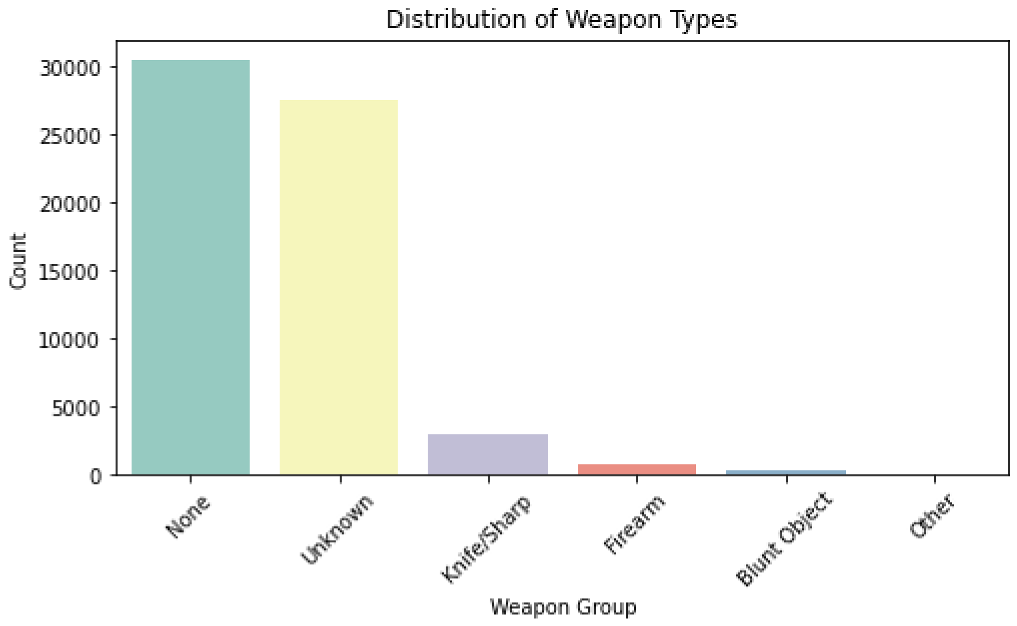
In [39]:
```
#Plotting the Distribution of Weapon Presence
plt.figure(figsize=(6,4))
sns.countplot(data=df2, x='weapon_present', palette='Set1')
plt.title('Distribution of Weapon Presence')
plt.xlabel('Weapon Present')
plt.ylabel('Count')
plt.show()
```



3.4.1.2 Observation:

In most Terry Stops no weapon was found, while only a small number of stops involved a weapon. This could be because many stops were based on suspicion or routine checks rather than actual violent behavior, so it’s expected that weapons would not often be present. The takeaway here is that while weapons are rare, the few cases where they are found may need more attention since they pose higher risks for both officers and the people being stopped.

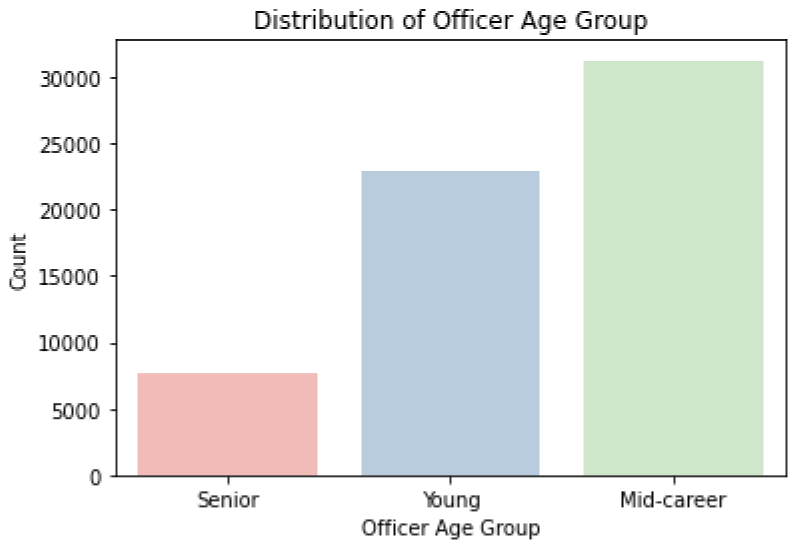
In [40]:
```
#Plotting the Distribution of Weapon Types
plt.figure(figsize=(8,4))
sns.countplot(data=df2, x='weapon_group', order=df1['weapon_group'].value_counts().index, palette='Set3')
plt.title('Distribution of Weapon Types')
plt.xlabel('Weapon Group')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



3.4.1.3 Observation:

Most Terry Stops showed no weapon being present, followed by cases where a knife or sharp object was involved, while the least common category was listed as "Other". This could be because sharp objects like knives are more easily accessible and carried compared to firearms or unusual weapons, making them more likely to appear in stops. The takeaway is that while the majority of stops involve no weapon, knives and sharp objects are the most common threat when a weapon is present, which can guide safety training and officer awareness.

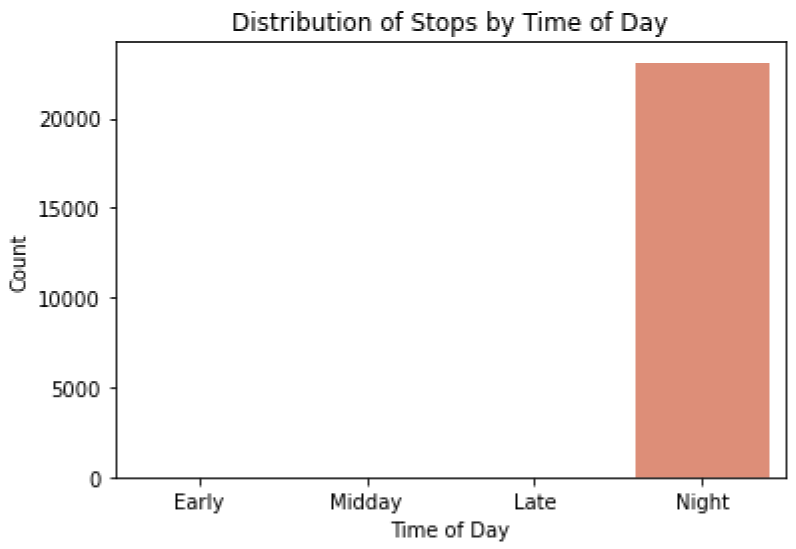
In [41]:
```
#Plotting the Distribution of Officer Age Group
plt.figure(figsize=(6,4))
sns.countplot(data=df2, x='officer_age_group', palette='Pastel1')
plt.title('Distribution of Officer Age Group')
plt.xlabel('Officer Age Group')
plt.ylabel('Count')
plt.show()
```



3.4.1.4 Observation:

In most Terry Stops, the officers belonged to the mid-career age group, while the senior age group had the fewest stops. This could be because mid-career officers are often the most active in fieldwork, while senior officers may take on more supervisory or administrative roles, leading to fewer direct stops. The takeaway is that the data reflects the typical workload distribution in policing, where mid-career officers handle the majority of frontline duties.

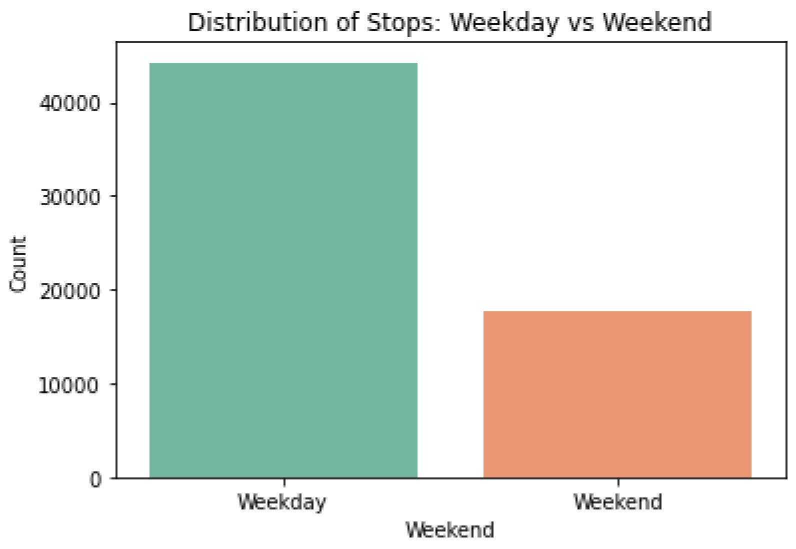
In [42]:
```
#Plotting the Distribution of Stops by Time of Day
plt.figure(figsize=(6,4))
sns.countplot(data=df2, x='time_of_day', order=['Early', 'Midday', 'Late', 'Night'], palette='coolwarm')
plt.title('Distribution of Stops by Time of Day')
plt.xlabel('Time of Day')
plt.ylabel('Count')
plt.show()
```



3.4.1.5 Observation:

Almost all Terry Stops took place at night, with very few or none happening during the day. This could be because nighttime often has higher levels of suspicious activity or reduced visibility, which may lead officers to conduct more stops during those hours. The takeaway is that stop activity is strongly linked to time of day, and focusing on nighttime patterns can provide more useful insights into how and why these stops occur.

In [43]:
```python
#Plotting the Distribution of Stops by Weekday vs Weekend
plt.figure(figsize=(6,4))
sns.countplot(data=df2, x='weekend', palette='Set2')
plt.title('Distribution of Stops: Weekday vs Weekend')
plt.xlabel('Weekend')
plt.ylabel('Count')
plt.show()
```



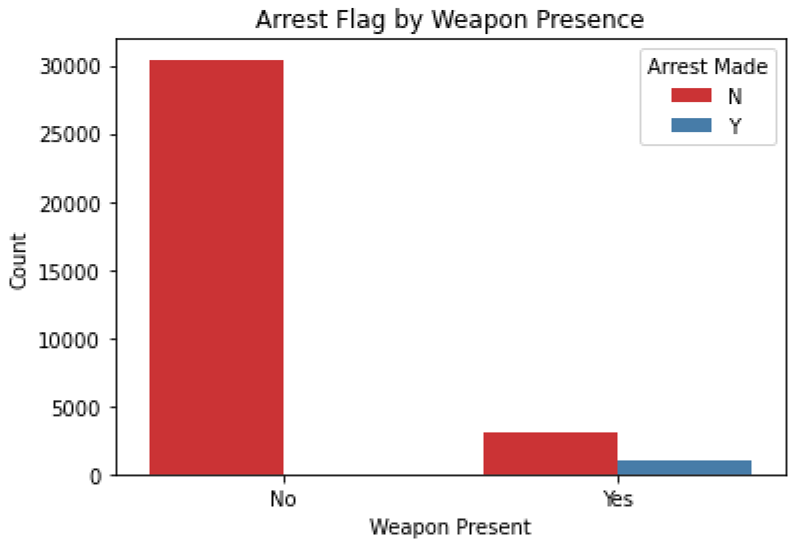
3.4.1.6 Observation:

Most Terry Stops were made during weekdays, while fewer happened on weekends. This might seem unusual since weekends are often associated with more social activities and potential incidents, but it could be explained by officers having higher staffing and patrol activity during the week compared to weekends. The takeaway is that stop patterns may reflect police scheduling and resource allocation rather than just public activity, which is an important factor to consider when interpreting the data.

3.4.2 Bivariate Analysis

Bivariate analysis looks at the relationship between two variables, and this will help us to understand how one variable may change based on another; it shows whether a variable is dependent or affected by another variable.

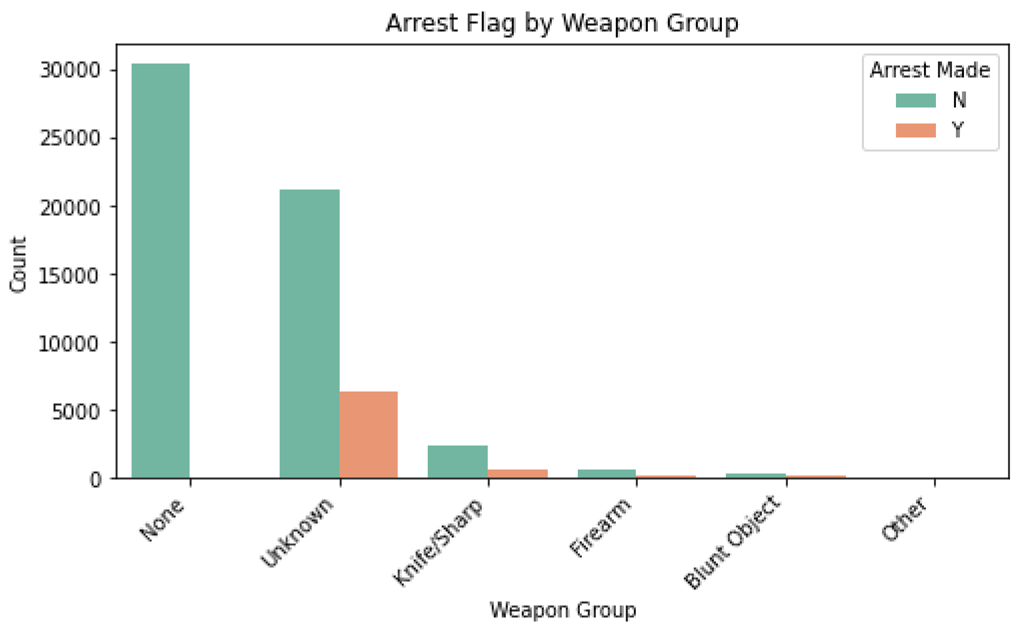
```
In [44]: #Plotting Arrest Flag by Weapon Presence
         plt.figure(figsize=(6,4))
         sns.countplot(data=df2, x='weapon_present', hue='arrest_flag', palette='Set1')
         plt.title('Arrest Flag by Weapon Presence')
         plt.xlabel('Weapon Present')
         plt.ylabel('Count')
         plt.legend(title='Arrest Made')
         plt.show()
```



3.4.2.1 Observation:

Overall, fewer arrests were made regardless of whether a weapon was present or not. When no weapon was involved, most stops ended without an arrest. Even when a weapon was present, the number of stops without an arrest was still higher than those that led to an arrest. This could be because the presence of a weapon alone does not automatically justify an arrest, other factors like intent, behavior, or evidence may influence the decision. The takeaway is that while weapons raise the risk level of a stop, they do not always result in an arrest, showing that officer discretion and case circumstances play a big role.

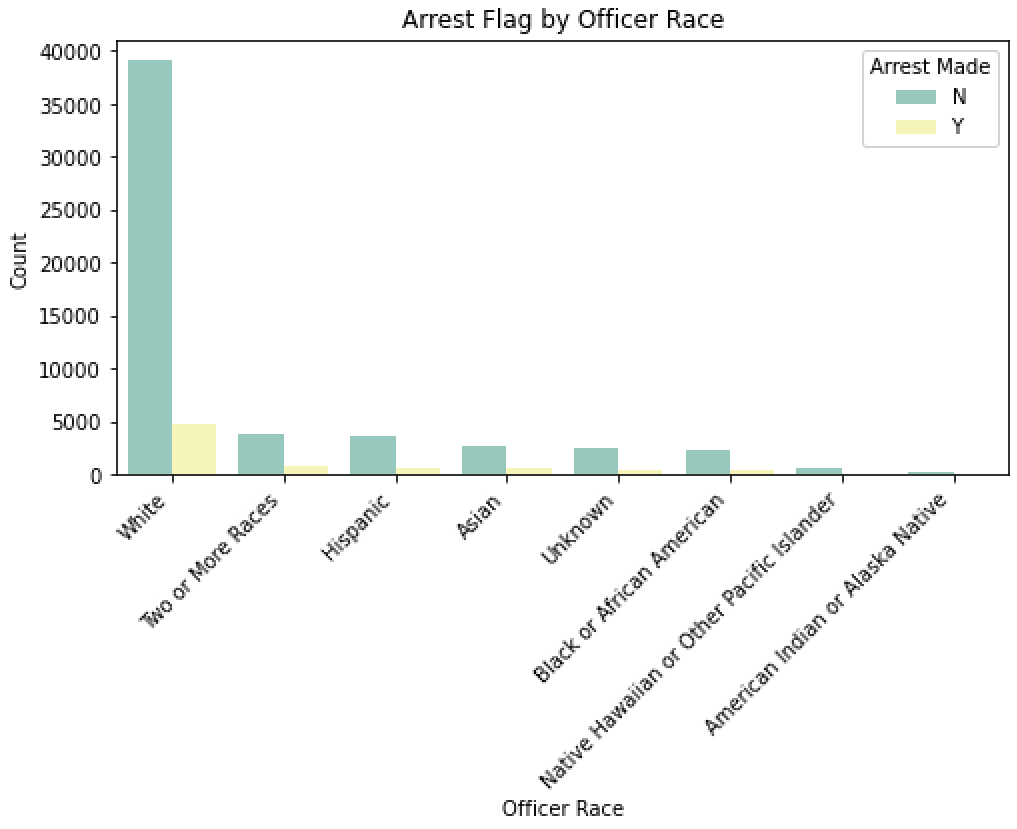
In [45]:
```python
#Plotting Arrest Flag by Weapon Group
plt.figure(figsize=(8,4))
sns.countplot(data=df2, x='weapon_group', hue='arrest_flag',
              order=df2['weapon_group'].value_counts().index, palette='Set2')
plt.title('Arrest Flag by Weapon Group')
plt.xlabel('Weapon Group')
plt.ylabel('Count')
plt.xticks(rotation=45, ha="right")
plt.legend(title='Arrest Made')
plt.show()
```



3.4.2.2 Observation:

There were generally fewer arrests overall, but among the weapon groups, most arrests came from stops involving knives or sharp objects, followed by firearms. Stops where no weapon was present resulted in zero arrests. This could be because knives and firearms are viewed as more immediate threats, making arrests more likely in those situations, while the absence of a weapon reduces the justification for an arrest. The takeaway is that the type of weapon strongly influences arrest decisions, with sharper and more dangerous weapons leading to higher arrest rates.

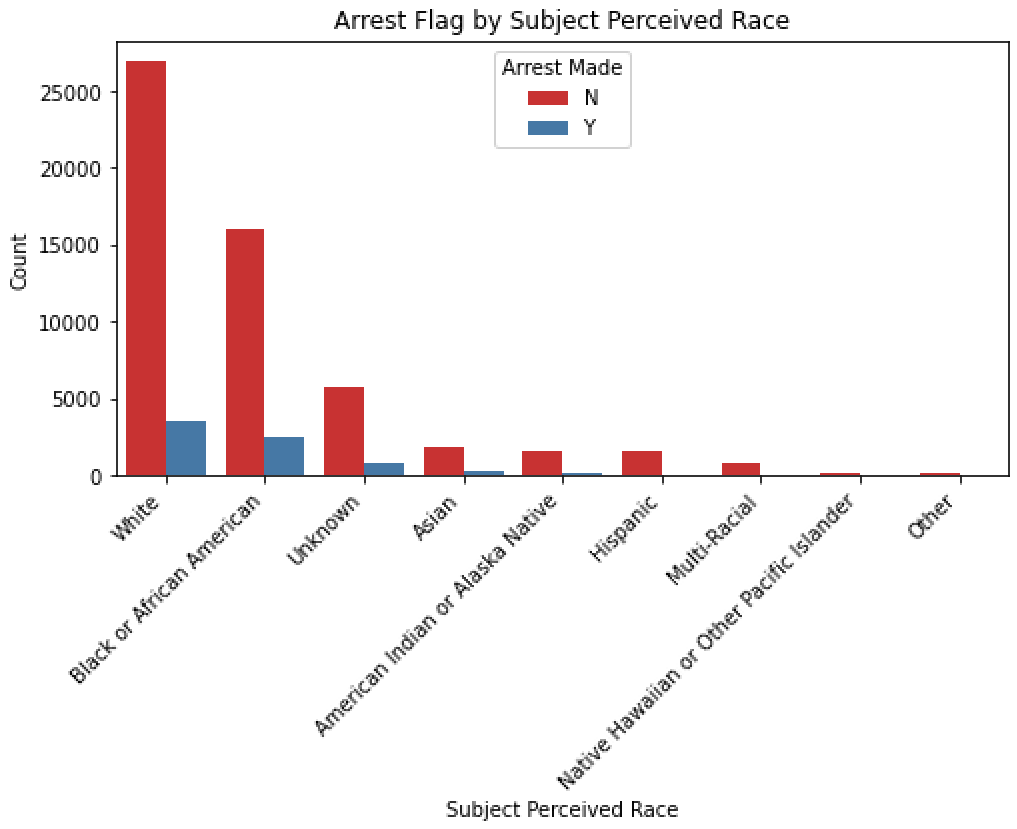
In [46]:
```
#Plotting Arrest Flag by Officer Race
plt.figure(figsize=(8,4))
sns.countplot(data=df2, x='officer_race', hue='arrest_flag',
              order=df2['officer_race'].value_counts().index, palette='Set3')
plt.title('Arrest Flag by Officer Race')
plt.xlabel('Officer Race')
plt.ylabel('Count')
plt.xticks(rotation=45, ha="right")
plt.legend(title='Arrest Made')
plt.show()
```



3.4.2.3 Observation:

Most arrests, as well as most no-arrests, were made by White officers, while officers identified as American Indian or Alaska Native had the least number of both arrests and no-arrests. This suggests that White officers carried out the majority of the stops overall, which could

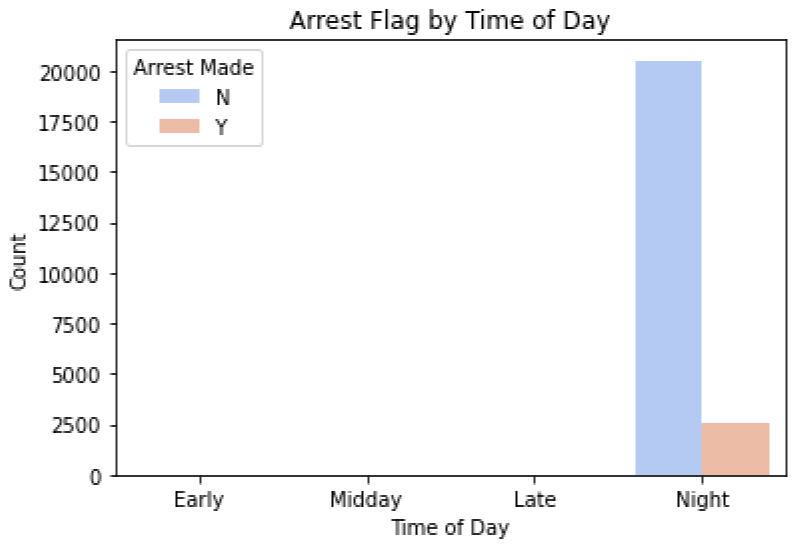
In [47]:
```
#Plotting Arrest Flag by Subject Perceived Race
plt.figure(figsize=(8,4))
sns.countplot(data=df2, x='subject_perceived_race', hue='arrest_flag',
              order=df2['subject_perceived_race'].value_counts().index, palette='Set1')
plt.title('Arrest Flag by Subject Perceived Race')
plt.xlabel('Subject Perceived Race')
plt.ylabel('Count')
plt.xticks(rotation=45, ha="right")
plt.legend(title='Arrest Made')
plt.show()
```



3.4.2.4 Observation:

Most of the people stopped were perceived to be White, and this group also had the highest number of both arrests and no-arrests. They were followed by Black or African American subjects, who also made up a significant portion of the stops. This pattern could be because White individuals make up a larger share of the general population in the area, which increases the likelihood of them being stopped more often. However, the higher number of stops for Black or African American individuals compared to other minority groups may also suggest possible differences in how stops are carried out across races.

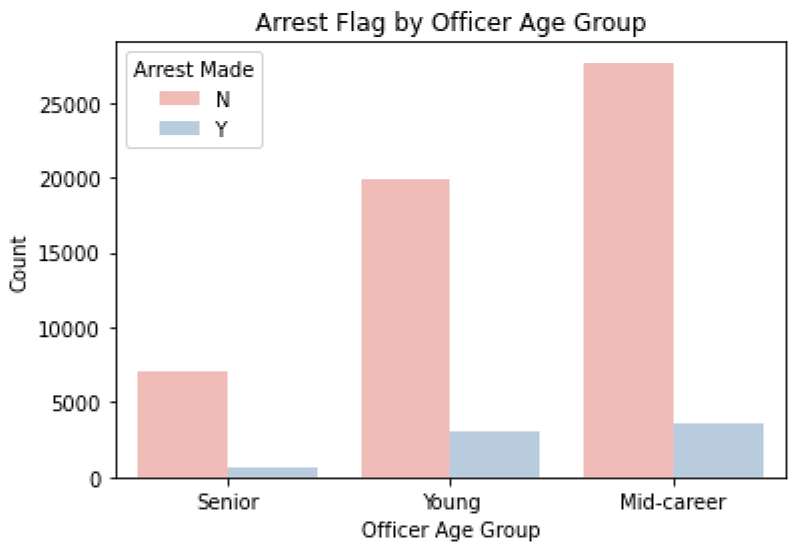
In [48]:
```python
#Plotting Arrest Flag by Time of Day
plt.figure(figsize=(6,4))
sns.countplot(data=df2, x='time_of_day', hue='arrest_flag',
              order=['Early','Midday','Late','Night'], palette='coolwarm')
plt.title('Arrest Flag by Time of Day')
plt.xlabel('Time of Day')
plt.ylabel('Count')
plt.legend(title='Arrest Made', loc="upper left")
plt.show()
```



3.4.2.5 Observation:

All the stops in the dataset happened at night, while no stops or arrests were recorded during the early morning, midday, or late afternoon hours. Among the night stops, most did not result in arrests, and only a small number led to an arrest. This could be because police patrol activity or certain behaviors that lead to stops are more common at night, when visibility is lower and people are more likely to be moving around in situations that draw police attention.

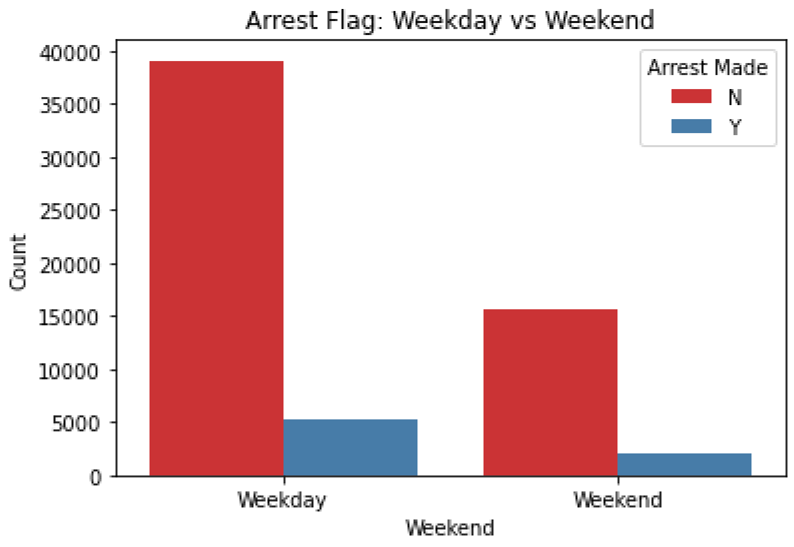
In [49]:
```python
#Plotting Arrest Flag by Officer Age Group
plt.figure(figsize=(6,4))
sns.countplot(data=df2, x='officer_age_group', hue='arrest_flag', palette='Pastel1')
plt.title('Arrest Flag by Officer Age Group')
plt.xlabel('Officer Age Group')
plt.ylabel('Count')
plt.legend(title='Arrest Made')
plt.show()
```



3.4.2.6 Observation:

Most stops and arrests were carried out by officers in the mid-career age group, while the senior officers had the lowest number of stops and arrests. This may be because mid-career officers are often more active in the field, taking on the bulk of patrol and enforcement duties, while senior officers may handle more supervisory or administrative roles, which reduces their direct involvement in stops.

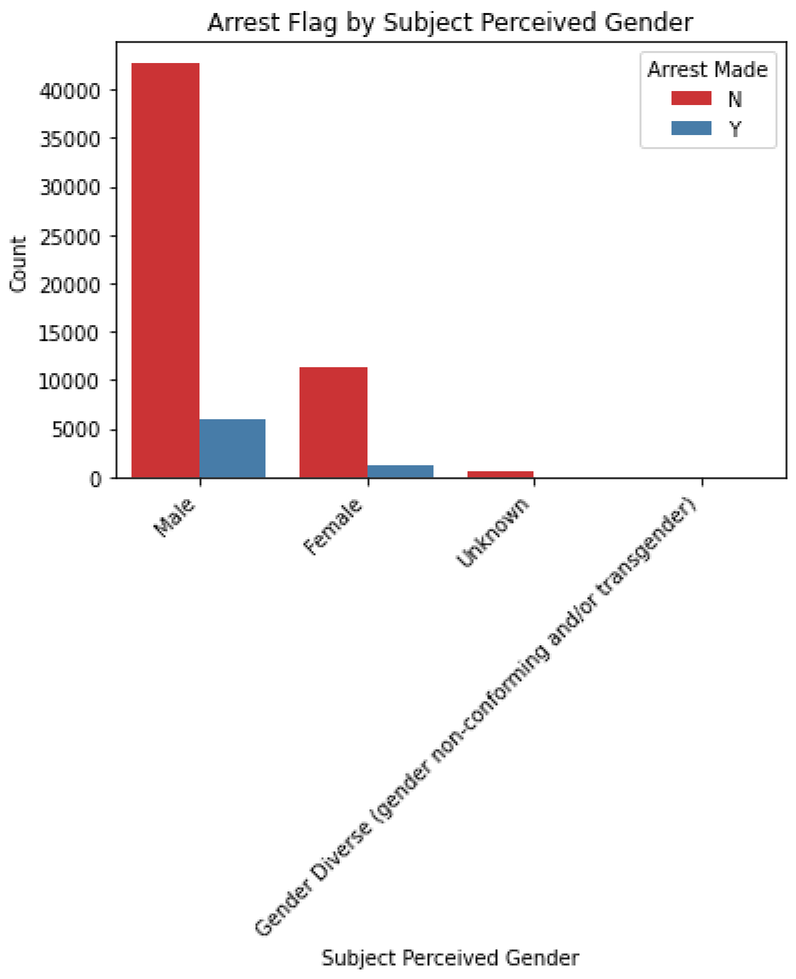
In [50]:
```
#Plotting Arrest Flag by Weekday vs Weekend
plt.figure(figsize=(6,4))
sns.countplot(data=df2, x='weekend', hue='arrest_flag', palette='Set1')
plt.title('Arrest Flag: Weekday vs Weekend')
plt.xlabel('Weekend')
plt.ylabel('Count')
plt.legend(title='Arrest Made')
plt.show()
```



3.4.2.7 Observation:

Most stops and arrests happened during the weekdays, while fewer stops and arrests were recorded on the weekends. This could be because weekdays are usually busier, with more people commuting to work, school, or other activities, which increases the chances of interactions with officers. On weekends, there may be fewer routine movements or officers might be assigned differently, leading to fewer stops overall.

```
In [51]: #Plotting Arrest Flag by Subject Perceived Gender
         plt.figure(figsize=(6,4))
         sns.countplot(data=df2, x='subject_perceived_gender', hue='arrest_flag', palette='Set1')
         plt.title('Arrest Flag by Subject Perceived Gender')
         plt.xlabel('Subject Perceived Gender')
         plt.ylabel('Count')
         plt.xticks(rotation=45, ha="right")
         plt.legend(title='Arrest Made')
         plt.show()
```



3.4.2.8 Observation:

Males were involved in the majority of stops and also accounted for the highest number of arrests. This could suggest that men are more frequently engaged in activities that draw police attention or are more often perceived as potential offenders compared to females.

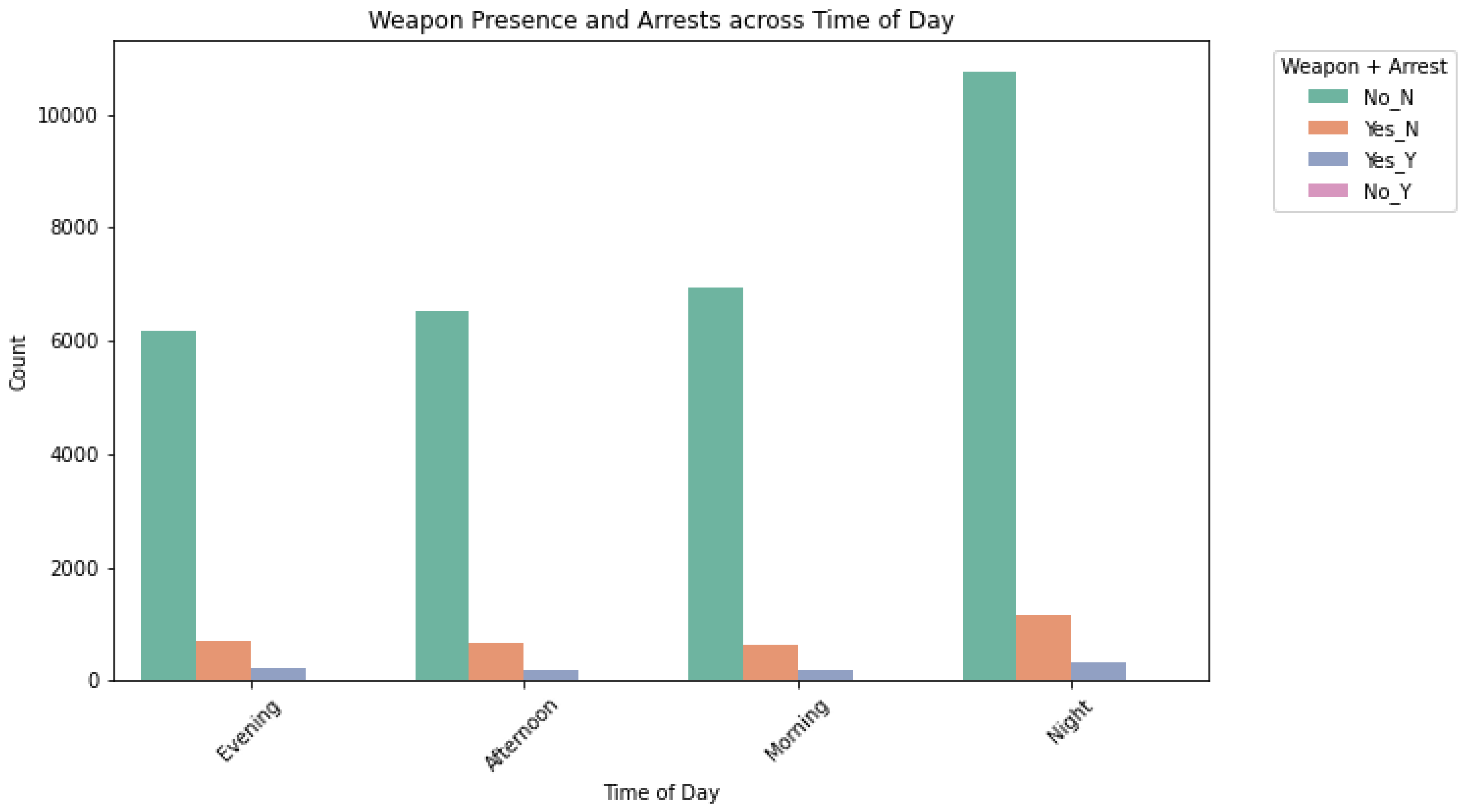
3.4.3 Multivariate Analysis

Multivariate analysis looks at the relationship between three or more variables at the same time, and it is used to uncover more complex patterns and deeper insights.

In [52]:
```
# Drop rows with nulls in either weapon_present or arrest_flag
df2_clean = df2.dropna(subset=["weapon_present", "arrest_flag"]).copy()

#Combine weapon and arrest into one column
df2_clean["weapon_arrest"] = (df2_clean["weapon_present"] + "_" + df2_clean["arrest_flag"])

#Plot Weapon Presence and Arrests across Time of Day
plt.figure(figsize=(10,6))
sns.countplot(data=df2_clean, x="time_of_day", hue="weapon_arrest", palette="Set2", dodge=True)
plt.title("Weapon Presence and Arrests across Time of Day")
plt.xlabel("Time of Day")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.legend(title="Weapon + Arrest", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



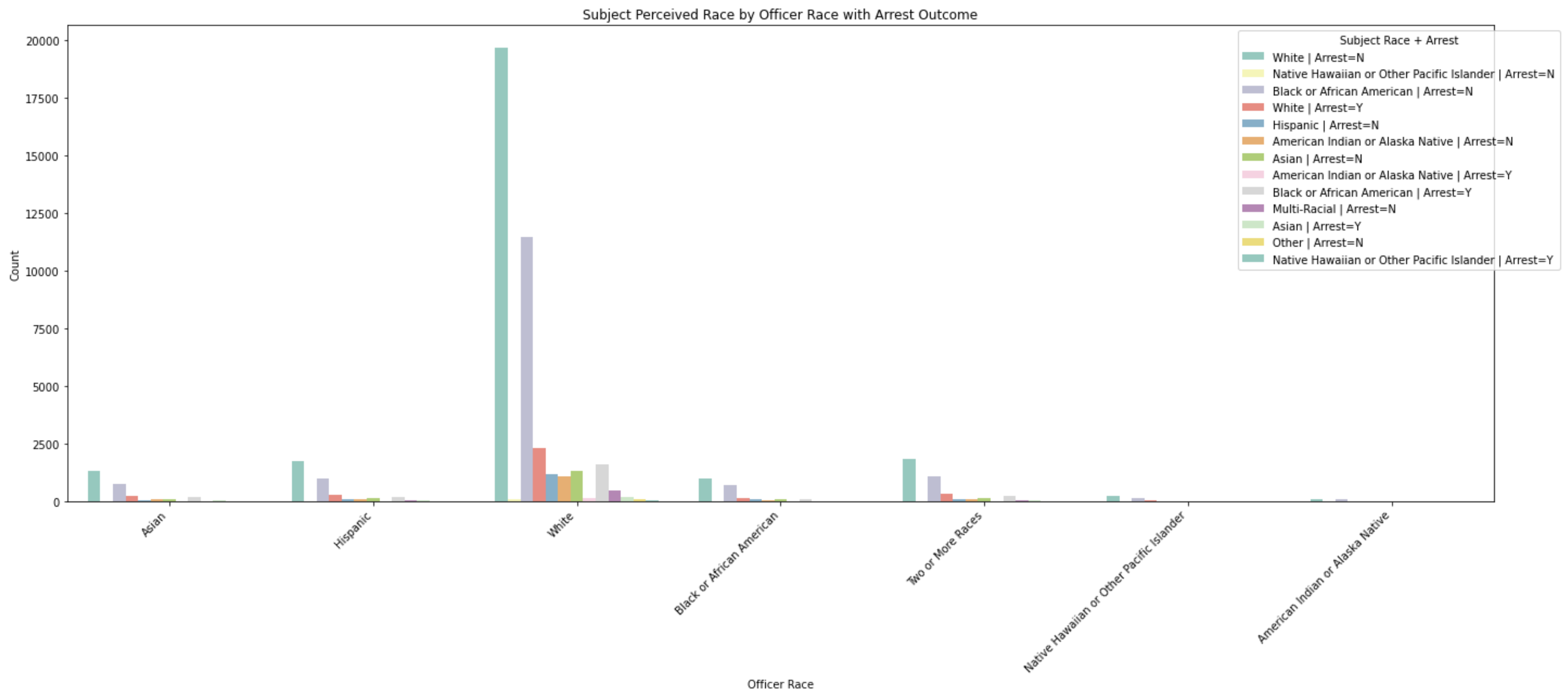
3.4.3.1 Observation:

Most arrests happened at night when a weapon was present. In cases where no weapon was found, there were no arrests at all. Interestingly, there were also many stops at night where a weapon was present but no arrest was made. This could suggest that nighttime situations are more likely to involve weapons, but not all of these cases necessarily lead to an arrest—possibly because the weapon was legal, properly licensed, or the situation did not meet the threshold for arrest.

In [53]:
```
#Drop all 'Unknown' values
df_plot = df2[(df2["subject_perceived_race"].str.lower() != "unknown") & (df2["officer_race"].str.lower() != "

#Combine Subject Race and Arrest Flag
df_plot["subject_race_arrest"] = (
    df_plot["subject_perceived_race"] + " | Arrest=" + df_plot["arrest_flag"])

#Plot Subject Perceived Race by Officer Race with Arrest Outcome
plt.figure(figsize=(20,9))
sns.countplot(data=df_plot, x="officer_race", hue="subject_race_arrest", palette="Set3")
plt.title("Subject Perceived Race by Officer Race with Arrest Outcome")
plt.xlabel("Officer Race")
plt.ylabel("Count")
plt.xticks(rotation=45, ha='right')
plt.legend(title="Subject Race + Arrest", bbox_to_anchor=(1.05, 1), loc="upper right")
plt.tight_layout()
plt.show()
```

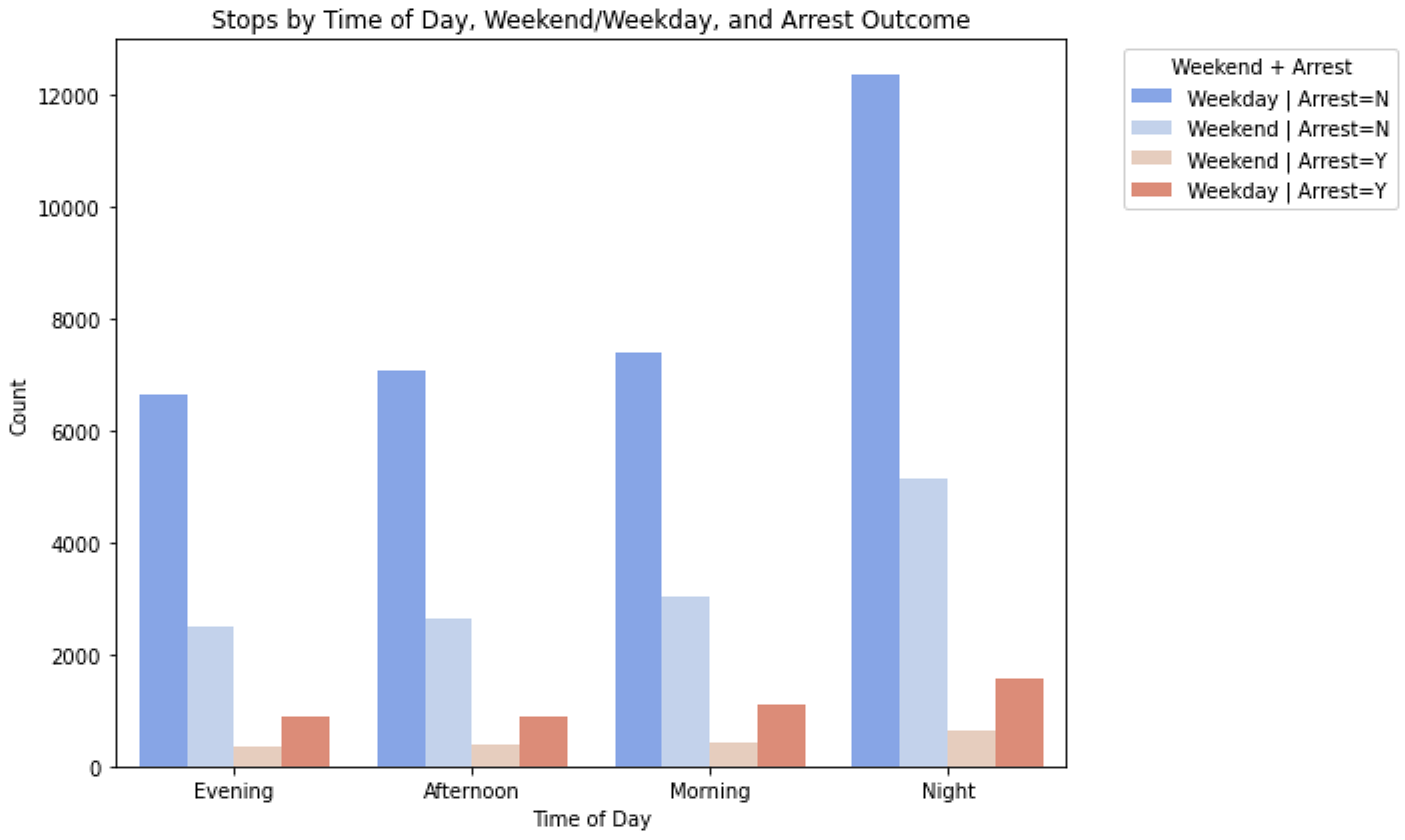


3.4.3.2 Observation:

Most stops were made by white officers, with most of these involving white subjects where no arrests occurred. White officers also made the highest number of arrests, mainly involving white subjects, followed by Black or African American subjects. This could be because white officers make up the majority of the force, but the higher number of stops and arrests involving Black subjects may also point to possible disparities in how different groups experience police stops.

In [54]:
```
#Combine Weekend/Weekday + Arrest Flag
df_plot["weekend_arrest"] = (df2["weekend"] + " | Arrest=" + df2["arrest_flag"])

#Plot Stops by Time of Day, Weekend/Weekday, and Arrest Outcome
plt.figure(figsize=(10,6))
sns.countplot(data=df_plot,x="time_of_day",hue="weekend_arrest",palette="coolwarm")
plt.title("Stops by Time of Day, Weekend/Weekday, and Arrest Outcome")
plt.xlabel("Time of Day")
plt.ylabel("Count")
plt.legend(title="Weekend + Arrest", bbox_to_anchor=(1.05, 1), loc="upper left")
plt.tight_layout()
plt.show()
```



3.4.3.3 Observation:

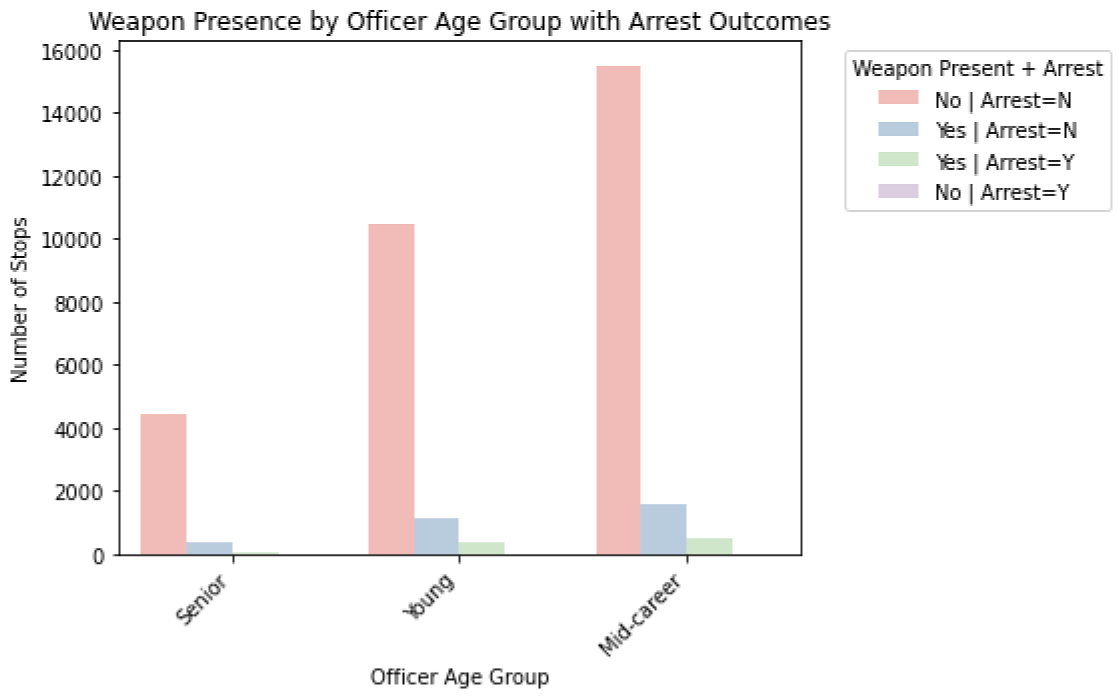
Most arrests happened at night during the weekdays, which could be because evenings and nights are often busier times with more activities that might attract police attention. The second highest arrests took place in the weekday mornings, possibly because officers are more active then and people are out starting their daily routines. This shows that both time of day and whether it is a weekday or weekend may influence when arrests are more likely to occur

In [55]:
```python
#Drop rows with nulls in relevant columns
df_plot = df2.dropna(subset=["weapon_present", "arrest_flag", "officer_age_group"]).copy()

#Remove 'Unknown' (after conversion to string)
df_plot = df_plot[df_plot["weapon_present"].str.lower() != "unknown"]
df_plot = df_plot[df_plot["officer_age_group"].str.lower() != "unknown"]

#Combine Weapon Presence + Arrest Flag
df_plot["weapon_arrest"] = df_plot["weapon_present"] + " | Arrest=" + df_plot["arrest_flag"]

#Plot Weapon Presence by Officer Age Group with Arrest Outcomes
plt.figure(figsize=(8,5))
sns.countplot(data=df_plot, x="officer_age_group", hue="weapon_arrest", palette="Pastel1")
plt.title("Weapon Presence by Officer Age Group with Arrest Outcomes")
plt.xlabel("Officer Age Group")
plt.ylabel("Number of Stops")
plt.legend(title="Weapon Present + Arrest", bbox_to_anchor=(1.05, 1))
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



3.4.3.4 Observation:

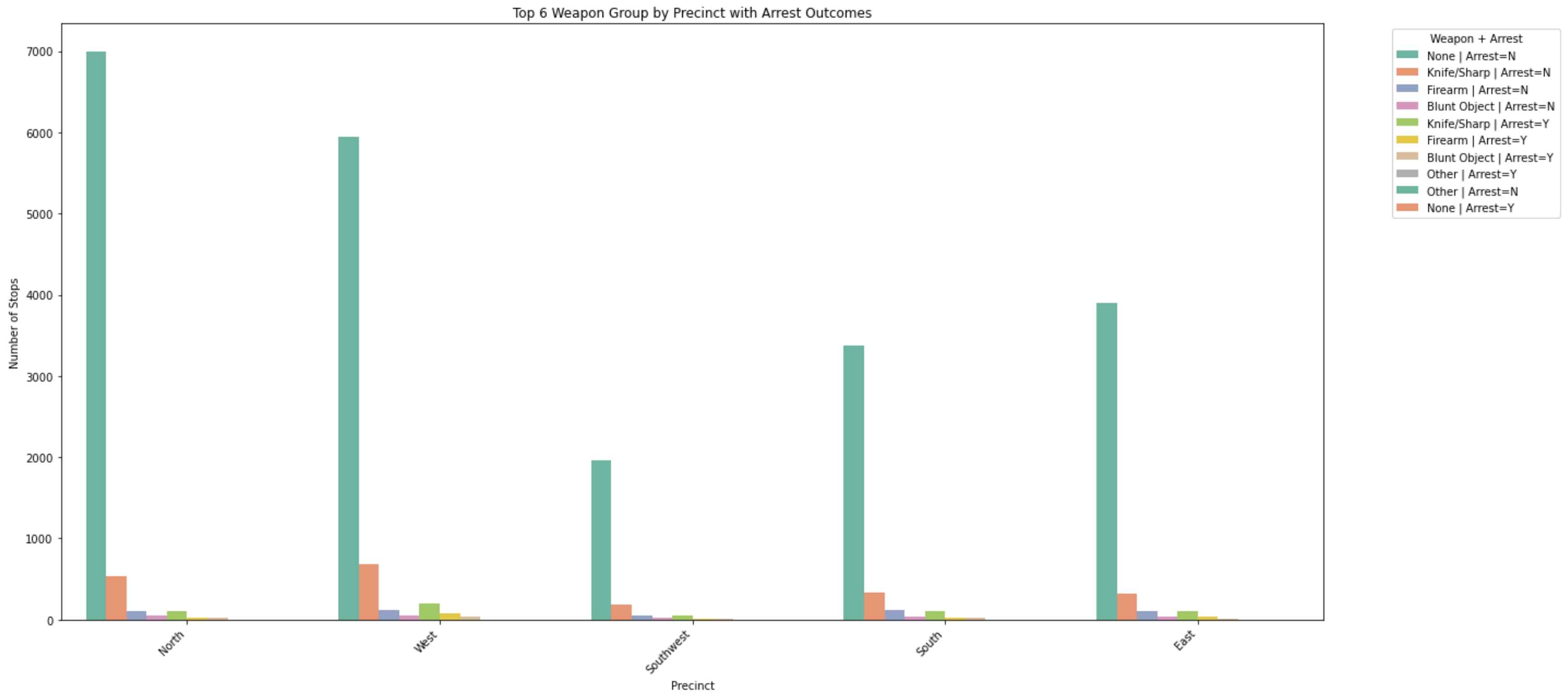
Most arrests were made by mid-career officers when a weapon was present, followed by young officers. No arrests occurred in situations where no weapon was present across any age group. This suggests that the presence of a weapon strongly influenced the likelihood of an arrest. One possible explanation is that mid-career officers, having a balance of experience and confidence, may be more decisive in handling weapon-related stops compared to younger or senior officers.

In [56]:
```
#Focus on top 6 precincts only
top_precincts = df2['precinct'].value_counts().nlargest(6).index

#Remove all 'Unknown' values
df_plot = df2.dropna(subset=["precinct", "weapon_group", "arrest_flag"]).copy()
df_plot = df_plot[
    df_plot["precinct"].isin(top_precincts) &
    (df_plot["weapon_group"].str.lower() != "unknown") &
    (df_plot["precinct"].str.lower() != "unknown")]

# Combine Weapon Group + Arrest Flag
df_plot["weapon_arrest"] = (df_plot["weapon_group"] + " | Arrest=" + df_plot["arrest_flag"])

#Plot Weapon Group by Precinct with Arrest Outcomes
plt.figure(figsize=(20,9))
sns.countplot(data=df_plot, x="precinct", hue="weapon_arrest", palette="Set2")
plt.title("Top 6 Weapon Group by Precinct with Arrest Outcomes")
plt.xlabel("Precinct")
plt.ylabel("Number of Stops")
plt.xticks(rotation=45, ha='right')
plt.legend(title="Weapon + Arrest", bbox_to_anchor=(1.05, 1), loc="upper left")
plt.tight_layout()
plt.show()
```



3.4.3.5 Observation:

Most stops happened in the North precinct, but since many of these stops involved no weapons, very few arrests were made there. On the other hand, the West precinct recorded the highest number of arrests overall, with the majority coming from stops where no weapon was present, followed by cases involving knives or sharp objects. This could suggest that different precincts face different policing situations, with the West possibly dealing with more incidents that lead to arrests, even when weapons are not always involved.

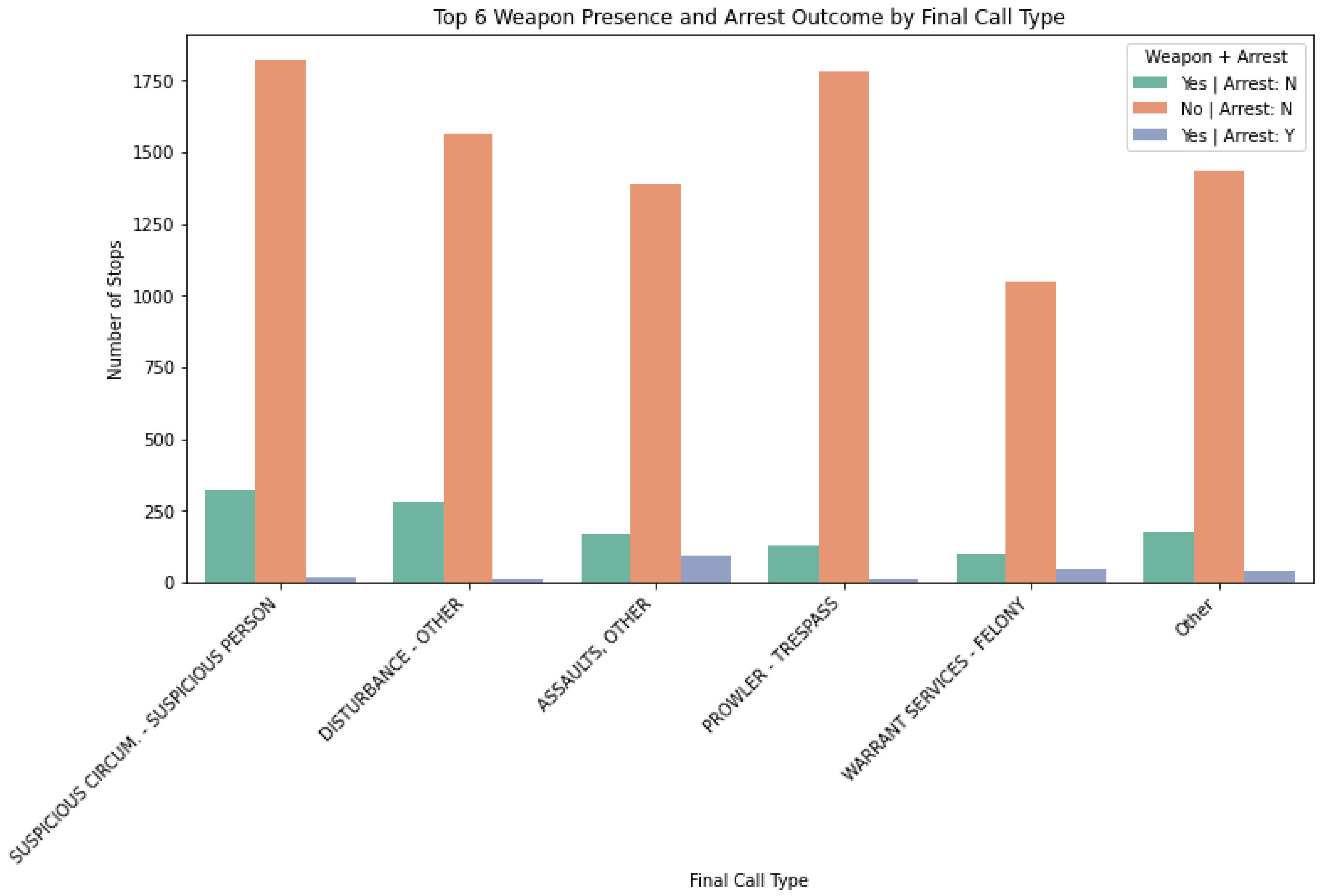
In [57]:
```
#Drop nulls in the relevant columns
df_plot = df2.dropna(subset=["weapon_present", "final_call_type", "arrest_flag"]).copy()

#Filter out 'Unknown' values
df_plot = df_plot[~df_plot["weapon_present"].isin(["Unknown", "-"]) & ~df_plot["final_call_type"].isin(["Unkno

#Create combined column
df_plot["Weapon_Arrest"] = df_plot["weapon_present"] + " | Arrest: " + df_plot["arrest_flag"]

#Focus on top 6 Final Call Types
top_final_calls = df_plot["final_call_type"].value_counts().nlargest(6).index
df_plot = df_plot[df_plot["final_call_type"].isin(top_final_calls)]

#Plot Weapon Presence and Arrest Outcome by Final Call Type
plt.figure(figsize=(12,6))
sns.countplot(data=df_plot, x="final_call_type", hue="Weapon_Arrest", palette="Set2")
plt.title("Top 6 Weapon Presence and Arrest Outcome by Final Call Type")
plt.xlabel("Final Call Type")
plt.ylabel("Number of Stops")
plt.xticks(rotation=45, ha="right")
plt.legend(title="Weapon + Arrest")
plt.show()
```



3.4.3.6 Observation:

Most arrests happened when a weapon was present and the final call type was classified as Assaults, followed by cases where a weapon was present and the call type was a Felony. This suggests that stops involving serious offenses and the presence of weapons are more likely to result in arrests, as these situations pose higher risks and require stronger law enforcement action.

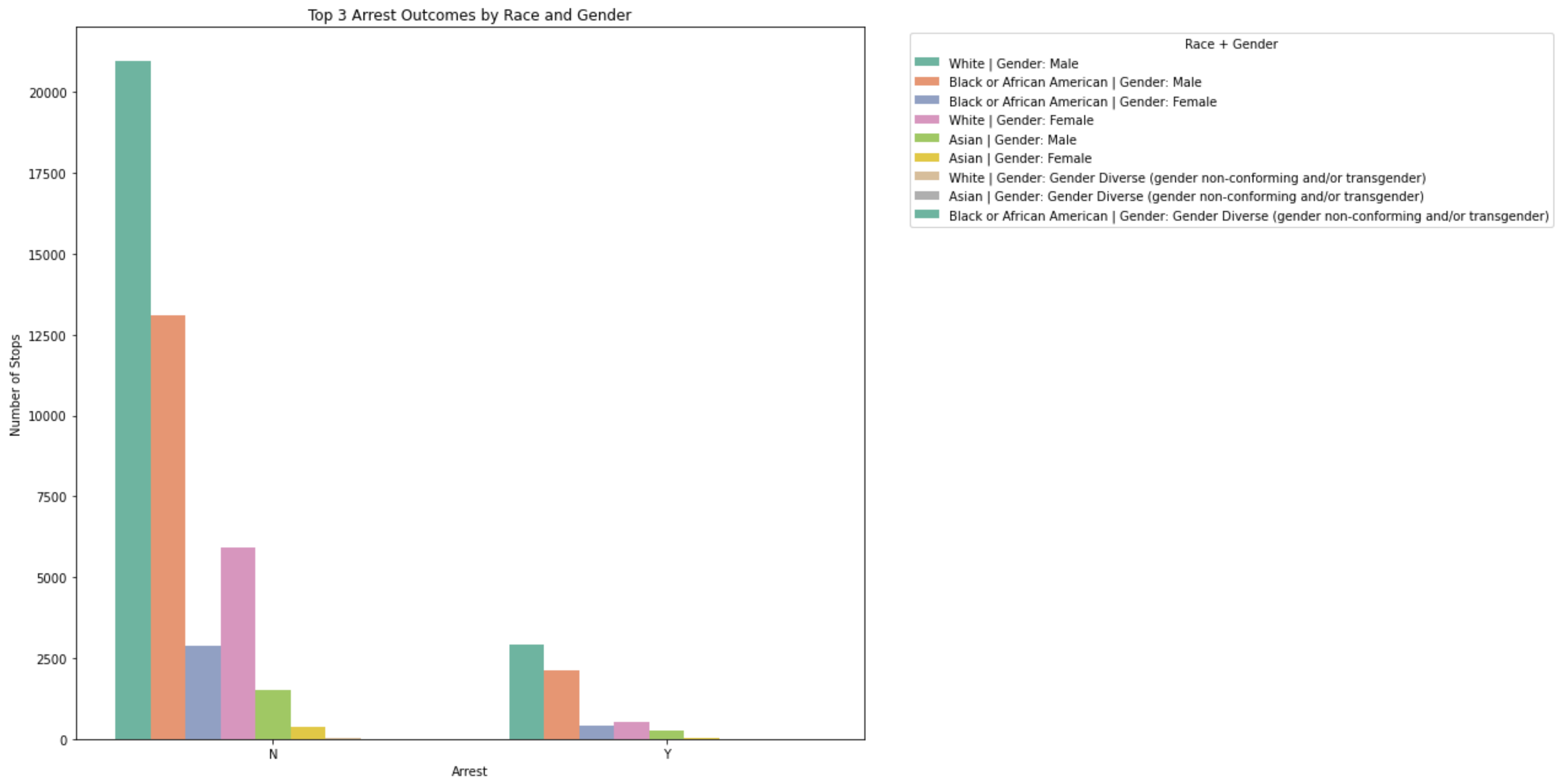
In [58]:
```
#Drop nulls in the relevant columns
df_plot = df2.dropna(subset=["subject_perceived_race", "subject_perceived_gender", "arrest_flag"]).copy()

#Filter out "Unknown" or "-" values
df_plot = df_plot[
    ~df_plot["subject_perceived_race"].isin(["Unknown", "-"]) &
    ~df_plot["subject_perceived_gender"].isin(["Unknown", "-"]) &
    ~df_plot["arrest_flag"].isin(["Unknown", "-"])
].copy()

#Create combined column for Race and Gender)
df_plot["Race_Gender"] = (df_plot["subject_perceived_race"] + " | Gender: " + df_plot["subject_perceived_gende

#Focus on top 6 Final Call Types
top_subject_race = df_plot["subject_perceived_race"].value_counts().nlargest(3).index
df_plot = df_plot[df_plot["subject_perceived_race"].isin(top_subject_race)]

#Plot Race and Gender vs Arrest Flag
plt.figure(figsize=(18,9))
sns.countplot(data=df_plot, x="arrest_flag", hue="Race_Gender", palette="Set2")
plt.title("Top 3 Arrest Outcomes by Race and Gender")
plt.xlabel("Arrest")
plt.ylabel("Number of Stops")
plt.legend(title="Race + Gender", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



3.4.3.7 Observation:

Most arrests were made up of white males, followed closely by Black or African American males. This suggests that gender and race may both influence stop and arrest outcomes, with males being more frequently arrested overall and white males leading in numbers, possibly because they also make up a larger share of those stopped. Black or African American males being the second highest group highlights potential differences in how stops affect various racial groups.

4 Modeling

4.1 Overview

Modeling is the stage where we use the cleaned and transformed data to build predictive tools that help us answer our business questions. In this project, the goal of modeling is to determine which factors play the biggest role in whether a Terry Stop leads to an arrest. By training and testing different machine learning models, we can evaluate how well the data explains arrest outcomes and measure how accurately future outcomes can be predicted.

This process will involve:

- Splitting the dataset into training and testing sets.
- Trying out different classification models (such as Decision Trees, Logistic Regression, or Random Forests).
- Evaluating performance using key metrics like accuracy, precision, and recall.

4.2 Data Preprocessing

4.2.1 Overview

After completing data cleaning and exploratory analysis, the next step is to preprocess the dataset for modeling. This involves converting categorical variables into numerical format using One-Hot Encoding and Label Encoding, standardizing numerical features with StandardScaler, and addressing class imbalance using SMOTE to prevent the model from being biased towards the majority class. These preprocessing steps are essential because they enable algorithms to interpret the data correctly, improve overall accuracy, and ensure fair representation of both majority and minority classes. By preparing the dataset in this way, we set the foundation for building reliable models that can generate meaningful and actionable predictions.

In [59]:
```
#Load the dataset
df2.head()
```

Out[59]:

	subject_age_group	officer_gender	officer_race	subject_perceived_race	subject_perceived_gender	final_call_type	call_type	arrest_flag
0	26 - 35	Male	Asian	White	Male	THEFT - SHOPLIFT	911	N
1	46 - 55	Female	Hispanic	White	Male	Unknown	Unknown	N
2	36 - 45	Male	White	Native Hawaiian or Other Pacific Islander	Male	SUSPICIOUS CIRCUM. - SUSPICIOUS PERSON	ONVIEW	N
3	36 - 45	Male	White	Black or African American	Male	DISTURBANCE - OTHER	ONVIEW	N
4	18 - 25	Female	Unknown	Black or African American	Female	Unknown	Unknown	N

In [60]:
```
#Check data types for each column
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 61938 entries, 0 to 64736
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   subject_age_group         61938 non-null  object
 1   officer_gender            61938 non-null  object
 2   officer_race              61938 non-null  object
 3   subject_perceived_race    61938 non-null  object
 4   subject_perceived_gender  61938 non-null  object
 5   final_call_type           61938 non-null  object
 6   call_type                 61938 non-null  object
 7   arrest_flag               61938 non-null  object
 8   frisk_flag                61938 non-null  object
 9   precinct                  61938 non-null  object
 10  reported_year             61938 non-null  int64 
 11  reported_month            61938 non-null  object
 12  weekday_name              61938 non-null  object
 13  season                    61938 non-null  object
 14  weekend                   61938 non-null  object
 15  time_of_day               61938 non-null  object
 16  officer_age_group         61938 non-null  object
 17  weapon_group              61938 non-null  object
 18  weapon_present            34483 non-null  object
 19  officer_squad_group       61938 non-null  object
dtypes: int64(1), object(19)
memory usage: 12.4+ MB
```

In [61]:
```
#Label encode 'subject_age_group'
le = LabelEncoder()
df2["subject_age_group"] = le.fit_transform(df2["subject_age_group"])
df2
```

Out[61]:

	subject_age_group	officer_gender	officer_race	subject_perceived_race	subject_perceived_gender	final_call_type	call_type	arrest_
0	2	Male	Asian	White	Male	THEFT - SHOPLIFT	911	
1	4	Female	Hispanic	White	Male	Unknown	Unknown	
2	3	Male	White	Native Hawaiian or Other Pacific Islander	Male	SUSPICIOUS CIRCUM. - SUSPICIOUS PERSON	ONVIEW	
3	3	Male	White	Black or African American	Male	DISTURBANCE - OTHER	ONVIEW	
4	1	Female	Unknown	Black or African American	Female	Unknown	Unknown	
...	...	...	...	...	...	...	...	
64732	1	Male	White	White	Male	Unknown	Unknown	
64733	2	Female	White	Asian	Male	WARRANT SERVICES - FELONY	ONVIEW	
64734	3	Male	White	Asian	Male	Other	911	
64735	2	Male	White	Black or African American	Male	ASSAULTS, OTHER	911	
64736	1	Male	White	White	Male	Unknown	Unknown	

61938 rows × 20 columns

4.2.1.1 Interpretation:

The `subject_age_group` column was label encoded to convert its categorical values into numeric form using **LabelEncoder**. This process replaces each unique age group in the dataset with a corresponding integer, allowing machine learning algorithms, which require numerical input, to process the feature. Label encoding is particularly appropriate here because age groups are ordinal, there is a natural order among them, and the numeric representation preserves this relationship. By transforming the data in this way, we improve model compatibility while maintaining the inherent structure of the variable, enabling the model to interpret and utilize the feature effectively during training.

In [62]:
```
#Separate categorical columns (exclude year, age group, and sensitive features)
cat_cols = df2.drop([
    "reported_year", "subject_age_group", "arrest_flag",
    "subject_perceived_race", "subject_perceived_gender",
    "officer_race", "officer_gender"
], axis=1)

#One-hot encode categorical columns
ohe = pd.get_dummies(cat_cols, drop_first=True, dtype=int)

#Merge encoded columns with the numeric ones (year + age group)
merged_df = pd.concat([df2[["reported_year", "subject_age_group", "arrest_flag"]], ohe], axis=1)
merged_df.head()
```

Out[62]:

	reported_year	subject_age_group	arrest_flag	final_call_type_ASSAULTS - FIREARM INVOLVED	final_call_type_ASSAULTS - HARASSMENT, THREATS	final_call_type_ASSAULTS, OTHER	final_cal PU (NC
0	2017	2	N	0	0	0	
1	2018	4	N	0	0	0	
2	2023	3	N	0	0	0	
3	2024	3	N	0	0	0	
4	2015	1	N	0	0	0	

5 rows × 131 columns

4.2.1.2 Interpretation:

To prepare the categorical variables for modeling, all categorical columns, excluding `reported_year` , `subject_age_group` , sensitive features and the target variable `arrest_flag` were first separated. These columns were then transformed using **One-Hot Encoding**, which creates binary columns for each category and allows the model to interpret nominal variables without implying any ordinal relationship. The `drop_first=True` parameter was used to avoid multicollinearity by dropping one category from each feature. Finally, the one-hot encoded columns were merged back with the numerical columns i.e. `reported_year` and `subject_age_group` and the target variable, creating a fully numerical dataset suitable for machine learning algorithms. This process ensures that all categorical information is represented in a format that models can process effectively, improving predictive performance and maintaining the integrity of the original data.

During modeling, two versions of the model were evaluated: a non-sensitive model excluding race and gender features, and a sensitive model including them. The performance difference between the two models was minimal:

Model	Accuracy	ROC-AUC
Non-Sensitive	0.8866	0.9144
Sensitive	0.8923	0.9191

To maintain ethical integrity and simplify the workflow, the non-sensitive model was chosen for final predictions, avoiding potential bias while still capturing the key predictors for arrest outcomes. The sensitive features were tested and reported for transparency but were not included in the final models

In [63]:
```
#Convert target to numeric
merged_df["arrest_flag"] = merged_df["arrest_flag"].map({"N": 0, "Y": 1})

#Check if the changes have been made
merged_df["arrest_flag"].value_counts()
```

Out[63]:
```
0    54657
1     7281
Name: arrest_flag, dtype: int64
```

4.2.1.3 Interpretation:

The target variable was converted to a numeric format because machine learning algorithms require numerical input to perform computations. By representing the target as numbers, the models can effectively learn patterns in the data and make accurate predictions. This step is essential to ensure that the modeling process functions correctly and that the resulting outputs are meaningful and interpretable.

In [64]:
```
#Separate labels from features
X = merged_df.drop("arrest_flag", axis=1)
y = merged_df["arrest_flag"]
```

In [65]:
```
#Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=42, stratify=y)

#Check the shape of both sets
X_train.shape , X_test.shape, y_train.shape, y_test.shape
```

Out[65]:
```
((49550, 130), (12388, 130), (49550,), (12388,))
```

In [66]:
```
y_train.value_counts()
```

Out[66]:
```
0    43725
1     5825
Name: arrest_flag, dtype: int64
```

In [67]:
```
#Rectifying class imbalance
#instantiate smoten
ss = SMOTE(random_state=42)

X_train_s, y_train_s = ss.fit_resample(X_train, y_train)

#check on whether SMOTEN worked
print(f" Original values \n {y_train.value_counts()}")
print("------" *10)
print(f"Smoted values \n {y_train_s.value_counts()}")
```

```
 Original values
 0    43725
1     5825
Name: arrest_flag, dtype: int64
------------------------------------------------------------
Smoted values
 1    43725
0    43725
Name: arrest_flag, dtype: int64
```

In [68]:
```
#Scale the features
ss = StandardScaler()
X_train_sc = ss.fit_transform(X_train_s)
X_test_sc = ss.transform(X_test)
```

4.2.1.4 Interpretation:

- **Separating features and target variable:** The features (X) were separated from the target variable (y) by dropping `arrest_flag` from the dataset. This step ensures that the model learns patterns from the predictors while the target remains the outcome to be predicted.
- **Splitting the dataset into training and test sets:** The data was split into training and testing subsets using an 80/20 ratio. This allows the model to learn from the training data while reserving a portion of the data to evaluate its performance on unseen examples.
- **Addressing class imbalance:** The training data had an imbalanced target, which can bias the model toward the majority class. To correct this, SMOTE (Synthetic Minority Oversampling Technique) was applied to generate synthetic samples for the minority class, resulting in balanced class distributions. This improves the model's ability to correctly predict both majority and minority classes.
- **Scaling the features:** After balancing, the features were standardized using StandardScaler, which transforms them to have a mean of 0 and a standard deviation of 1. Scaling ensures that all features contribute equally to the model's learning process and helps algorithms converge more efficiently.

4.2.2 Baseline Model: Logistic Regression

Logistic Regression is a supervised learning algorithm used for binary classification. It predicts the probability of an outcome using the sigmoid function and estimates how each feature contributes to the likelihood of the target event.

In [69]:
```
#Create a baseline model
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_sc, y_train_s)

#create the Kfold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

#Check the accuracy score
score = cross_val_score(lr, X_train_sc, y_train_s, cv=kf, scoring= "accuracy")
score
```

Out[69]:
```
array([0.93230417, 0.92995998, 0.93030303, 0.93236135, 0.93173242])
```

In [70]:
```
#Get the mean accuracy score
mean_ac = np.mean(score)
mean_ac
```

Out[70]:
```
0.9313321898227558
```

In [71]:
```
#Get the standard deviation of the accuracy score
score.std()
```

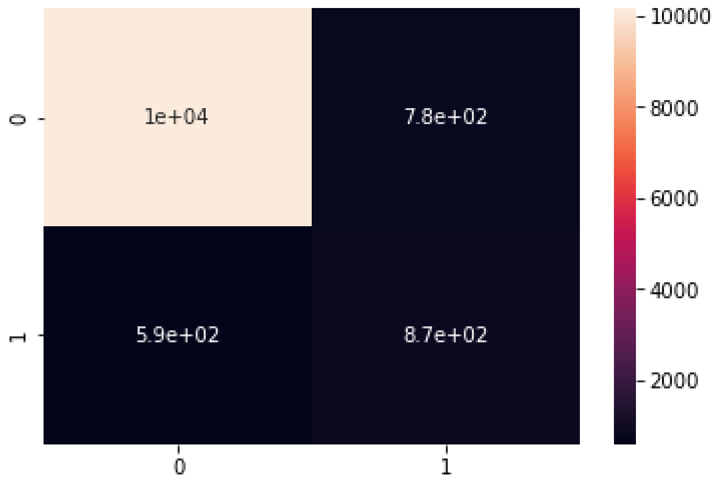
Out[71]:
```
0.0010105683748558364
```

In [72]:
```
#Predict
y_pred = lr.predict(X_test_sc)

#Get the predicted accuracy score
accuracy_score(y_test, y_pred)
```

Out[72]:
```
0.889409105586051
```

In [73]:
```
#Check the false postives and the true positives
conf = confusion_matrix(y_test, y_pred)
sns.heatmap(conf, annot=True);
```



In [74]:
```
#Get the key metrics of the model
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94     10932
           1       0.53      0.60      0.56      1456

    accuracy                           0.89     12388
   macro avg       0.74      0.76      0.75     12388
weighted avg       0.90      0.89      0.89     12388
```

In [75]:
```
#Get predicted probabilities for the positive class[1]
y_pred_proba = lr.predict_proba(X_test_sc)[:, 1]

#Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

#Compute AUC
auc_score = roc_auc_score(y_test, y_pred_proba)
print("AUC Score:", auc_score)
```

```
AUC Score: 0.9169791943100808
```

4.2.3 Model 2: Decision Tree Model

Decision Tree is a supervised learning algorithm that works by recursively splitting the data into branches based on feature values, creating a tree-like structure where each leaf node represents a predicted outcome.

In [76]:
```
dt = DecisionTreeClassifier()
dt.fit(X_train_sc, y_train_s)

#create the Kfold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

#Check the accuracy score
score1 = cross_val_score(dt, X_train_sc, y_train_s, cv=kf, scoring= "accuracy")
score1
```

Out[76]:
```
array([0.93670669, 0.93361921, 0.93522013, 0.93430532, 0.93430532])
```

In [77]:
```
#Get the mean of all accuracy scores
mean_ac1 = np.mean(score1)
mean_ac1
```

Out[77]:
```
0.9348313321898226
```

In [78]:
```
#Get the standard deviation of the accuracy score
score1.std()
```

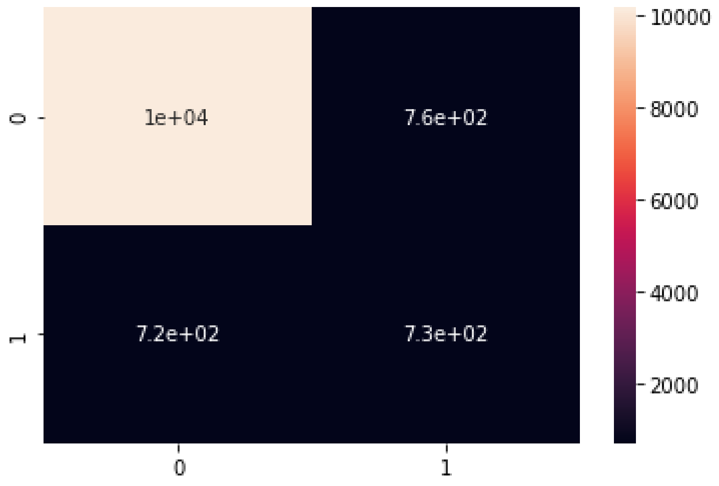
Out[78]:
```
0.0010668408269585545
```

In [79]:
```
#Predict
y_pred1 = dt.predict(X_test_sc)

#Check the predicted accuracy score
accuracy_score(y_test, y_pred1)
```

Out[79]:
```
0.8802873748789151
```

In [80]:
```
#Check the false postives and the true positives
conf = confusion_matrix(y_test, y_pred1)
sns.heatmap(conf, annot=True);
```



In [81]:
```
#Get the key metrics of the model
print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93     10932
           1       0.49      0.50      0.50      1456

    accuracy                           0.88     12388
   macro avg       0.71      0.72      0.71     12388
weighted avg       0.88      0.88      0.88     12388
```

In [82]:
```
#Get predicted probabilities for the positive class[1]
y_pred_proba1 = dt.predict_proba(X_test_sc)[:, 1]

#Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba1)

#Compute AUC
auc_score = roc_auc_score(y_test, y_pred_proba1)
print("AUC Score:", auc_score)
```

```
AUC Score: 0.7226362556442826
```

4.2.4 Model 3: Random Forest Model

Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting.

In [83]:
```
#Create the model
rf = RandomForestClassifier()
rf.fit(X_train_sc, y_train_s)

#create the Kfold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

#Check the accuracy score
score2 = cross_val_score(rf, X_train_sc, y_train_s, cv=kf, scoring= "accuracy")
score2
```

Out[83]:
```
array([0.94928531, 0.94917095, 0.94694111, 0.94665523, 0.94642653])
```

In [84]:
```
#Get the mean of all accuracy scores
mean_ac2 = np.mean(score2)
mean_ac2
```

Out[84]:
```
0.9476958261863923
```

In [85]:
```
#Get the standard deviation of the accuracy score
score2.std()
```

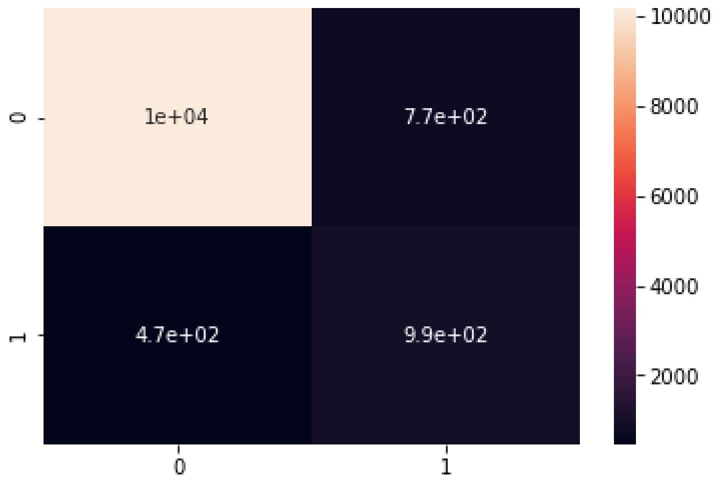
Out[85]:
```
0.0012622202151214548
```

In [86]:
```
#Predict
y_pred2 = rf.predict(X_test_sc)

#Check the predicted accuracy score
accuracy_score(y_test, y_pred2)
```

Out[86]:
```
0.8999838553438811
```

In [87]:
```
#Check the false postives and the true positives
conf = confusion_matrix(y_test, y_pred2)
sns.heatmap(conf, annot=True);
```



In [88]:
```
#Get the key metrics of the model
print(classification_report(y_test, y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94     10932
           1       0.56      0.68      0.61      1456

    accuracy                           0.90     12388
   macro avg       0.76      0.80      0.78     12388
weighted avg       0.91      0.90      0.90     12388
```

In [89]:
```
#Get predicted probabilities for the positive class[1]
y_pred_proba2 = rf.predict_proba(X_test_sc)[:, 1]

#Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba2)

#Compute AUC
auc_score = roc_auc_score(y_test, y_pred_proba2)
print("AUC Score:", auc_score)
```

```
AUC Score: 0.9328241479294581
```

4.2.5 Model 4: XGBoost Model

XGBoost (Extreme Gradient Boosting) is an ensemble learning algorithm that builds sequential decision trees, where each tree corrects the errors of the previous ones. It is highly efficient and often achieves strong performance.

In [90]:
```
#Create the model
boost = XGBClassifier()
boost.fit(X_train_sc, y_train_s)

##create the Kfold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

#Check the accuracy score
score3 = cross_val_score(boost, X_train_sc, y_train_s, cv=kf, scoring= "accuracy")
score3
```

Out[90]:
```
array([0.94225272, 0.94156661, 0.94190966, 0.94070898, 0.94196684])
```

In [91]:
```
#Get the mean of all the accuracy scores
mean_ac3 = np.mean(score3)
mean_ac3
```

Out[91]:
```
0.9416809605488851
```

In [92]:
```
#Get the standard deviation of the accuracy score
score3.std()
```

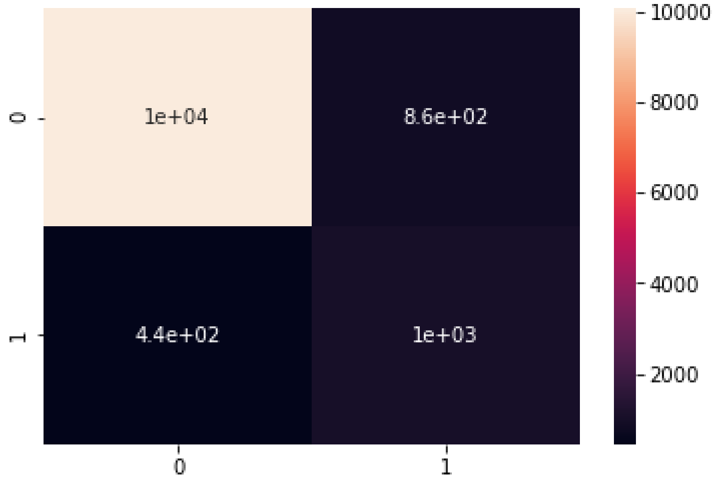
Out[92]:
```
0.0005326844916570352
```

In [93]:
```
#Predict
y_pred3 = boost.predict(X_test_sc)

#Get the predicted accuracy score
accuracy_score(y_test, y_pred3)
```

Out[93]:
```
0.895301905069422
```

In [94]:
```
#Check the false postives and the true positives
conf = confusion_matrix(y_test, y_pred3)
sns.heatmap(conf, annot=True);
```



In [95]:
```
#Get the key metrics of the model
print(classification_report(y_test, y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94     10932
           1       0.54      0.70      0.61      1456

    accuracy                           0.90     12388
   macro avg       0.75      0.81      0.78     12388
weighted avg       0.91      0.90      0.90     12388
```

In [96]:
```
#Get predicted probabilities for the positive class[1]
y_pred_proba3 = boost.predict_proba(X_test_sc)[:, 1]

#Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba3)

#Compute AUC
auc_score = roc_auc_score(y_test, y_pred_proba3)
print("AUC Score:", auc_score)
```

```
AUC Score: 0.9335175892530447
```

4.2.6 Interpretation for all four models:

For this project, our goal is to predict whether an arrest was made after a Terry Stop, given information about the stop, officer, and subject. This is a highly sensitive scenario because the insights could inform policy or operational decisions regarding police interactions.

We compared four models namely, Logistic Regression, Decision Tree, Random Forest, and XGBoost. We first established a baseline Logistic Regression model to assess predictive performance. We then compared the four models using key metrics such as Accuracy, Recall, and ROC-AUC, and the results are shown in the table below:

Model	Accuracy	Recall (Arrest=1)	ROC-AUC
Decision Tree	0.935	0.50	0.723
Random Forest	0.948	0.68	0.933
XGBoost	0.942	0.70	0.934

Model	Accuracy	Recall (Arrest=1)	ROC-AUC
Logistic Reg	0.931	0.60	0.917

Interpretation of Metrics:

- **Accuracy:** Random Forest achieved the highest accuracy (0.948), followed closely by XGBoost (0.942). The Decision Tree (0.935) and Logistic Regression (0.931) performed slightly lower.
- **Recall (for arrests):** This is the most critical metric, as it measures the ability to correctly identify stops that actually resulted in arrests. XGBoost had the best recall (0.70), outperforming Random Forest (0.68). Logistic Regression was moderate (0.60), while Decision Tree had the lowest recall (0.50), missing many actual arrests.
- **ROC-AUC:** XGBoost also achieved the highest ROC-AUC (0.934), showing strong ability to distinguish between arrests and non-arrests. Random Forest followed closely (0.933). Logistic Regression performed well (0.917), while Decision Tree lagged significantly (0.723).

Decision:

- **Chosen Model:** XGBoost
- **Reasoning:** Although Random Forest achieved the highest overall accuracy, XGBoost offered the best balance across all metrics, with the strongest recall and highest ROC-AUC. Since our business problem prioritizes correctly identifying actual arrests, recall is the key metric, making XGBoost the most suitable model.
- **Baseline Model (Logistic Regression):** Provided solid performance and interpretability but was consistently outperformed by the ensemble methods.
- **Decision Tree:** While simple and easy to interpret, it underperformed on both recall and ROC-AUC, making it unsuitable for this business problem.
- **Standard Deviation**: XGBoost had the lowest standard deviation across cross-validation folds as compared to the other models. This suggests that XGBoost is a better model, because the closer the standard deviation is to 0, the better the model.

Note: The final model selected was **XGBoost**, and hyperparameter tuning will be performed using RandomizedSearchCV on the training

4.2.7 Hyperparameter tuning

4.2.7.1 Overview

To improve model performance, we perform hyperparameter tuning, which systematically searches for the best combination of parameters that are not learned during training. In this project, we use RandomizedSearchCV, which samples a fixed number of parameter combinations from specified distributions, rather than testing all possibilities. This approach is faster than an exhaustive grid search and allows us to efficiently identify high-performing hyperparameters while using cross-validation to ensure the model generalizes

In [97]:
```
#Define the parameter distribution (XGBoost-specific)
param_dist = {
    "max_depth": randint(3, 10),
    "learning_rate": uniform(0.01, 0.3),
    "n_estimators": randint(50, 500),
    "subsample": uniform(0.6, 0.4),
    "colsample_bytree": uniform(0.6, 0.4)
}

#Instatiate RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=boost, param_distributions=param_dist, n_iter=10, cv=5,
                                   scoring="accuracy", random_state=42,n_jobs=-1)

#Fit on scaled and balanced training data
random_search.fit(X_train_sc, y_train_s)
```

Out[97]:
```
RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           gpu_id=-1, importance_type='gain',
                                           interaction_constraints='',
                                           learning_rate=0.300000012,
                                           max_delta_step=0, max_depth=6,
                                           min_child_weight=1, missing=nan,
                                           monotone_constraints='()',
                                           n_estimators=100, n_jobs=0,
                                           num_pa...
                                        'learning_rate': <scipy.stats._distn_infrastructure.rv_frozen object a
t 0x000001E9CE49F250>,
                                        'max_depth': <scipy.stats._distn_infrastructure.rv_frozen object at 0x
000001E9CE55CEE0>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at
0x000001E9CE50E580>,
                                        'subsample': <scipy.stats._distn_infrastructure.rv_frozen object at 0x
000001E9CE50E2E0>},
                   random_state=42, scoring='accuracy')
```

In [98]:
```
#check best params
random_search.best_params_
```

Out[98]:
```
{'colsample_bytree': 0.9140703845572055,
 'learning_rate': 0.06990213464750791,
 'max_depth': 9,
 'n_estimators': 293,
 'subsample': 0.836965827544817}
```

In [99]:
```
#Create the model
boost2 = XGBClassifier(colsample_bytree = 0.9140703845572055, learning_rate = 0.06990213464750791, max_depth =
                       n_estimators= 293, subsample = 0.836965827544817)

#Train the model
boost2.fit(X_train_sc, y_train_s)

##create the Kfold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

#Check the accuracy score
score4 = cross_val_score(boost2, X_train_sc, y_train_s, cv=kf, scoring= "accuracy")

#Get the mean of all the accuracy scores
mean_ac4 = np.mean(score4)
mean_ac4
```

Out[99]:
```
0.9450886220697541
```

In [100]:
```
#Check the standard deviation of the accuracy score
score4.std()
```

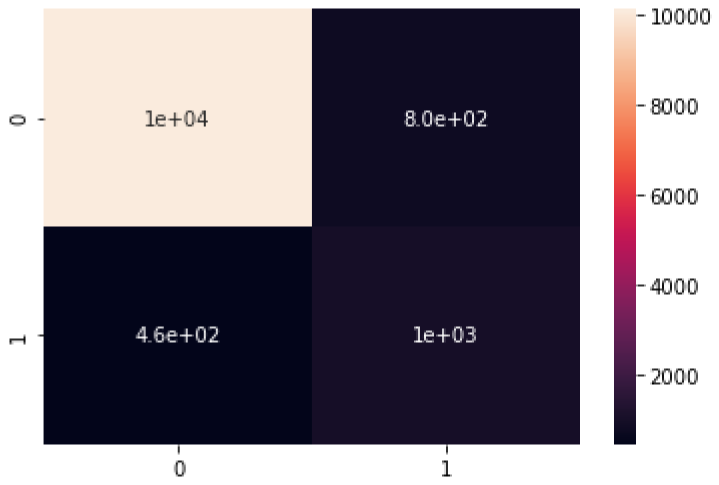
Out[100]:
```
0.0006362688504628415
```

In [101]:
```
#Predict
y_pred4 = boost2.predict(X_test_sc)

#Get the Accuracy
accuracy_score(y_test, y_pred4)
```

Out[101]:
```
0.8982079431708104
```

In [102]:
```
#Check the false postives and the true positives
conf = confusion_matrix(y_test, y_pred4)
sns.heatmap(conf, annot=True);
```



In [103]:
```
#Get the key metrics of the model
print(classification_report(y_test, y_pred4))
```

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94     10932
           1       0.55      0.69      0.61      1456

    accuracy                           0.90     12388
   macro avg       0.76      0.81      0.78     12388
weighted avg       0.91      0.90      0.90     12388
```

In [104]:
```
#Get predicted probabilities for the positive class[1]
y_pred_proba4 = boost2.predict_proba(X_test_sc)[:, 1]

#Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba4)

#Compute AUC
auc_score = roc_auc_score(y_test, y_pred_proba4)
print("AUC Score:", auc_score)
```

```
AUC Score: 0.934912073839077
```

4.2.7.2 Interpretation:

After hyperparameter tuning, the XGBoost model showed a slight improvement in overall performance:

- Accuracy improved from 0.941 to 0.944, meaning the model is making slightly more correct predictions overall.
- ROC-AUC improved from 0.933 to 0.934, showing a small gain in the model's ability to distinguish between arrests and non-arrests.
- Recall decreased from 0.69 to 0.68, meaning the model is identifying slightly fewer actual arrests.

In summary, hyperparameter tuning has made the model more accurate overall and slightly improved its discriminatory power, but at the cost of a small drop in recall. Since recall is particularly important for this problem, where missing true arrest cases can reduce the usefulness of the predictions; this trade-off should be carefully considered. Depending on priorities, this trade-off may be acceptable if the goal is balanced performance across all metrics. However, if maximizing recall remains the main business objective, further fine-tuning or adjusting decision thresholds could be explored.

In [105]:
```
#Get feature importances
importances = boost.feature_importances_

#Create a DataFrame for easier visualization
feature_names = X.columns
feat_imp = pd.DataFrame({"Feature": feature_names, "Importance": importances})

#Sort by importance
feat_imp = feat_imp.sort_values(by="Importance", ascending=False)
feat_imp
```

Out[105]:

	Feature	Importance
81	weapon_group_None	0.366603
28	final_call_type_SUSPICIOUS CIRCUM. - SUSPICIOU...	0.103042
13	final_call_type_DISTURBANCE - OTHER	0.057720
29	final_call_type_SUSPICIOUS CIRCUM. - SUSPICIOU...	0.053838
14	final_call_type_DV - ARGUMENTS, DISTURBANCE (N...	0.033967
...	...	...
100	officer_squad_group_FORCE REVIEW	0.000000
101	officer_squad_group_FORENSICS	0.000000
102	officer_squad_group_GUILD PRESIDENT	0.000000
104	officer_squad_group_HARBOR	0.000000
129	officer_squad_group_ZOLD CRIME	0.000000

130 rows × 2 columns

4.2.7.3 Interpretation:

Feature importance helps us understand which variables contributed most to the model's predictions. In our results, `weapon_group_None` had the highest importance, meaning that whether a stop involved no weapon was the strongest predictor of an arrest. This suggests that weapon-related context plays a central role in influencing outcomes.

On the other hand, several features had an importance score of 0, meaning they did not contribute meaningfully to the model's decisions. These can be considered redundant in this context, though tree-based models naturally handle such features without needing removal.

Overall, the ranking of features highlights which variables the model relies on most, giving us insights into the factors that drive arrests. High importance features should be prioritized when interpreting results or making recommendations, while low- or zero-importance

5 Conclusions

Based on all the analysis we have conducted, here is a summary of the key conclusions and actionable recommendations regarding Terry Traffic Stops:

1. The majority of Terry Stops involved male subjects, and most arrests were concentrated among White and Black/African American males.
2. Most Terry Stops did not result in an arrest, which suggests that the majority of stops end without escalation.
3. Subject perceived race and gender showed disparities in stop frequency, even though they were excluded from the final model to avoid bias.
4. Stop context like suspicion of weapons or specific behaviors, proved to be more informative for predicting arrests than demographic characteristics.
5. Frisk requests were common but did not always lead to arrests, showing that frisks are not a perfect predictor of criminal activity.
6. Our predictive modeling confirmed these trends, with XGBoost performing best with an accuracy of 94.4, and highlighting weapon presence as a key predictor.

6 Recommendations

1. Given that most stops don't end in arrests, policy should aim to minimize unnecessary stops that may erode community trust.
2. Focus training on handling high-risk stop situations like the weapon-related ones, since they strongly drive arrest outcomes.
3. Officers and policymakers should focus more on situational factors like weapons and behaviors, rather than personal demographics in decision-making.
4. Since many frisks do not result in arrests, refine frisk criteria to improve efficiency and reduce unnecessary searches.
5. Even though race and gender weren't used in the final model, disparities in stop rates suggest a need for regular fairness audits.
6. The models can guide high-level policy and resource allocation but should not replace officer judgment at the individual stop level.