Classification Project - A study of Terry Stops in Seattle, WA

Facts of the case of Terry vs Ohio

> The Fourth Amendment of the U.S. Constitution provides that
>
> > [t]he right of the people to be secure in their persons, houses, papers, and effects, against unreasonable searches and seizures, shall not be violated, and no Warrants shall issue, but upon probable cause, supported by Oath or affirmation, and particularly describing the place to be searched, and the persons or things to be seized.
>
> The ultimate goal of this provision is to protect people's right to privacy and freedom from unreasonable intrusions by the government. However, the Fourth Amendment does not guarantee protection from all searches and seizures, but only those done by the government and deemed unreasonable under the law.

[law.cornell.edu](https://www.law.cornell.edu/wex/fourth_amendment (https://www.law.cornell.edu/wex/fourth_amendment) "Cornell Law)

> In the case of Terry vs Ohio, decided on June 10, 1968, by the United States Supreme Court, Terry and two other men were observed by a plain clothes policeman in what the officer believed to be "casing a job, a stick-up." The officer stopped and frisked the three men, and found weapons on two of them. Terry was convicted of carrying a concealed weapon and sentenced to three years in jail.
>
> In an 8-to-1 decision, the Court held that the search undertaken by the officer was reasonable under the Fourth Amendment and that the weapons seized could be introduced into evidence against Terry. Attempting to focus narrowly on the facts of this particular case, the Court found that the officer acted on more than a "hunch" and that "a reasonably prudent man would have been warranted in believing [Terry] was armed and thus presented a threat to the officer's safety while he was investigating his suspicious behavior." The Court found that the searches undertaken were limited in scope and designed to protect the officer's safety incident to the investigation.

Oyez.org (https://www.oyez.org/cases/1967/67)

> A Terry stop in the United States allows the police to briefly detain a person based on reasonable suspicion of involvement in criminal activity. Reasonable suspicion is a lower standard than probable cause which is needed for arrest. When police stop and search a pedestrian, this is commonly known as a stop and frisk. When police stop an automobile, this is known as a traffic stop. If the police stop a motor vehicle on minor infringements in order to investigate other suspected criminal activity, this is known as a pretextual stop.

Wikipedia (https://en.wikipedia.org/wiki/Terry_stop#:~:text=A%20Terry%20stop%20in%20the,as%20a%20stop%20and%20frisk)

> A terry stop is another name for stop and frisk; the name was generated from the U.S Supreme Court case Terry v. Ohio. When a police officer has a reasonable suspicion that an individual is armed, engaged, or about to be engaged, in criminal conduct, the officer may briefly stop and detain an individual for a pat-down search of outer clothing. A Terry stop is a seizure within the meaning of Fourth Amendment.
>
> In a traffic stop setting, the Terry condition of a lawful investigatory stop is met whenever it is lawful for the police to detain an automobile and its occupants pending inquiry into a vehicular violation. The police do not need to believe that any occupant of the vehicle is involved in criminal activity.
>
> In a recent case, Floyd v. City of New York 813 F. Supp.2d 417 (2011), the court held the New York stop-and-frisk policy violated the Fourth Amendment because it rendered stop and frisks more frequent for blacks and Hispanics.

Legal Information Institute (https://www.law.cornell.edu/wex/terry_stop/stop_and_frisk)

Dataset information

This data represents records of police reported stops under Terry v. Ohio, 392 U.S. 1 (1968). The dataset was created on 04/12/2017 and first published on 05/22/2018 and is provided by the city of Seattle, WA.

Each row represents a unique stop. Each record contains perceived demographics of the subject, as reported by the officer making the stop and officer demographics as reported to the Seattle Police Department, for employment purposes.

Where available, data elements from the associated Computer Aided Dispatch (CAD) event (e.g. Call Type, Initial Call Type, Final Call Type) are included.

There are 45,317 rows and 23 variables:

- ***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 were 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 perceived 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.

Some ideas for exploration:

1. How does the probability of arrest vary by categories of different demographic variables?
2. Which variables are the strongest predictors of arrest for this dataset?
3. NOTE - these models cannot be used to predict arrest outside of the actual data recorded, as the model would build in and perpetuate any inherent bias of the officers

Data cleaning and EDA

Import and basic info

Output - terry

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
from user_functions import *
from datetime import datetime
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression, Ridge, Lasso, LassoCV, LassoLarsCV, LassoLarsIC, L
from sklearn.metrics import r2_score, mean_squared_error, mean_squared_log_error, plot_confusion_matr
from sklearn.model_selection import train_test_split, cross_val_score, KFold, GridSearchCV
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler, OneHotEncoder, StandardScaler, sc
from sklearn import metrics
from sklearn.feature_selection import VarianceThreshold, f_regression, mutual_info_regression, Select
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.neighbors import KNeighborsClassifier
from collections import defaultdict
from sklearn.naive_bayes import GaussianNB, ComplementNB, MultinomialNB, BernoulliNB, CategoricalNB
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, AdaBoostClassifier, GradientB
from xgboost import XGBClassifier
```

In [2]:

```
# Set default visualization parameters

CB91_Blue = '#2CBDFE'
CB91_Green = '#47DBCD'
CB91_Pink = '#F3A0F2'
CB91_Purple = '#9D2EC5'
CB91_Violet = '#661D98'
CB91_Amber = '#F5B14C'
color_list = [CB91_Blue, CB91_Pink, CB91_Green, CB91_Amber, CB91_Purple, CB91_Violet]
plt.rcParams['axes.prop_cycle'] = plt.cycler(color=color_list)
sns.set_context("notebook", rc={"font.size":16, "axes.titlesize":20, "axes.labelsize":18})
sns.set(font='Franklin Gothic Book',
rc={'axes.axisbelow': False,
'axes.edgecolor': 'lightgrey',
# 'axes.edgecolor': 'white',
'axes.facecolor': 'None',
'axes.grid': False,
'axes.labelcolor': 'dimgrey',
# 'axes.labelcolor': 'white',
'axes.spines.right': False,
'axes.spines.top': False,
'axes.prop_cycle': plt.cycler(color=color_list),
'figure.facecolor': 'white',
'lines.solid_capstyle': 'round',
'patch.edgecolor': 'w',
'patch.force_edgecolor': True,
'text.color': 'dimgrey',
# 'text.color': 'white',
'xtick.bottom': False,
'xtick.color': 'dimgrey',
# 'xtick.color': 'white',
'xtick.direction': 'out',
'xtick.top': False,
'ytick.color': 'dimgrey',
# 'ytick.color': 'white',
'ytick.direction': 'out',
'ytick.left': False,
'ytick.right': False})
%matplotlib inline
```

In [3]:
```
# Some null entries in Officer Squad, 45317 records
terry = pd.read_csv('Terry_Stops.csv')
terry.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45317 entries, 0 to 45316
Data columns (total 23 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Subject Age Group        45317 non-null  object
 1   Subject ID               45317 non-null  int64 
 2   GO / SC Num              45317 non-null  int64 
 3   Terry Stop ID            45317 non-null  int64 
 4   Stop Resolution          45317 non-null  object
 5   Weapon Type              45317 non-null  object
 6   Officer ID               45317 non-null  object
 7   Officer YOB              45317 non-null  int64 
 8   Officer Gender           45317 non-null  object
 9   Officer Race             45317 non-null  object
 10  Subject Perceived Race   45317 non-null  object
 11  Subject Perceived Gender 45317 non-null  object
 12  Reported Date            45317 non-null  object
 13  Reported Time            45317 non-null  object
 14  Initial Call Type        45317 non-null  object
 15  Final Call Type          45317 non-null  object
 16  Call Type                45317 non-null  object
 17  Officer Squad            44714 non-null  object
 18  Arrest Flag              45317 non-null  object
 19  Frisk Flag               45317 non-null  object
 20  Precinct                 45317 non-null  object
 21  Sector                   45317 non-null  object
 22  Beat                     45317 non-null  object
dtypes: int64(4), object(19)
memory usage: 8.0+ MB
```

In [4]:
```
terry['Arrest Flag'].value_counts()
```

Out[4]:
```
N    42585
Y     2732
Name: Arrest Flag, dtype: int64
```

In [5]:
```
# Import the data, data appears to be sorted by GO/SC Number, in which the 1st four numbers are the y
# Or possibly Terry Stop ID but I do see some out of order toward the end
# Perhaps it is sorted on Reported Date which I cannot see, or not really sorted
terry.tail(10)
```

Out[5]:

	Subject Age Group	Subject ID	GO / SC Num	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Off R
45307	56 and Above	16366865981	20200000304239	16366787296	Field Contact	-	8758	1995	M	W
45308	56 and Above	16689873637	20200000306763	16688826004	Field Contact	-	8308	1987	M	Hispa or La
45309	56 and Above	16700206747	20200000329461	18068162403	Field Contact	-	8704	1988	M	W
45310	56 and Above	16834950496	20200000307517	16833080841	Field Contact	-	7723	1987	M	W
45311	56 and Above	17560315357	20200000316857	17560309580	Field Contact	-	8646	1996	M	W
45312	56 and Above	17705067875	20200000321463	17722624502	Arrest	-	8486	1992	M	As
45313	56 and Above	18018113199	20200000327585	18018069307	Field Contact	-	8668	1990	F	W
45314	56 and Above	18036883066	20200000328353	18036796582	Field Contact	-	8747	1991	M	W
45315	56 and Above	18763121119	20200000334915	18760675122	Field Contact	-	7456	1979	M	W
45316	56 and Above	19145427342	20200000345283	19145423883	Field Contact	Knife/Cutting/Stabbing Instrument	8646	1996	M	W

10 rows × 23 columns

In [6]:
```
# Only mostly sorted on Date, some out of order
terry['Reported Date'][45300:45318]
```

Out[6]:
```
45300    2020-10-08T00:00:00
45301    2020-10-15T00:00:00
45302    2020-10-18T00:00:00
45303    2020-10-19T00:00:00
45304    2020-10-19T00:00:00
45305    2020-10-19T00:00:00
45306    2020-10-28T00:00:00
45307    2020-10-26T00:00:00
45308    2020-10-29T00:00:00
45309    2020-11-26T00:00:00
45310    2020-10-30T00:00:00
45311    2020-11-11T00:00:00
45312    2020-11-17T00:00:00
45313    2020-11-24T00:00:00
45314    2020-11-25T00:00:00
45315    2020-12-03T00:00:00
45316    2020-12-15T00:00:00
Name: Reported Date, dtype: object
```

In [7]:
```
terry.isna().sum()
```

Out[7]:
```
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              603
Arrest Flag                  0
Frisk Flag                   0
Precinct                     0
Sector                       0
Beat                         0
dtype: int64
```

Analyze 'Subject Age Group'

Changed age group of '-' to 'Unknown'.
All of these values occur at the beginning of the dataset, so they may have more to do with WHEN they were recorded.

In [8]:
```
# Starting analysis of first variable, Subject Age Group
# Information is nicely binned into groups, presumably this is perceived age unless the subject provi
# Will change '-' values to unknown
terry['Subject Age Group'].value_counts()
```

Out[8]:
```
26 - 35         15054
36 - 45          9557
18 - 25          9169
46 - 55          5852
56 and Above     2301
1 - 17           1935
-                1449
Name: Subject Age Group, dtype: int64
```

In [9]:
```
# Is this a coincidence?  The unknown age records are 1449 of the first 1459 records.
# Probably they didn't start keeping track of Subject Age right away
# If the unknown ages appear relevant, it may have more to do with the date than the actual subject's
init_terry=terry[:1459]
init_terry[init_terry['Subject Age Group'] != '-']
# Some of these early records with ages belong to the same GO/SC Num
# Seems like they just transitioned to start recording age group at this point
```

Out[9]:

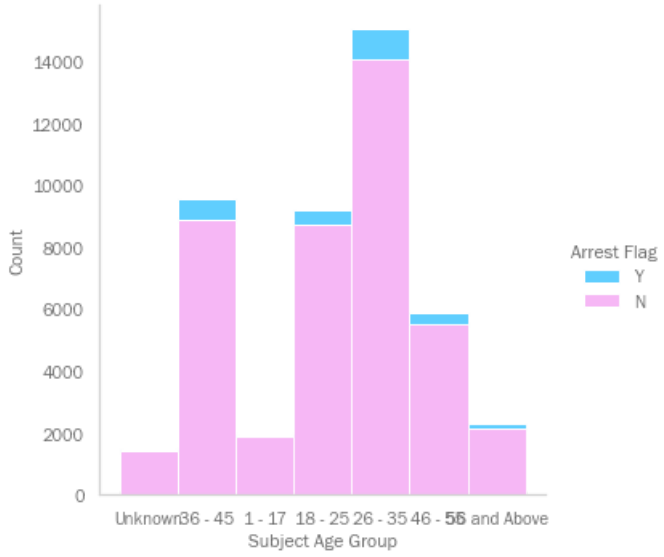
	Subject Age Group	Subject ID	GO / SC Num	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	...	Reported Time	Initial Call Type
214	36 - 45	-1	20150000001902	35955	Field Contact	None	7595	1978	M	White	...	04:59:00	-
334	36 - 45	-1	20150000001920	36225	Field Contact	None	7595	1978	M	White	...	05:20:00	-
1067	1 - 17	-1	20150000002531	47538	Field Contact	None	7415	1978	M	White	...	01:25:00	-
1095	1 - 17	-1	20150000002613	49001	Field Contact	None	7673	1985	M	White	...	23:19:00	-
1151	1 - 17	-1	20150000002531	47539	Field Contact	None	7415	1978	M	White	...	01:58:00	-
1235	1 - 17	-1	20150000002531	47540	Field Contact	None	7415	1978	M	White	...	01:59:00	-
1288	1 - 17	-1	20150000002531	47541	Field Contact	None	7415	1978	M	White	...	02:01:00	-
1369	1 - 17	-1	20150000002611	48895	Field Contact	None	7726	1990	M	White	...	17:17:00	-
1422	1 - 17	-1	20150000002613	48899	Field Contact	None	7673	1985	M	White	...	23:13:00	-
1448	1 - 17	-1	20150000002613	48900	Field Contact	None	7673	1985	M	White	...	23:18:00	-

10 rows × 23 columns

In [10]:
```
# Think I am ok to replace '-' with 'unknown'
terry['Subject Age Group'] = terry['Subject Age Group'].replace(to_replace='-',value='Unknown')
terry['Subject Age Group'].value_counts()
```

Out[10]:
```
26 - 35         15054
36 - 45          9557
18 - 25          9169
46 - 55          5852
56 and Above     2301
1 - 17           1935
Unknown          1449
Name: Subject Age Group, dtype: int64
```

In [11]:
```
sns.displot(terry, x='Subject Age Group', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack')
plt.savefig('Images/subject_age_group.png');
```



Analyze 'Subject ID'

Output - terry_sort

Binned Subject ID into a new column Subject Known with values of Unidentified, First, Repeat, then deleted Subject ID column

In [12]:
```
# Now look at duplicate Subject ID
terry['Subject ID'].value_counts() # 8249 unique values
```

Out[12]:
```
-1              34718
 7726859935        19
 7753260438        13
 7727117712        12
 7726559999         9
                 ...
 7725797630         1
 16168302851        1
 7704469768         1
 7733768490         1
 16219707395        1
Name: Subject ID, Length: 8249, dtype: int64
```

In [13]:
```
# There are 8249 different "subject IDs" even though most of the are -1, which I assume is their vers
# Many seem to be duplicated several times.  Does this indicate multiple times the same person was st
# Can I somehow look at people who were stopped multiple times?
# Let's first change the -1 to unidentified so I don't think they are duplicates
terry['Subject ID'] = terry['Subject ID'].astype(str)
terry['Subject ID'] = terry['Subject ID'].replace(to_replace='-1',value='Unidentified')
terry['Subject ID'].value_counts()
```

Out[13]:
```
Unidentified    34718
7726859935         19
7753260438         13
7727117712         12
7726559999          9
                ...
7815600937          1
7739437979          1
7728777839          1
7729969068          1
7727212388          1
Name: Subject ID, Length: 8249, dtype: int64
```

My initial reaction was that Subject ID doesn't matter and can be removed. There are 8249 unique values and I certainly don't want that many dummy variables. But I wonder if it is possible that by a subject NOT identifying themselves (i.e. Subject ID = -1), they could be more or less likely to be arrested. Same concern about Subject IDs that have many repeat encounters... maybe they are more or less likely to be arrested. I'm considering binning the info somehow... into 'unidentified', 'first encounter', 'repeat encounter'. Does the officer know it is a repeat encounter if it happened with a different officer?

In [14]:
```
# This value is interesting to see who has been stopped more than once
# Several people appear to be repeatedly stopped
# The question is: Do I want to consider Subject ID in this analysis?
# If I encode it I will have 8249 more variables
subject_known = terry[terry['Subject ID'] != 'Unidentified']
subject_known[subject_known.duplicated(subset=['Subject ID'], keep=False)]['Subject ID'].value_counts
```

Out[14]:
```
7726859935    19
7753260438    13
7727117712    12
7726559999     9
7727600619     9
              ..
7725923694     2
8216706891     2
7725525905     2
7727594822     2
8749257941     2
Name: Subject ID, Length: 1438, dtype: int64
```

In [15]:
```
# There are 1438 unique subject IDs that are repeated at least once. Can I map the Subject ID to a ne
terry_sort = terry.sort_values(['Reported Date', 'Reported Time']) #sort by date and time first
Subject_First = []
Subjects = []
for subject in range(0,len(terry_sort)):
    if terry_sort['Subject ID'][subject] == "Unidentified":
        Subjects.append('Unidentified')
    elif terry_sort['Subject ID'][subject] in Subject_First:
        Subjects.append('Repeat')
    else:
        Subjects.append('First')
        Subject_First.append(terry_sort['Subject ID'][subject])
```

In [16]:
```
terry_sort['Subject Known'] = pd.Series(Subjects, index = terry.index)
```

In [17]:
```
terry_sort['Subject Known'].value_counts()
```

Out[17]:
```
Unidentified    34718
First            8248
Repeat           2351
Name: Subject Known, dtype: int64
```

In [18]:
```
# Just checking that my sort worked and the index matched up
(terry_sort[(terry_sort.duplicated(subset=['Subject ID'], keep=False)) &
            (terry_sort['Subject ID']!='Unidentified')]).sort_values('Subject ID')
```

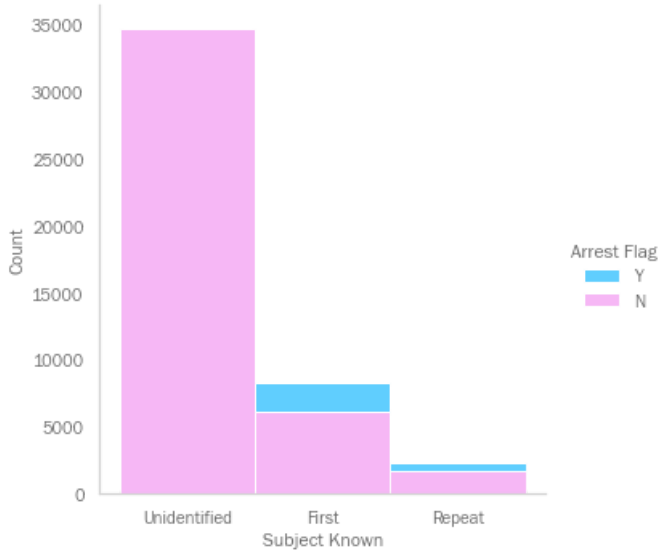
Out[18]:

	Subject Age Group	Subject ID	GO / SC Num	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Off R
12009	18 - 25	10042368279	20200000095287	12775150617	Arrest	-	8394	1991	M	W
12010	18 - 25	10042368279	20200000182733	13357757706	Field Contact	Knife/Cutting/Stabbing Instrument	6334	1974	M	W
36633	36 - 45	10045452578	20190000326602	10045469204	Arrest	-	8632	1997	M	W
36634	36 - 45	10045452578	20200000202621	13806256374	Arrest	-	7690	1977	M	W
36637	36 - 45	10045502278	20200000111289	12802772614	Field Contact	-	8696	1996	M	W
...	...	...	...	...	...	...	...	...	...	
42731	46 - 55	9879252608	20200000171525	13254286419	Field Contact	-	8459	1990	M	Hispa or La
26737	26 - 35	9928085824	20190000322078	9928079386	Field Contact	-	8656	1988	M	W
26738	26 - 35	9928085824	20200000096497	12778521884	Field Contact	Knife/Cutting/Stabbing Instrument	6800	1972	M	W
26740	26 - 35	9972151245	20190000444352	11910224956	Field Contact	-	7773	1978	M	W
26739	26 - 35	9972151245	20190000323932	9969555078	Offense Report	-	8418	1984	M	W

3789 rows × 24 columns

In [19]:
```
# OK I'm going to reset the index for clarity and I do not need Subject ID anymore
terry_sort.reset_index(drop=True, inplace=True)
terry_sort.drop(columns='Subject ID', axis=1, inplace=True)
```

In [20]:
```
sns.displot(terry_sort, x='Subject Known', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack')
plt.savefig('Images/subject_id.png');
```



In [21]:
```
terry_sort[terry_sort['Subject Known'] == "Unidentified"]['Arrest Flag'].value_counts()

# None of the Unidentified were arrested? None?
```

Out[21]:
```
N    34718
Name: Arrest Flag, dtype: int64
```

In [22]:
```
# fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(15,5))
# sns.histplot(terry_sort, x='Subject Known', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack
# axes[0].set_title("Price Distribution")
# axes[0].set_xlabel('Sales Price')
# plt.savefig('images/price_dist.png'); # This is our current price distribution
# sns.scatterplot(terry_sort, x='Subject Known',  ax=axes[1])
# axes[1].set_title("Price Distribution after Outliers Removed")
# axes[1].set_xlabel('Sales Price')
# plt.savefig('images/outlier_comparison.png');
```

Analyze 'GO / SC Num'

Parent report ID number seems irrelevant to my model. Dropped this column.

In [23]:
```
# This field is related to the parent report id number.  There are 35439 unique values.
# I do not see how this field could be relevent and I do not want 35439 dummy variables.
# So I will drop this variable.
terry_sort['GO / SC Num'].value_counts()
```

Out[23]:
```
20150000190790    16
20160000378750    16
20180000134604    14
20170000132836    13
20190000441736    13
                  ..
20160000292906     1
20150000281640     1
20190000105516     1
20200000057387     1
20180000071981     1
Name: GO / SC Num, Length: 35439, dtype: int64
```

In [24]:
```
terry_sort.drop(columns='GO / SC Num', axis=1, inplace=True)
terry_sort.head()
```

Out[24]:

	Subject Age Group	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	...	Initial Call Type	Final Call Type	Call Type
0	1 - 17	28020	Referred for Prosecution	Lethal Cutting Instrument	4585	1955	M	Hispanic or Latino	Black or African American	Female	...	-	-	-
1	18 - 25	305901	Arrest	None	7661	1984	M	White	Black or African American	Male	...	-	-	-
2	36 - 45	28092	Field Contact	None	7634	1977	M	White	Multi-Racial	Male	...	-	-	-
3	18 - 25	28093	Field Contact	None	7634	1977	M	White	White	Male	...	-	-	-
4	26 - 35	28381	Field Contact	None	7634	1977	M	White	White	Male	...	-	-	-

5 rows × 22 columns

Analyze 'Terry Stop ID'

Found duplicates that appear to be same subject, same stop, but a different weapon.
Kept the first duplicate and changed Weapon Type to Multiple. Then drop Terry Stop ID column.

In [25]:
```
terry_sort['Terry Stop ID'].value_counts()
```

Out[25]:
```
15045077325    3
13080077761    3
12119304761    2
12105013403    2
15595812669    2
              ..
490528         1
8705875089     1
13103094430    1
154270         1
65536          1
Name: Terry Stop ID, Length: 45292, dtype: int64
```

In [26]:
```
terry_sort[terry_sort.duplicated
           (subset=['Terry Stop ID'], keep=False)
          ].loc[:,['Subject Known','Terry Stop ID', 'Weapon Type', 'Stop Resolution', 'Reported Date'
# 48 rows of duplicates, otherwise a unique key
# But what do these duplicates mean?
# Same subject, same stop, same resolution, different weapons, same officer
# It appears the officer enters a new record for each weapon found
# But this isn't really a new arrest... it is the same subject, same officer, same stop, same arrest
# Need to check, are the outcomes ever actually different? No
# The outcomes are not different.  It just looks like two (or more) weapons for the same subject at t
# What will this do to my models?
# Can I somehow capture that 2 weapons were found?
# My instinct says to drop these duplicates.  The stop id is not actually relevent.
# Perhaps change the Weapon Type to include both types
# Given that only 6% of the stops result in arrest, and half of these 2 weapon stops result in arrest
# Since I am considering changing the weapon to to 'Multiple', let's analyze that variable first.
```

Out[26]:

	Subject Known	Terry Stop ID	Weapon Type	Stop Resolution	Reported Date	Arrest Flag
35968	First	8611673538	Blunt Object/Striking Implement	Field Contact	2019-07-12T00:00:00	N
35969	Repeat	8611673538	Knife/Cutting/Stabbing Instrument	Field Contact	2019-07-12T00:00:00	N
36162	First	8677596250	Knife/Cutting/Stabbing Instrument	Offense Report	2019-07-22T00:00:00	N
36163	Repeat	8677596250	Taser/Stun Gun	Offense Report	2019-07-22T00:00:00	N
36428	First	9585545373	Firearm	Field Contact	2019-08-03T00:00:00	N
36429	Repeat	9585545373	Handgun	Field Contact	2019-08-03T00:00:00	N
38928	First	12034618758	Knife/Cutting/Stabbing Instrument	Arrest	2019-12-08T00:00:00	Y
38929	Repeat	12034618758	Other Firearm	Arrest	2019-12-08T00:00:00	Y
39126	First	12105013403	Knife/Cutting/Stabbing Instrument	Arrest	2019-12-17T00:00:00	Y
39127	Repeat	12105013403	Mace/Pepper Spray	Arrest	2019-12-17T00:00:00	Y
39341	Repeat	12119304761	Blunt Object/Striking Implement	Field Contact	2019-12-28T00:00:00	N
39342	Repeat	12119304761	Knife/Cutting/Stabbing Instrument	Field Contact	2019-12-28T00:00:00	N
40035	First	12601385662	Blunt Object/Striking Implement	Field Contact	2020-02-02T00:00:00	N
40036	Repeat	12601385662	Knife/Cutting/Stabbing Instrument	Field Contact	2020-02-02T00:00:00	N
40129	First	12608907801	Firearm	Arrest	2020-02-07T00:00:00	Y
40130	Repeat	12608907801	Knife/Cutting/Stabbing Instrument	Arrest	2020-02-07T00:00:00	Y
40595	Repeat	12686594000	Blunt Object/Striking Implement	Arrest	2020-03-03T00:00:00	Y
40596	Repeat	12686594000	Knife/Cutting/Stabbing Instrument	Arrest	2020-03-03T00:00:00	Y
40616	First	12689034912	Blunt Object/Striking Implement	Field Contact	2020-03-04T00:00:00	N
40617	Repeat	12689034912	Knife/Cutting/Stabbing Instrument	Field Contact	2020-03-04T00:00:00	N
40994	First	12781960580	Firearm	Offense Report	2020-03-21T00:00:00	N
40995	Repeat	12781960580	Knife/Cutting/Stabbing Instrument	Offense Report	2020-03-21T00:00:00	N
41505	First	12851512661	Knife/Cutting/Stabbing Instrument	Field Contact	2020-04-11T00:00:00	N
41506	Repeat	12851512661	Other Firearm	Field Contact	2020-04-11T00:00:00	N
41894	Repeat	13080077761	Blunt Object/Striking Implement	Arrest	2020-04-24T00:00:00	Y
41895	Repeat	13080077761	Knife/Cutting/Stabbing Instrument	Arrest	2020-04-24T00:00:00	Y
41896	Repeat	13080077761	Mace/Pepper Spray	Arrest	2020-04-24T00:00:00	Y
43080	Repeat	13645398605	Blunt Object/Striking Implement	Offense Report	2020-06-22T00:00:00	N
43081	Repeat	13645398605	Knife/Cutting/Stabbing Instrument	Offense Report	2020-06-22T00:00:00	N
43430	First	14266198833	Blunt Object/Striking Implement	Arrest	2020-07-26T00:00:00	Y
43431	Repeat	14266198833	Knife/Cutting/Stabbing Instrument	Arrest	2020-07-26T00:00:00	Y
43666	Repeat	14935106231	Knife/Cutting/Stabbing Instrument	Arrest	2020-08-18T00:00:00	Y

	Subject Known	Terry Stop ID	Weapon Type	Stop Resolution	Reported Date	Arrest Flag
43667	Repeat	14935106231	Mace/Pepper Spray	Arrest	2020-08-18T00:00:00	Y
43719	First	15045077325	Blunt Object/Striking Implement	Arrest	2020-08-21T00:00:00	Y
43720	Repeat	15045077325	Fire/Incendiary Device	Arrest	2020-08-21T00:00:00	Y
43721	Repeat	15045077325	Knife/Cutting/Stabbing Instrument	Arrest	2020-08-21T00:00:00	Y
43764	First	15055860719	Knife/Cutting/Stabbing Instrument	Field Contact	2020-08-26T00:00:00	N
43765	Repeat	15055860719	Mace/Pepper Spray	Field Contact	2020-08-26T00:00:00	N
44049	First	15595812669	Blunt Object/Striking Implement	Offense Report	2020-09-17T00:00:00	N
44050	Repeat	15595812669	Knife/Cutting/Stabbing Instrument	Offense Report	2020-09-17T00:00:00	N
44401	First	16076863718	Firearm	Arrest	2020-10-12T00:00:00	Y
44402	Repeat	16076863718	Knife/Cutting/Stabbing Instrument	Arrest	2020-10-12T00:00:00	Y
44687	First	16677515049	Knife/Cutting/Stabbing Instrument	Field Contact	2020-10-29T00:00:00	N
44688	Repeat	16677515049	Taser/Stun Gun	Field Contact	2020-10-29T00:00:00	N
44807	First	17542218019	Knife/Cutting/Stabbing Instrument	Arrest	2020-11-05T00:00:00	Y
44808	Repeat	17542218019	Other Firearm	Arrest	2020-11-05T00:00:00	Y
45201	Repeat	18800151229	Blunt Object/Striking Implement	Field Contact	2020-12-06T00:00:00	N
45202	Repeat	18800151229	Knife/Cutting/Stabbing Instrument	Field Contact	2020-12-06T00:00:00	N

In [27]:
```
# These are not really Repeat stops as my new Subject Known indicates
# I am going to change these records Weapon Type to Multiple and then delete the duplicate record

terry_sort.loc[terry_sort[terry_sort.duplicated(subset=['Terry Stop ID'], keep=False)].index,'Weapon
terry_sort.drop_duplicates(subset=['Terry Stop ID'], keep='first', inplace=True)
terry_sort.shape
```

Out[27]: (45292, 22)

In [28]:
```
# Now delete the Terry Stop ID column
terry_sort.reset_index(drop=True, inplace=True)
terry_sort.drop(columns=['Terry Stop ID'], inplace=True)
terry_sort.tail()
```

Out[28]:

	Subject Age Group	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	...
45287	26 - 35	Field Contact	-	6353	1972	M	White	White	Male	2020-12-15T00:00:00	...
45288	26 - 35	Field Contact	-	6353	1972	M	White	White	Male	2020-12-15T00:00:00	...
45289	36 - 45	Arrest	-	8719	1996	M	Asian	Black or African American	Male	2020-12-15T00:00:00	...
45290	18 - 25	Field Contact	-	8639	1995	M	White	Black or African American	Male	2020-12-15T00:00:00	...
45291	56 and Above	Field Contact	Knife/Cutting/Stabbing Instrument	8646	1996	M	White	Black or African American	Male	2020-12-15T00:00:00	...

5 rows × 21 columns

Analyze 'Weapon Type'

Binned some obvious types together. May consider just using weapon found or no weapon found.

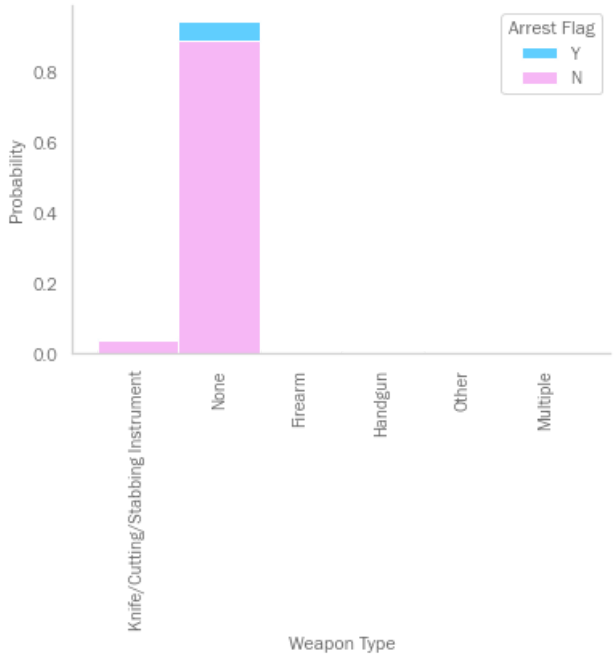
In [29]:
```
terry_sort['Weapon Type'].value_counts()
# The '-' should probably be changed to 'Unknown" or maybe it actually means 'None'. I'm going to cho
# believe that is mostly likely what the reports meant.
# None/Not Applicable should be combined with None
# Firearm Other should be combined with Other Firearm
# Should Club and Blackjack and Brass Knuckles be combined with Club, Blackjack, Brass Knuckles?
# I need to think backward from what I would like my results to tell me.  Is it useful to know the 'F
# but Other Firearm is not? I don't think so. Perhaps the best solution is just weapon vs no weapon.
# some categories that seem obvious
```

Out[29]:
```
None                                32565
-                                   10119
Lethal Cutting Instrument            1482
Knife/Cutting/Stabbing Instrument     499
Handgun                               280
Firearm Other                         100
Blunt Object/Striking Implement        59
Club, Blackjack, Brass Knuckles        49
Firearm                                30
Multiple                               23
Mace/Pepper Spray                      17
Other Firearm                          16
Firearm (unk type)                     15
Club                                    9
None/Not Applicable                     7
Rifle                                   7
Taser/Stun Gun                          5
Fire/Incendiary Device                  3
Shotgun                                 3
Automatic Handgun                       2
Blackjack                               1
Brass Knuckles                          1
Name: Weapon Type, dtype: int64
```

In [30]:
```
# Going to group some together
weapon_dict = {'-' : 'None',
               'None/Not Applicable' : 'None',
               'Lethal Cutting Instrument' : 'Knife/Cutting/Stabbing Instrument',
               'Firearm Other' : 'Firearm',
               'Other Firearm' : 'Firearm',
               'Blunt Object/Striking Implement' : 'Other',
               'Club, Blackjack, Brass Knuckles' : 'Other',
               'Club' : 'Other',
               'Blackjack' : 'Other',
               'Brass Knuckles' : 'Other',
               'Mace/Pepper Spray' : 'Other',
               'Firearm (unk type)' : 'Firearm',
               'Rifle' : 'Other',
               'Taser/Stun Gun' : 'Other',
               'Fire/Incendiary Device' : 'Other',
               'Shotgun' : 'Other',
               'Automatic Handgun' : 'Handgun'}

for k, v in weapon_dict.items():
    terry_sort['Weapon Type'] = terry_sort['Weapon Type'].replace(to_replace = k, value = v)

terry_sort['Weapon Type'].value_counts()
```

Out[30]:
```
None                                42691
Knife/Cutting/Stabbing Instrument    1981
Handgun                               282
Firearm                               161
Other                                 154
Multiple                               23
Name: Weapon Type, dtype: int64
```

In [31]:
```
sns.histplot(terry_sort, x='Weapon Type', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack', s
plt.xticks(rotation=90);
```



In [32]:
```
# Only 283 cases where Arrest Flag is Y and Weapon Type is NOT None
terry_sort[(terry_sort['Arrest Flag'] == 'Y') & (terry_sort['Weapon Type'] != 'None')]
```

Out[32]:

	Subject Age Group	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	...	
34576	26 - 35	Arrest	Knife/Cutting/Stabbing Instrument	7765	1985	M	White	White	Male	2019-05-09T00:00:00	...	TH
34616	46 - 55	Arrest	Handgun	8507	1987	M	White	White	Male	2019-05-11T00:00:00	...	
34662	36 - 45	Arrest	Other	7792	1987	M	White	White	Male	2019-05-14T00:00:00	...	F
34802	1 - 17	Arrest	Knife/Cutting/Stabbing Instrument	7782	1986	M	White	Black or African American	Male	2019-05-20T00:00:00	...	T
34863	18 - 25	Arrest	Handgun	7677	1983	M	White	-	Male	2019-05-23T00:00:00	...	T
...	...	...	...	...	...	...	...	...	...	...	...	
45108	46 - 55	Arrest	Other	8597	1990	M	Asian	Unknown	Male	2020-11-28T00:00:00	...	
45110	56 and Above	Arrest	Knife/Cutting/Stabbing Instrument	8701	1994	M	Two or More Races	American Indian or Alaska Native	Male	2020-11-28T00:00:00	...	
45145	26 - 35	Arrest	Knife/Cutting/Stabbing Instrument	8719	1996	M	Asian	Black or African American	Male	2020-12-03T00:00:00	...	PH
45170	26 - 35	Arrest	Other	8643	1989	F	White	Black or African American	Male	2020-12-05T00:00:00	...	
45202	18 - 25	Arrest	Other	8762	1978	M	White	Black or African American	Male	2020-12-08T00:00:00	...	

283 rows × 21 columns

Analyze 'Stop Resolution'

I was a bit concerned that Stop Resolution is a perfect predictor for Arrest Flag, since only 2 of the Arrest Flag = Y do NOT have Stop Resolution = arrest. However, since there are so many Stop Resolutions of 'arrest' that don't lead to an arrest flag of Y, I guess the models will have to predict on other information. I'm going to choose to leave it as is for now.

In [33]:
```
terry_sort['Stop Resolution'].value_counts()
```

Out[33]:
```
Field Contact               18265
Offense Report              15194
Arrest                      10928
Referred for Prosecution      728
Citation / Infraction         177
Name: Stop Resolution, dtype: int64
```

In [34]:
```
terry_sort[(terry_sort['Stop Resolution']=='Arrest') & (terry_sort['Arrest Flag']=='Y')]
```

Out[34]:

	Subject Age Group	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	...	
34541	26 - 35	Arrest	None	7758	1987	M	White	White	Male	2019-05-08T00:00:00	...	F
34546	56 and Above	Arrest	None	8527	1990	M	Hispanic or Latino	Black or African American	Male	2019-05-08T00:00:00	...	
34560	18 - 25	Arrest	None	7794	1991	M	White	White	Male	2019-05-09T00:00:00	...	T
34567	46 - 55	Arrest	None	7765	1985	M	White	White	Male	2019-05-09T00:00:00	...	
34576	26 - 35	Arrest	Knife/Cutting/Stabbing Instrument	7765	1985	M	White	White	Male	2019-05-09T00:00:00	...	T
...	...	...	...	...	...	...	...	...	...	...	...	
45273	36 - 45	Arrest	None	8702	1979	M	White	Unknown	Male	2020-12-14T00:00:00	...	
45279	18 - 25	Arrest	None	8749	1993	M	Asian	White	Male	2020-12-14T00:00:00	...	
45283	26 - 35	Arrest	None	8759	1996	M	Asian	Black or African American	Male	2020-12-14T00:00:00	...	F
45285	36 - 45	Arrest	None	8425	1989	M	White	White	Male	2020-12-15T00:00:00	...	
45289	36 - 45	Arrest	None	8719	1996	M	Asian	Black or African American	Male	2020-12-15T00:00:00	...	

2718 rows × 21 columns

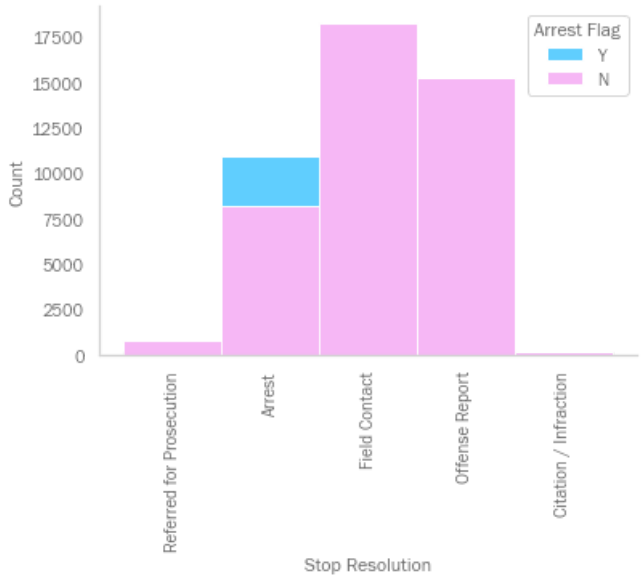
In [35]:
```
terry_sort[(terry_sort['Stop Resolution']!='Arrest') & (terry_sort['Arrest Flag']=='Y')]
```

Out[35]:

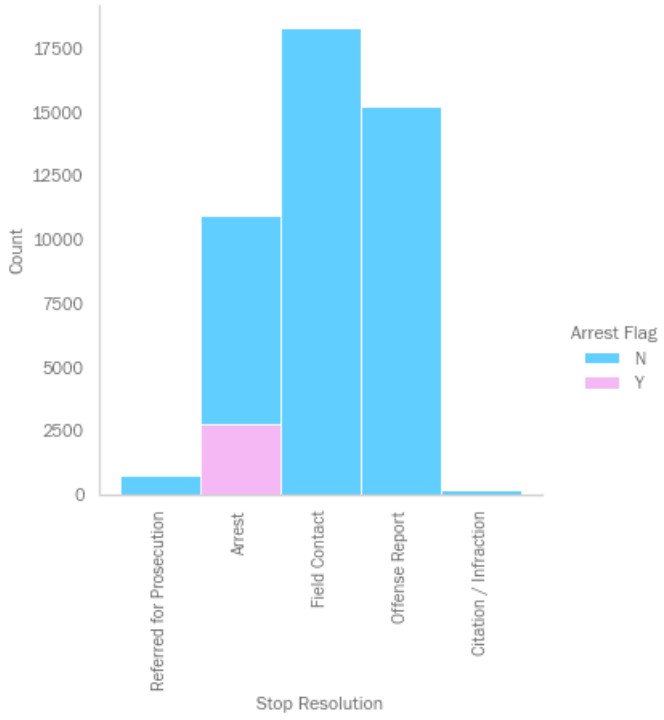
	Subject Age Group	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	...	Initi
36377	56 and Above	Referred for Prosecution	None	8643	1989	F	White	White	Male	2019-08-02T00:00:00	...	BEHAVIORAL/E
39812	36 - 45	Referred for Prosecution	None	8582	1991	M	White	Black or African American	Female	2020-01-23T00:00:00	...	DIST MISCELLANEC

2 rows × 21 columns

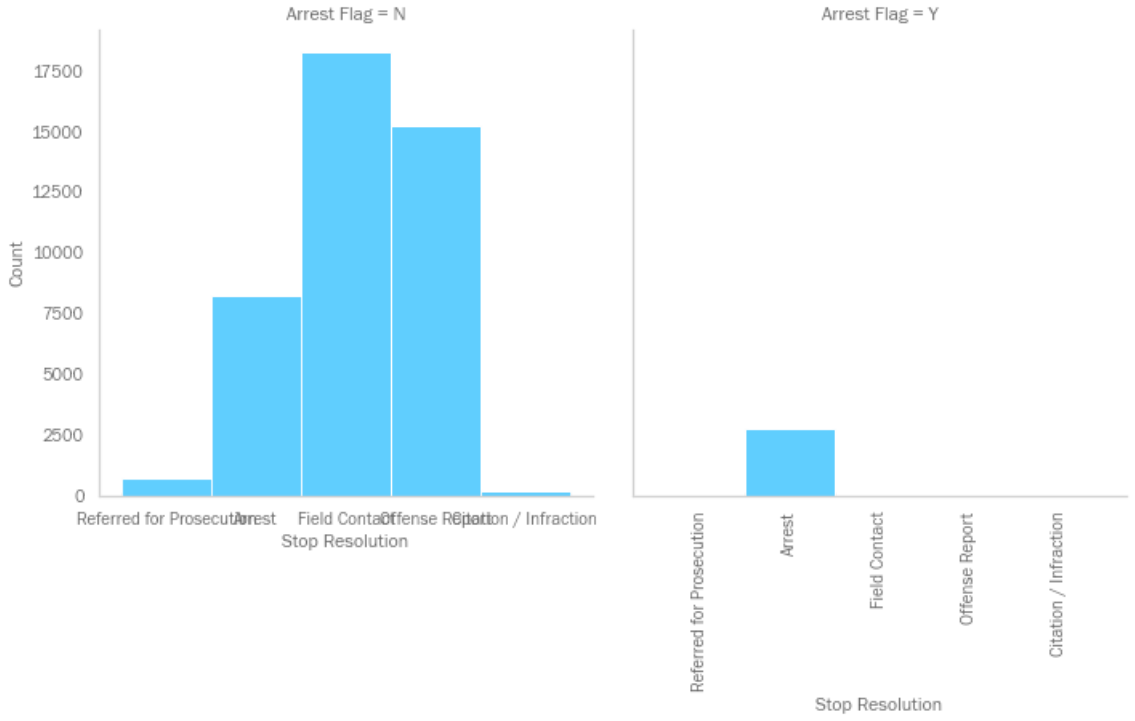
In [36]:
```
sns.histplot(terry_sort, x='Stop Resolution', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack
plt.xticks(rotation=90)
plt.savefig('Images/stop_resolution.png');
```



In [37]:
```
# Examining Stop Resolution again.  I think it may be a confounding variable. Will probably have to r
sns.displot(terry_sort, x='Stop Resolution', hue='Arrest Flag', multiple='stack')
plt.xticks(rotation=90)
plt.savefig('Images/stop_resolution_dist.png');
```



In [38]:
```
sns.displot(terry_sort, x='Stop Resolution', col='Arrest Flag', multiple='dodge')
plt.xticks(rotation=90)
plt.savefig('Images/stop_resolution_dodge.png');
```



Analyze 'Officer ID'

Strip spaces, replace - and -9 with Unknown. Binned top 10 and changed the rest to Other.

In [39]:
```
terry_sort['Officer ID'].unique()
# There are 1183 unique values, with some officers generating hundreds of stops and some only one
```

Out[39]:
```
array(['4585  ', '7661  ', '7634  ', ..., '8765  ', '8772  ', '8751  '],
      dtype=object)
```

In [40]:
```
# I noticed some value of -9.  Let's look at them.
indices=[]
for i,x in enumerate(terry_sort['Officer ID']):
    if x.startswith('-'):
        indices.append(i)

terry_sort.iloc[indices,:]
```

Out[40]:

	Subject Age Group	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	...	Initial Call Type	Fina Ca Typ
427	18 - 25	Field Contact	None	-9	1900	N	Unknown	Black or African American	Male	2015-05-19T00:00:00	...	-	
7101	36 - 45	Field Contact	None	-9	1900	N	Unknown	White	Female	2016-01-02T00:00:00	...	-	
12857	36 - 45	Field Contact	None	-9	1900	N	Unknown	White	Male	2016-09-14T00:00:00	...	-	
17770	18 - 25	Field Contact	None	-9	1900	N	Unknown	Unknown	Female	2017-06-06T00:00:00	...	-	
23085	18 - 25	Arrest	None	-9	1900	N	Unknown	White	Male	2018-02-07T00:00:00	...	-	
35225	36 - 45	Field Contact	None	-	1900	N	Unknown	Black or African American	Male	2019-06-06T00:00:00	...	-	
35904	36 - 45	Field Contact	None	-	1900	N	Unknown	-	Female	2019-07-09T00:00:00	...	-	
35995	46 - 55	Field Contact	None	-	1900	N	Unknown	Black or African American	Male	2019-07-13T00:00:00	...	-	
36138	36 - 45	Field Contact	None	-	1900	N	Unknown	White	Male	2019-07-21T00:00:00	...	-	
36238	Unknown	Field Contact	None	-	1900	N	Unknown	-	-	2019-07-27T00:00:00	...	-	
36239	Unknown	Field Contact	None	-	1900	N	Unknown	-	-	2019-07-27T00:00:00	...	-	
36240	Unknown	Field Contact	None	-	1900	N	Unknown	-	-	2019-07-27T00:00:00	...	-	
36666	26 - 35	Arrest	None	-	1900	N	Unknown	White	Female	2019-08-15T00:00:00	...	-	
37177	Unknown	Field Contact	None	-	1900	N	Unknown	White	Male	2019-09-10T00:00:00	...	-	
37420	56 and Above	Field Contact	None	-	1900	N	Unknown	-	Male	2019-09-22T00:00:00	...	-	
37421	46 - 55	Field Contact	None	-	1900	N	Unknown	White	Male	2019-09-22T00:00:00	...	-	
37515	46 - 55	Field Contact	None	-	1900	N	Unknown	White	Male	2019-09-29T00:00:00	...	-	
37541	36 - 45	Field Contact	None	-	1900	N	Unknown	-	Female	2019-09-30T00:00:00	...	-	
37967	46 - 55	Offense Report	None	-	1900	N	Unknown	American Indian or Alaska Native	Female	2019-10-23T00:00:00	...	-	
37969	26 - 35	Arrest	None	-	1900	N	Unknown	White	Male	2019-10-23T00:00:00	...	-	
37985	46 - 55	Field Contact	None	-	1900	N	Unknown	-	Male	2019-10-24T00:00:00	...	-	
38066	36 - 45	Field Contact	None	-	1900	N	Unknown	-	Female	2019-10-28T00:00:00	...	-	

	Subject Age Group	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	...	Initial Call Type	Fina Ca Typ
38090	18 - 25	Offense Report	None	-	1900	N	Unknown	American Indian or Alaska Native	Female	2019-10-29T00:00:00	...	-	
38135	26 - 35	Field Contact	None	-	1900	N	Unknown	White	Male	2019-11-02T00:00:00	...	-	
38321	36 - 45	Field Contact	None	-	1900	N	Unknown	White	Male	2019-11-10T00:00:00	...	-	
38324	26 - 35	Field Contact	None	-	1900	N	Unknown	White	Male	2019-11-10T00:00:00	...	-	
38570	36 - 45	Field Contact	None	-	1900	N	Unknown	White	Male	2019-11-22T00:00:00	...	-	
38571	36 - 45	Field Contact	None	-	1900	N	Unknown	White	Female	2019-11-22T00:00:00	...	-	
38691	18 - 25	Field Contact	None	-	1900	N	Unknown	White	Female	2019-11-27T00:00:00	...	-	

29 rows × 21 columns

In [41]:
```
terry_sort[~terry_sort.duplicated(subset = 'Officer ID', keep=False)]
# 79 Officer IDs only occur once
```

Out[41]:

	Subject Age Group	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	...	Ini
30	26 - 35	Arrest	None	5143	1957	M	Black or African American	Black or African American	Male	2015-03-19T00:00:00	...	ROB (INCLUI
90	26 - 35	Field Contact	None	4320	1956	M	White	Black or African American	Male	2015-04-01T00:00:00	...	
603	26 - 35	Field Contact	None	7448	1972	M	White	Black or African American	Male	2015-05-24T00:00:00	...	
764	26 - 35	Field Contact	None	5712	1961	M	White	White	Male	2015-05-28T00:00:00	...	
823	26 - 35	Field Contact	None	5458	1966	M	White	White	Male	2015-05-29T00:00:00	...	
...	...	...	...	...	...	...	...	...	...	...	...	
42286	26 - 35	Offense Report	None	8732	1992	M	Two or More Races	Unknown	Male	2020-05-08T00:00:00	...	DIS MISCELLANE
42506	46 - 55	Arrest	None	6735	1968	M	White	Black or African American	Male	2020-05-16T00:00:00	...	ASL
42909	26 - 35	Field Contact	None	6421	1972	F	White	American Indian or Alaska Native	Female	2020-06-06T00:00:00	...	DUI - DRI
42987	18 - 25	Offense Report	None	8715	1994	M	White	-	Male	2020-06-14T00:00:00	...	SUSPICIO VEHICLE (
43565	18 - 25	Arrest	None	8745	1991	M	Black or African American	White	Male	2020-08-08T00:00:00	...	SUSPICIO VEHICLE (

79 rows × 21 columns

In [42]:
```
terry_sort['Officer ID'] = terry_sort['Officer ID'].map(lambda x: x.strip())
terry_sort['Officer ID'] = terry_sort['Officer ID'].replace(to_replace='-',value='Unknown')
terry_sort['Officer ID'] = terry_sort['Officer ID'].replace(to_replace='-9',value='Unknown')
top_ten = terry_sort['Officer ID'].value_counts()[:10].index.tolist()
```

In [43]:
```
# Bin into top 10 and then Other
Officer_Bin =[]
for i in range(len(terry_sort)):
    if terry_sort.iloc[i]['Officer ID'] not in top_ten:
        Officer_Bin.append('Other')
    else:
        Officer_Bin.append(terry_sort.iloc[i]['Officer ID'])

terry_sort['Officer_Bin'] = Officer_Bin
```

In [44]:
```
terry_sort['Officer_Bin'].value_counts()
```

Out[44]:
```
Other    42314
7456       405
7634       341
7773       309
7765       305
7758       301
7690       291
7774       284
7792       264
8302       240
7713       238
Name: Officer_Bin, dtype: int64
```

In [45]:
```
terry_sort.drop(columns='Officer ID', inplace=True)
```

Analyze 'Officer YOB'

Bin into decades in 'Officer DOB' with datatype category. Values of 1900 obviously mean unknown so I don't feel the need to change them.

In [46]:
```
terry_sort['Officer YOB'].value_counts() # 52 unique years

# I don't imagine the specific year is valuable.  For convenience I will bin into decades.
```

Out[46]:
```
1986    3185
1987    2900
1984    2681
1991    2623
1985    2433
1992    2296
1990    2157
1988    1998
1989    1928
1982    1824
1983    1675
1979    1457
1981    1379
1993    1350
1971    1215
1978    1128
1995    1002
1976     987
1977     983
1973     901
1994     831
1980     789
1967     707
1968     621
1970     582
1974     548
1996     533
1969     532
1975     521
1962     452
1972     419
1965     415
1964     411
1997     338
1963     256
1966     223
1958     218
1961     208
1959     174
1960     161
1900      64
1954      44
1957      43
1953      32
1955      21
1956      17
1948      11
1952       9
1949       5
1998       2
1946       2
1951       1
Name: Officer YOB, dtype: int64
```

In [47]:
```
terry_sort['Officer DOB'] = terry_sort['Officer YOB'].map(lambda x:((x-1900)//10)*10).astype('categor
```

In [48]:
```
terry_sort['Officer DOB'].value_counts() # Hmm. We had 64 officers born in 1900?
```

Out[48]:
```
80    20792
90    11132
70     8741
60     3986
50      559
0        64
40       18
Name: Officer DOB, dtype: int64
```

In [49]:
```
terry_sort[terry_sort['Officer DOB'] == 0] # Mostly occurs when Officer ID = '-9' or '-'
```

Out[49]:

	Subject Age Group	Stop Resolution	Weapon Type	Officer YOB	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Date	Reported Time	...
427	18 - 25	Field Contact	None	1900	N	Unknown	Black or African American	Male	2015-05-19T00:00:00	20:50:00	...
7101	36 - 45	Field Contact	None	1900	N	Unknown	White	Female	2016-01-02T00:00:00	07:05:00	...
12857	36 - 45	Field Contact	None	1900	N	Unknown	White	Male	2016-09-14T00:00:00	21:33:00	...
17770	18 - 25	Field Contact	None	1900	N	Unknown	Unknown	Female	2017-06-06T00:00:00	07:43:00	...
23085	18 - 25	Arrest	None	1900	N	Unknown	White	Male	2018-02-07T00:00:00	04:11:00	...
...	...	...	...	...	...	...	...	...	...	...	...
44760	36 - 45	Field Contact	None	1900	M	Unknown	White	Male	2020-11-03T00:00:00	20:50:19	...
44918	36 - 45	Field Contact	None	1900	M	Unknown	Black or African American	Male	2020-11-14T00:00:00	22:39:46	...
44925	18 - 25	Offense Report	Knife/Cutting/Stabbing Instrument	1900	M	Unknown	Black or African American	Male	2020-11-15T00:00:00	14:20:35	...
45093	26 - 35	Field Contact	Knife/Cutting/Stabbing Instrument	1900	M	Unknown	White	Male	2020-11-27T00:00:00	09:56:46	...
45137	26 - 35	Field Contact	None	1900	M	Unknown	Black or African American	Male	2020-12-02T00:00:00	11:54:24	...

64 rows × 22 columns

In [50]:
```
terry_sort.drop(columns='Officer YOB', axis=1, inplace=True)
```

In [51]:
```
terry_sort.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45292 entries, 0 to 45291
Data columns (total 21 columns):
 #   Column                   Non-Null Count  Dtype   
---  ------                   --------------  -----   
 0   Subject Age Group        45292 non-null  object  
 1   Stop Resolution          45292 non-null  object  
 2   Weapon Type              45292 non-null  object  
 3   Officer Gender           45292 non-null  object  
 4   Officer Race             45292 non-null  object  
 5   Subject Perceived Race   45292 non-null  object  
 6   Subject Perceived Gender 45292 non-null  object  
 7   Reported Date            45292 non-null  object  
 8   Reported Time            45292 non-null  object  
 9   Initial Call Type        45292 non-null  object  
 10  Final Call Type          45292 non-null  object  
 11  Call Type                45292 non-null  object  
 12  Officer Squad            44689 non-null  object  
 13  Arrest Flag              45292 non-null  object  
 14  Frisk Flag               45292 non-null  object  
 15  Precinct                 45292 non-null  object  
 16  Sector                   45292 non-null  object  
 17  Beat                     45292 non-null  object  
 18  Subject Known            45292 non-null  object  
 19  Officer_Bin              45292 non-null  object  
 20  Officer DOB              45292 non-null  category
dtypes: category(1), object(20)
memory usage: 7.0+ MB
```

Analyze 'Officer Gender'

Fine as is... values M, F, N.

In [52]:
```
terry_sort['Officer Gender'].value_counts()
```

Out[52]:
```
M    40097
F     5166
N       29
Name: Officer Gender, dtype: int64
```

Analyze 'Officer Race'

Fine as is.

In [53]:
```
terry_sort['Officer Race'].value_counts()
```

Out[53]:
```
White                            34402
Hispanic or Latino                2581
Two or More Races                 2522
Asian                             1895
Black or African American         1802
Not Specified                     1271
Nat Hawaiian/Oth Pac Islander      441
American Indian/Alaska Native      314
Unknown                             64
Name: Officer Race, dtype: int64
```

Analyze 'Subject Perceived Race'

Changed '-' to 'Unknown'

In [54]:
```
terry_sort['Subject Perceived Race'].value_counts()
```

Out[54]:
```
White                                       22127
Black or African American                   13495
Unknown                                      2423
-                                            1797
Hispanic                                     1684
Asian                                        1449
American Indian or Alaska Native             1313
Multi-Racial                                  809
Other                                         152
Native Hawaiian or Other Pacific Islander      43
Name: Subject Perceived Race, dtype: int64
```

In [55]:
```
terry_sort['Subject Perceived Race'] = terry_sort['Subject Perceived Race'].replace(to_replace = '-',
```

In [56]:
```
terry_sort['Subject Perceived Race'].value_counts()
```

Out[56]:
```
White                                       22127
Black or African American                   13495
Unknown                                      4220
Hispanic                                     1684
Asian                                        1449
American Indian or Alaska Native             1313
Multi-Racial                                  809
Other                                         152
Native Hawaiian or Other Pacific Islander      43
Name: Subject Perceived Race, dtype: int64
```

Analyze 'Subject Perceived Gender'

Grouped together Unknown, Unable to Determine, and '-'

In [57]:
```
terry_sort['Subject Perceived Gender'].value_counts()
```

Out[57]:
```
Male                                                    35427
Female                                                   9245
Unable to Determine                                       326
-                                                         269
Unknown                                                    21
Gender Diverse (gender non-conforming and/or transgender)       4
Name: Subject Perceived Gender, dtype: int64
```

In [58]:
```
# Going to group some together
gender_dict = {'-' : 'Unknown',
               'Unable to Determine' : 'Unknown'}

for k, v in gender_dict.items():
    terry_sort['Subject Perceived Gender'] = terry_sort['Subject Perceived Gender'].replace(to_replac

terry_sort['Subject Perceived Gender'].value_counts()
```

Out[58]:
```
Male                                                    35427
Female                                                   9245
Unknown                                                   616
Gender Diverse (gender non-conforming and/or transgender)       4
Name: Subject Perceived Gender, dtype: int64
```

Analyze 'Reported Date'

I don't see a particular date as being more likely to get arrested. However, maybe certain days of the week, or certain months, or even certain years. I will break these out, and include an ordinal value of Date.

In [59]:
```
terry_sort['Reported Date'].value_counts()
```

Out[59]:
```
2015-10-01T00:00:00    101
2015-09-29T00:00:00     66
2015-05-28T00:00:00     57
2015-07-18T00:00:00     55
2019-04-26T00:00:00     54
                      ...
2015-04-14T00:00:00      1
2015-03-15T00:00:00      1
2015-03-24T00:00:00      1
2015-04-28T00:00:00      1
2015-05-13T00:00:00      1
Name: Reported Date, Length: 2102, dtype: int64
```

In [60]:
```
terry_sort["Date"] = terry_sort["Reported Date"].map(lambda date: datetime.strptime(date[:10], '%Y-%m
terry_sort['Date'].value_counts()
```

Out[60]:
```
2015-10-01    101
2015-09-29     66
2015-05-28     57
2015-07-18     55
2019-04-26     54
              ...
2015-05-10      1
2015-03-28      1
2015-03-15      1
2015-03-24      1
2015-04-28      1
Name: Date, Length: 2102, dtype: int64
```

In [61]:
```
terry_sort['Month'] = terry_sort['Date'].map(lambda date: date.month).astype(str)
terry_sort['Year'] = terry_sort['Date'].map(lambda date: date.year).astype(str)
terry_sort['Day'] = terry_sort['Date'].map(lambda date: date.day_name())
terry_sort['Date'] = terry_sort['Date'].map(datetime.toordinal)
terry_sort.drop(columns=['Reported Date'], inplace=True)
terry_sort.head()
```

Out[61]:

	Subject Age Group	Stop Resolution	Weapon Type	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Reported Time	Initial Call Type	Final Call Type	...	Precinct
0	1 - 17	Referred for Prosecution	Knife/Cutting/Stabbing Instrument	M	Hispanic or Latino	Black or African American	Female	16:10:00	-	-	...	East
1	18 - 25	Arrest	None	M	White	Black or African American	Male	01:13:00	-	-	...	West
2	36 - 45	Field Contact	None	M	White	Multi-Racial	Male	05:49:00	-	-	...	-
3	18 - 25	Field Contact	None	M	White	White	Male	05:55:00	-	-	...	-
4	26 - 35	Field Contact	None	M	White	White	Male	10:38:00	-	-	...	-

5 rows × 24 columns

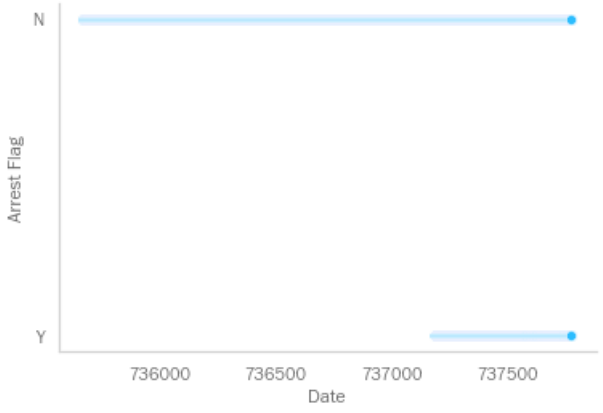
In [62]:
```
terry_sort.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45292 entries, 0 to 45291
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Subject Age Group        45292 non-null  object
 1   Stop Resolution          45292 non-null  object
 2   Weapon Type              45292 non-null  object
 3   Officer Gender           45292 non-null  object
 4   Officer Race             45292 non-null  object
 5   Subject Perceived Race   45292 non-null  object
 6   Subject Perceived Gender 45292 non-null  object
 7   Reported Time            45292 non-null  object
 8   Initial Call Type        45292 non-null  object
 9   Final Call Type          45292 non-null  object
 10  Call Type                45292 non-null  object
 11  Officer Squad            44689 non-null  object
 12  Arrest Flag              45292 non-null  object
 13  Frisk Flag               45292 non-null  object
```

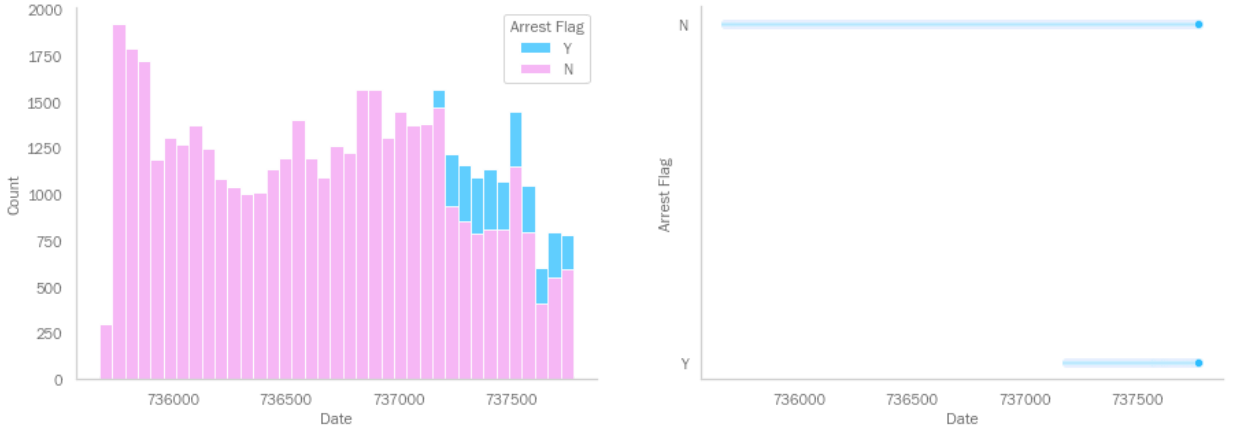
In [63]:
```
terry_sort.Year.value_counts()
```

Out[63]:
```
2018    8883
2019    8242
2016    7715
2017    7488
2015    7059
2020    5905
Name: Year, dtype: int64
```

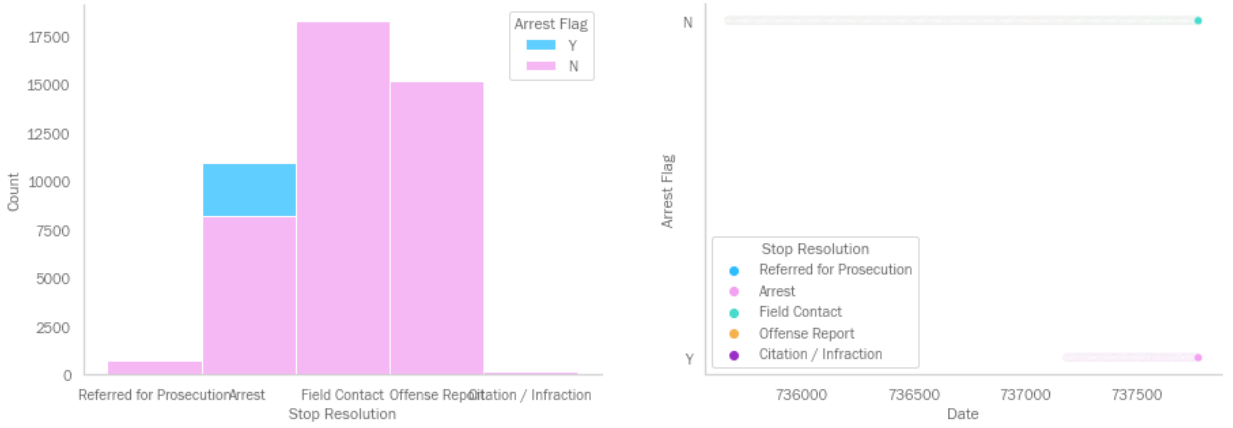
In [64]:
```
sns.scatterplot(x=terry_sort.Date, y=terry_sort['Arrest Flag'])
plt.savefig('Images/date_arrestflag.png');

# This is a clear problem, all the arrests happen after a certain date.  I will subset the data to in
# range where an Arrest Flag of Y is possible.
```



In [65]:
```
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(15,5))
sns.histplot(terry_sort, x='Date', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack', ax=axes[
sns.scatterplot(x=terry_sort.Date, y=terry_sort['Arrest Flag'], ax=axes[1])
plt.savefig('images/date_dual_plot.png')
plt.show()
```



In [66]:
```
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(15,5))
sns.histplot(terry_sort, x='Stop Resolution', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack
axes[1] = sns.scatterplot(x="Date", y="Arrest Flag", hue="Stop Resolution", data=terry_sort)
# plt.savefig('images/date_dual_plot.png')
plt.show()
```

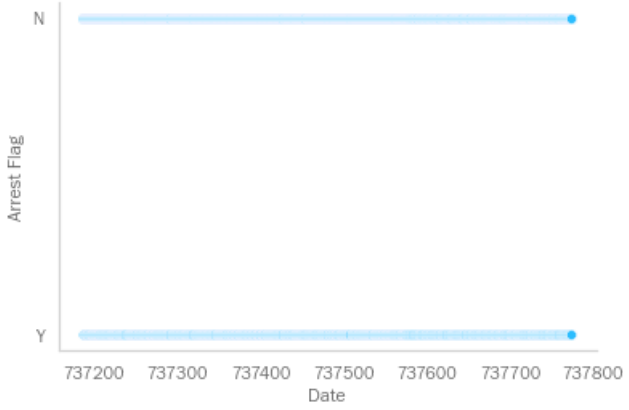


In [67]:
```
terry_sort[terry_sort['Arrest Flag'] == 'Y']['Date'][:10] # this was already sorted on reported date
```

Out[67]:
```
34541    737187
34546    737187
34560    737188
34567    737188
34576    737188
34578    737189
34587    737189
34591    737189
34610    737190
34611    737190
Name: Date, dtype: int64
```

In [68]:
```
terry_sort2 = terry_sort[terry_sort['Date']>=737187]
```

In [69]:
```
sns.scatterplot(x=terry_sort2.Date, y=terry_sort2['Arrest Flag']);
```



In [70]:
```
terry_sort2.info() # We have gone from 45000 entries down to just over 10000
```

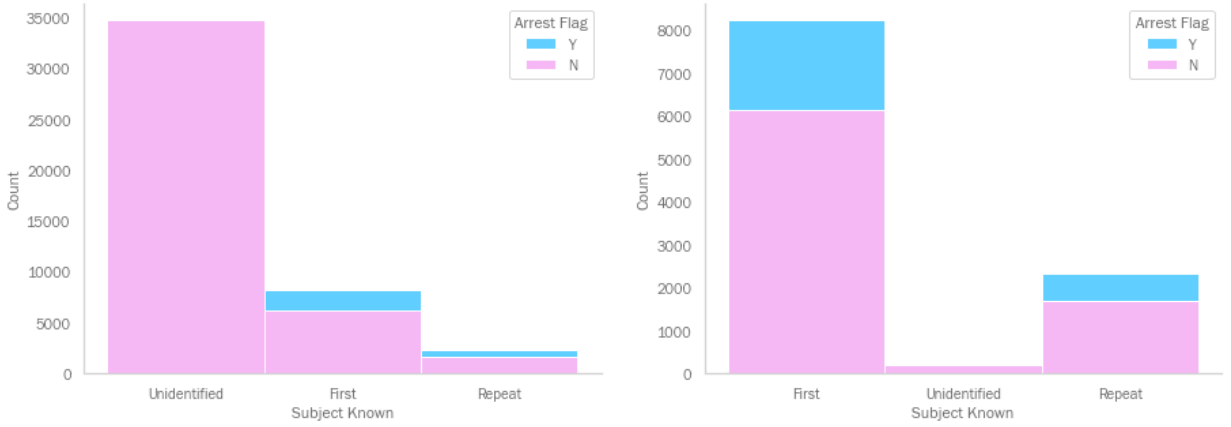
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10756 entries, 34536 to 45291
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Subject Age Group        10756 non-null  object
 1   Stop Resolution          10756 non-null  object
 2   Weapon Type              10756 non-null  object
 3   Officer Gender           10756 non-null  object
 4   Officer Race             10756 non-null  object
 5   Subject Perceived Race   10756 non-null  object
 6   Subject Perceived Gender 10756 non-null  object
 7   Reported Time            10756 non-null  object
 8   Initial Call Type        10756 non-null  object
 9   Final Call Type          10756 non-null  object
 10  Call Type                10756 non-null  object
 11  Officer Squad            10680 non-null  object
 12  Arrest Flag              10756 non-null  object
 13  Frisk Flag               10756 non-null  object
```

In [71]:
```
terry_sort[terry_sort['Stop Resolution'] == 'Arrest']['Date'][:10]

# If we wanted to use Stop Resolution as the target instead of Arrest Flag, then we would have arrest
# back to the beginning of the dataset.  But for the purposes of my model I am looking for a binanry
# Now that we understand the data better, we should also remove Stop Resolution as it is surely confo
```

Out[71]:
```
1     735673
8     735675
13    735675
16    735675
17    735675
30    735676
31    735676
34    735677
45    735679
49    735679
Name: Date, dtype: int64
```

In [72]:
```
terry_sort3 = terry_sort2.drop(columns='Stop Resolution')
```

In [73]:
```
# Now that we have subset, what does Subject Known look like
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(15,5))
sns.histplot(terry_sort, x='Subject Known', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack',
sns.histplot(terry_sort3, x='Subject Known', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack'
plt.savefig('Images/subj_known_comparison.png');
```



Analyze 'Reported Time'

Given that the Reported Time could be anywhere up to 10 hours after the occurrence, I choose to drop this column.

In [74]:
```
terry_sort3['Reported Time'].value_counts()
```

Out[74]:
```
01:41:04    4
17:17:53    3
23:26:15    3
16:44:44    3
19:36:00    3
           ..
23:45:36    1
18:32:14    1
07:58:54    1
06:29:13    1
09:07:05    1
Name: Reported Time, Length: 10077, dtype: int64
```

In [75]:
```
terry_sort3.drop(columns=['Reported Time'], inplace=True)
```

Analyze 'Initial Call Type'

166 unique values, changed - to Unknown.

There are exactly 13071 missing values here and exactly the same missing from Initial Call Type and Call Type. Seems like these were added as a required field in the reporting at some point or these records were all merged over from another system.

There are too many to model on so I will just use 'Call Type'.

In [76]:
```
terry_sort3['Initial Call Type'].value_counts()
```

Out[76]:
```
SUSPICIOUS STOP - OFFICER INITIATED ONVIEW    1334
SUSPICIOUS PERSON, VEHICLE OR INCIDENT         911
DISTURBANCE, MISCELLANEOUS/OTHER               633
TRESPASS                                       562
-                                              525
                                              ...
TRACKING ALARM                                   1
ASSIST SPD - URGENT SERVICE                      1
PURSE SNATCH - ROBBERY                           1
VICE - PORNOGRAPHY                               1
SUSPICIOUS PACKAGE                               1
Name: Initial Call Type, Length: 141, dtype: int64
```

In [77]:
```
terry_sort3['Initial Call Type'] = terry_sort3['Initial Call Type'].replace(to_replace='-',value='Unk
terry_sort3['Initial Call Type'].value_counts()
```

Out[77]:
```
SUSPICIOUS STOP - OFFICER INITIATED ONVIEW    1334
SUSPICIOUS PERSON, VEHICLE OR INCIDENT         911
DISTURBANCE, MISCELLANEOUS/OTHER               633
TRESPASS                                       562
Unknown                                        525
                                              ...
TRU - THEFT                                      1
TRACKING ALARM                                   1
ANIMAL - INJURED, DEAD HAZARD, OTHER             1
MISSING - (ALZHEIMER, ENDANGERED, ELDERLY)       1
SUSPICIOUS PACKAGE                               1
Name: Initial Call Type, Length: 141, dtype: int64
```

In [78]:
```
terry_sort3['Initial Call Type'].value_counts()[:10]
```

Out[78]:
```
SUSPICIOUS STOP - OFFICER INITIATED ONVIEW        1334
SUSPICIOUS PERSON, VEHICLE OR INCIDENT             911
DISTURBANCE, MISCELLANEOUS/OTHER                   633
TRESPASS                                           562
Unknown                                            525
ASLT - IP/JO - WITH OR W/O WPNS (NO SHOOTINGS)     516
SHOPLIFT - THEFT                                   383
THEFT (DOES NOT INCLUDE SHOPLIFT OR SVCS)          341
WEAPN-IP/JO-GUN,DEADLY WPN (NO THRT/ASLT/DIST)     338
FIGHT - IP - PHYSICAL (NO WEAPONS)                 324
Name: Initial Call Type, dtype: int64
```

In [79]:
```
terry_sort3[(terry_sort3['Initial Call Type'] == 'Unknown') & (terry_sort3['Call Type'] != 'Unknown')
```

Out[79]:

	Subject Age Group	Weapon Type	Officer Gender	Officer Race	Subject Perceived Race	Subject Perceived Gender	Initial Call Type	Final Call Type	Call Type	Officer Squad	...	Precinc
34565	46 - 55	None	M	Two or More Races	White	Male	Unknown	-	-	NORTH PCT 1ST W - BOY (JOHN)	...	
34594	18 - 25	None	F	Black or African American	Black or African American	Male	Unknown	-	-	EAST PCT OPS - NIGHT ACT	...	
34595	18 - 25	None	M	White	Unknown	Male	Unknown	-	-	EAST PCT OPS - NIGHT ACT	...	
34618	1 - 17	None	M	White	Black or African American	Female	Unknown	-	-	SOUTH PCT 2ND W - OCEAN RELIEF	...	Sout
34650	36 - 45	Knife/Cutting/Stabbing Instrument	M	Unknown	White	Male	Unknown	-	-	NaN	...	
...	...	...	...	...	...	...	...	...	...	...	...	.
45257	36 - 45	None	F	Two or More Races	Unknown	Male	Unknown	-	-	CRG - SQUAD 82A	...	Nort
45260	18 - 25	None	M	White	White	Female	Unknown	-	-	EAST PCT 3RD W - GEORGE	...	Eas
45261	26 - 35	None	M	White	Unknown	Male	Unknown	-	-	CRG - SQUAD 82D	...	Nort
45264	Unknown	None	M	White	Black or African American	Female	Unknown	-	-	SOUTH PCT 3RD W - SAM	...	Sout
45275	18 - 25	None	M	White	White	Male	Unknown	-	-	WEST PCT 1ST W - KQ/DM RELIEF	...	Wes

525 rows × 22 columns

In [80]:
```
terry_sort3.drop(columns='Initial Call Type', inplace=True)
```

Analyze 'Final Call Type'

205 unique values, changed - to Unknown. There are too many to model on so I will just use 'Call Type'.

In [81]:
```
terry_sort3['Final Call Type'].value_counts()
```

Out[81]:
```
--SUSPICIOUS CIRCUM. - SUSPICIOUS PERSON            1462
--PROWLER - TRESPASS                                1219
--DISTURBANCE - OTHER                                778
--ASSAULTS, OTHER                                    597
-                                                    525
                                                    ...
WARRANT PICKUP - FROM OTHER AGENCY                     1
--CASUALTY NON-TRAF NON-CRIM - DRUG RELATED (OD)       1
--DV - DOMESTIC THREATS BY PHONE OR WRITING            1
--MISC MISD AND VIOLS - PARKS EXCLUSION               1
TRAFFIC - BLOCKING ROADWAY                             1
Name: Final Call Type, Length: 180, dtype: int64
```

In [82]:
```
terry_sort3['Final Call Type'] = terry_sort3['Final Call Type'].replace(to_replace='-',value='Unknown
terry_sort3['Final Call Type'].value_counts()
```

Out[82]:
```
--SUSPICIOUS CIRCUM. - SUSPICIOUS PERSON       1462
--PROWLER - TRESPASS                           1219
--DISTURBANCE - OTHER                           778
--ASSAULTS, OTHER                               597
Unknown                                         525
                                               ...
--CROWD MGMNT (STAND BY ONLY)                     1
PROWLER                                           1
PEACE-STANDBY TO ASSURE (NO COURT ORDR SVC)       1
--ANIMAL COMPLAINT - INJURED,DEAD,DANGEROUS       1
--CRISIS COMPLAINT - PICK-UP OR TRANSPORT         1
Name: Final Call Type, Length: 180, dtype: int64
```

In [83]:
```
terry_sort3.drop(columns='Final Call Type', inplace=True)
```

Analyze 'Call Type'

7 unique values, changed - to Unknown

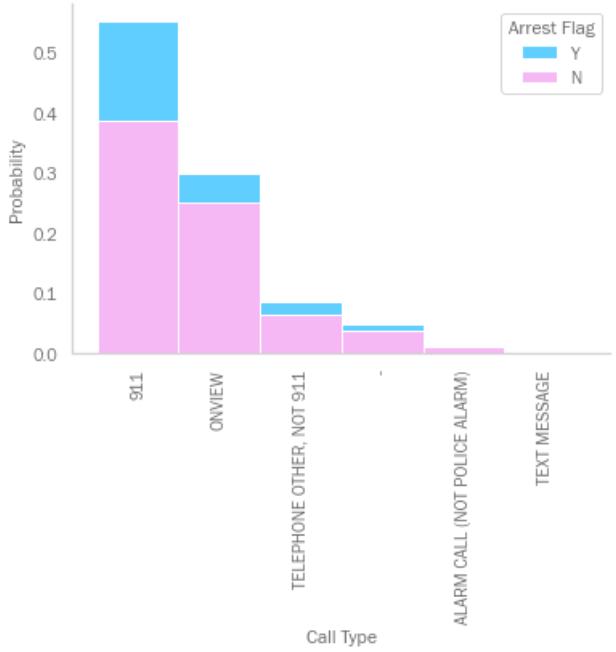
In [84]:
```
terry_sort3['Call Type'].value_counts()
```

Out[84]:
```
911                            5949
ONVIEW                         3212
TELEPHONE OTHER, NOT 911        926
-                               525
ALARM CALL (NOT POLICE ALARM)   138
TEXT MESSAGE                      6
Name: Call Type, dtype: int64
```

In [85]:
```
terry_sort3['Call Type'] = terry_sort3['Call Type'].replace(to_replace='-',value='Unknown')
terry_sort3['Call Type'].value_counts()
```

Out[85]:
```
911                            5949
ONVIEW                         3212
TELEPHONE OTHER, NOT 911        926
Unknown                         525
ALARM CALL (NOT POLICE ALARM)   138
TEXT MESSAGE                      6
Name: Call Type, dtype: int64
```

In [86]:
```
sns.histplot(terry_sort2, x='Call Type', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack', st
plt.xticks(rotation=90)
plt.savefig('Images/call_type.png');
```



Analyze 'Officer Squad'

Replace nan values with Unknown, 170 unique values. Binned into Training, Unknown, and Other. Training was most common.

In [87]:
```
terry_sort3['Officer Squad'].unique()
```

Out[87]:
```
array(['WEST PCT 3RD W - K/Q RELIEF', 'SOUTH PCT 3RD W - SAM',
       'NORTH PCT OPS - ACT DAY', 'EAST PCT 1ST W - E/G RELIEF (CHARLIE)',
       'SOUTH PCT 1ST W - R/S RELIEF', 'SOUTH PCT 1ST W - SAM',
       'WEST PCT 1ST W - DAVID/MARY', 'SOUTHWEST PCT 2ND W - FRANK',
       'SOUTHWEST PCT - 1ST WATCH - F/W RELIEF', 'WEST PCT 2ND W - DAVID',
       'NORTH PCT 2ND WATCH - NORTH BEATS',
       'SOUTHWEST PCT 2ND WATCH - F/W RELIEF',
       'SOUTHWEST PCT - 3RD WATCH - F/W RELIEF',
       'NORTH PCT 3RD W - B/N RELIEF', 'NORTH PCT 1ST W - LINCOLN',
       'NORTH PCT 1ST W - BOY (JOHN)', 'SOUTH PCT 1ST W - OCEAN',
       'NORTH PCT 2ND W - NORA', 'SOUTH PCT 2ND W - SAM',
       'SOUTHWEST PCT 2ND W - SOUTHWEST BEATS',
       'SOUTH PCT 3RD W - ROBERT', 'WEST PCT 3RD W - QUEEN',
       'EAST PCT 2ND W - EDWARD', 'NAVIGATION TEAM - SQUAD A',
       'SOUTH PCT 2ND W - SAM BEATS', 'NORTH PCT 2ND W - BOY',
       'EAST PCT 2ND W - CHARLIE RELIEF', 'EAST PCT OPS - NIGHT ACT',
       'EAST PCT 3RD W - EAST BEATS', 'EAST PCT 3RD W - CHARLIE',
       'SOUTHWEST PCT 3RD W - WILLIAM', 'TRAINING - FIELD TRAINING SQUAD',
       'SOUTH PCT 2ND W - ROBERT', 'WEST PCT 2ND W - KING BEATS',
       'SOUTHWEST PCT 2ND W - WILLIAM', 'EAST PCT 3RD W - GEORGE',
       'WEST PCT 3RD W - KING', 'SOUTH PCT 3RD W - OCEAN',
       'SOUTH PCT 2ND W - OCEAN RELIEF', 'WEST PCT 1ST W - KING/QUEEN',
       'WEST PCT 3RD W - DAVID BEATS', 'WEST PCT 2ND W - D/M RELIEF',
       'EAST PCT 3RD W - E/G RELIEF', 'SOUTHWEST PCT 3RD W - FRANK', nan,
       'WEST PCT 2ND W - K/Q RELIEF', 'WEST PCT 2ND W - KING',
       'NORTH PCT 3RD W - JOHN', 'WEST PCT 3RD W - MARY',
       'SOUTH PCT 1ST W - ROBERT', 'WEST PCT 1ST W - KQ/DM RELIEF',
       'EAST PCT 1ST W - EDWARD (CHARLIE)', 'WEST PCT 2ND W - MARY BEATS',
       'NORTH PCT 3RD W - NORA', 'EAST PCT 3RD W - EDWARD',
       'NORTH PCT 2ND W - JOHN', 'NORTH PCT 2ND W - L/U RELIEF',
       'NORTH PCT 3RD W - UNION', 'SOUTHWEST PCT 1ST W - WILLIAM',
       'EAST PCT 2ND W - E/G RELIEF', 'JOINT ENFORCEMENT TEAM',
       'SOUTH PCT 2ND W - R/S RELIEF',
       'EAST PCT 3RD WATCH - CHARLIE RELIEF',
       'NORTH PCT 2ND W - JOHN RELIEF', 'NORTH PCT 3RD W - L/U RELIEF',
       'NORTH PCT 1ST W - UNION', 'SOUTHWEST PCT 1ST W - FRANK',
       'SOUTH PCT 3RD W - R/S RELIEF', 'WEST PCT 3RD W - D/M RELIEF',
       'NORTH PCT 2ND W - UNION', 'WEST PCT 2ND W - QUEEN',
       'WEST PCT 3RD W - DAVID', 'SOUTH PCT OPS - DAY ACT',
       'HR - BLEA - ACADEMY RECRUITS', 'NORTH PCT 3RD W - BOY',
       'SOUTHWEST PCT OPS - NIGHT ACT', 'WEST PCT 2ND W - DAVID BEATS',
       'NORTH PCT 3RD W - JOHN RELIEF',
       'EAST PCT 1ST W - GEORGE (CHARLIE)', 'WEST PCT 2ND W - MARY',
       'SWAT - NIGHT SQUAD 1', 'NORTH PCT 1ST W - L/U RELIEF',
       'EAST PCT 2ND W - CHARLIE', 'WEST PCT OPS - ACT NIGHT',
       'WEST PCT OPS - CPT', 'NORTH PCT 2ND WATCH - B/N RELIEF',
       'CANINE - SQUAD C', 'EAST PCT 2ND W - BEATS',
       'NORTH PCT 1ST W - NORA (JOHN)', 'NORTH PCT 2ND W - LINCOLN',
       'SOUTH PCT 3RD W - OCEAN RELIEF', 'SOUTH PCT OPS - CPT',
       'CRISIS RESPONSE SQUAD', 'GANG SQUAD A', 'SOUTH PCT 2ND W - OCEAN',
       'CANINE - SQUAD A', 'AUTO THEFT', 'NORTH PCT OPS - CPT',
       'NORTH PCT 3RD W - LINCOLN',
       'COMM - INTERNET AND TELEPHONE REPORTING (ITRU)',
       'EAST PCT 2ND W - GEORGE', 'MAJOR CRIMES TASK FORCE',
       'SOUTH PCT OPS - BURG/THEFT', 'GANG SQUAD C',
       'NORTH PCT 1ST W - B/N RELIEF (JOHN)', 'WEST PCT OPS - BURG/THEFT',
       'TRAF - PM ENFORCEMENT', 'HR - DEPARTMENT UNAVAILABLE PERSONNEL',
       'RECORDS - DAY SHIFT', 'COMMUNITY OUTREACH UNIT',
       'WEST PCT 2ND W - SPECIAL BEATS', 'SWAT - DAY SQUAD 2',
       'TRAF - DUI SQUAD', 'CRISIS INTERVENTION COORDINATION SQUAD',
       'DV SQUAD C - ELDER ABUSE', 'TRAINING - ADVANCED - SQUAD C',
       'TRAF - MOTORCYCLE UNIT - T4 SQUAD', 'SAU SQUAD B',
       'HARBOR - SQUAD D', 'SOUTHWEST PCT OPS - CPT', 'ROBBERY SQUAD A',
       'EAST PCT OPS - CPT', 'DV SQUAD D - ORDER SERVICE',
       'NAVIGATION TEAM - SQUAD B', 'TRAF - AM ENFORCEMENT',
       'CRG - SQUAD 82A', 'CRG - SQUAD 82D', 'CRG - SQUAD 82E',
       'CRG - SQUAD 81B', 'CRG - SQUAD 81A', 'CRG - SQUAD 81C',
```

```
       'CRG - SQUAD 81E', 'CRG - SQUAD 81D', 'CRG - SQUAD 82C',
       'CRG - SQUAD 82B'], dtype=object)
```

In [88]:
```
terry_sort3['Officer Squad'].fillna('Unknown', inplace=True)
```

In [89]:
```
terry_sort3['Officer Squad'].value_counts()[:30]
```

Out[89]:
```
TRAINING - FIELD TRAINING SQUAD          1206
WEST PCT 3RD W - D/M RELIEF               400
WEST PCT 1ST W - DAVID/MARY               351
WEST PCT 2ND W - D/M RELIEF               309
WEST PCT 1ST W - KING/QUEEN               293
WEST PCT 2ND W - KING                     264
WEST PCT 3RD W - KING                     228
WEST PCT 3RD W - QUEEN                    212
WEST PCT 3RD W - DAVID                    207
WEST PCT 3RD W - K/Q RELIEF               206
NORTH PCT 3RD W - B/N RELIEF              197
SOUTHWEST PCT 2ND W - FRANK               193
WEST PCT 2ND W - K/Q RELIEF               190
WEST PCT 3RD W - MARY                     189
WEST PCT 2ND W - DAVID                    187
EAST PCT 3RD W - EDWARD                   184
WEST PCT 1ST W - KQ/DM RELIEF             176
EAST PCT 1ST W - E/G RELIEF (CHARLIE)     169
SOUTH PCT 3RD W - SAM                     166
NORTH PCT OPS - ACT DAY                   163
NORTH PCT 1ST W - BOY (JOHN)              156
SOUTH PCT 1ST W - R/S RELIEF              155
EAST PCT 3RD W - E/G RELIEF               154
EAST PCT 2ND W - CHARLIE RELIEF           147
SOUTH PCT 1ST W - SAM                     147
NORTH PCT 3RD W - UNION                   145
SOUTH PCT 1ST W - OCEAN                   139
SOUTH PCT 2ND W - R/S RELIEF              124
SOUTHWEST PCT 2ND W - WILLIAM             114
WEST PCT 2ND W - MARY BEATS               110
Name: Officer Squad, dtype: int64
```

In [90]:
```
terry_sort3[['Officer Squad','Precinct']][:-100]
```

Out[90]:

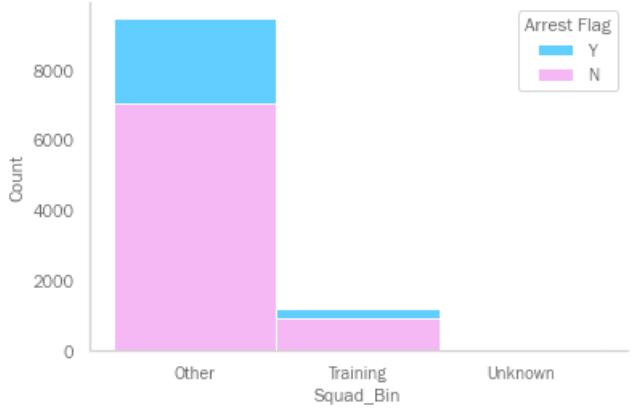
	Officer Squad	Precinct
34536	WEST PCT 3RD W - K/Q RELIEF	-
34537	SOUTH PCT 3RD W - SAM	South
34538	NORTH PCT OPS - ACT DAY	North
34539	NORTH PCT OPS - ACT DAY	North
34540	NORTH PCT OPS - ACT DAY	North
...	...	...
45187	EAST PCT 2ND W - E/G RELIEF	East
45188	WEST PCT 2ND W - DAVID BEATS	West
45189	SOUTH PCT 2ND W - ROBERT	-
45190	CRG - SQUAD 81A	West
45191	NORTH PCT 3RD W - UNION	North

10656 rows × 2 columns

In [91]:
```
# It seems to me like most Squads begin with the Precinct except for the Training - Field Training Sq
# Rather than just deleting the squad, I think binning into Field Training, Other, and Unknown will g

# Bin into top 10 and then Other
Squad_Bin =[]
for i in range(len(terry_sort3)):
    if terry_sort3.iloc[i]['Officer Squad'] == 'Unknown':
        Squad_Bin.append('Unknown')
    elif terry_sort3.iloc[i]['Officer Squad'] == 'TRAINING - FIELD TRAINING SQUAD':
        Squad_Bin.append('Training')
    else:
        Squad_Bin.append('Other')

terry_sort3['Squad_Bin'] = Squad_Bin
```

In [92]:
```
terry_sort3.drop(columns=('Officer Squad'), inplace=True)
```

In [93]:
```
sns.histplot(terry_sort3, x='Squad_Bin', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack');
```



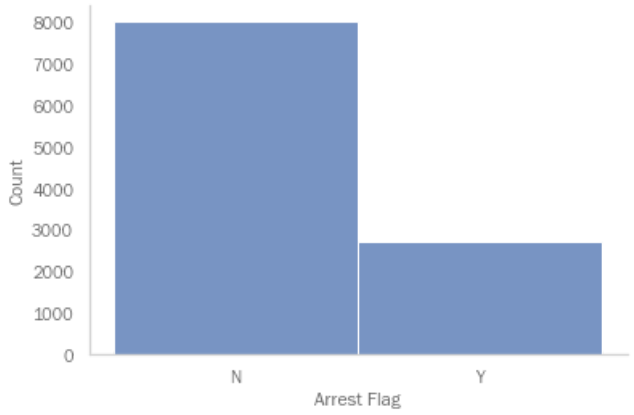
Analyze 'Arrest Flag'

Target variable. Fine as is... values N, Y. Need to address class imbalance.

In [94]:
```
terry_sort3['Arrest Flag'].value_counts() # Definite class imbalance, may need to address
```

Out[94]:
```
N    8036
Y    2720
Name: Arrest Flag, dtype: int64
```

In [95]:
```
sns.histplot(terry_sort3, x='Arrest Flag');
```



Analyze 'Frisk Flag'

Changed - to Unknown. Values N, Y, Unknown

In [96]:
```
terry_sort3['Frisk Flag'].value_counts()
```

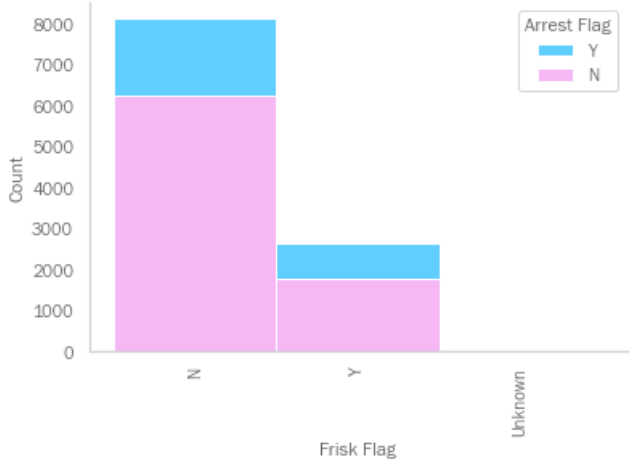
Out[96]:
```
N    8116
Y    2639
-       1
Name: Frisk Flag, dtype: int64
```

In [97]:
```
terry_sort3['Frisk Flag'] = terry_sort['Frisk Flag'].replace(to_replace='-',value='Unknown')
```

In [98]:
```
terry_sort3['Frisk Flag'].value_counts()
```

Out[98]:
```
N          8116
Y          2639
Unknown       1
Name: Frisk Flag, dtype: int64
```

In [99]:
```
sns.histplot(terry_sort3, x='Frisk Flag', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack')
plt.xticks(rotation=90);
```



Analyze 'Precinct'

Change - to Unknown, combine Southwest and SouthWest. Eight values including error.

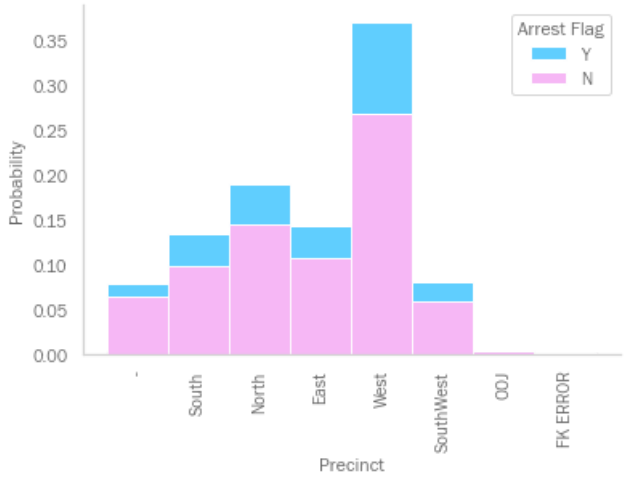
In [100]:
```
terry_sort3['Precinct'].value_counts()
```

Out[100]:
```
West          3984
North         2045
East          1530
South         1448
SouthWest      859
-              845
OOJ             30
FK ERROR        15
Name: Precinct, dtype: int64
```

In [101]:
```
terry_sort3['Precinct'] = terry_sort3['Precinct'].replace(to_replace='-',value='Unknown')
# terry_sort3['Precinct'] = terry_sort3['Precinct'].replace(to_replace='SouthWest',value='Southwest')
terry_sort3['Precinct'].value_counts()
```

Out[101]:
```
West          3984
North         2045
East          1530
South         1448
SouthWest      859
Unknown        845
OOJ             30
FK ERROR        15
Name: Precinct, dtype: int64
```

In [102]:
```
# Interesting amount of Arrests in the West precinct

sns.histplot(terry_sort2, x='Precinct', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack', sta
plt.xticks(rotation=90)
plt.savefig('Images/precinct.png');
```



Analyze 'Sector'

Strip spaces and replace - with Unknown. Left with under 20 values. Too many to work with, will drop.

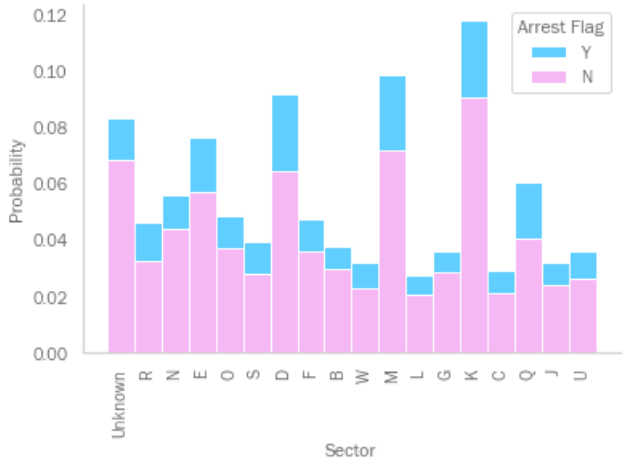
In [103]:
```
terry_sort3['Sector'].value_counts()
```

Out[103]:
```
K    1270
M    1064
D     993
-     896
E     823
Q     653
N     603
O     523
F     513
R     498
S     428
B     407
G     389
U     386
J     349
W     345
C     317
L     299
Name: Sector, dtype: int64
```

In [104]:
```
terry_sort3['Sector'] = terry_sort3['Sector'].replace(to_replace='-',value='Unknown')
terry_sort3['Sector'] = terry_sort3['Sector'].map(lambda x: x.strip())
terry_sort3['Sector'].value_counts()
```

Out[104]:
```
K          1270
M          1064
D           993
Unknown     896
E           823
Q           653
N           603
O           523
F           513
R           498
S           428
B           407
G           389
U           386
J           349
W           345
C           317
L           299
Name: Sector, dtype: int64
```

In [105]:
```
sns.histplot(terry_sort3, x='Sector', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack', stat=
plt.xticks(rotation=90);
```



In [106]:
```
terry_sort3.drop(columns=('Sector'), inplace = True)
```

Analyze 'Beat'

Strip spaces and replace - with Unknown. I think this is getting very granular in the data. Perhaps if a particular Precinct and Sector are significant to the models then we can also look at Beat. But for now I think it just complicates the model too much so I will drop it.

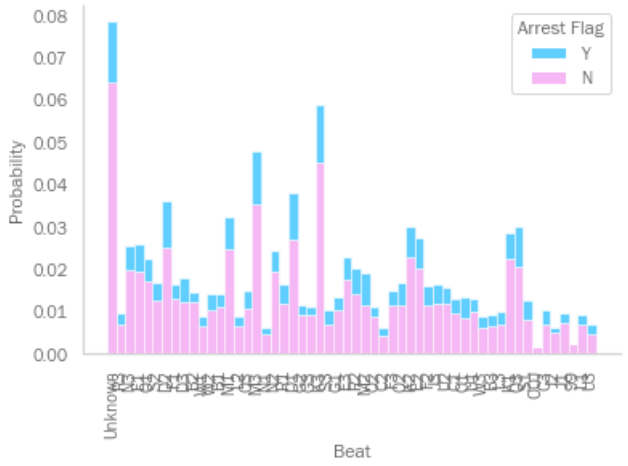
In [107]:
```
terry_sort3['Beat'].value_counts()
```

Out[107]:
```
-      845
K3     635
M3     515
D1     408
D2     390
M1     348
K2     326
Q3     325
K1     309
E2     294
E1     280
N3     276
N2     262
E3     248
O1     243
R2     216
M2     205
D3     195
Q2     182
S2     180
F1     179
R1     177
J3     176
F2     175
U2     168
O3     161
F3     159
B2     157
B1     153
W2     152
Q1     146
G1     143
U1     141
C1     140
S1     136
G2     123
G3     122
O2     119
S3     112
C3     111
L3     106
J1     105
R3     105
B3      98
L1      98
W1      97
W3      96
L2      95
U3      77
J2      68
C2      67
N1      65
99      27
OOJ     20
Name: Beat, dtype: int64
```

In [108]:
```
terry_sort3['Beat'] = terry_sort3['Beat'].replace(to_replace='-',value='Unknown')
terry_sort3['Beat'] = terry_sort3['Beat'].map(lambda x: x.strip())
terry_sort3['Beat'].unique()
```

Out[108]:
```
array(['Unknown', 'R3', 'N3', 'E1', 'O1', 'S2', 'D2', 'F1', 'D3', 'B2',
       'W1', 'W2', 'B1', 'M1', 'L2', 'O3', 'M3', 'N1', 'N2', 'R1', 'D1',
       'G2', 'G3', 'K3', 'S3', 'G1', 'E3', 'R2', 'M2', 'O2', 'C2', 'F3',
       'Q2', 'K2', 'E2', 'F2', 'J3', 'U2', 'C1', 'Q1', 'U1', 'W3', 'B3',
       'L3', 'K1', 'Q3', 'S1', 'OOJ', 'C3', 'J2', 'J1', '99', 'L1', 'U3'],
      dtype=object)
```

In [109]:
```
# Don't see an obvious standout values
sns.histplot(terry_sort3, x='Beat', hue_order=('Y','N'), hue='Arrest Flag', multiple='stack', stat='p
plt.xticks(rotation=90);
```



In [110]:
```
terry_sort3.drop(columns='Beat', inplace=True)
```

Rename variables and visualize relationships

In [111]:
```
# First let's fix all of the variable names so they don't have spaces
terry_sort3.columns = terry_sort3.columns.str.replace(' ', '_')
terry_sort3.columns
```

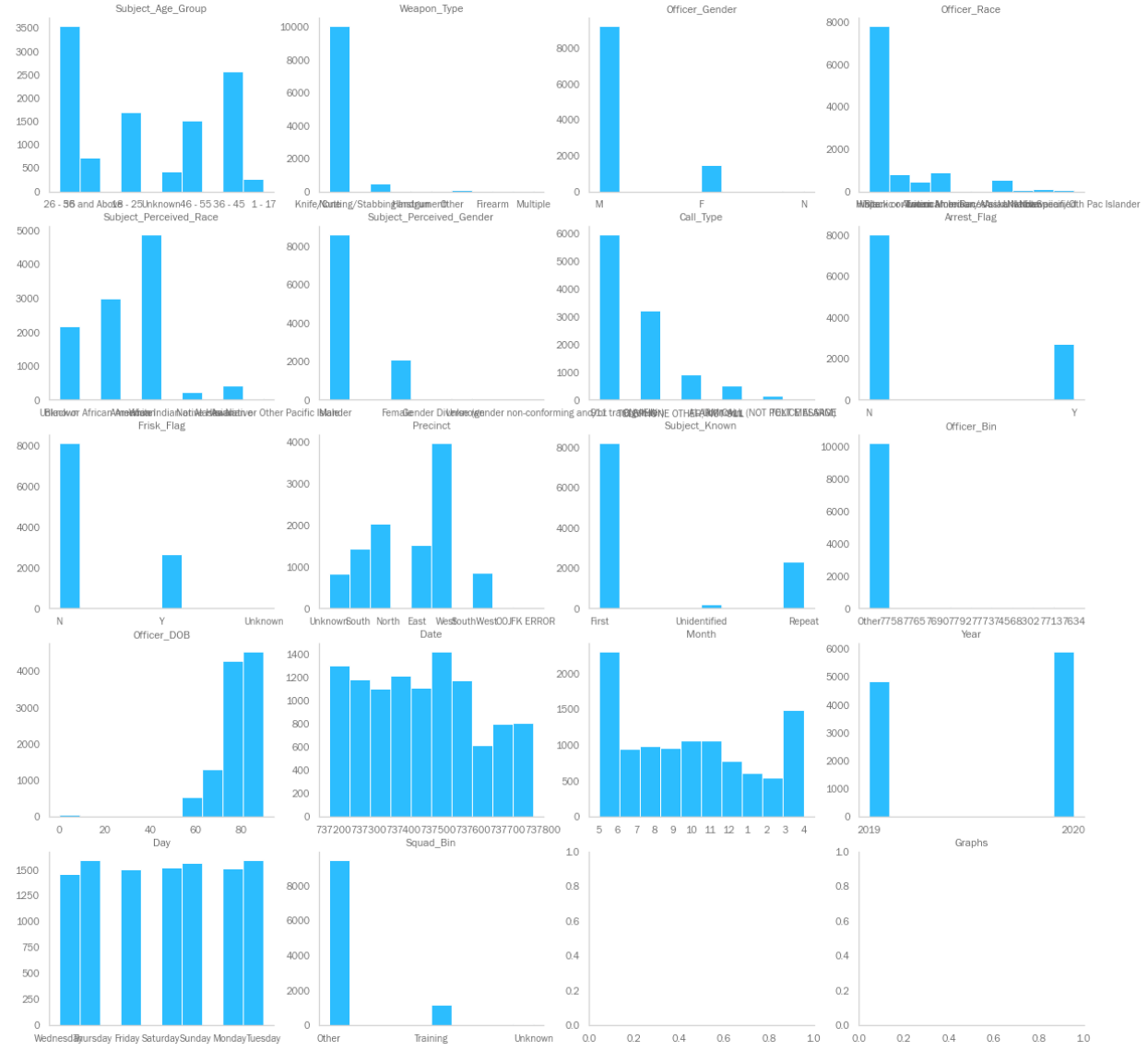
Out[111]:
```
Index(['Subject_Age_Group', 'Weapon_Type', 'Officer_Gender', 'Officer_Race',
       'Subject_Perceived_Race', 'Subject_Perceived_Gender', 'Call_Type',
       'Arrest_Flag', 'Frisk_Flag', 'Precinct', 'Subject_Known', 'Officer_Bin',
       'Officer_DOB', 'Date', 'Month', 'Year', 'Day', 'Squad_Bin'],
      dtype='object')
```

In [112]:
```
terry_sort3.info()
```

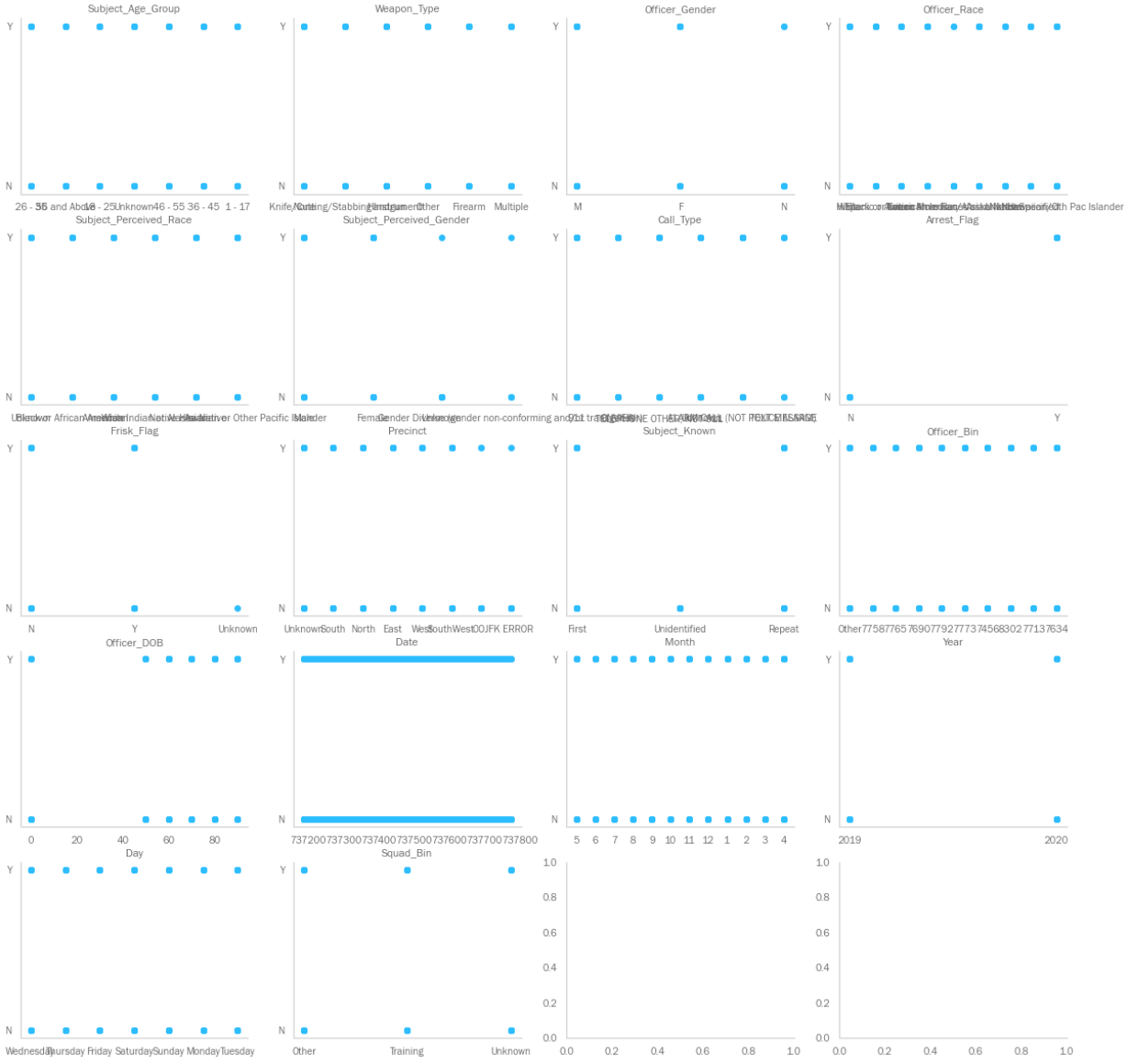
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10756 entries, 34536 to 45291
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype   
---  ------                   --------------  -----   
 0   Subject_Age_Group        10756 non-null  object  
 1   Weapon_Type              10756 non-null  object  
 2   Officer_Gender           10756 non-null  object  
 3   Officer_Race             10756 non-null  object  
 4   Subject_Perceived_Race   10756 non-null  object  
 5   Subject_Perceived_Gender 10756 non-null  object  
 6   Call_Type                10756 non-null  object  
 7   Arrest_Flag              10756 non-null  object  
 8   Frisk_Flag               10756 non-null  object  
 9   Precinct                 10756 non-null  object  
 10  Subject_Known            10756 non-null  object  
 11  Officer_Bin              10756 non-null  object  
 12  Officer_DOB              10756 non-null  category
 13  Date                     10756 non-null  int64   
 14  Month                    10756 non-null  object  
 15  Year                     10756 non-null  object  
 16  Day                      10756 non-null  object  
 17  Squad_Bin                10756 non-null  object  
dtypes: category(1), int64(1), object(16)
memory usage: 1.5+ MB
```

In [113]:

```
fig, axes = plt.subplots(nrows=5, ncols=4,figsize=(20,20))
plt.title('Graphs')
y = terry_sort3['Arrest_Flag']
for n in range(len(terry_sort3.columns)):
    row=(n)//4
    col=n%4
    ax=axes[row][col]
    x=terry_sort3.iloc[:,n]
    ax.hist(x)
    ax.set_title(terry_sort3.columns.values[n])
# plt.savefig('images/initial_distributions.png')
plt.show()
```



In [114]:
```
fig, axes = plt.subplots(nrows=5, ncols=4,figsize=(20,20))
for n, column in enumerate(terry_sort3.columns):
    row=(n)//4
    col=n%4
    ax=axes[row][col]
    x=terry_sort3[column]
    y=terry_sort3.Arrest_Flag
    ax.scatter(x, y)
    ax.set_title(terry_sort3.columns.values[n])
plt.show()
```



In [115]:
```
# Then change our target variable to binary encoding (0 or 1 instead of N or Y)
terry_sort3['Arrest_Flag']=terry_sort3['Arrest_Flag'].replace(to_replace = 'Y', value = 1).replace(to
y = terry_sort3['Arrest_Flag']
```

In [116]:
```
terry_sort3.columns
```

Out[116]:
```
Index(['Subject_Age_Group', 'Weapon_Type', 'Officer_Gender', 'Officer_Race',
       'Subject_Perceived_Race', 'Subject_Perceived_Gender', 'Call_Type',
       'Arrest_Flag', 'Frisk_Flag', 'Precinct', 'Subject_Known', 'Officer_Bin',
       'Officer_DOB', 'Date', 'Month', 'Year', 'Day', 'Squad_Bin'],
      dtype='object')
```

In [238]:
```
# Check ANOVA test to examine possible confounding variables

import statsmodels.api as sm
from statsmodels.formula.api import ols

formula = 'Arrest_Flag ~ C(Subject_Age_Group) + C(Weapon_Type) + C(Officer_Gender) + C(Officer_Race)

lm = ols(formula, terry_sort3).fit()
table = sm.stats.anova_lm(lm, typ=2)
print(table)

# With the data already subset and Stop Resolution removed, I do not see any major issues with the AN
# Subject Known had previously been an issue for me as nearly all arrests happened outside of Subject
```

	sum_sq	df	F	PR(>F)
C(Subject_Age_Group)	12.259443	6.0	11.692641	4.207503e-13
C(Weapon_Type)	6.507831	5.0	7.448338	5.506503e-07
C(Officer_Gender)	1.683814	2.0	4.817895	8.101370e-03
C(Officer_Race)	4.222644	8.0	3.020561	2.167795e-03
C(Subject_Perceived_Race)	36.960724	5.0	42.302265	2.668606e-43
C(Subject_Perceived_Gender)	0.134763	3.0	0.257064	8.563421e-01
C(Call_Type)	34.283302	5.0	39.237902	4.399923e-40
C(Frisk_Flag)	4.718219	2.0	13.500233	1.394199e-06
C(Precinct)	8.506451	7.0	6.954141	2.742377e-08
C(Subject_Known)	0.579100	2.0	1.656978	1.907635e-01
C(Officer_Bin)	10.730852	9.0	6.823149	7.685538e-10
C(Officer_DOB)	2.521361	6.0	2.404789	3.453019e-02
C(Month)	4.196245	11.0	2.183038	1.274772e-02
C(Year)	0.519937	1.0	2.975391	8.456866e-02
C(Day)	0.602115	6.0	0.574277	7.511726e-01
C(Squad_Bin)	0.040441	2.0	0.115713	8.907315e-01
Date	0.451261	1.0	2.582385	1.080883e-01
Residual	1865.586620	10676.0	NaN	NaN

```
C:\Users\melod\anaconda3\envs\learn-env\lib\site-packages\statsmodels\base\model.py:1832: ValueWarni
ng: covariance of constraints does not have full rank. The number of constraints is 6, but rank is 5
  warnings.warn('covariance of constraints does not have full '
```

Train, test, split

Output - X, y, X_train, X_test, y_train, y_test, train_df, test_df

In [118]:
```
terry_sort3.reset_index(drop=True, inplace=True)
terry_sort3.head()
```

Out[118]:

	Subject_Age_Group	Weapon_Type	Officer_Gender	Officer_Race	Subject_Perceived_Race	Subject_Perceived_Gender	Cal
0	26 - 35	None	M	White	Unknown	Male	
1	56 and Above	None	M	White	Black or African American	Male	OI
2	26 - 35	None	M	White	White	Female	OI
3	26 - 35	None	M	White	White	Male	OI
4	18 - 25	None	M	White	White	Male	OI

In [119]:
```
X = terry_sort3.drop(columns='Arrest_Flag')
y = terry_sort3['Arrest_Flag']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=17)
train_df = pd.concat([X_train, y_train], axis=1)
test_df = pd.concat([X_test, y_test], axis=1)

print('X_train: ', X_train.shape, '\nX_test: ', X_test.shape)
```

```
X_train:  (8067, 17)
X_test:  (2689, 17)
```

Perform SMOTENC for class imbalance

Output - X_train_SMOTE, y_train_SMOTE (still X_test and y_test)

In [120]:
```
X_train.columns # Column 12 is our one numerical column, the rest are categorical.  Need this for SMO
```

Out[120]:
```
Index(['Subject_Age_Group', 'Weapon_Type', 'Officer_Gender', 'Officer_Race',
       'Subject_Perceived_Race', 'Subject_Perceived_Gender', 'Call_Type',
       'Frisk_Flag', 'Precinct', 'Subject_Known', 'Officer_Bin', 'Officer_DOB',
       'Date', 'Month', 'Year', 'Day', 'Squad_Bin'],
      dtype='object')
```

In [121]:
```
#  One of the labs did SMOTE before the train test split.  I think maybe it is better to do after the
#   not making any changes to my test data.  However, I have learned to do it before to OHE or my dum
# get incorrect values.
from imblearn.over_sampling import SMOTENC
# Previous original class distribution
print('Original class distribution: \n')
print(y.value_counts())

smote = SMOTENC(random_state=10, categorical_features=[0,1,2,3,4,5,6,7,8,9,10,11,13,14,15,16])
X_train_SMOTE, y_train_SMOTE = smote.fit_resample(X_train, y_train)
# Preview synthetic sample class distribution
print('----------------------------------------')
print('Synthetic sample class distribution: \n')
print(pd.Series(y_train_SMOTE).value_counts())
```

```
Original class distribution:

0    8036
1    2720
Name: Arrest_Flag, dtype: int64
----------------------------------------
Synthetic sample class distribution:

1    6055
0    6055
Name: Arrest_Flag, dtype: int64
```

In [122]:
```
X_train_SMOTE.Weapon_Type.value_counts() # Just checking SMOTEd data values
```

Out[122]:
```
None                                11469
Knife/Cutting/Stabbing Instrument     483
Other                                  61
Handgun                                44
Firearm                                33
Multiple                               20
Name: Weapon_Type, dtype: int64
```

In [123]:
```
X_train_SMOTE.isna().sum() # No NaN values introduced in the SMOTE process
```

Out[123]:
```
Subject_Age_Group          0
Weapon_Type                0
Officer_Gender             0
Officer_Race               0
Subject_Perceived_Race     0
Subject_Perceived_Gender   0
Call_Type                  0
Frisk_Flag                 0
Precinct                   0
Subject_Known              0
Officer_Bin                0
Officer_DOB                0
Date                       0
Month                      0
Year                       0
Day                        0
Squad_Bin                  0
dtype: int64
```

In [124]:
```
X_train_SMOTE.shape
```

Out[124]:
```
(12110, 17)
```

(OHE) Get dummy variables for categoricals

Output - X_train_encoded, X_test_encoded dataframes (still y_train_SMOTE and y_test)

In [125]:
```
# Hang on

# I did the SMOTE before the OHE so I didn't get non-binary values in my categoricals
# But now I want to fit my models to the SMOTEd data, but then predict on non-SMOTEd data, but I need
# Can I go back to the non-SMOTEd data and do the OHE on that for predictions?
# SMOTE shouldnt have changed any variables, just added data points, and I need to get rid of them to
# What about scaling? Do I go back and scale on non-SMOTEd data too?

# Going to try it.  Will have X_train_encoded for non-SMOTE and X_train_encoded_SMOTE for SMOTEd.
# Same for test... no test didn't get SMOTEd, just X_train_resampled and y_train_resampled.
# And I have a regular y_train and a y_train_resampled, but they don't get SMOTEd or scaled
```

In [126]:
```
#Originally I used OHE when I had Officer ID as a variable instead of Officer_Bin.  In that case it w
#on all of the X data in case there were values in Test that were not in Train.  Now with the binning
enc = OneHotEncoder(sparse=False)
X_cat = X.drop(columns='Date')
enc.fit(X_cat)
X_train_code_SMOTE = enc.transform(X_train_SMOTE.drop(columns='Date'))
X_train_code = enc.transform(X_train.drop(columns='Date'))
X_test_code = enc.transform(X_test.drop(columns='Date'))
```

In [127]:
```
# The transforms are not dataframes yet and we have to add back in the continuous column of Date
X_train_encoded_SMOTE = pd.DataFrame(X_train_code_SMOTE, columns=enc.get_feature_names(X_cat.columns)
X_train_encoded_SMOTE['Date'] = X_train_SMOTE.Date
X_train_encoded = pd.DataFrame(X_train_code, columns=enc.get_feature_names(X_cat.columns), index=X_tr
X_train_encoded['Date'] = X_train.Date
X_test_encoded = pd.DataFrame(X_test_code, columns=enc.get_feature_names(X_cat.columns), index=X_test
X_test_encoded['Date'] = X_test.Date
X_test_encoded.head()
```

Out[127]:

	Subject_Age_Group_1 - 17	Subject_Age_Group_18 - 25	Subject_Age_Group_26 - 35	Subject_Age_Group_36 - 45	Subject_Age_Group_46 - 55
8213	0.0	0.0	0.0	0.0	0.0
2520	0.0	1.0	0.0	0.0	0.0
6475	0.0	0.0	0.0	0.0	0.0
3849	0.0	0.0	0.0	0.0	1.0
5695	0.0	0.0	1.0	0.0	0.0

5 rows × 96 columns

In [128]:
```
X_train_SMOTE.iloc[0]
```

Out[128]:
```
Subject_Age_Group                      36 - 45
Weapon_Type                               None
Officer_Gender                               M
Officer_Race                  Hispanic or Latino
Subject_Perceived_Race                    White
Subject_Perceived_Gender                   Male
Call_Type                                   911
Frisk_Flag                                    N
Precinct                                   East
Subject_Known                            Repeat
Officer_Bin                               Other
Officer_DOB                                  80
Date                                     737460
Month                                         2
Year                                       2020
Day                                   Wednesday
Squad_Bin                                 Other
Name: 0, dtype: object
```

In [129]:
```
# Checking that the df looks right
X_train_encoded.iloc[0]
```

Out[129]:
```
Subject_Age_Group_1 - 17          0.0
Subject_Age_Group_18 - 25         0.0
Subject_Age_Group_26 - 35         0.0
Subject_Age_Group_36 - 45         1.0
Subject_Age_Group_46 - 55         0.0
                                 ...
Day_Wednesday                     1.0
Squad_Bin_Other                   1.0
Squad_Bin_Training                0.0
Squad_Bin_Unknown                 0.0
Date                         737460.0
Name: 5556, Length: 96, dtype: float64
```

In [130]:
```
X_train_encoded.shape #Now have 96 columns
```

Out[130]:
```
(8067, 96)
```

In [131]:
```
X_train_encoded_SMOTE.shape
```

Out[131]:
```
(12110, 96)
```

In [132]:
```
X_train_encoded.columns # Some column names have spaces and special characters now.  Should address.
```

Out[132]:
```
Index(['Subject_Age_Group_1 - 17', 'Subject_Age_Group_18 - 25',
       'Subject_Age_Group_26 - 35', 'Subject_Age_Group_36 - 45',
       'Subject_Age_Group_46 - 55', 'Subject_Age_Group_56 and Above',
       'Subject_Age_Group_Unknown', 'Weapon_Type_Firearm',
       'Weapon_Type_Handgun', 'Weapon_Type_Knife/Cutting/Stabbing Instrument',
       'Weapon_Type_Multiple', 'Weapon_Type_None', 'Weapon_Type_Other',
       'Officer_Gender_F', 'Officer_Gender_M', 'Officer_Gender_N',
       'Officer_Race_American Indian/Alaska Native', 'Officer_Race_Asian',
       'Officer_Race_Black or African American',
       'Officer_Race_Hispanic or Latino',
       'Officer_Race_Nat Hawaiian/Oth Pac Islander',
       'Officer_Race_Not Specified', 'Officer_Race_Two or More Races',
       'Officer_Race_Unknown', 'Officer_Race_White',
       'Subject_Perceived_Race_American Indian or Alaska Native',
       'Subject_Perceived_Race_Asian',
       'Subject_Perceived_Race_Black or African American',
       'Subject_Perceived_Race_Native Hawaiian or Other Pacific Islander',
       'Subject_Perceived_Race_Unknown', 'Subject_Perceived_Race_White',
       'Subject_Perceived_Gender_Female',
       'Subject_Perceived_Gender_Gender Diverse (gender non-conforming and/or transgender)',
       'Subject_Perceived_Gender_Male', 'Subject_Perceived_Gender_Unknown',
       'Call_Type_911', 'Call_Type_ALARM CALL (NOT POLICE ALARM)',
       'Call_Type_ONVIEW', 'Call_Type_TELEPHONE OTHER, NOT 911',
       'Call_Type_TEXT MESSAGE', 'Call_Type_Unknown', 'Frisk_Flag_N',
       'Frisk_Flag_Unknown', 'Frisk_Flag_Y', 'Precinct_East',
       'Precinct_FK ERROR', 'Precinct_North', 'Precinct_OOJ', 'Precinct_South',
       'Precinct_SouthWest', 'Precinct_Unknown', 'Precinct_West',
       'Subject_Known_First', 'Subject_Known_Repeat',
       'Subject_Known_Unidentified', 'Officer_Bin_7456', 'Officer_Bin_7634',
       'Officer_Bin_7690', 'Officer_Bin_7713', 'Officer_Bin_7758',
       'Officer_Bin_7765', 'Officer_Bin_7773', 'Officer_Bin_7792',
       'Officer_Bin_8302', 'Officer_Bin_Other', 'Officer_DOB_0',
       'Officer_DOB_50', 'Officer_DOB_60', 'Officer_DOB_70', 'Officer_DOB_80',
       'Officer_DOB_90', 'Month_1', 'Month_10', 'Month_11', 'Month_12',
       'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6', 'Month_7',
       'Month_8', 'Month_9', 'Year_2019', 'Year_2020', 'Day_Friday',
       'Day_Monday', 'Day_Saturday', 'Day_Sunday', 'Day_Thursday',
       'Day_Tuesday', 'Day_Wednesday', 'Squad_Bin_Other', 'Squad_Bin_Training',
       'Squad_Bin_Unknown', 'Date'],
      dtype='object')
```

In [133]:
```
X_train_encoded_SMOTE.Year_2019.value_counts() # just 0s and 1s, unlike when I ohe before resampling
```

Out[133]:
```
0.0    6591
1.0    5519
Name: Year_2019, dtype: int64
```

In [134]:
```
# let's fix all of the TRAIN variable names so they don't have spaces
X_train_encoded.columns = X_train_encoded.columns.str.replace(' ', '_')
X_train_encoded.columns = X_train_encoded.columns.str.replace('-', '_')
X_train_encoded.columns = X_train_encoded.columns.str.replace('/', '_')

X_train_encoded_SMOTE.columns = X_train_encoded_SMOTE.columns.str.replace(' ', '_')
X_train_encoded_SMOTE.columns = X_train_encoded_SMOTE.columns.str.replace('-', '_')
X_train_encoded_SMOTE.columns = X_train_encoded_SMOTE.columns.str.replace('/', '_')

X_train_encoded.columns
```

Out[134]:
```
Index(['Subject_Age_Group_1___17', 'Subject_Age_Group_18___25',
       'Subject_Age_Group_26___35', 'Subject_Age_Group_36___45',
       'Subject_Age_Group_46___55', 'Subject_Age_Group_56_and_Above',
       'Subject_Age_Group_Unknown', 'Weapon_Type_Firearm',
       'Weapon_Type_Handgun', 'Weapon_Type_Knife_Cutting_Stabbing_Instrument',
       'Weapon_Type_Multiple', 'Weapon_Type_None', 'Weapon_Type_Other',
       'Officer_Gender_F', 'Officer_Gender_M', 'Officer_Gender_N',
       'Officer_Race_American_Indian_Alaska_Native', 'Officer_Race_Asian',
       'Officer_Race_Black_or_African_American',
       'Officer_Race_Hispanic_or_Latino',
       'Officer_Race_Nat_Hawaiian_Oth_Pac_Islander',
       'Officer_Race_Not_Specified', 'Officer_Race_Two_or_More_Races',
       'Officer_Race_Unknown', 'Officer_Race_White',
       'Subject_Perceived_Race_American_Indian_or_Alaska_Native',
       'Subject_Perceived_Race_Asian',
       'Subject_Perceived_Race_Black_or_African_American',
       'Subject_Perceived_Race_Native_Hawaiian_or_Other_Pacific_Islander',
       'Subject_Perceived_Race_Unknown', 'Subject_Perceived_Race_White',
       'Subject_Perceived_Gender_Female',
       'Subject_Perceived_Gender_Gender_Diverse_(gender_non_conforming_and_or_transgender)',
       'Subject_Perceived_Gender_Male', 'Subject_Perceived_Gender_Unknown',
       'Call_Type_911', 'Call_Type_ALARM_CALL_(NOT_POLICE_ALARM)',
       'Call_Type_ONVIEW', 'Call_Type_TELEPHONE_OTHER,_NOT_911',
       'Call_Type_TEXT_MESSAGE', 'Call_Type_Unknown', 'Frisk_Flag_N',
       'Frisk_Flag_Unknown', 'Frisk_Flag_Y', 'Precinct_East',
       'Precinct_FK_ERROR', 'Precinct_North', 'Precinct_OOJ', 'Precinct_South',
       'Precinct_SouthWest', 'Precinct_Unknown', 'Precinct_West',
       'Subject_Known_First', 'Subject_Known_Repeat',
       'Subject_Known_Unidentified', 'Officer_Bin_7456', 'Officer_Bin_7634',
       'Officer_Bin_7690', 'Officer_Bin_7713', 'Officer_Bin_7758',
       'Officer_Bin_7765', 'Officer_Bin_7773', 'Officer_Bin_7792',
       'Officer_Bin_8302', 'Officer_Bin_Other', 'Officer_DOB_0',
       'Officer_DOB_50', 'Officer_DOB_60', 'Officer_DOB_70', 'Officer_DOB_80',
       'Officer_DOB_90', 'Month_1', 'Month_10', 'Month_11', 'Month_12',
       'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6', 'Month_7',
       'Month_8', 'Month_9', 'Year_2019', 'Year_2020', 'Day_Friday',
       'Day_Monday', 'Day_Saturday', 'Day_Sunday', 'Day_Thursday',
       'Day_Tuesday', 'Day_Wednesday', 'Squad_Bin_Other', 'Squad_Bin_Training',
       'Squad_Bin_Unknown', 'Date'],
      dtype='object')
```

In [135]:
```
# let's fix all of the TEST variable names so they don't have spaces
X_test_encoded.columns = X_test_encoded.columns.str.replace(' ', '_')
X_test_encoded.columns = X_test_encoded.columns.str.replace('-', '_')
X_test_encoded.columns = X_test_encoded.columns.str.replace('/', '_')
X_test_encoded.columns
```

Out[135]:
```
Index(['Subject_Age_Group_1___17', 'Subject_Age_Group_18___25',
       'Subject_Age_Group_26___35', 'Subject_Age_Group_36___45',
       'Subject_Age_Group_46___55', 'Subject_Age_Group_56_and_Above',
       'Subject_Age_Group_Unknown', 'Weapon_Type_Firearm',
       'Weapon_Type_Handgun', 'Weapon_Type_Knife_Cutting_Stabbing_Instrument',
       'Weapon_Type_Multiple', 'Weapon_Type_None', 'Weapon_Type_Other',
       'Officer_Gender_F', 'Officer_Gender_M', 'Officer_Gender_N',
       'Officer_Race_American_Indian_Alaska_Native', 'Officer_Race_Asian',
       'Officer_Race_Black_or_African_American',
       'Officer_Race_Hispanic_or_Latino',
       'Officer_Race_Nat_Hawaiian_Oth_Pac_Islander',
       'Officer_Race_Not_Specified', 'Officer_Race_Two_or_More_Races',
       'Officer_Race_Unknown', 'Officer_Race_White',
       'Subject_Perceived_Race_American_Indian_or_Alaska_Native',
       'Subject_Perceived_Race_Asian',
       'Subject_Perceived_Race_Black_or_African_American',
       'Subject_Perceived_Race_Native_Hawaiian_or_Other_Pacific_Islander',
       'Subject_Perceived_Race_Unknown', 'Subject_Perceived_Race_White',
       'Subject_Perceived_Gender_Female',
       'Subject_Perceived_Gender_Gender_Diverse_(gender_non_conforming_and_or_transgender)',
       'Subject_Perceived_Gender_Male', 'Subject_Perceived_Gender_Unknown',
       'Call_Type_911', 'Call_Type_ALARM_CALL_(NOT_POLICE_ALARM)',
       'Call_Type_ONVIEW', 'Call_Type_TELEPHONE_OTHER,_NOT_911',
       'Call_Type_TEXT_MESSAGE', 'Call_Type_Unknown', 'Frisk_Flag_N',
       'Frisk_Flag_Unknown', 'Frisk_Flag_Y', 'Precinct_East',
       'Precinct_FK_ERROR', 'Precinct_North', 'Precinct_OOJ', 'Precinct_South',
       'Precinct_SouthWest', 'Precinct_Unknown', 'Precinct_West',
       'Subject_Known_First', 'Subject_Known_Repeat',
       'Subject_Known_Unidentified', 'Officer_Bin_7456', 'Officer_Bin_7634',
       'Officer_Bin_7690', 'Officer_Bin_7713', 'Officer_Bin_7758',
       'Officer_Bin_7765', 'Officer_Bin_7773', 'Officer_Bin_7792',
       'Officer_Bin_8302', 'Officer_Bin_Other', 'Officer_DOB_0',
       'Officer_DOB_50', 'Officer_DOB_60', 'Officer_DOB_70', 'Officer_DOB_80',
       'Officer_DOB_90', 'Month_1', 'Month_10', 'Month_11', 'Month_12',
       'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6', 'Month_7',
       'Month_8', 'Month_9', 'Year_2019', 'Year_2020', 'Day_Friday',
       'Day_Monday', 'Day_Saturday', 'Day_Sunday', 'Day_Thursday',
       'Day_Tuesday', 'Day_Wednesday', 'Squad_Bin_Other', 'Squad_Bin_Training',
       'Squad_Bin_Unknown', 'Date'],
      dtype='object')
```

Scale Variables

Output - scaler, scaled_X_train_SMOTE, scaled_X_train, scaled_X_test (still y_train_SMOTE and y_test and y_train)

In [136]:
```
X_train_encoded['Date'].std()
```

Out[136]:
```
162.39780024784656
```

In [137]:
```
X_train_encoded_SMOTE['Date'].std()
```

Out[137]:
```
161.34496463198812
```

In [138]:
```
# Instantiate StandardScaler
scaler = StandardScaler()

# I am only going to scale the continuous variable of Date, so my categorical variables still look li
# though as long as they are still binary it shouldn't affect the outcome.

X_train_cont = np.array(X_train_encoded['Date'])
X_train_cont_SMOTE = np.array(X_train_encoded_SMOTE['Date'])
X_test_cont = np.array(X_test_encoded['Date'])

# Transform the training and test sets
# Does this matter if I fit to the regular data and transform the smote data? Means and STd are almos
scaled_data_train = scaler.fit_transform(X_train_cont.reshape(-1,1))
scaled_data_train_SMOTE = scaler.transform(X_train_cont_SMOTE.reshape(-1,1))
scaled_data_test = scaler.transform(X_test_cont.reshape(-1,1))

# Convert into a DataFrame
scaled_X_train = pd.concat([X_train_encoded.drop(columns='Date'),
                            pd.DataFrame(scaled_data_train, columns=['Date'], index=X_train_encoded.i
scaled_X_train_SMOTE = pd.concat([X_train_encoded_SMOTE.drop(columns='Date'),
                            pd.DataFrame(scaled_data_train_SMOTE, columns=['Date'], index=X_train_enc
scaled_X_test = pd.concat([X_test_encoded.drop(columns='Date'),
                           pd.DataFrame(scaled_data_test, columns=['Date'], index=X_test_encoded.inde
scaled_X_test.head()
```

Out[138]:

	Subject_Age_Group_1___17	Subject_Age_Group_18___25	Subject_Age_Group_26___35	Subject_Age_Group_36___45	Su
8213	0.0	0.0	0.0	0.0	
2520	0.0	1.0	0.0	0.0	
6475	0.0	0.0	0.0	0.0	
3849	0.0	0.0	0.0	0.0	
5695	0.0	0.0	1.0	0.0	

5 rows × 96 columns

In [139]:
```
#Kept categorical as 0 and 1
scaled_X_train['Subject_Age_Group_56_and_Above'].value_counts()
```

Out[139]:
```
0.0    7514
1.0     553
Name: Subject_Age_Group_56_and_Above, dtype: int64
```

In [140]:
```
scaled_X_train_SMOTE['Subject_Age_Group_56_and_Above'].value_counts()
```

Out[140]:
```
0.0    11427
1.0      683
Name: Subject_Age_Group_56_and_Above, dtype: int64
```

In [141]:
```
#Date is now scaled
scaled_X_train_SMOTE.Date.value_counts()
```

Out[141]:
```
 0.571575    58
 0.485362    45
 0.232880    44
-0.536883    44
 0.614682    44
             ..
 1.877092     6
 0.811741     5
 0.805583     5
 1.883251     5
 1.901725     5
Name: Date, Length: 588, dtype: int64
```

In [142]:
```
scaled_X_train.isna().sum().sum() # Making sure concat didnt create any NaN values because index was
```

Out[142]: 0

In [143]:
```
scaled_X_train_SMOTE.isna().sum().sum() # Making sure concat didnt create any NaN values because inde
```

Out[143]: 0

Define model metrics_df

Output - print_roc(), print_confusion_matrices(), print_metrics_df(), run_model(), plot_feature_importances(), metrics_df

In [144]:
```
metric_labels = ['Model','Name','Fit_Time','Pred_Time','Train_Precision','Test_Precision',
        'Train_Recall','Test_Recall','Train_Accuracy','Test_Accuracy','Train_F1','Test_F1']
metrics_df = pd.DataFrame(columns=metric_labels)
```

In [145]:
```
# Functions for running models and evaluating metrics_df have been moved to user_functions.py
```

Try models to check for more data cleaning

Output - logreg1, logreg2, dt1, dt2

This section originally showed me models with perfect scores. I had to go back and revisit Date and Stop Resolution.

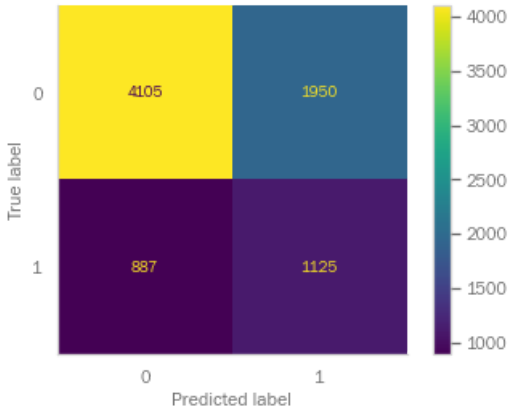
Will end up removing Date from dataset after these models

Initial Logistic Regression model

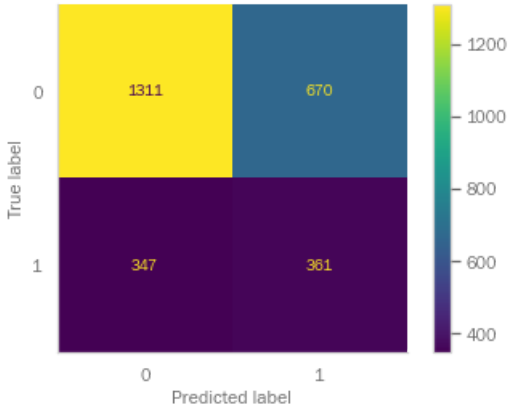
```
Fit time:        0.2253859043121338
Prediction time: 0.0059587955474853516
Precision Score: Train 0.36585, Test 0.35015
Recall Score:    Train 0.55915, Test 0.50989
Accuracy Score:  Train 0.64832, Test 0.62179
F1 Score:        Train 0.44230, Test 0.41518
```

In [146]:
```
# Need to fit to SMOTE data, but predict on non-SMOTE data
# run_model function will do this if I pass fit_X and fit_y parameters

logreg1 = LogisticRegression(fit_intercept=False, C=1e20, solver='liblinear')

metrics_df = run_model(logreg1, 'logreg1', scaled_X_train, scaled_X_test,
                    y_train, y_test, metrics_df, fit_X = scaled_X_train_SMOTE, fit_y = y_train_SMOTE)

# Don't think I can use cross_val_score when I am fitting on one set of data, but predicting on anoth
# print(cross_val_score(logreg1, scaled_X_train_SMOTE, y_train_SMOTE))
```

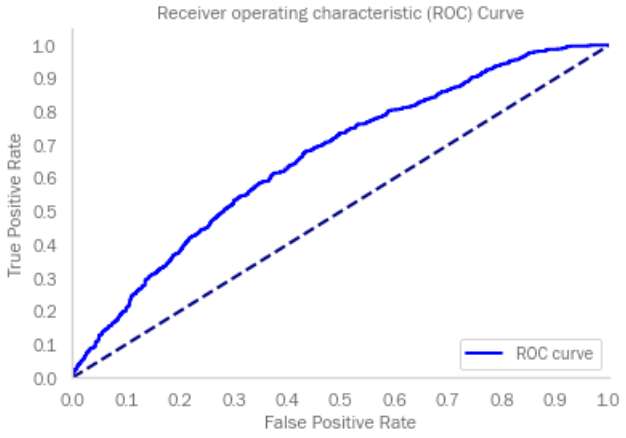
```
Fit time:  0.2653214931488037
Prediction time:  0.009940862655639648
Precision Score: Train 0.36585, Test 0.35015
Recall Score:    Train 0.55915, Test 0.50989
Accuracy Score:  Train 0.64832, Test 0.62179
F1 Score:        Train 0.44230, Test 0.41518
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2837, percentage = 35.1680%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1017, percentage = 37.8208%
```



```
----------------
              precision    recall  f1-score   support

           0       0.79      0.66      0.72      1981
           1       0.35      0.51      0.42       708

    accuracy                           0.62      2689
   macro avg       0.57      0.59      0.57      2689
weighted avg       0.67      0.62      0.64      2689

AUC: 0.6614778959436682
------------------------------------------------
```



In [147]:
```
metrics_df
```

Out[147]:

	Model	Name	Fit_Time	Pred_Time	Train_Precision	Test_Precision	Train_Recall	Test_Recall	Train
0	logreg1	LogisticRegression(C=1e+20, fit_intercept=Fals...	0.265321	0.009941	0.365854	0.350145	0.559145	0.509887	

Address multicollinearity for Logistic regression and rerun

```
Fit time:          0.15558338165283203
Prediction time:   0.004987239837646484
Precision Score: Train 0.36597, Test 0.35112
Recall Score:    Train 0.55915, Test 0.51130
Accuracy Score:  Train 0.64844, Test 0.62254
F1 Score:        Train 0.44239, Test 0.41633
```

Logistic Regression - scaled_X_train_logreg_SMOTE, scaled_X_test_logreg, scaled_X_test_logreg, y_train_SMOTE, y_train, y_test

In [148]:
```
corr_data = scaled_X_train.corr().abs().stack().reset_index().sort_values(0, ascending=False)
corr_data[corr_data[0] < 1][:20]
```

Out[148]:

	level_0	level_1	0
1343	Officer_Gender_M	Officer_Gender_F	0.989814
1249	Officer_Gender_F	Officer_Gender_M	0.989814
3166	Subject_Perceived_Gender_Male	Subject_Perceived_Gender_Female	0.980152
2978	Subject_Perceived_Gender_Female	Subject_Perceived_Gender_Male	0.980152
8831	Squad_Bin_Training	Squad_Bin_Other	0.962581
8737	Squad_Bin_Other	Squad_Bin_Training	0.962581
4991	Subject_Known_Repeat	Subject_Known_First	0.950207
4897	Subject_Known_First	Subject_Known_Repeat	0.950207
7979	Year_2020	Date	0.844799
7884	Year_2019	Date	0.844799
9013	Date	Year_2020	0.844799
9012	Date	Year_2019	0.844799
1054	Weapon_Type_None	Weapon_Type_Knife_Cutting_Stabbing_Instrument	0.832626
866	Weapon_Type_Knife_Cutting_Stabbing_Instrument	Weapon_Type_None	0.832626
3550	Call_Type_ONVIEW	Call_Type_911	0.726815
3362	Call_Type_911	Call_Type_ONVIEW	0.726815
6529	Officer_DOB_80	Officer_DOB_90	0.698750
6623	Officer_DOB_90	Officer_DOB_80	0.698750
6095	Officer_DOB_0	Officer_Gender_N	0.658314
1448	Officer_Gender_N	Officer_Race_Unknown	0.658314

In [149]:
```
# Clearly multicolinearity is a problem here.  Lets perform the OHE and scale again but drop the firs
# Perhaps this is something we can do in a pipeline later!

enc2 = OneHotEncoder(drop='first', sparse=False) # Dropping first column on OHE
X_cat = X.drop(columns='Date')
enc2.fit(X_cat)
X_train_code_logreg_SMOTE = enc2.transform(X_train_SMOTE.drop(columns='Date'))
X_train_code_logreg = enc2.transform(X_train.drop(columns='Date'))
X_test_code_logreg = enc2.transform(X_test.drop(columns='Date'))

# The transforms are not dataframes yet and we have to add back in the continuous column of Date
X_train_encoded_logreg_SMOTE = pd.DataFrame(X_train_code_logreg_SMOTE, columns=enc2.get_feature_names
X_train_encoded_logreg_SMOTE['Date'] = X_train_SMOTE.Date
X_train_encoded_logreg = pd.DataFrame(X_train_code_logreg, columns=enc2.get_feature_names(X_cat.colum
X_train_encoded_logreg['Date'] = X_train.Date
X_test_encoded_logreg = pd.DataFrame(X_test_code_logreg, columns=enc2.get_feature_names(X_cat.columns
X_test_encoded_logreg['Date'] = X_test.Date


# let's fix all of the TRAIN variable names so they don't have spaces
X_train_encoded_logreg_SMOTE.columns = X_train_encoded_logreg_SMOTE.columns.str.replace(' ', '_')
X_train_encoded_logreg_SMOTE.columns = X_train_encoded_logreg_SMOTE.columns.str.replace('-', '_')
X_train_encoded_logreg_SMOTE.columns = X_train_encoded_logreg_SMOTE.columns.str.replace('/', '_')

X_train_encoded_logreg.columns = X_train_encoded_logreg.columns.str.replace(' ', '_')
X_train_encoded_logreg.columns = X_train_encoded_logreg.columns.str.replace('-', '_')
X_train_encoded_logreg.columns = X_train_encoded_logreg.columns.str.replace('/', '_')

# let's fix all of the TEST variable names so they don't have spaces
X_test_encoded_logreg.columns = X_test_encoded_logreg.columns.str.replace(' ', '_')
X_test_encoded_logreg.columns = X_test_encoded_logreg.columns.str.replace('-', '_')
X_test_encoded_logreg.columns = X_test_encoded_logreg.columns.str.replace('/', '_')

# Instantiate StandardScaler
scaler2 = StandardScaler()

# I am only going to scale the continuous variable of Date, so my categorical variables still look li
# though as long as they are still binary it shouldn't affect the outcome.

X_train_cont_logreg_SMOTE = np.array(X_train_encoded_logreg_SMOTE['Date'])
X_train_cont_logreg = np.array(X_train_encoded_logreg['Date'])
X_test_cont_logreg = np.array(X_test_encoded_logreg['Date'])

# Transform the training and test sets
scaled_data_train_logreg = scaler2.fit_transform(X_train_cont_logreg.reshape(-1,1))
scaled_data_train_logreg_SMOTE = scaler2.transform(X_train_cont_logreg_SMOTE.reshape(-1,1))
scaled_data_test_logreg = scaler2.transform(X_test_cont_logreg.reshape(-1,1))

# Convert into a DataFrame
scaled_X_train_logreg_SMOTE = pd.concat([X_train_encoded_logreg_SMOTE.drop(columns='Date'),
                          pd.DataFrame(scaled_data_train_logreg_SMOTE, columns=['Date'], index=X_tr
scaled_X_train_logreg = pd.concat([X_train_encoded_logreg.drop(columns='Date'),
                          pd.DataFrame(scaled_data_train_logreg, columns=['Date'], index=X_train_en
scaled_X_test_logreg = pd.concat([X_test_encoded_logreg.drop(columns='Date'),
                         pd.DataFrame(scaled_data_test_logreg, columns=['Date'], index=X_test_encod
scaled_X_train_logreg_SMOTE.head()
```

Out[149]:

	Subject_Age_Group_18___25	Subject_Age_Group_26___35	Subject_Age_Group_36___45	Subject_Age_Group_46___55	Subj
0	0.0	0.0	1.0	0.0	
1	0.0	0.0	0.0	1.0	
2	0.0	0.0	1.0	0.0	
3	0.0	0.0	0.0	1.0	
4	0.0	1.0	0.0	0.0	

5 rows × 80 columns

In [150]:
```
corr_data = scaled_X_train_logreg.corr().abs().stack().reset_index().sort_values(0, ascending=False)
corr_data[corr_data[0] < 1][:10]
```

Out[150]:

	level_0	level_1	0
5529	Year_2020	Date	0.844799
6231	Date	Year_2020	0.844799
718	Weapon_Type_None	Weapon_Type_Knife_Cutting_Stabbing_Instrument	0.832626
562	Weapon_Type_Knife_Cutting_Stabbing_Instrument	Weapon_Type_None	0.832626
4481	Officer_DOB_80	Officer_DOB_90	0.698750
4559	Officer_DOB_90	Officer_DOB_80	0.698750
1513	Officer_Race_Unknown	Officer_Gender_N	0.658314
967	Officer_Gender_N	Officer_Race_Unknown	0.658314
3402	Subject_Known_Unidentified	Subject_Age_Group_Unknown	0.649783
438	Subject_Age_Group_Unknown	Subject_Known_Unidentified	0.649783

In [151]:
```
# Still some issues.  Let's drop Date and Weapon_Type_Knife_Cutting_Stabbing_Instrument
scaled_X_train_logreg_SMOTE.drop(columns=['Weapon_Type_Knife_Cutting_Stabbing_Instrument', 'Date'], i
scaled_X_train_logreg.drop(columns=['Weapon_Type_Knife_Cutting_Stabbing_Instrument', 'Date'], inplace
scaled_X_test_logreg.drop(columns=['Weapon_Type_Knife_Cutting_Stabbing_Instrument', 'Date'], inplace=
```

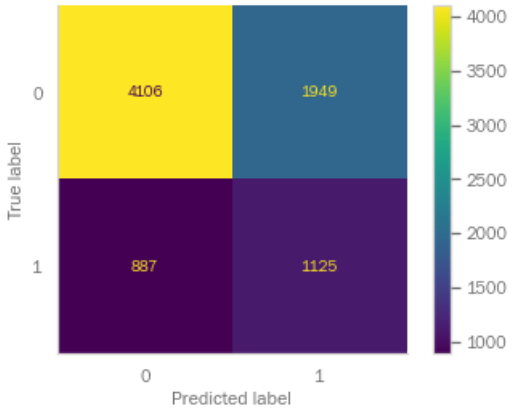
In [152]:
```
corr_data = scaled_X_train_logreg.corr().abs().stack().reset_index().sort_values(0, ascending=False)
corr_data[corr_data[0] < 1][:10]
```

Out[152]:

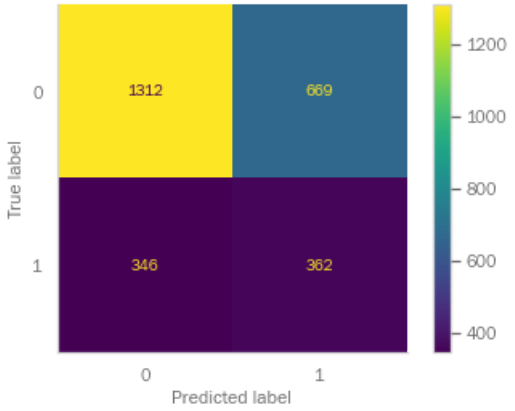
	level_0	level_1	0
4291	Officer_DOB_80	Officer_DOB_90	0.698750
4367	Officer_DOB_90	Officer_DOB_80	0.698750
1397	Officer_Race_Unknown	Officer_Gender_N	0.658314
865	Officer_Gender_N	Officer_Race_Unknown	0.658314
427	Subject_Age_Group_Unknown	Subject_Known_Unidentified	0.649783
3239	Subject_Known_Unidentified	Subject_Age_Group_Unknown	0.649783
1641	Subject_Perceived_Race_Black_or_African_American	Subject_Perceived_Race_White	0.569463
1869	Subject_Perceived_Race_White	Subject_Perceived_Race_Black_or_African_American	0.569463
923	Officer_Gender_N	Squad_Bin_Unknown	0.561929
5863	Squad_Bin_Unknown	Officer_Gender_N	0.561929

In [153]:
```
logreg2 = LogisticRegression(fit_intercept=False, C=1e20, solver='liblinear')

metrics_df = run_model(logreg2, 'logreg2', scaled_X_train_logreg, scaled_X_test_logreg,
                   y_train, y_test, metrics_df, fit_X = scaled_X_train_logreg_SMOTE, fit_y = y_train

# print(cross_val_score(logreg2,scaled_X_train_logreg,y_train_resampled))
```

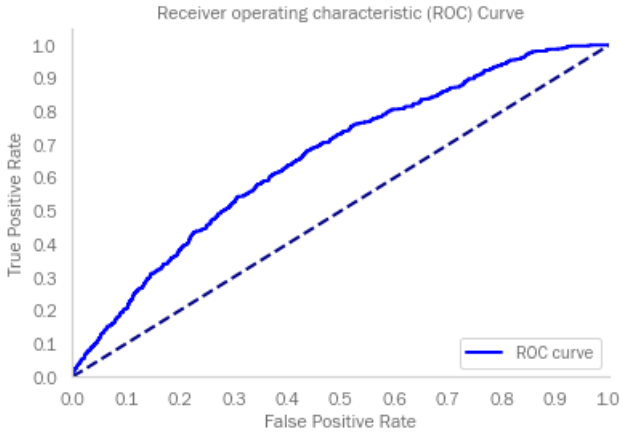
```
Fit time:  0.16456103324890137
Prediction time:  0.005988359451293945
Precision Score: Train 0.36597, Test 0.35112
Recall Score:    Train 0.55915, Test 0.51130
Accuracy Score:  Train 0.64844, Test 0.62254
F1 Score:        Train 0.44239, Test 0.41633
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2836, percentage = 35.1556%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1015, percentage = 37.7464%
```




```
----------------
              precision    recall  f1-score   support

           0       0.79      0.66      0.72      1981
           1       0.35      0.51      0.42       708

    accuracy                           0.62      2689
   macro avg       0.57      0.59      0.57      2689
weighted avg       0.68      0.62      0.64      2689

AUC: 0.6613855639878279
------------------------------------------------
```

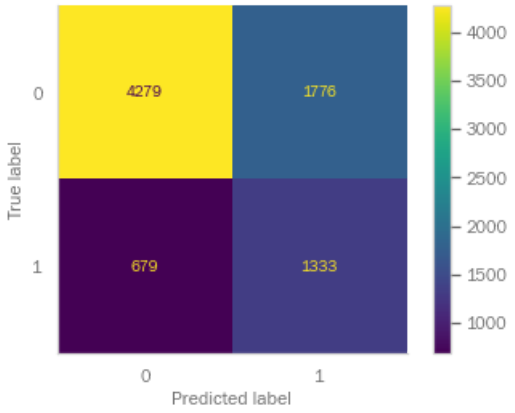


Initial Decision Tree to check feature importances

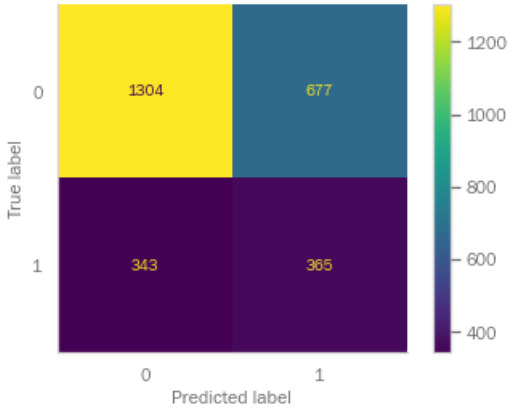
```
Fit time:        0.08673977851867676
Prediction time: 0.009972333908081055
Precision Score: Train 0.42876, Test 0.35029
Recall Score:    Train 0.66252, Test 0.51554
Accuracy Score:  Train 0.69567, Test 0.62068
F1 Score:        Train 0.52060, Test 0.41714
```

In [154]:
```
# I want to drop the variables that might be confounding (if any) but I want to run one decision tree
# I can see which variable it thinks gives the most info gain.

dt1 = DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=10)

metrics_df = run_model(dt1, 'dt1', X_train_encoded, X_test_encoded, y_train, y_test,
                       metrics_df, fit_X=X_train_encoded_SMOTE, fit_y=y_train_SMOTE)
```

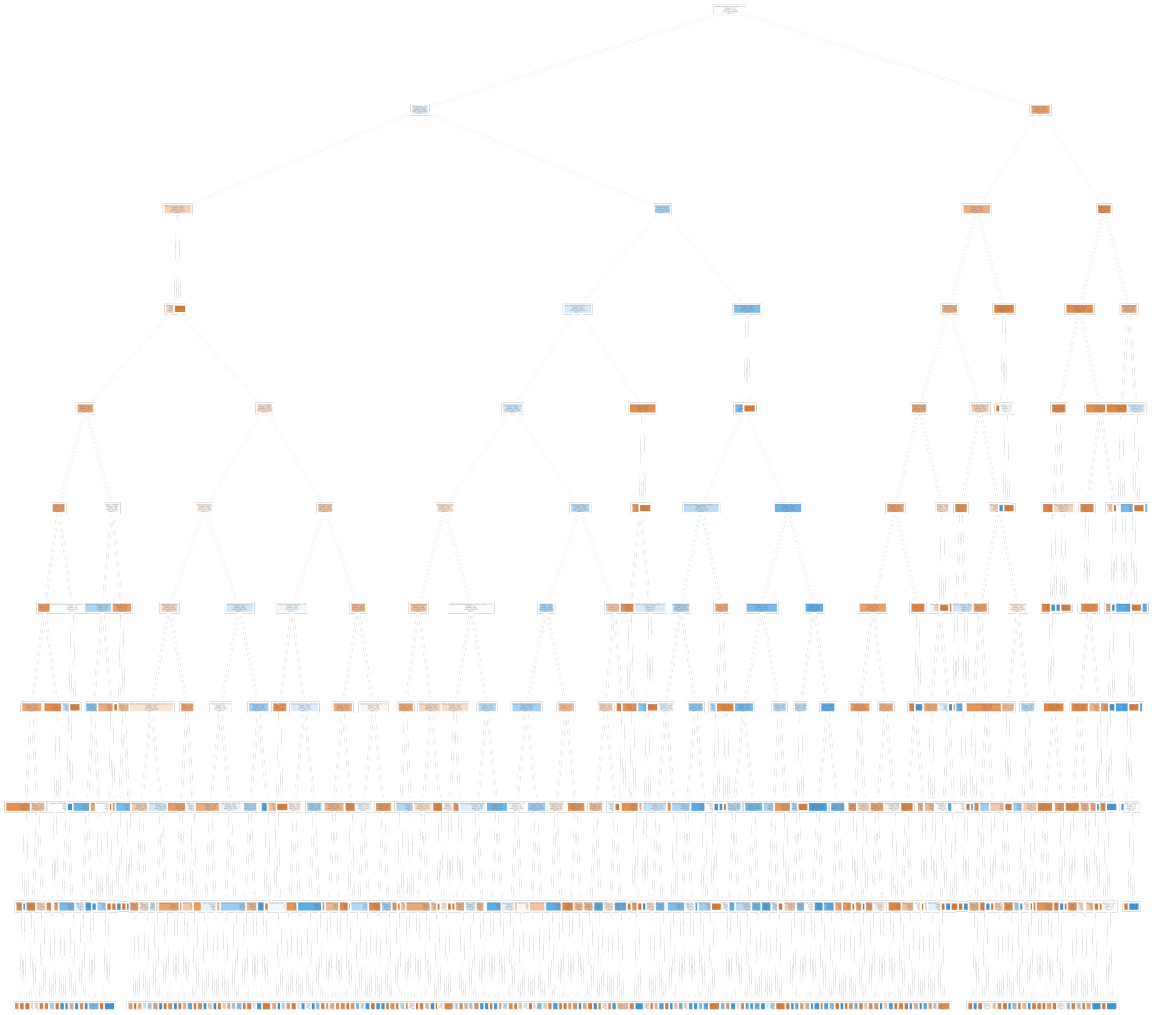
```
Fit time:  0.13122129440307617
Prediction time:  0.013988971710205078
Precision Score: Train 0.42876, Test 0.35029
Recall Score:    Train 0.66252, Test 0.51554
Accuracy Score:  Train 0.69567, Test 0.62068
F1 Score:        Train 0.52060, Test 0.41714
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2455, percentage = 30.4326%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1020, percentage = 37.9323%
```

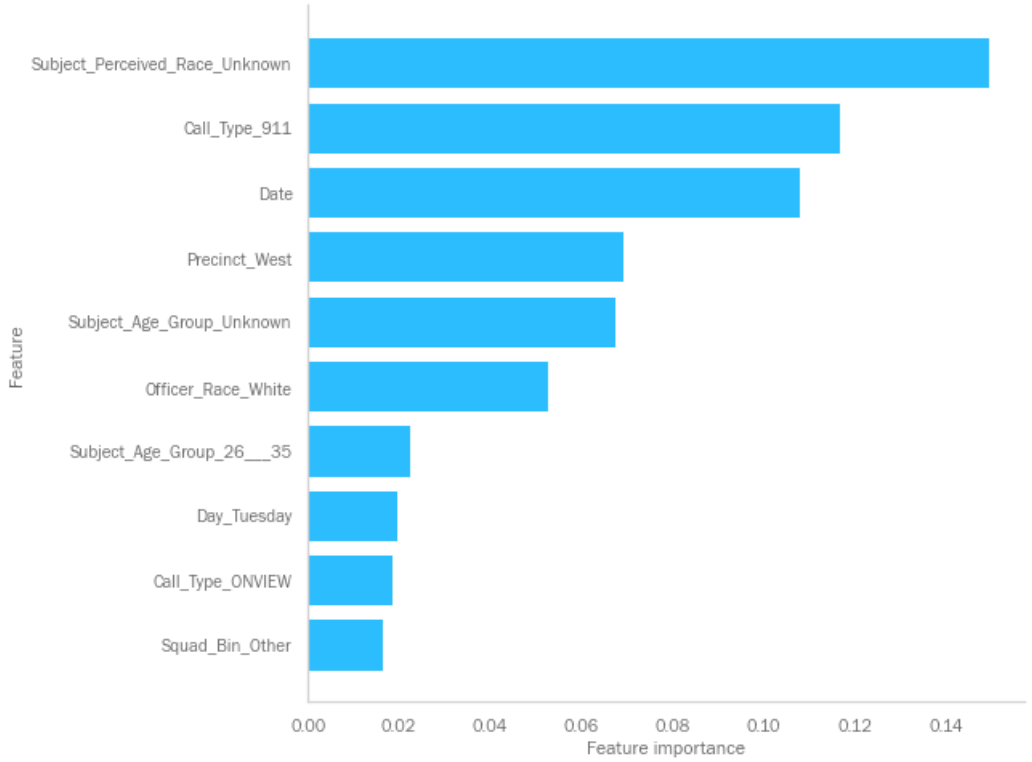



```
----------------
              precision    recall  f1-score   support

           0       0.79      0.66      0.72      1981
           1       0.35      0.52      0.42       708

    accuracy                           0.62      2689
   macro avg       0.57      0.59      0.57      2689
weighted avg       0.68      0.62      0.64      2689


AUC is :0.59
```



In [155]:
```
plot_ten_feature_importances(dt1, X_train_encoded, 'dt1')
```

```
Images/feature_importancedt1.png
```

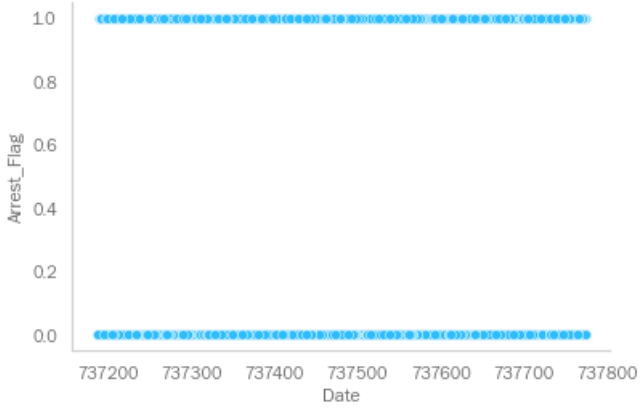


Remove Date variable and rerun Decision Tree

```
Fit time:        0.10272502899169922
Prediction time: 0.006981611251831055
Precision Score: Train 0.98467, Test 0.34138
Recall Score:    Train 0.98956, Test 0.41949
Accuracy Score:  Train 0.99355, Test 0.63406
F1 Score:        Train 0.98711, Test 0.37643
```

Decision Tree - X_train_encoded_cleaned, y_train, X_test_encoded_cleaned, y_test (after removing Date)

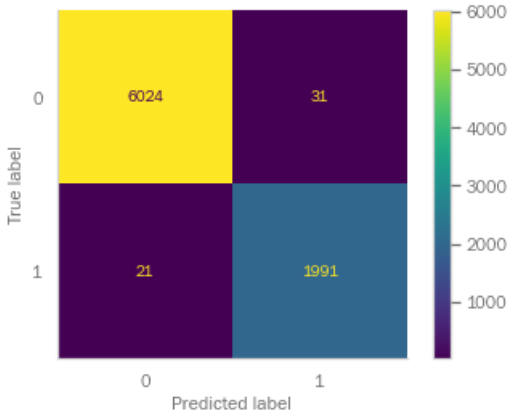
In [156]:
```
#  Hmm is Date a problem here? I recall hearing that date shouldnt be used in a model but I am unclea
# If there was something cyclical, like weekly or seasonality, the day and month features should catc

sns.scatterplot(x=X_train_encoded_SMOTE['Date'], y=y_train_SMOTE)
plt.savefig('Images/dates_after_subset.png');
```



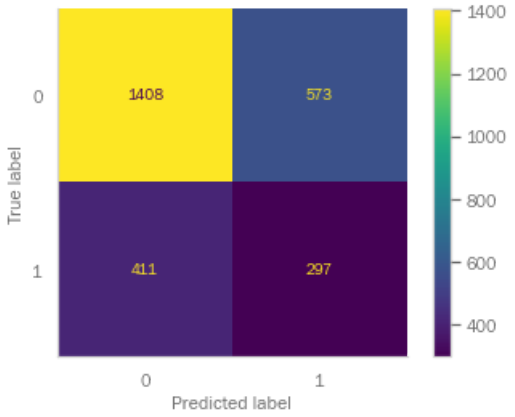
In [157]:
```
# Let's try removing Date
X_train_encoded_cleaned = X_train_encoded.drop(columns=['Date'])
X_train_encoded_cleaned_SMOTE = X_train_encoded_SMOTE.drop(columns=['Date'])
X_test_encoded_cleaned = X_test_encoded.drop(columns=['Date'])
```

In [158]:
```
dt2 = DecisionTreeClassifier(criterion='entropy', random_state=10)

metrics_df = run_model(dt2, 'dt2', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

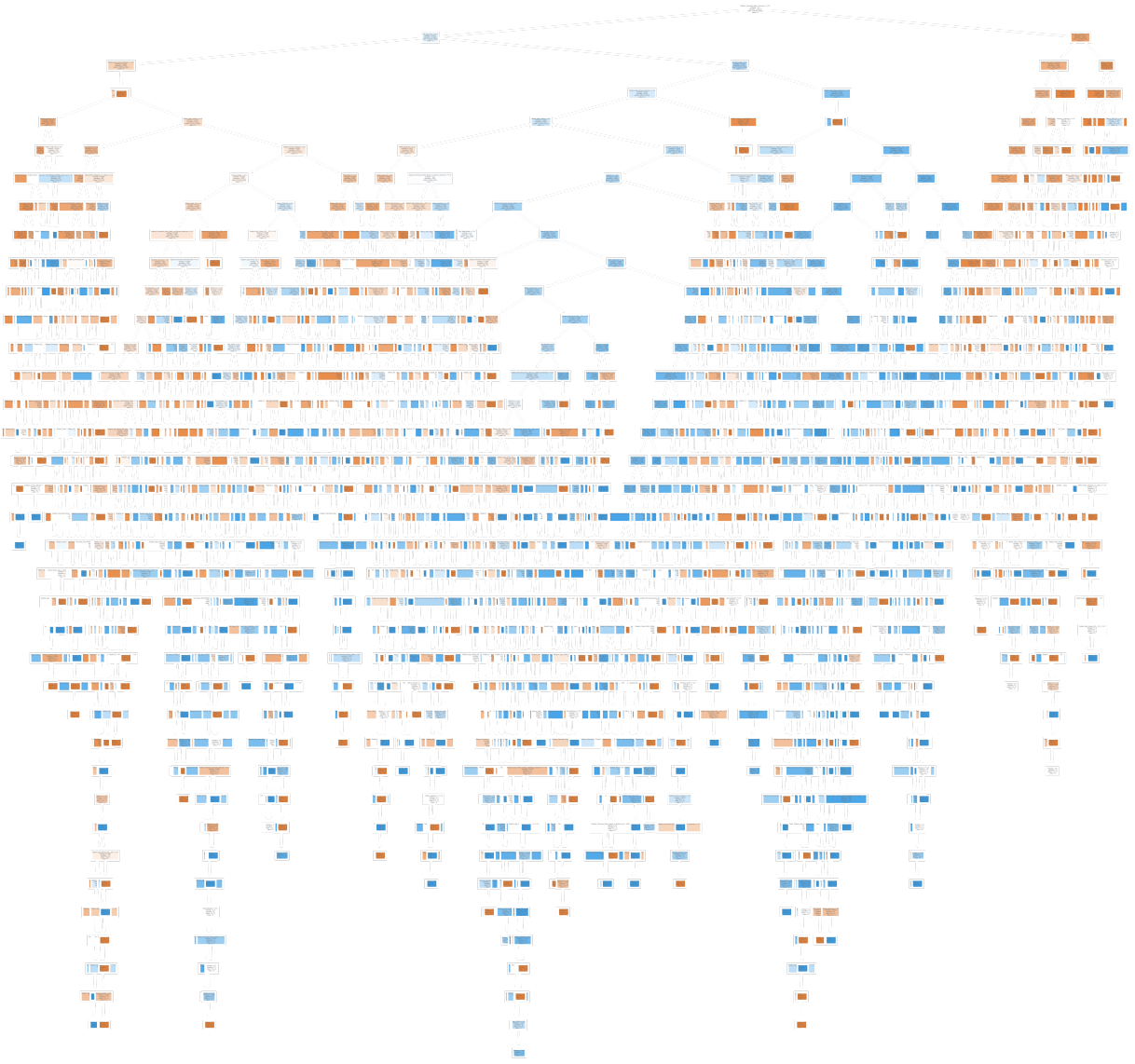
```
Fit time:  0.08779525756835938
Prediction time:  0.007948637008666992
Precision Score: Train 0.98467, Test 0.34138
Recall Score:    Train 0.98956, Test 0.41949
Accuracy Score:  Train 0.99355, Test 0.63406
F1 Score:        Train 0.98711, Test 0.37643
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 52, percentage = 0.6446%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 984, percentage = 36.5935%
```

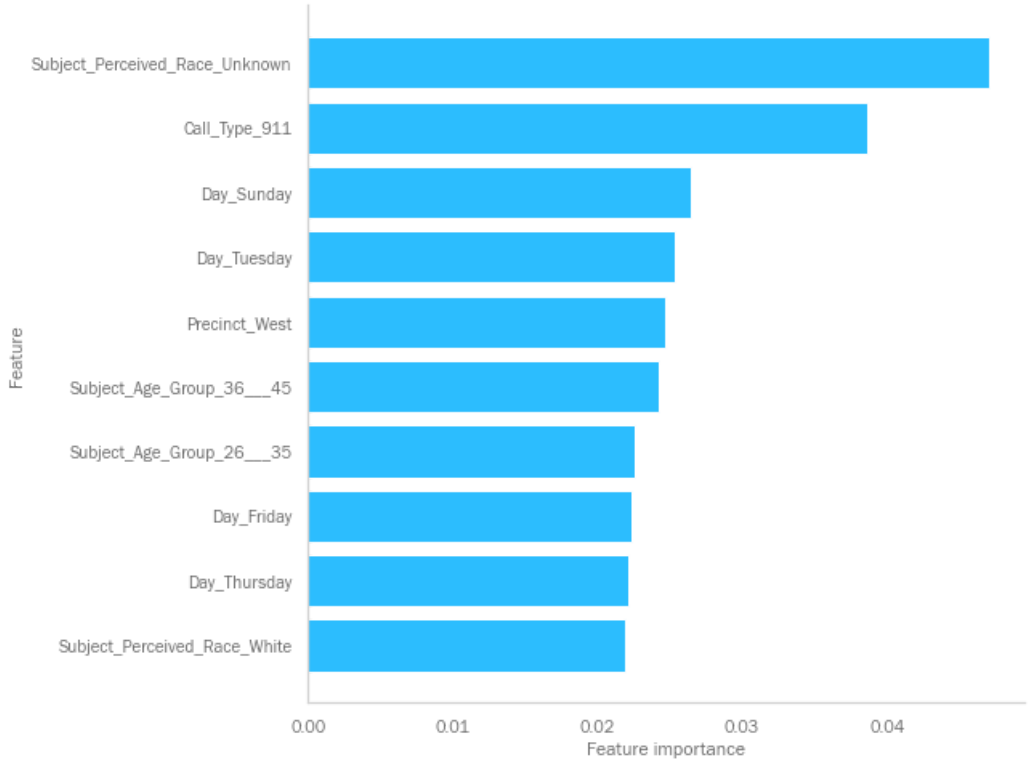



```
----------------
              precision    recall  f1-score   support

           0       0.77      0.71      0.74      1981
           1       0.34      0.42      0.38       708

    accuracy                           0.63      2689
   macro avg       0.56      0.57      0.56      2689
weighted avg       0.66      0.63      0.65      2689


AUC is :0.57
```



In [159]:
```
# Well it has shockingly raised our training scores.  Does that make sense?

plot_ten_feature_importances(dt2, X_train_encoded_cleaned, 'dt2')
```

Images/feature_importancedt2.png



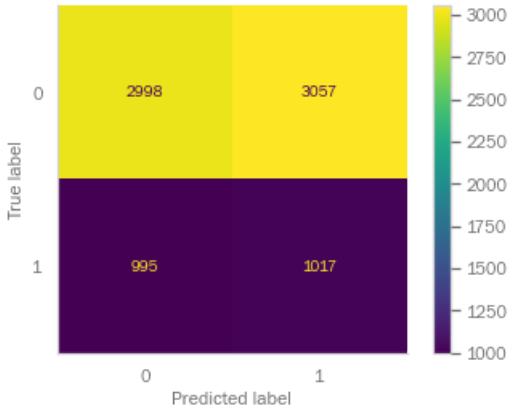
Create baseline model Dummy Classifier

```
Fit time:        0.0009984970092773438
Prediction time: 0.0019943714141845703
Precision Score: Train 0.24963, Test 0.26897
Recall Score:    Train 0.50547, Test 0.50565
Accuracy Score:  Train 0.49771, Test 0.50800
F1 Score:        Train 0.33421, Test 0.35115
```

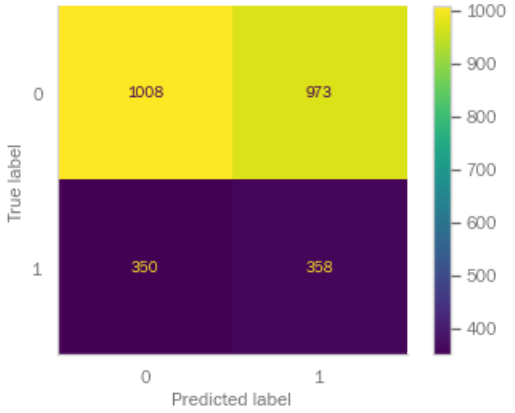
Dummy Classifier - X_train_encoded_cleaned, y_train, X_test_encoded_cleaned, y_test

In [160]:
```
from sklearn.dummy import DummyClassifier

dummy1 = DummyClassifier(strategy="uniform", random_state=10) #“stratified”, “most_frequent”, “prior”

metrics_df = run_model(dummy1, 'dummy1', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  0.0019910335540771484
Prediction time:  0.0029938220977783203
Precision Score: Train 0.24963, Test 0.26897
Recall Score:    Train 0.50547, Test 0.50565
Accuracy Score:  Train 0.49771, Test 0.50800
F1 Score:        Train 0.33421, Test 0.35115
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 4052, percentage = 50.2293%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1323, percentage = 49.2004%
```



```
----------------
              precision    recall  f1-score   support

           0       0.74      0.51      0.60      1981
           1       0.27      0.51      0.35       708

    accuracy                           0.51      2689
   macro avg       0.51      0.51      0.48      2689
weighted avg       0.62      0.51      0.54      2689
```

In [161]:
```
metrics_df
```

Out[161]:

	Model	Name	Fit_Time	Pred_Time	Train_Precision	Test_Precision	Train_Recall	Test_R
0	logreg1	LogisticRegression(C=1e+20, fit_intercept=Fals...	0.265321	0.009941	0.365854	0.350145	0.559145	0.50
1	logreg2	LogisticRegression(C=1e+20, fit_intercept=Fals...	0.164561	0.005988	0.365973	0.351115	0.559145	0.51
2	dt1	DecisionTreeClassifier(criterion='entropy', ma...	0.131221	0.013989	0.428755	0.350288	0.662525	0.51
3	dt2	DecisionTreeClassifier(criterion='entropy', ra...	0.087795	0.007949	0.984669	0.341379	0.989563	0.41
4	dummy1	DummyClassifier(random_state=10, strategy='uni...	0.001991	0.002994	0.249632	0.268971	0.505467	0.50

Contextualize Misclassified Data

This seems like a good place to look at what a false positive and false negative would be. I have coded an arrest as the positive case. So a false positive is predicting an arrest when no arrest was made. A false negative would be predicting NO arrest when an arrest was made. This model won't be used to predict future cases of Terry Stops, it is just to analyze past decisions. Therefore, I think the most important metric will be accuracy. How accurately did my model explain the actual data that I was given? This will give us the most useful information about how the decisions appear to have been influenced by the variables examined.

F1 is possibly also the right metric for my model as it gives us a more generalized view of the model without as much potential for overfitting. If we just focus on accuracy and fit exactly to our data, we cannot make any generalizations about observations outside out data.

Accuracy answers the question of 'How many classifications did my model get right?'
F1 is the harmonic mean of precision and recall

- Precision - how many of my predicted positives are true positives? (Only classify arrests if certain to avoid false positives)
- Recall - how many of the actual positives are true positives? (Classify everyone as arrest to catch all of the actual arrests)

Logistic Regression

Output - logreg3

Logistic Regression different solver (date already removed for collinearity)

```
Fit time:        1.494966745376587
Prediction time: 0.005983829498291016
Precision Score: Train 0.36633, Test 0.35112
Recall Score:    Train 0.55915, Test 0.51130
Accuracy Score:  Train 0.64882, Test 0.62254
F1 Score:        Train 0.44265, Test 0.41633
```

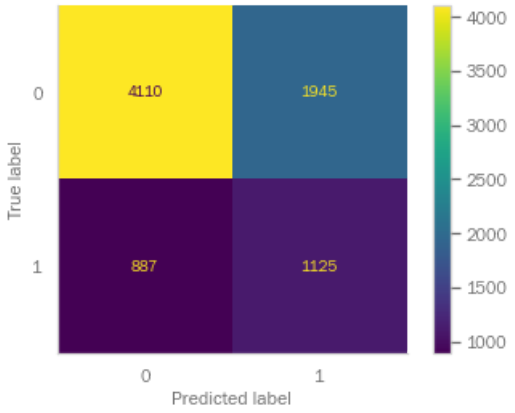
Failed to converge, and scores aren't good. Moving on to a different classifier.

Also use this scaled data for KNN

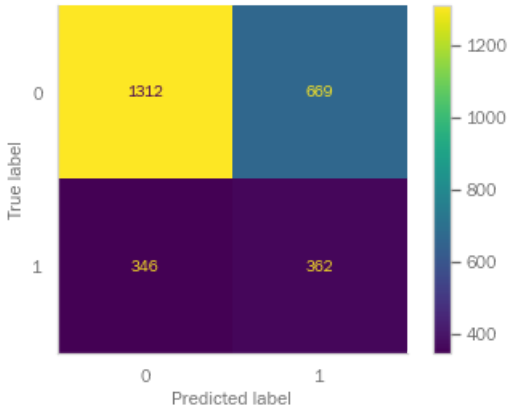
scaled_X_train_logreg, scaled_X_test_logreg, y_train, y_test

In [162]:
```
logreg3 = LogisticRegression(fit_intercept=False, C=1e20, penalty='elasticnet', solver='saga', l1_rat

metrics_df = run_model(logreg3, 'logreg3', scaled_X_train_logreg, scaled_X_test_logreg,
                   y_train, y_test, metrics_df, fit_X = scaled_X_train_logreg_SMOTE, fit_y = y_train
```

```
Fit time:  1.5348923206329346
Prediction time:  0.0043735504150390625
Precision Score: Train 0.36645, Test 0.35112
Recall Score:    Train 0.55915, Test 0.51130
Accuracy Score:  Train 0.64894, Test 0.62254
F1 Score:        Train 0.44274, Test 0.41633
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2832, percentage = 35.1060%

C:\Users\melod\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_sag.py:329: Converge
nceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

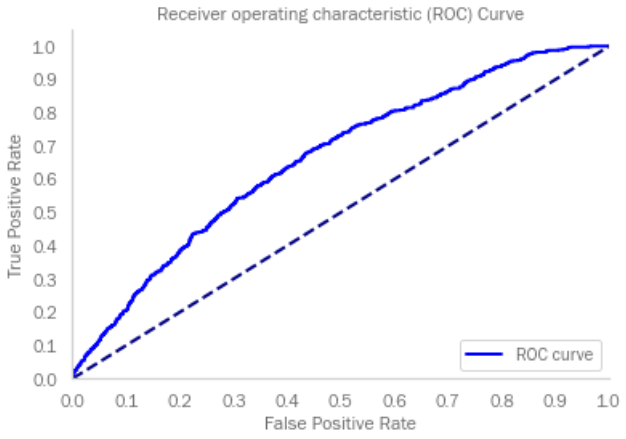



```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1015, percentage = 37.7464%
```




```
----------------
              precision    recall  f1-score   support

           0       0.79      0.66      0.72      1981
           1       0.35      0.51      0.42       708

    accuracy                           0.62      2689
   macro avg       0.57      0.59      0.57      2689
weighted avg       0.68      0.62      0.64      2689

AUC: 0.6611324532208525
------------------------------------------------
```

```
C:\Users\melod\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_sag.py:329: Converge
nceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```



KNN models

Output - knn1 - knn3, find_best_k()

Ran on ohe and scaled data

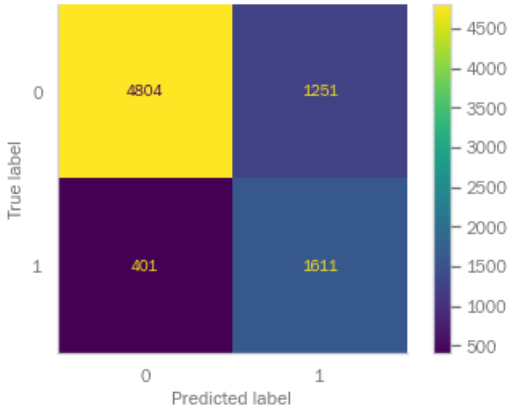
Initial KNN (default of 5)

```
Fit time:        0.3311138153076172
Prediction time: 14.89331841468811
Precision Score: Train 0.56289, Test 0.33468
Recall Score:    Train 0.80070, Test 0.46893
Accuracy Score:  Train 0.79522, Test 0.61473
F1 Score:        Train 0.66106, Test 0.39059
```

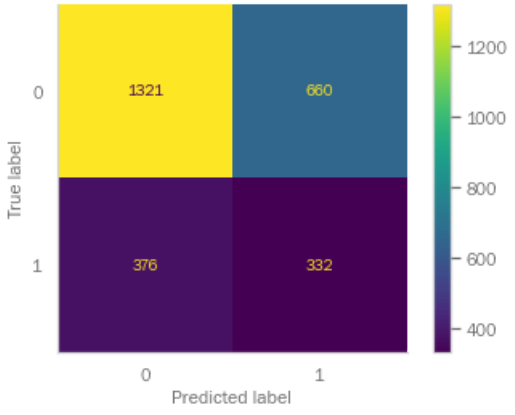
Using logreg data because it is scaled and has Date removed

In [163]:
```
knn1 = KNeighborsClassifier()

metrics_df = run_model(knn1, 'knn1', scaled_X_train_logreg, scaled_X_test_logreg,
                   y_train, y_test, metrics_df, fit_X = scaled_X_train_logreg_SMOTE, fit_y = y_train
```

```
Fit time:  0.34507298469543457
Prediction time:  16.584353923797607
Precision Score: Train 0.56289, Test 0.33468
Recall Score:    Train 0.80070, Test 0.46893
Accuracy Score:  Train 0.79522, Test 0.61473
F1 Score:        Train 0.66106, Test 0.39059
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 1652, percentage = 20.4785%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1036, percentage = 38.5273%
```




```
----------------
              precision    recall  f1-score   support

           0       0.78      0.67      0.72      1981
           1       0.33      0.47      0.39       708

    accuracy                           0.61      2689
   macro avg       0.56      0.57      0.55      2689
weighted avg       0.66      0.61      0.63      2689
```

In [164]:
```
# Looking for model with best F1 score

def find_best_k_f1_test(X_train, X_test, y_train, y_test, fit_X, fit_y, min_k=1, max_k=25):
    best_k = 0
    best_score = 0.0
    tic = time.time()
    for k in range(min_k, max_k+1, 2):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(fit_X, fit_y)
        preds = knn.predict(X_test)
        f1 = f1_score(y_test, preds)
        if f1 > best_score:
            best_k = k
            best_score = f1
    toc = time.time()
    run_time = toc-tic
    print('Run time: ', run_time)
    print("Best Value for k: {}".format(best_k))
    print("F1-Score: {}".format(best_score))
```

In [165]:
```
find_best_k_f1_test(scaled_X_train_logreg, scaled_X_test_logreg,
                    y_train, y_test, scaled_X_train_logreg_SMOTE, y_train_SMOTE)
```

```
Run time:  65.4840874671936
Best Value for k: 11
F1-Score: 0.4285714285714286
```

In [166]:
```
# What if I looked for best f1 training score?

def find_best_k_f1_train(X_train, X_test, y_train, y_test, fit_X, fit_y, min_k=1, max_k=25):
    best_k = 0
    best_score = 0.0
    tic = time.time()
    for k in range(min_k, max_k+1, 2):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(fit_X, fit_y)
        preds = knn.predict(X_train)
        f1 = f1_score(y_train, preds)
        if f1 > best_score:
            best_k = k
            best_score = f1
    toc = time.time()
    run_time = toc-tic
    print('Run time: ', run_time)
    print("Best Value for k: {}".format(best_k))
    print("F1-Score: {}".format(best_score))

find_best_k_f1_train(scaled_X_train_logreg, scaled_X_test_logreg,
                    y_train, y_test, scaled_X_train_logreg_SMOTE, y_train_SMOTE)
```

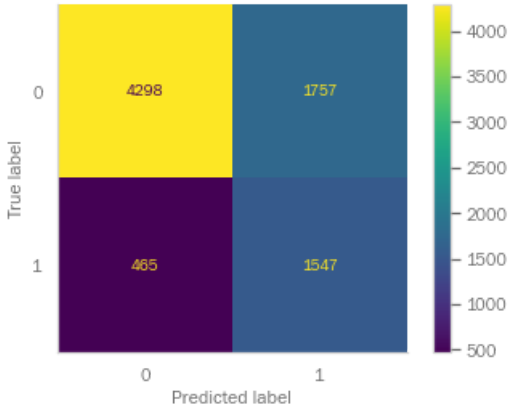
```
Run time:  170.56651163101196
Best Value for k: 1
F1-Score: 0.9903441445902451
```

KNN with calculated best k of 11

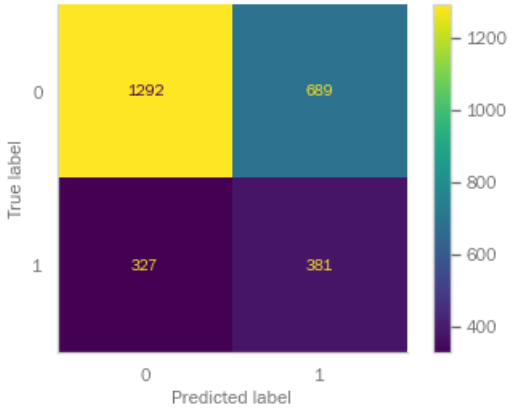
```
Fit time:        0.3370969295501709
Prediction time: 16.59153127670288
Precision Score: Train 0.46822, Test 0.35607
Recall Score:    Train 0.76889, Test 0.53814
Accuracy Score:  Train 0.72456, Test 0.62216
F1 Score:        Train 0.58202, Test 0.42857
```

In [167]:
```
knn2 = KNeighborsClassifier(n_neighbors=11, weights='uniform', algorithm='auto',
                            leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)

metrics_df = run_model(knn2, 'knn2', scaled_X_train_logreg, scaled_X_test_logreg,
                       y_train, y_test, metrics_df, fit_X = scaled_X_train_logreg_SMOTE, fit_y = y_train
```

```
Fit time:  0.3450734615325928
Prediction time:  20.29995822906494
Precision Score: Train 0.46822, Test 0.35607
Recall Score:    Train 0.76889, Test 0.53814
Accuracy Score:  Train 0.72456, Test 0.62216
F1 Score:        Train 0.58202, Test 0.42857
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2222, percentage = 27.5443%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1016, percentage = 37.7836%
```



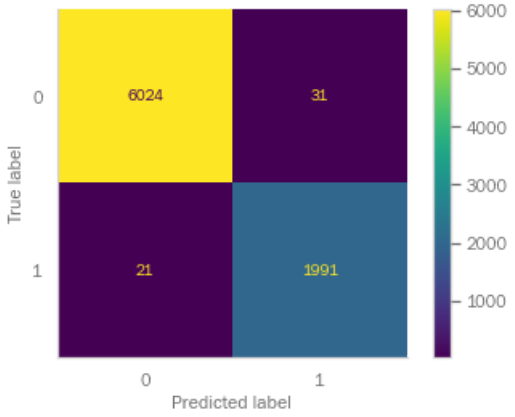

```
----------------
              precision    recall  f1-score   support

           0       0.80      0.65      0.72      1981
           1       0.36      0.54      0.43       708

    accuracy                           0.62      2689
   macro avg       0.58      0.60      0.57      2689
weighted avg       0.68      0.62      0.64      2689
```

KNN with calculated best k of 11, weights = 'distance'

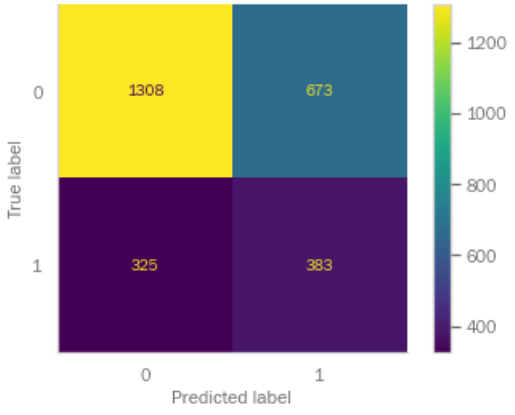
```
Fit time:        0.32915329933166504
Prediction time: 15.610223770141602
Precision Score: Train 0.98467, Test 0.36269
Recall Score:    Train 0.98956, Test 0.54096
Accuracy Score:  Train 0.99355, Test 0.62886
F1 Score:        Train 0.98711, Test 0.43424
```

In [168]:
```
knn3 = KNeighborsClassifier(n_neighbors=11, weights='distance', algorithm='auto',
                            leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)

metrics_df = run_model(knn3, 'knn3', scaled_X_train_logreg, scaled_X_test_logreg,
                    y_train, y_test, metrics_df, fit_X = scaled_X_train_logreg_SMOTE, fit_y = y_train
```

```
Fit time:  0.3670165538787842
Prediction time:  23.042906761169434
Precision Score: Train 0.98467, Test 0.36269
Recall Score:    Train 0.98956, Test 0.54096
Accuracy Score:  Train 0.99355, Test 0.62886
F1 Score:        Train 0.98711, Test 0.43424
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 52, percentage = 0.6446%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 998, percentage = 37.1142%
```




```
----------------
              precision    recall  f1-score   support

           0       0.80      0.66      0.72      1981
           1       0.36      0.54      0.43       708

    accuracy                           0.63      2689
   macro avg       0.58      0.60      0.58      2689
weighted avg       0.69      0.63      0.65      2689
```

Bayes Classification

Output - gnb1-4

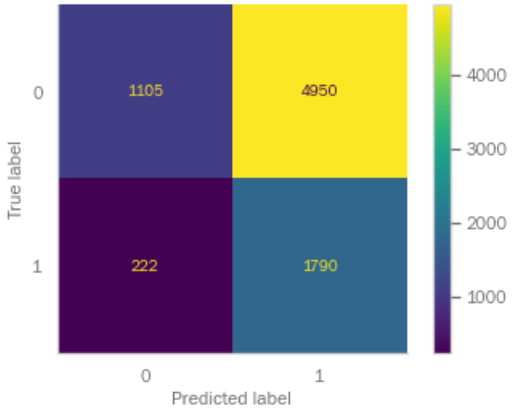
Ran on SMOTE and ohe data **but not scaled** data, scaling won't affect probabilities

Initial Gaussian Naive Bayes

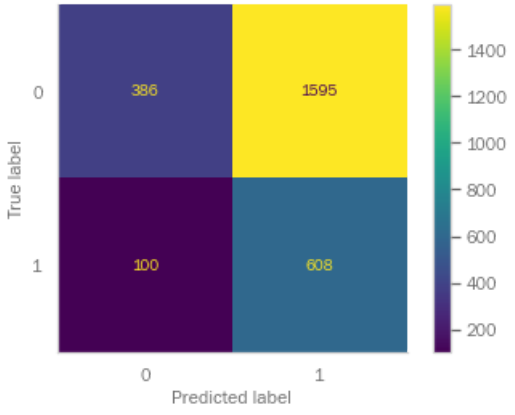
```
Fit time:        0.025929689407348633
Prediction time: 0.023937463760375977
Precision Score: Train 0.26558, Test 0.27599
Recall Score:    Train 0.88966, Test 0.85876
Accuracy Score:  Train 0.35887, Test 0.36965
F1 Score:        Train 0.40905, Test 0.41773
```

In [169]:
```
gnb1 = GaussianNB()

metrics_df = run_model(gnb1, 'gnb1', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  0.032955169677734375
Prediction time:  0.04488039016723633
Precision Score: Train 0.26558, Test 0.27599
Recall Score:    Train 0.88966, Test 0.85876
Accuracy Score:  Train 0.35887, Test 0.36965
F1 Score:        Train 0.40905, Test 0.41773
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 5172, percentage = 64.1131%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1695, percentage = 63.0346%
```




```
----------------
              precision    recall  f1-score   support

           0       0.79      0.19      0.31      1981
           1       0.28      0.86      0.42       708

    accuracy                           0.37      2689
   macro avg       0.54      0.53      0.37      2689
weighted avg       0.66      0.37      0.34      2689
```

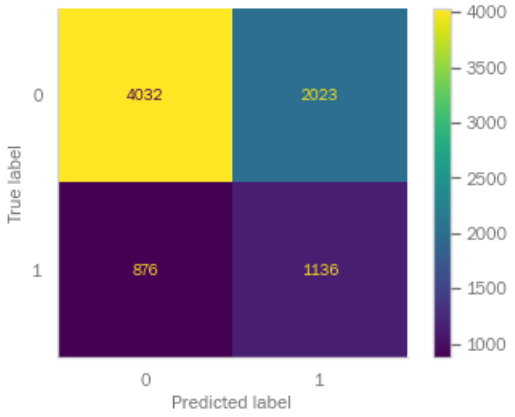
Bernoulli Naive Bayes

Can't seem to run ComplementNB, CategoricalNB, or MultinomialNB because of negative values. Need to understand how to choose which model.

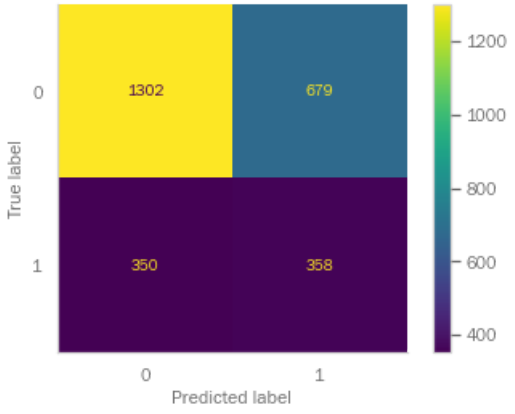
```
Fit time:        0.01994633674621582
Prediction time: 0.01795172691345215
Precision Score: Train 0.35961, Test 0.34523
Recall Score:    Train 0.56461, Test 0.50565
Accuracy Score:  Train 0.64063, Test 0.61733
F1 Score:        Train 0.43937, Test 0.41032
```

In [170]:
```
bnb1 = BernoulliNB()

metrics_df = run_model(bnb1,'bnb1', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  0.0269317626953125
Prediction time:  0.02597784996032715
Precision Score: Train 0.35961, Test 0.34523
Recall Score:    Train 0.56461, Test 0.50565
Accuracy Score:  Train 0.64063, Test 0.61733
F1 Score:        Train 0.43937, Test 0.41032
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2899, percentage = 35.9365%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1029, percentage = 38.2670%
```




```
----------------
              precision    recall  f1-score   support

           0       0.79      0.66      0.72      1981
           1       0.35      0.51      0.41       708

    accuracy                           0.62      2689
   macro avg       0.57      0.58      0.56      2689
weighted avg       0.67      0.62      0.64      2689
```

Decision Tree

Output - dt3 - dt7, plot_feature_performances()

Already ran initial decision trees above during Exploratory modeling

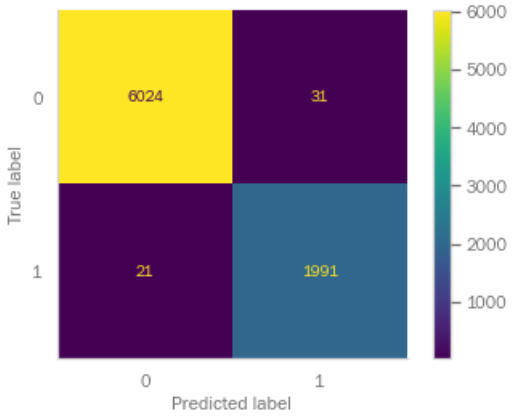
Ran on SMOTE and ohe data * **but not scaled** * data

Decision Tree with Gini instead of Entropy

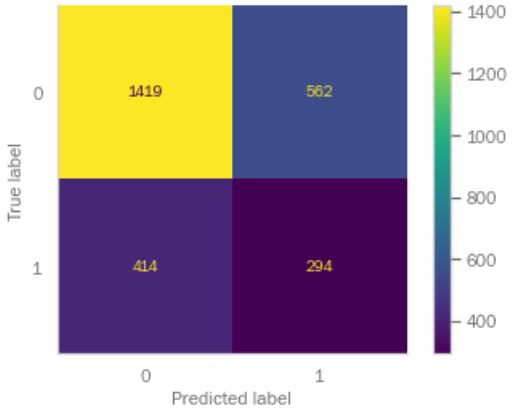
```
Fit time:        0.09470987319946289
Prediction time: 0.007979393005371094
Precision Score: Train 0.98467, Test 0.34346
Recall Score:    Train 0.98956, Test 0.41525
Accuracy Score:  Train 0.99355, Test 0.63704
F1 Score:        Train 0.98711, Test 0.37596
```

In [171]:
```
dt3 = DecisionTreeClassifier(random_state=10) # with GINI default

metrics_df = run_model(dt3, 'dt3', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

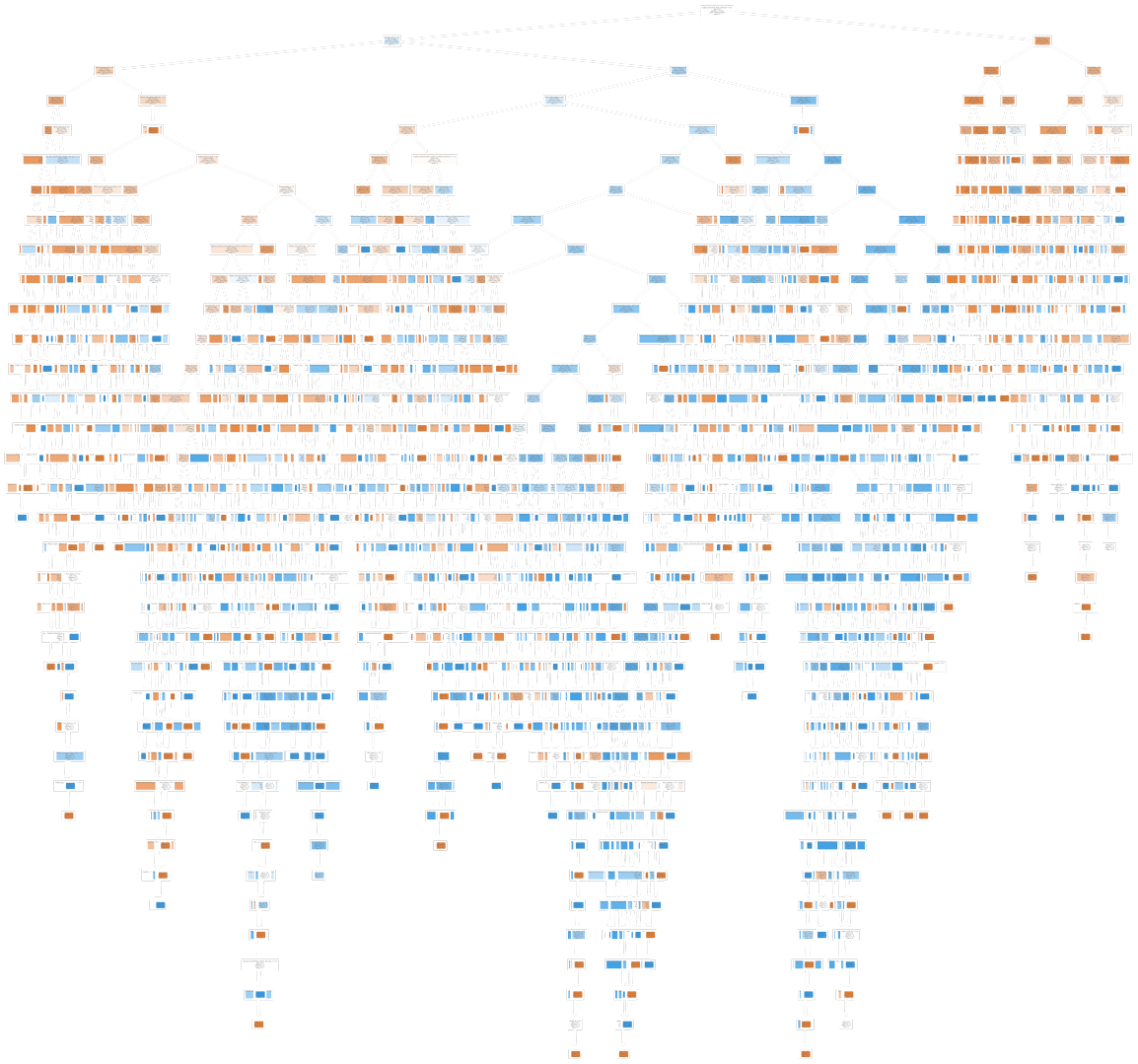
```
Fit time:  0.14694476127624512
Prediction time:  0.01695537567138672
Precision Score: Train 0.98467, Test 0.34346
Recall Score:    Train 0.98956, Test 0.41525
Accuracy Score:  Train 0.99355, Test 0.63704
F1 Score:        Train 0.98711, Test 0.37596
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 52, percentage = 0.6446%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 976, percentage = 36.2960%
```




```
----------------
              precision    recall  f1-score   support

           0       0.77      0.72      0.74      1981
           1       0.34      0.42      0.38       708

    accuracy                           0.64      2689
   macro avg       0.56      0.57      0.56      2689
weighted avg       0.66      0.64      0.65      2689


AUC is :0.57
```

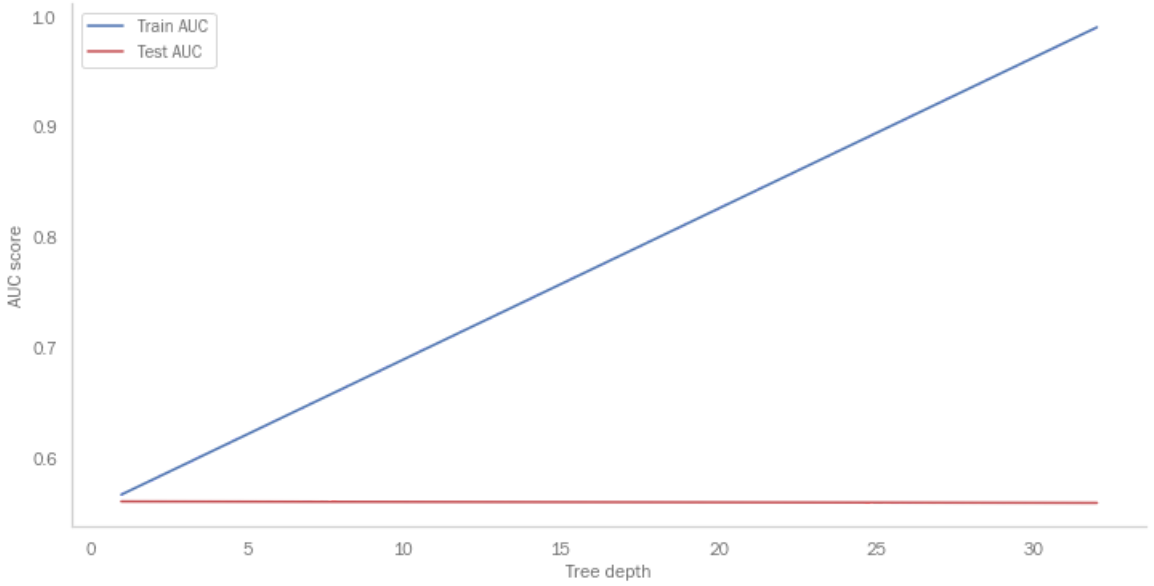


Decision Tree with Max Depth

Optimal Max_depth = 1

```
Fit time:        0.018917322158813477
Prediction time: 0.006983041763305664
Precision Score: Train 0.28143, Test 0.29343
Recall Score:    Train 0.89911, Test 0.88983
Accuracy Score:  Train 0.40226, Test 0.40684
F1 Score:        Train 0.42867, Test 0.44133
```

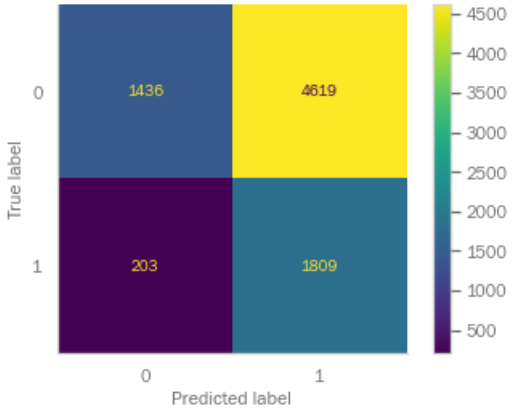
In [172]:
```
# Identify the optimal tree depth for given data
max_depths = np.linspace(1, 32, 2, endpoint=True)
train_results = []
test_results = []
for max_depth in max_depths:
   dt4 = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth, random_state=10)
   dt4.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)
   dt4_train_preds = dt4.predict(X_train_encoded_cleaned)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, dt4_train_preds)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous train results
   train_results.append(roc_auc)
   dt4_preds = dt4.predict(X_test_encoded_cleaned)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, dt4_preds)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous test results
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(max_depths, train_results, 'b', label='Train AUC')
plt.plot(max_depths, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('Tree depth')
plt.legend()
plt.show()
```



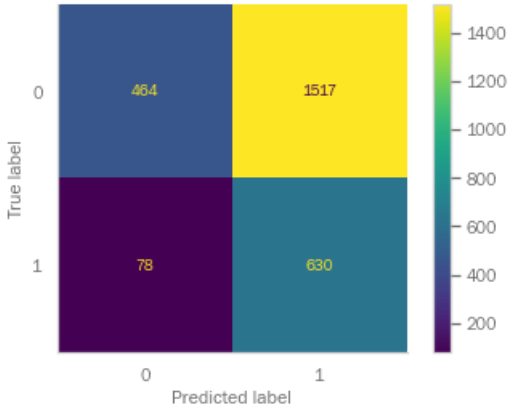
In [173]:
```
# Training error decreases with increasing tree depth
# Test error NEVER decreases? - nothing more to learn from deeper trees
# Optimal value seen here is 1
```

In [174]:
```
# Create the classifier, fit it on the training data and make predictions on the test set
dt4 = DecisionTreeClassifier(criterion='entropy', max_depth=1, random_state=10)
metrics_df = run_model(dt4, 'dt4', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

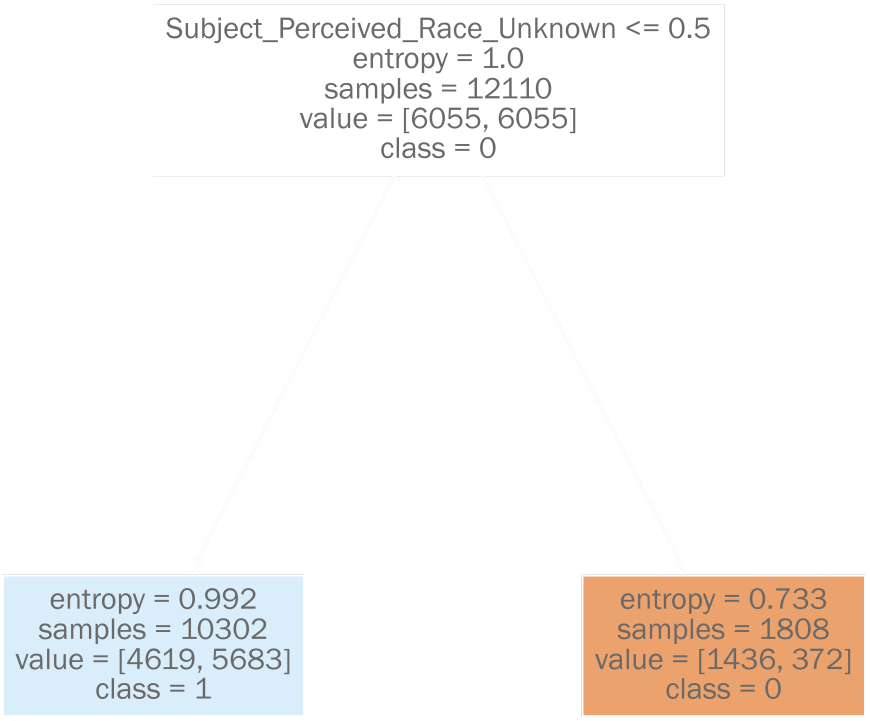
```
Fit time:  0.0189816951751709
Prediction time:  0.0069484710693359375
Precision Score: Train 0.28143, Test 0.29343
Recall Score:    Train 0.89911, Test 0.88983
Accuracy Score:  Train 0.40226, Test 0.40684
F1 Score:        Train 0.42867, Test 0.44133
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 4822, percentage = 59.7744%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1595, percentage = 59.3157%
```




```
----------------
              precision    recall  f1-score   support

           0       0.86      0.23      0.37      1981
           1       0.29      0.89      0.44       708

    accuracy                           0.41      2689
   macro avg       0.57      0.56      0.40      2689
weighted avg       0.71      0.41      0.39      2689


AUC is :0.56
```

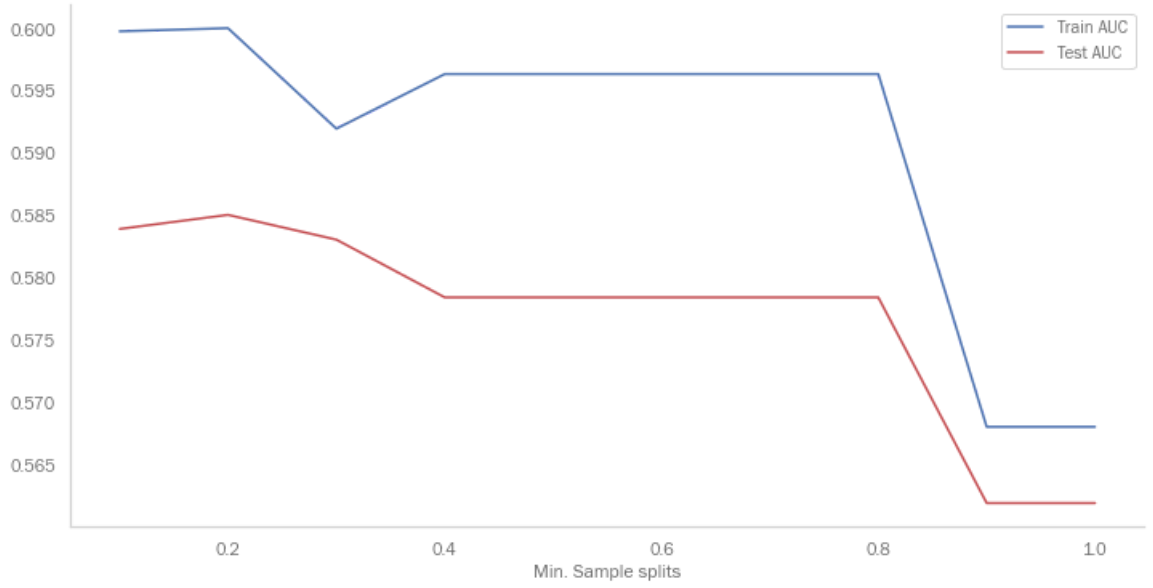


Decision Tree with Min Samples Split

Optimal min_samples_split = 0.4

```
Fit time:        0.03390908241271973
Prediction time: 0.005984306335449219
Precision Score: Train 0.33001, Test 0.33194
Recall Score:    Train 0.59245, Test 0.55932
Accuracy Score:  Train 0.59836, Test 0.58758
F1 Score:        Train 0.42390, Test 0.41662
```

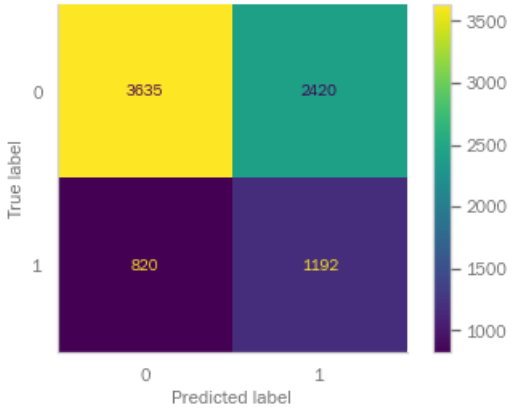
In [175]:
```
# Identify the optimal min-samples-split for given data
min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
train_results = []
test_results = []
for min_samples_split in min_samples_splits:
   dt5 = DecisionTreeClassifier(criterion='entropy', min_samples_split=min_samples_split, random_stat
   dt5.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)
   dt5_train_preds = dt5.predict(X_train_encoded_cleaned)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, dt5_train_preds)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   dt5_preds = dt5.predict(X_test_encoded_cleaned)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, dt5_preds)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(min_samples_splits, train_results, 'b', label='Train AUC')
plt.plot(min_samples_splits, test_results, 'r', label='Test AUC')
plt.xlabel('Min. Sample splits')
plt.legend()
plt.show()
```



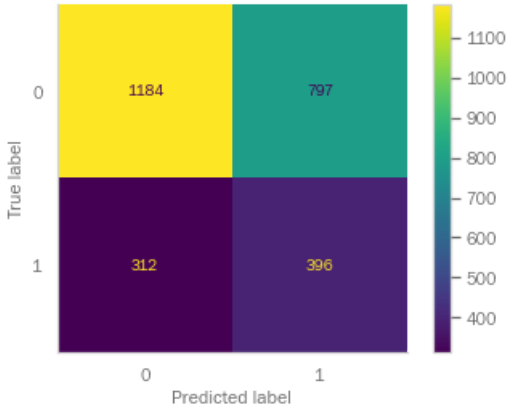
In [176]:
```
# AUC for both test and train data stabilizes at 0.4
# Further increase in minimum sample split does not improve learning
```

In [177]:
```
# Create the classifier, fit it on the training data and make predictions on the test set
dt5 = DecisionTreeClassifier(criterion='entropy', min_samples_split=0.4, random_state=10)

metrics_df = run_model(dt5, 'dt5', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  0.03390908241271973
Prediction time:  0.005984783172607422
Precision Score: Train 0.33001, Test 0.33194
Recall Score:    Train 0.59245, Test 0.55932
Accuracy Score:  Train 0.59836, Test 0.58758
F1 Score:        Train 0.42390, Test 0.41662
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 3240, percentage = 40.1636%
```

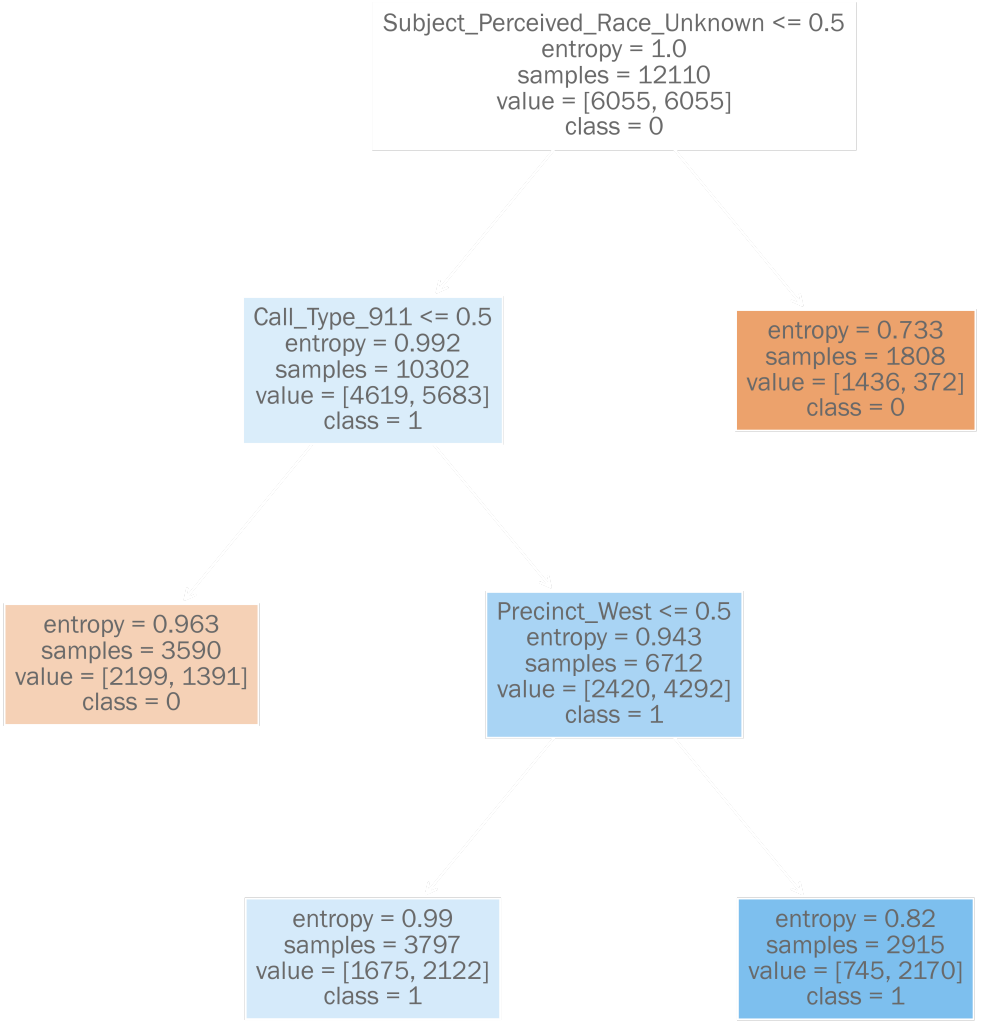



```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1109, percentage = 41.2421%
```




```
----------------
              precision    recall  f1-score   support
```

```
                0       0.79      0.60      0.68      1981
                1       0.33      0.56      0.42       708

         accuracy                           0.59      2689
        macro avg       0.56      0.58      0.55      2689
     weighted avg       0.67      0.59      0.61      2689


     AUC is :0.58
```

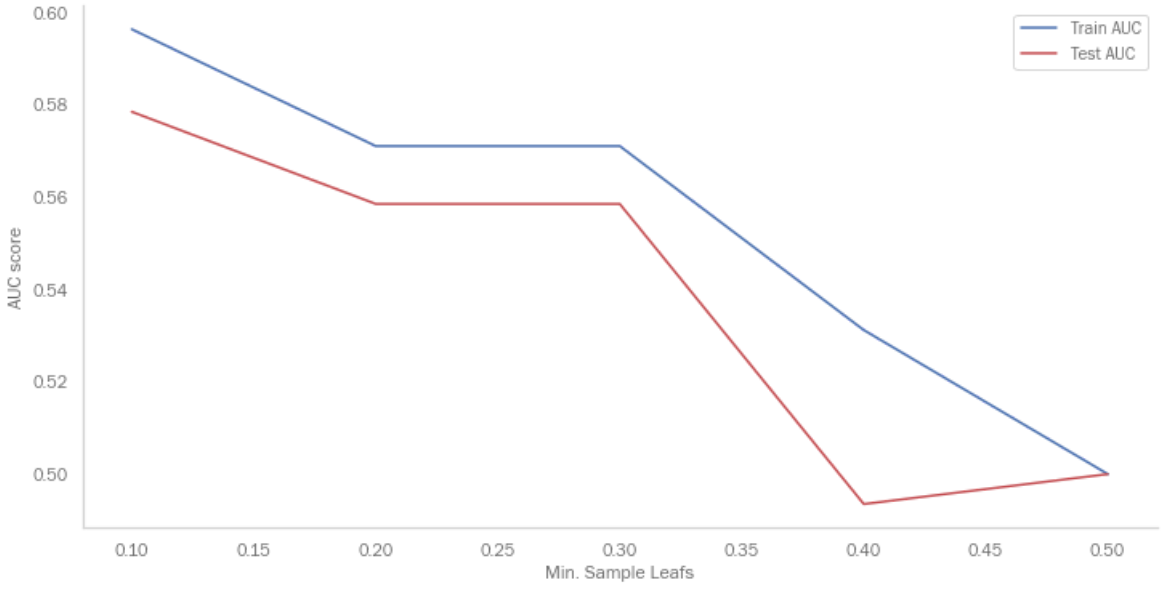


Decision Tree with Min Samples Leaf

Optimal min_samples_leaf = 0.10

```
Fit time:        0.034941673278808594
Prediction time: 0.0069773197174072266
Precision Score: Train 0.33001, Test 0.33194
Recall Score:    Train 0.59245, Test 0.55932
Accuracy Score:  Train 0.59836, Test 0.58758
F1 Score:        Train 0.42390, Test 0.41662
```

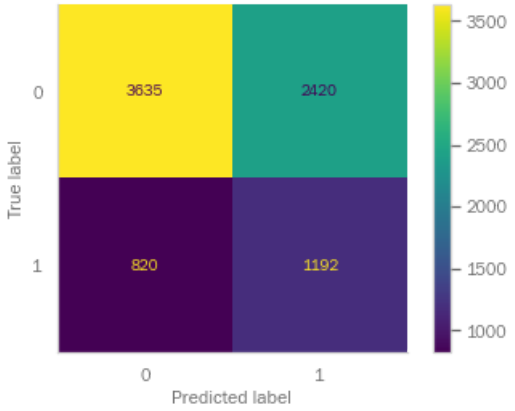
In [178]:
```
# Calculate the optimal value for minimum sample leafs
min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint=True)
train_results = []
test_results = []
for min_samples_leaf in min_samples_leafs:
   dt6 = DecisionTreeClassifier(criterion='entropy', min_samples_leaf=min_samples_leaf, random_state=
   dt6.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)
   dt6_train_preds = dt6.predict(X_train_encoded_cleaned)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, dt6_train_preds)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   dt6_preds = dt6.predict(X_test_encoded_cleaned)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, dt6_preds)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(min_samples_leafs, train_results, 'b', label='Train AUC')
plt.plot(min_samples_leafs, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('Min. Sample Leafs')
plt.legend()
plt.show()
```



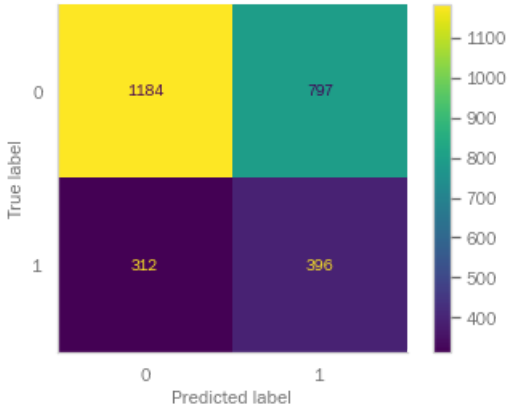
In [179]:
```
# AUC gives best value at 0.1 for both test and training sets
# Setting a higher minimum per leaf restricts our model too much
# The accuracy drops down if we continue to increase the parameter value
```

In [180]:
```
# Create the classifier, fit it on the training data and make predictions on the test set
dt6 = DecisionTreeClassifier(criterion='entropy', min_samples_leaf=0.10, random_state=10)

metrics_df = run_model(dt6, 'dt6', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

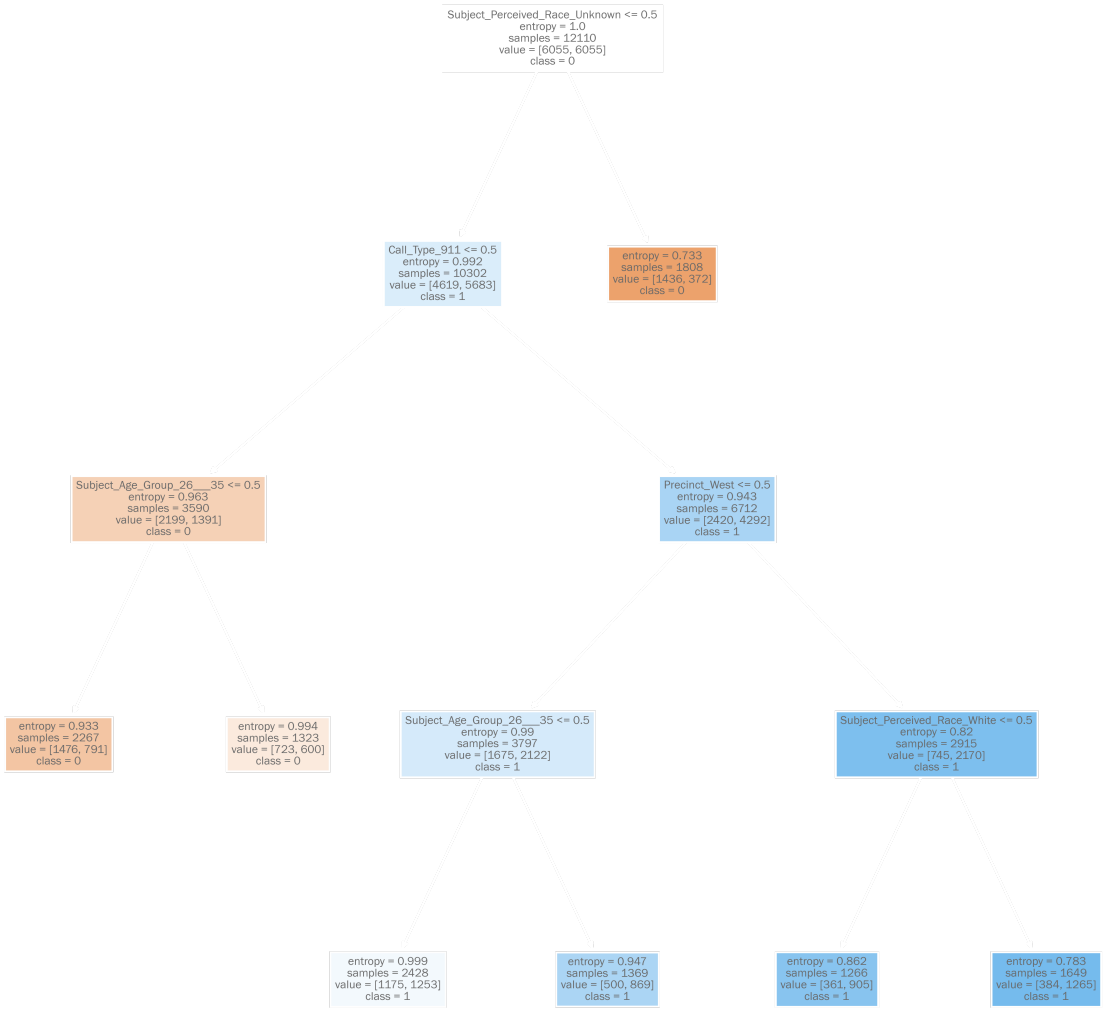
```
Fit time:  0.043915510177612305
Prediction time:  0.006954193115234375
Precision Score: Train 0.33001, Test 0.33194
Recall Score:    Train 0.59245, Test 0.55932
Accuracy Score:  Train 0.59836, Test 0.58758
F1 Score:        Train 0.42390, Test 0.41662
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 3240, percentage = 40.1636%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1109, percentage = 41.2421%
```




```
----------------
              precision    recall  f1-score   support

           0       0.79      0.60      0.68      1981
           1       0.33      0.56      0.42       708

    accuracy                           0.59      2689
   macro avg       0.56      0.58      0.55      2689
weighted avg       0.67      0.59      0.61      2689


AUC is :0.58
```

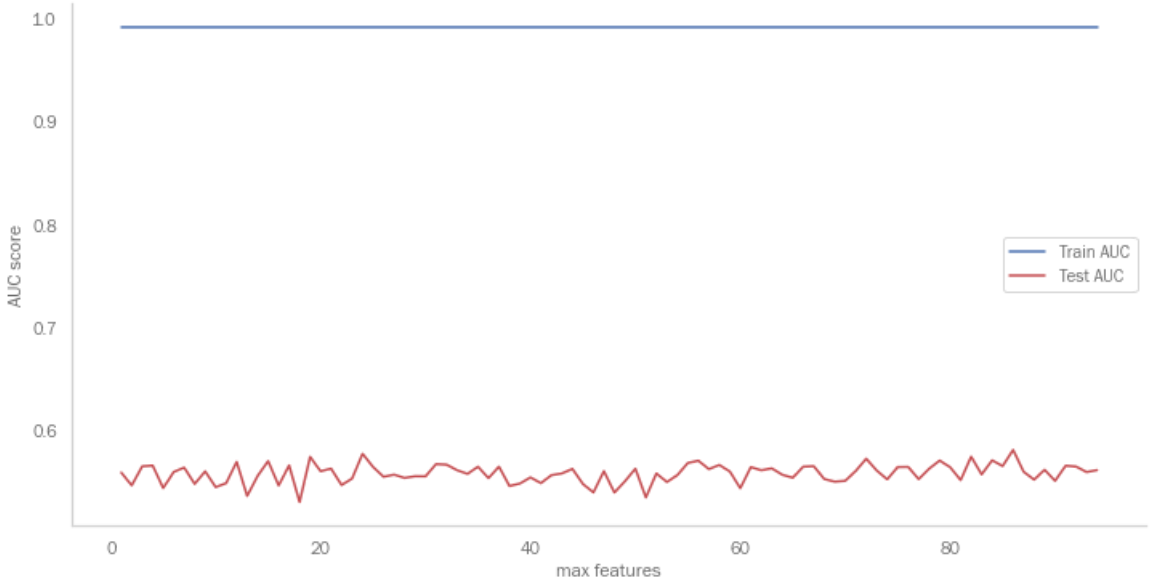


Decision Tree with Max Features

Optimal max_features = 25

```
Fit time:        0.0458531379699707
Prediction time: 0.007979154586791992
Precision Score: Train 0.98467, Test 0.34360
Recall Score:    Train 0.98956, Test 0.40960
Accuracy Score:  Train 0.99355, Test 0.63853
F1 Score:        Train 0.98711, Test 0.37371
```

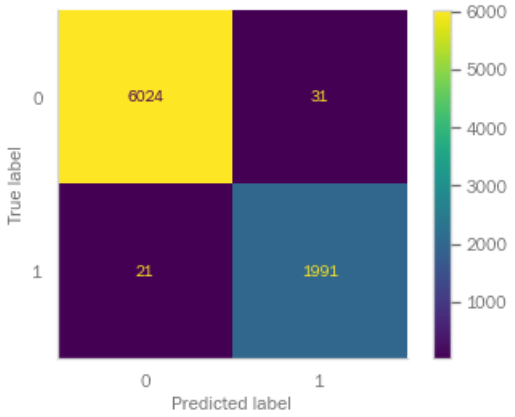
In [181]:
```
max_features = list(range(1, X_train_encoded_cleaned.shape[1]))
train_results = []
test_results = []
for max_feature in max_features:
   dt7 = DecisionTreeClassifier(criterion='entropy', max_features=max_feature, random_state=10)
   dt7.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)
   dt7_train_preds = dt7.predict(X_train_encoded_cleaned)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, dt7_train_preds)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   dt7_preds = dt7.predict(X_test_encoded_cleaned)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, dt7_preds)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(max_features, train_results, 'b', label='Train AUC')
plt.plot(max_features, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('max features')
plt.legend()
plt.show()
```



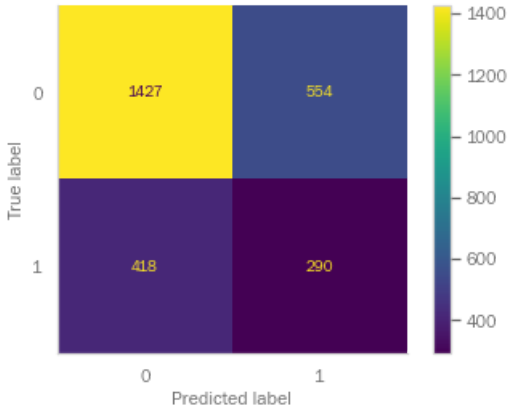
In [182]:
```
# No clear effect on the training dataset - flat AUC
# Some fluctuations in test AUC but not definitive enough to make a judgement
# Highest AUC value seen at 25
```

In [183]:
```
dt7 = DecisionTreeClassifier(criterion='entropy', max_features=25, random_state=10)

metrics_df = run_model(dt7, 'dt7', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  0.036902427673339844
Prediction time:  0.006979942321777344
Precision Score: Train 0.98467, Test 0.34360
Recall Score:    Train 0.98956, Test 0.40960
Accuracy Score:  Train 0.99355, Test 0.63853
F1 Score:        Train 0.98711, Test 0.37371
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 52, percentage = 0.6446%
```



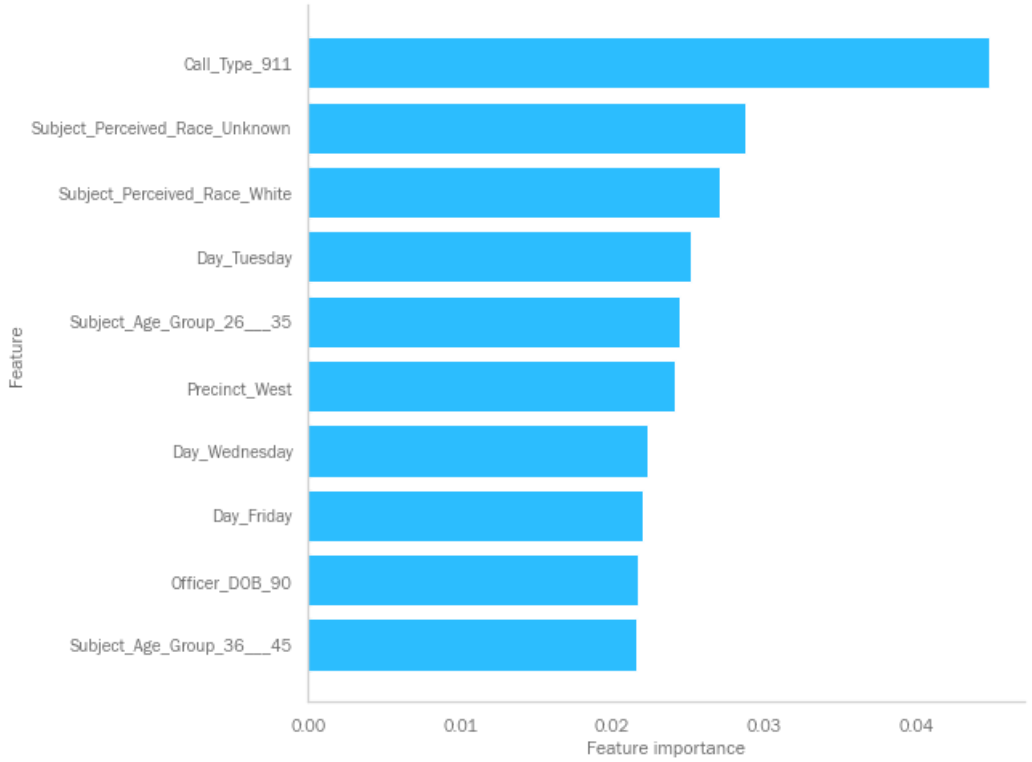

```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 972, percentage = 36.1473%
```




```
----------------
              precision    recall  f1-score   support

           0       0.77      0.72      0.75      1981
           1       0.34      0.41      0.37       708

    accuracy                           0.64      2689
   macro avg       0.56      0.56      0.56      2689
weighted avg       0.66      0.64      0.65      2689


AUC is :0.56
```

In [184]:
```
plot_ten_feature_importances(dt7, X_train_encoded_cleaned, 'dt3')
```

```
Images/feature_importancedt3.png
```



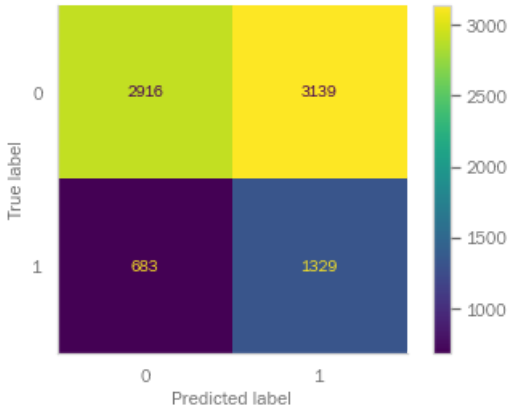
Decision Tree with All Optimal Parameters

Optimal Max_depth = 1
Optimal min_samples_split = 0.4
Optimal min_samples_leaf = 0.10
Optimal max_features = 25

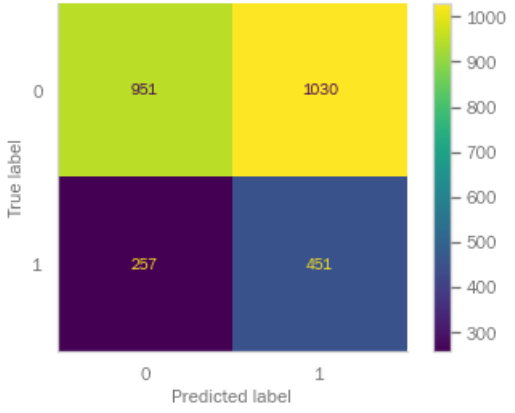
```
Fit time:        0.00797891616821289
Prediction time: 0.007016420364379883
Precision Score: Train 0.29745, Test 0.30452
Recall Score:    Train 0.66054, Test 0.63701
Accuracy Score:  Train 0.52622, Test 0.52138
F1 Score:        Train 0.41019, Test 0.41206
```

In [185]:
```
# Create the classifier, fit it on the training data and make predictions on the test set
dt8 = DecisionTreeClassifier(criterion='entropy', max_depth=1, min_samples_split=0.4,
                             min_samples_leaf=0.10, max_features=25, random_state=10)

metrics_df = run_model(dt8, 'dt8', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

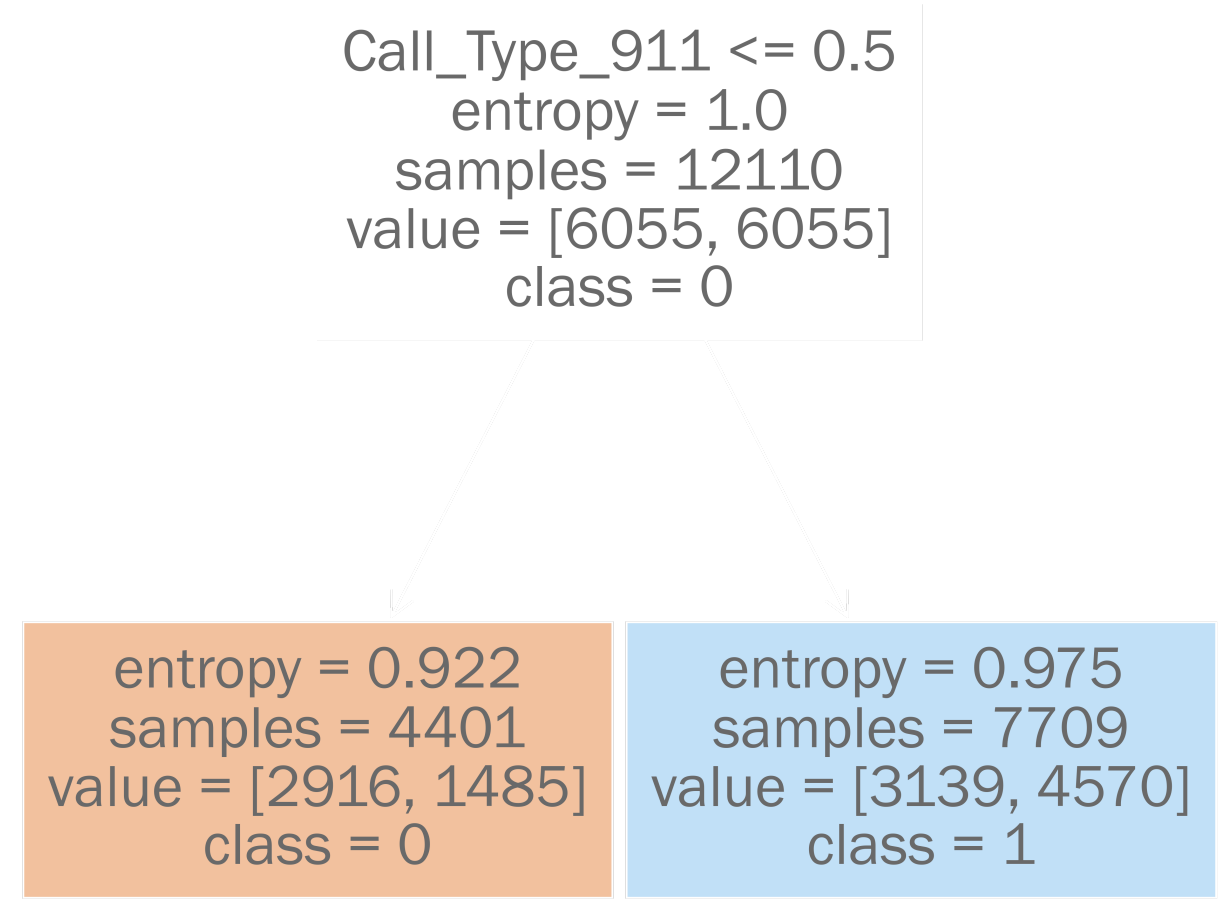
```
Fit time:  0.013962507247924805
Prediction time:  0.007980108261108398
Precision Score: Train 0.29745, Test 0.30452
Recall Score:    Train 0.66054, Test 0.63701
Accuracy Score:  Train 0.52622, Test 0.52138
F1 Score:        Train 0.41019, Test 0.41206
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 3822, percentage = 47.3782%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1287, percentage = 47.8617%
```



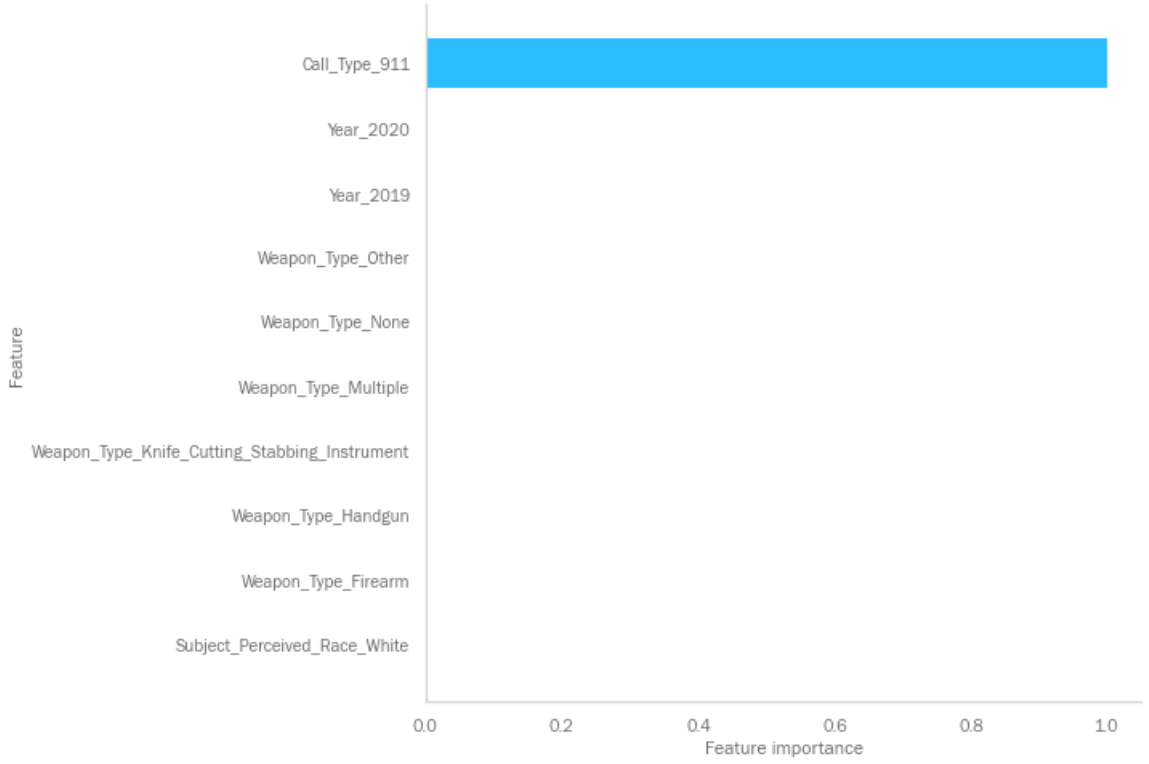
```
----------------
              precision    recall  f1-score   support

           0       0.79      0.48      0.60      1981
           1       0.30      0.64      0.41       708

    accuracy                           0.52      2689
   macro avg       0.55      0.56      0.50      2689
weighted avg       0.66      0.52      0.55      2689


AUC is :0.56
```



In [186]:
```
# Max Depth appears to have had the best impact on our model
```

In [187]:
```
plot_ten_feature_importances(dt8, X_train_encoded_cleaned, 'dt4')
```

```
Images/feature_importancedt4.png
```



Ensemble Methods

Output - bagged_tree, forest, rf_tree_1

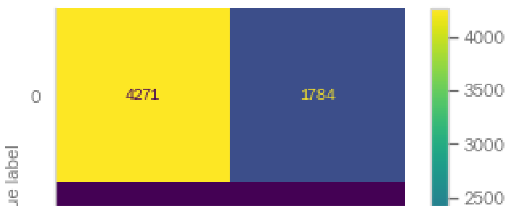
Ran on ohe data * **but not scaled** * data

Bagged Tree

```
Fit time:        0.6552150249481201
Prediction time: 0.14162206649780273
Precision Score: Train 0.35747, Test 0.35837
Recall Score:    Train 0.50547, Test 0.47175
Accuracy Score:  Train 0.65006, Test 0.63853
F1 Score:        Train 0.41878, Test 0.40732
```

In [188]:
```
bagged_tree =  BaggingClassifier(DecisionTreeClassifier(criterion='gini', max_depth=5), n_estimators=

metrics_df = run_model(bagged_tree, 'bagged_tree', X_train_encoded_cleaned, X_test_encoded_cleaned,
                       y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  0.7859272956848145
Prediction time:  0.176499605178833
Precision Score: Train 0.35943, Test 0.36066
Recall Score:    Train 0.49751, Test 0.46610
Accuracy Score:  Train 0.65353, Test 0.64187
F1 Score:        Train 0.41734, Test 0.40665
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2795, percentage = 34.6473%
```



In [189]:
```
bagged_tree.estimators_features_
```

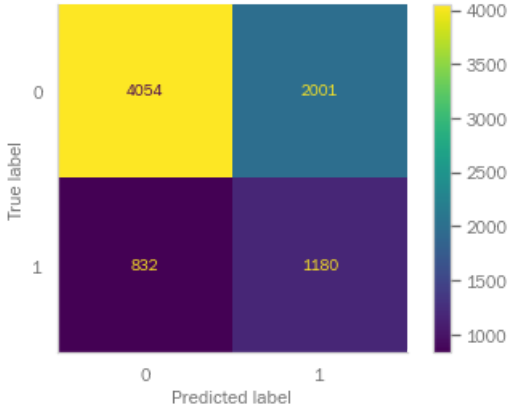
Out[189]:
```
[array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94]),
 array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94]),
 array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94]),
 array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
```

Random Forest

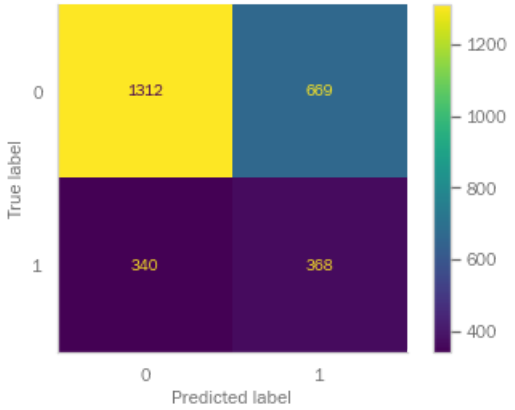
```
Fit time:        0.48969030380249023
Prediction time: 0.07782196998596191
Precision Score: Train 0.37111, Test 0.36214
Recall Score:    Train 0.58101, Test 0.52684
Accuracy Score:  Train 0.64993, Test 0.63109
F1 Score:        Train 0.45293, Test 0.42923
```

In [190]:
```
forest = RandomForestClassifier(n_estimators=100, max_depth= 5)

metrics_df = run_model(forest, 'forest', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  0.5465412139892578
Prediction time:  0.11173200607299805
Precision Score: Train 0.37095, Test 0.35487
Recall Score:    Train 0.58648, Test 0.51977
Accuracy Score:  Train 0.64882, Test 0.62477
F1 Score:        Train 0.45446, Test 0.42178
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2833, percentage = 35.1184%
```



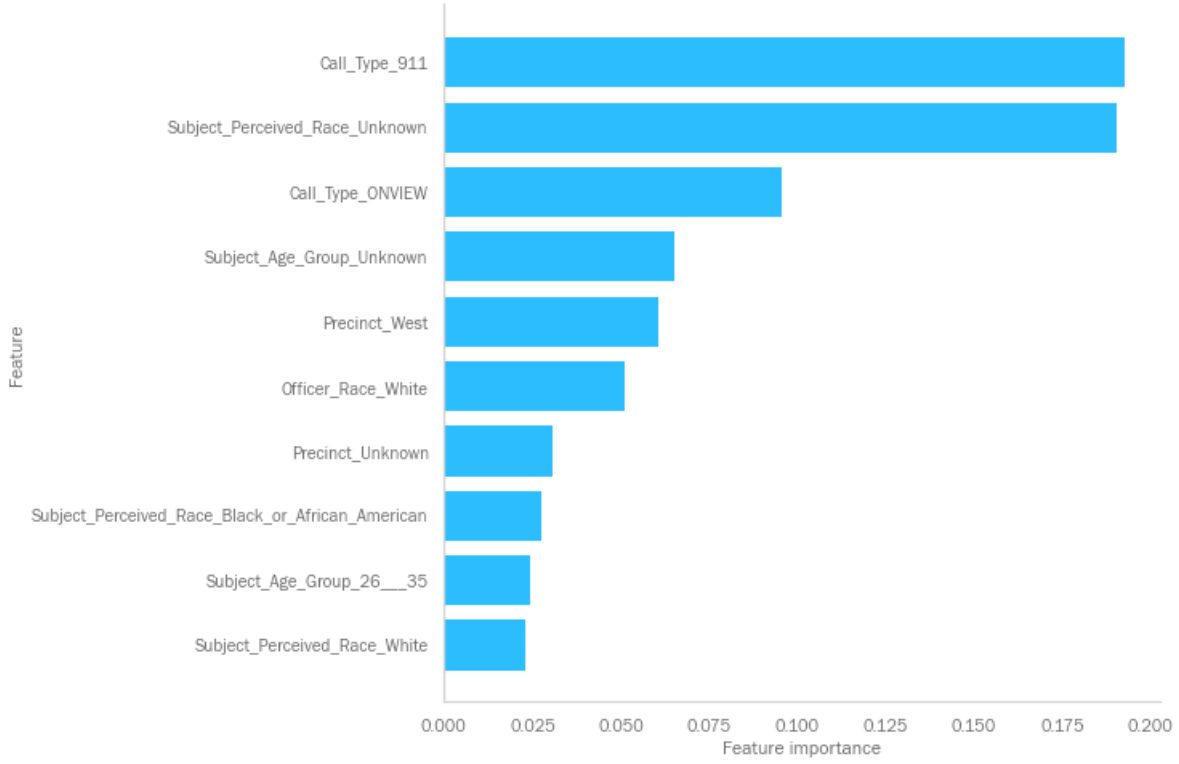

```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1009, percentage = 37.5232%
```



```
----------------
              precision    recall  f1-score   support

           0       0.79      0.66      0.72      1981
           1       0.35      0.52      0.42       708

    accuracy                           0.62      2689
   macro avg       0.57      0.59      0.57      2689
weighted avg       0.68      0.62      0.64      2689
```

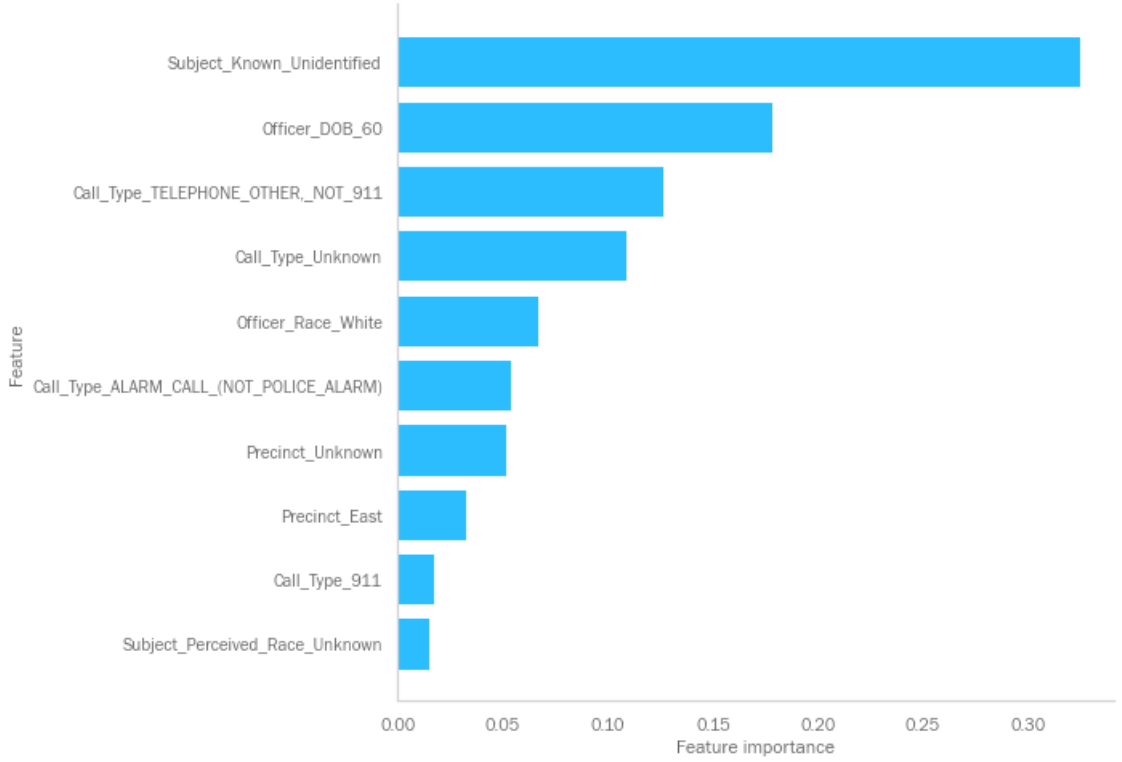
In [191]:
```
plot_ten_feature_importances(forest, X_train_encoded_cleaned, 'forest')
```

Images/feature_importanceforest.png



In [192]:
```
rf_tree_1 = forest.estimators_[0]
plot_ten_feature_importances(rf_tree_1, X_train_encoded_cleaned, 'rf_tree_1')
```

```
Images/feature_importancerf_tree_1.png
```



Grid Search

Output - gs_clf, gs_tree, gs_param_grid, dt_gs_training_score, dt_gs_testing_score, rf_clf, mean_rf_cv_score, rf_param_grid, rf_grid_search, dt_score, rf_score

Ran on ohe data * **but not scaled** * data

Grid Search with Decision Tree

Optimal criterion = 'gini'
Optimal min_samples_split = 0.1
Optimal max_depth = 10

```
Fit time:        2.5920615196228027
Prediction time: 0.0060138702392578125
Precision Score: Train 0.35304, Test 0.35579
Recall Score:    Train 0.51044, Test 0.47740
Accuracy Score:  Train 0.64460, Test 0.63481
F1 Score:        Train 0.41739, Test 0.40772
```

In [193]:

```
gs_clf = DecisionTreeClassifier()

gs_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 5, 10, 20],
    'min_samples_split': [.1, .3, .7, .9]
}

gs_tree = GridSearchCV(gs_clf, gs_param_grid, cv=3, return_train_score=True)
gs_tree.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)

# Mean training score
dt_gs_training_score = np.mean(gs_tree.cv_results_['mean_train_score'])

# Mean test score
dt_gs_testing_score = gs_tree.score(X_test_encoded_cleaned, y_test)

print(f"Mean Training Score: {dt_gs_training_score :.2%}")
print(f"Mean Test Score: {dt_gs_testing_score :.2%}")
print("Best Parameter Combination Found During Grid Search:")

gs_tree.best_params_
```

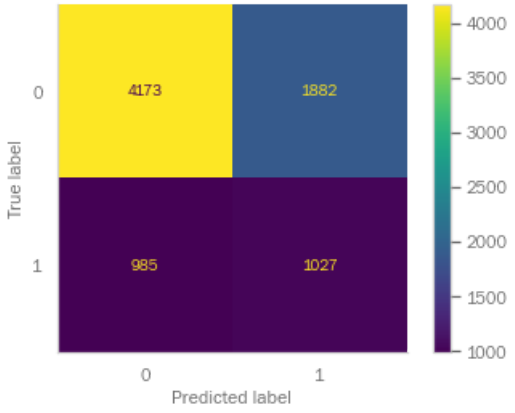
```
Mean Training Score: 64.69%
Mean Test Score: 63.48%
Best Parameter Combination Found During Grid Search:
```

Out[193]:

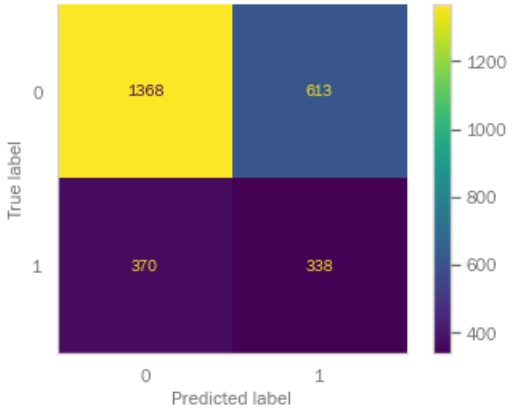
```
{'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 0.1}
```

In [194]:
```
metrics_df = run_model(gs_tree, 'gs_tree', X_train_encoded_cleaned, X_test_encoded_cleaned,
                       y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  2.6628737449645996
Prediction time:  0.005983829498291016
Precision Score: Train 0.35304, Test 0.35542
Recall Score:    Train 0.51044, Test 0.47740
Accuracy Score:  Train 0.64460, Test 0.63444
F1 Score:        Train 0.41739, Test 0.40747
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2867, percentage = 35.5399%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 983, percentage = 36.5563%
```




```
----------------
              precision    recall  f1-score   support

           0       0.79      0.69      0.74      1981
           1       0.36      0.48      0.41       708

    accuracy                           0.63      2689
   macro avg       0.57      0.58      0.57      2689
weighted avg       0.67      0.63      0.65      2689
```

In [195]:
```
# gs_tree_1 = gs_tree.estimators_[0]
# plot_ten_feature_importances(gs_tree_1, X_train_encoded_cleaned)
```

Grid Search with Random Forest

Optimal criterion = 'gini'
Optimal min_samples_split = 5
Optimal min_samples_leaf = 3
Optimal max_depth = None
Optimal num_estimators = 150

```
Fit time:        262.08292841911316
Prediction time: 0.15158867835998535
Precision Score: Train 0.63467, Test 0.38318
Recall Score:    Train 0.69334, Test 0.40537
Accuracy Score:  Train 0.82397, Test 0.67163
F1 Score:        Train 0.66271, Test 0.39396
```

In [196]:
```
rf_clf = RandomForestClassifier()
mean_rf_cv_score = np.mean(cross_val_score(rf_clf, X_train_encoded_cleaned, y_train, cv=3))

print(f"Mean Cross Validation Score for Random Forest Classifier: {mean_rf_cv_score :.2%}")
```

```
Mean Cross Validation Score for Random Forest Classifier: 74.01%
```

In [197]:
```
rf_param_grid = {
    'n_estimators': [100, 150, 200],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 6, 10, 20],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [3, 6]
}
```

In [198]:
```
rf_grid_search = GridSearchCV(rf_clf, rf_param_grid, cv=3)
rf_grid_search.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)

print(f"Training Accuracy: {rf_grid_search.best_score_ :.2%}")
print("")
print(f"Optimal Parameters: {rf_grid_search.best_params_}")
```

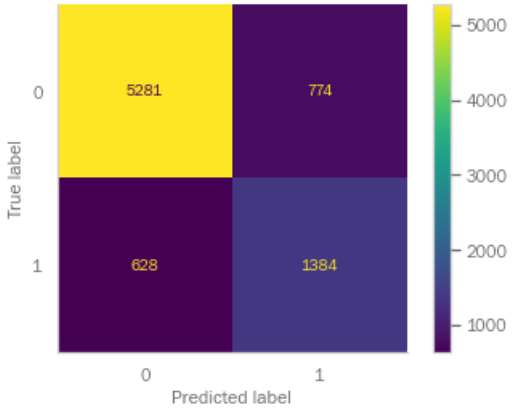
```
Training Accuracy: 76.14%

Optimal Parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 3, 'min_samples_spl
it': 5, 'n_estimators': 150}
```

In [199]:
```
dt_score = gs_tree.score(X_test_encoded_cleaned, y_test)
rf_score = rf_grid_search.score(X_test_encoded_cleaned, y_test)

print('Decision tree grid search: ', dt_score)
print('Random forest grid search: ', rf_score)
```

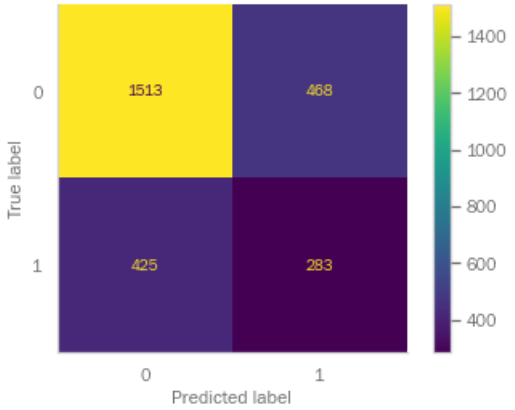
```
Decision tree grid search:  0.634436593529193
Random forest grid search:  0.6708813685384901
```

In [200]:
```
forest2 = RandomForestClassifier(criterion='gini', min_samples_leaf=3,min_samples_split=5,n_estimator
metrics_df = run_model(forest2, 'forest2', X_train_encoded_cleaned, X_test_encoded_cleaned,
                       y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  1.6455960273742676
Prediction time:  0.24434614181518555
Precision Score: Train 0.64133, Test 0.37683
Recall Score:    Train 0.68787, Test 0.39972
Accuracy Score:  Train 0.82621, Test 0.66791
F1 Score:        Train 0.66379, Test 0.38794
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 1402, percentage = 17.3794%
```



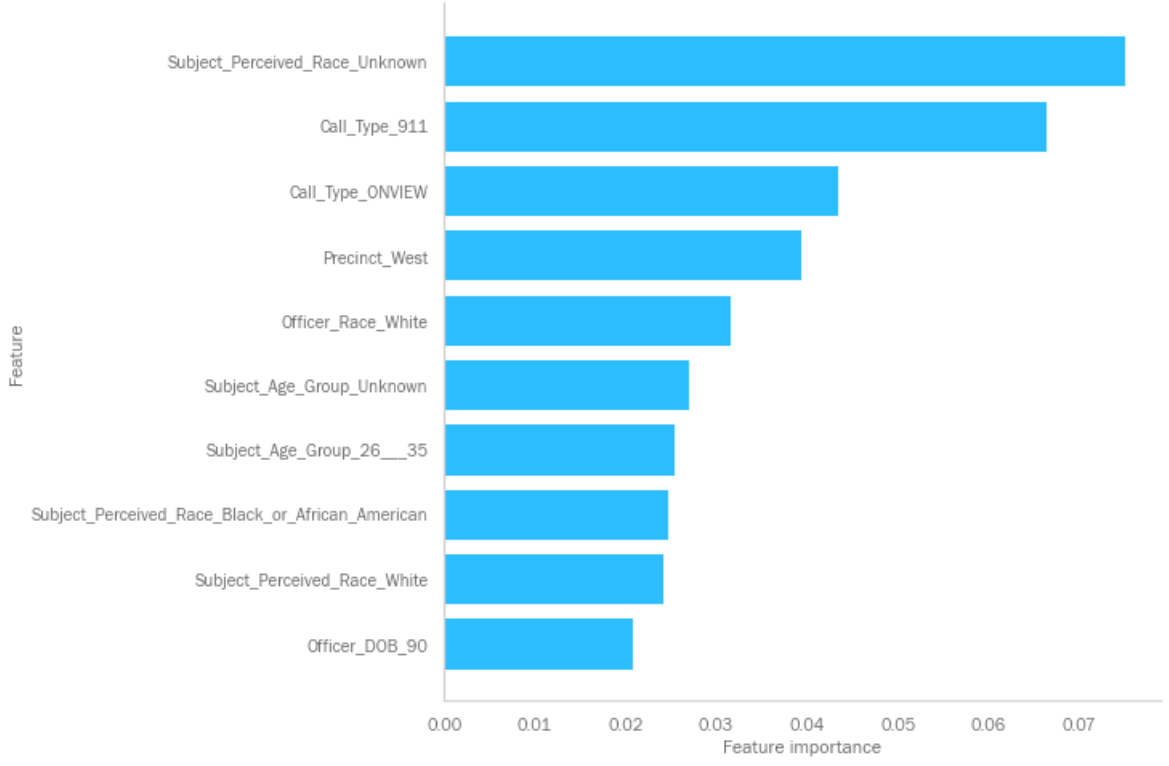

```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 893, percentage = 33.2094%
```



```
----------------
              precision    recall  f1-score   support

           0       0.78      0.76      0.77      1981
           1       0.38      0.40      0.39       708

    accuracy                           0.67      2689
   macro avg       0.58      0.58      0.58      2689
weighted avg       0.67      0.67      0.67      2689
```

In [201]:
```
# rf_tree_2 = rf_grid_search.estimators_[0]
plot_ten_feature_importances(forest2, X_train_encoded_cleaned, 'forest2')
```

Images/feature_importanceforest2.png



Boosting Models

Output - xgb_clf, grid_xgb, xgb_train_preds, xgb_test_preds, training_accuracy, test_accuracy, xgb_param_grid, grid_xgb_train_preds, grid_xgb_test_preds, xgb_best_parameters, adaboost_clf, gbt_clf

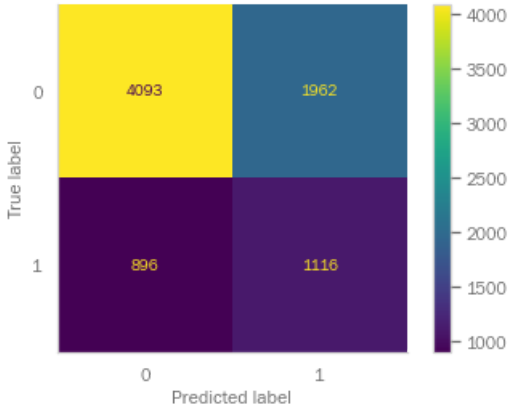
Ran on ohe data * **but not scaled** * data

AdaBoost

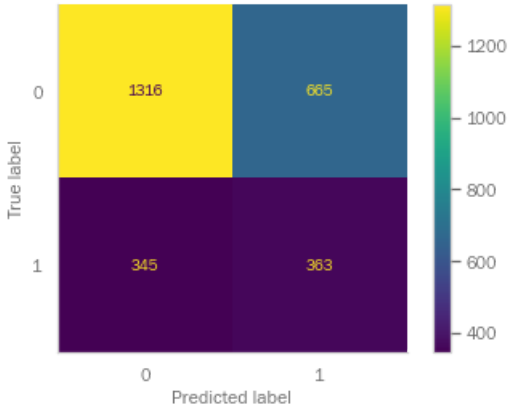
```
Fit time:        0.9584665298461914
Prediction time: 0.17253828048706055
Precision Score: Train 0.36257, Test 0.35311
Recall Score:    Train 0.55467, Test 0.51271
Accuracy Score:  Train 0.64572, Test 0.62440
F1 Score:        Train 0.43851, Test 0.41820
```

In [202]:
```
adaboost_clf = AdaBoostClassifier(random_state=42)
metrics_df = run_model(adaboost_clf, 'adaboost_clf', X_train_encoded_cleaned, X_test_encoded_cleaned,
                       y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  1.0922493934631348
Prediction time:  0.17450189590454102
Precision Score: Train 0.36257, Test 0.35311
Recall Score:    Train 0.55467, Test 0.51271
Accuracy Score:  Train 0.64572, Test 0.62440
F1 Score:        Train 0.43851, Test 0.41820
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2858, percentage = 35.4283%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 1010, percentage = 37.5604%
```



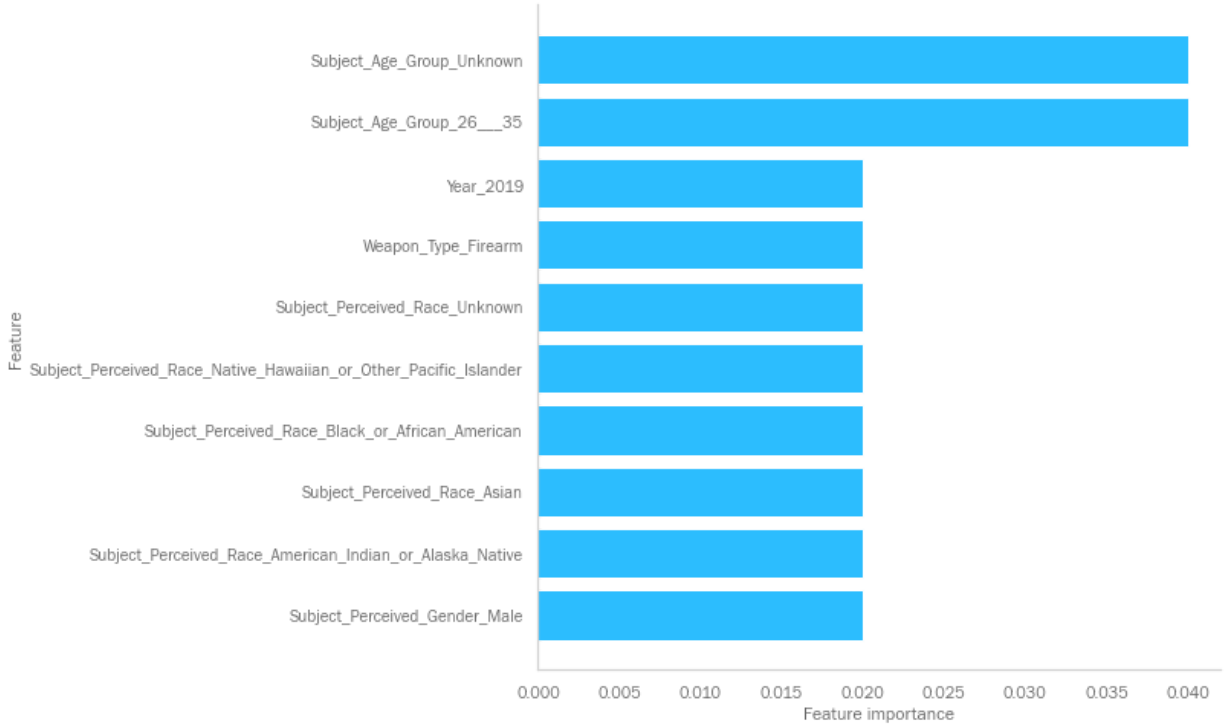

```
----------------
              precision    recall  f1-score   support

           0       0.79      0.66      0.72      1981
           1       0.35      0.51      0.42       708

    accuracy                           0.62      2689
   macro avg       0.57      0.59      0.57      2689
weighted avg       0.68      0.62      0.64      2689
```

In [203]:
```
print('Mean Adaboost Cross-Val Score (k=5):')
print(cross_val_score(adaboost_clf, X_train_encoded_cleaned, y_train, cv=5).mean())
```

```
Mean Adaboost Cross-Val Score (k=5):
0.750341747772705
```

In [204]:
```
plot_ten_feature_importances(adaboost_clf, X_train_encoded_cleaned, 'adaboost')
```

```
Images/feature_importanceadaboost.png
```

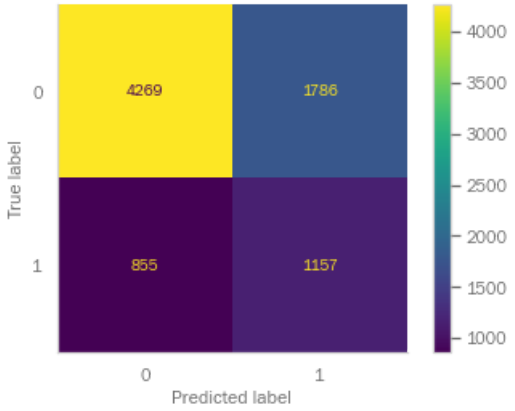


GradientBoost

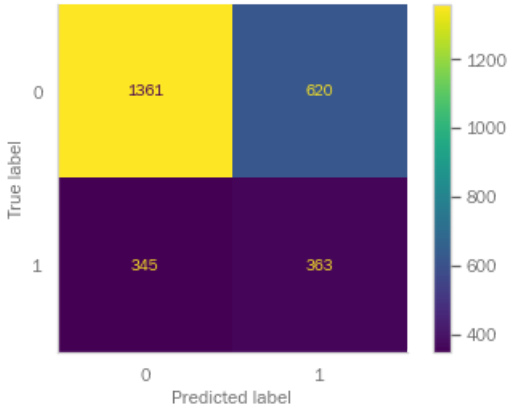
```
Fit time:        2.3008506298065186
Prediction time: 0.04802584648132324
Precision Score: Train 0.39314, Test 0.36928
Recall Score:    Train 0.57505, Test 0.51271
Accuracy Score:  Train 0.67262, Test 0.64113
F1 Score:        Train 0.46700, Test 0.42933
```

In [205]:
```
gbt_clf = GradientBoostingClassifier(random_state=42)
metrics_df = run_model(gbt_clf, 'gbt_clf', X_train_encoded_cleaned, X_test_encoded_cleaned,
                       y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  2.565162181854248
Prediction time:  0.021907567977905273
Precision Score: Train 0.39314, Test 0.36928
Recall Score:    Train 0.57505, Test 0.51271
Accuracy Score:  Train 0.67262, Test 0.64113
F1 Score:        Train 0.46700, Test 0.42933
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 2641, percentage = 32.7383%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 965, percentage = 35.8869%
```



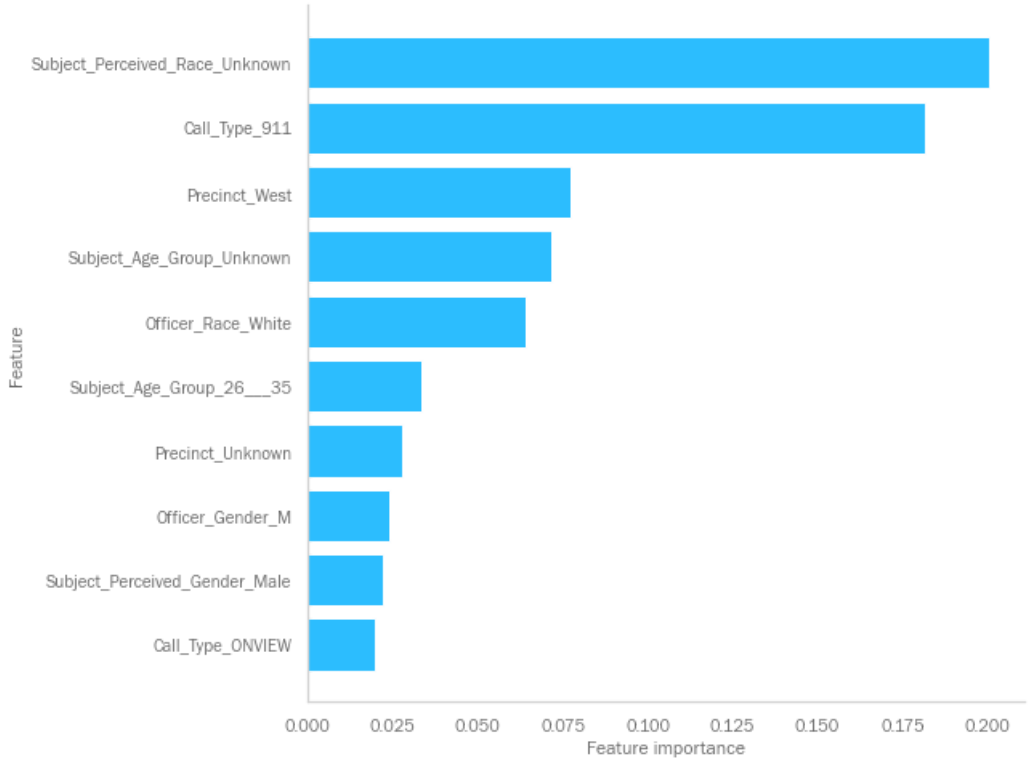

```
----------------
              precision    recall  f1-score   support

           0       0.80      0.69      0.74      1981
           1       0.37      0.51      0.43       708

    accuracy                           0.64      2689
   macro avg       0.58      0.60      0.58      2689
weighted avg       0.68      0.64      0.66      2689
```

In [206]:
```
print('Mean GBT Cross-Val Score (k=5):')
print(cross_val_score(gbt_clf, X_train_encoded_cleaned, y_train, cv=5).mean())
```

```
Mean GBT Cross-Val Score (k=5):
0.7495973314711402
```

In [207]:
```
plot_ten_feature_importances(gbt_clf, X_train_encoded_cleaned, 'gbt_clf')
```

```
Images/feature_importancegbt_clf.png
```

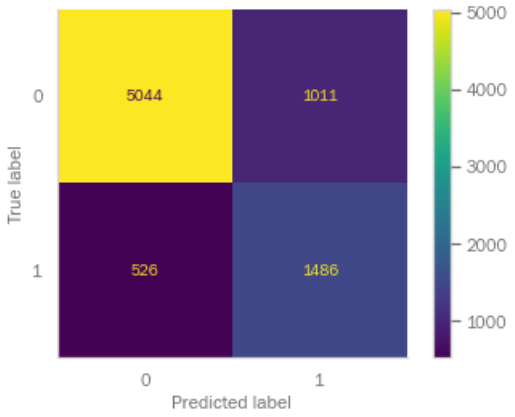


Initial XGBoost

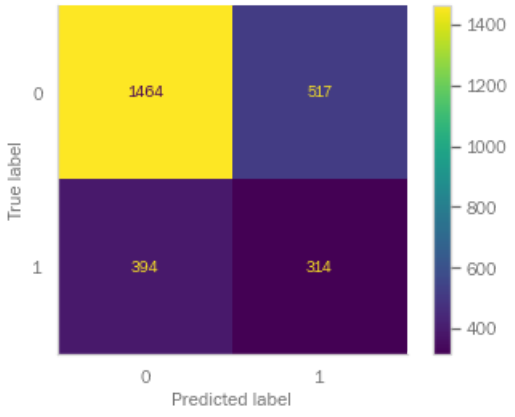
```
Fit time:         0.802851676940918
Prediction time: 0.032912254333496094
Precision Score: Train 0.59511, Test 0.37786
Recall Score:    Train 0.73857, Test 0.44350
Accuracy Score:  Train 0.80947, Test 0.66121
F1 Score:        Train 0.65913, Test 0.40806
```

In [208]:
```
xgb_clf = XGBClassifier()
xgb_clf.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)

# Predict on training and test sets
xgb_train_preds = xgb_clf.predict(X_train_encoded_cleaned)
xgb_test_preds = xgb_clf.predict(X_test_encoded_cleaned)

# Accuracy of training and test sets
xgb_training_accuracy = accuracy_score(y_train, xgb_train_preds)
xgb_test_accuracy = accuracy_score(y_test, xgb_test_preds)

print('Training Accuracy: {:.4}%'.format(xgb_training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(xgb_test_accuracy * 100))
print('------------------')
metrics_df = run_model(xgb_clf, 'xgb_clf', X_train_encoded_cleaned, X_test_encoded_cleaned,
                       y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Training Accuracy: 80.95%
Validation accuracy: 66.12%
------------------
Fit time:  1.024259090423584
Prediction time:  0.03789830207824707
Precision Score: Train 0.59511, Test 0.37786
Recall Score:    Train 0.73857, Test 0.44350
Accuracy Score:  Train 0.80947, Test 0.66121
F1 Score:        Train 0.65913, Test 0.40806
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 1537, percentage = 19.0529%
```



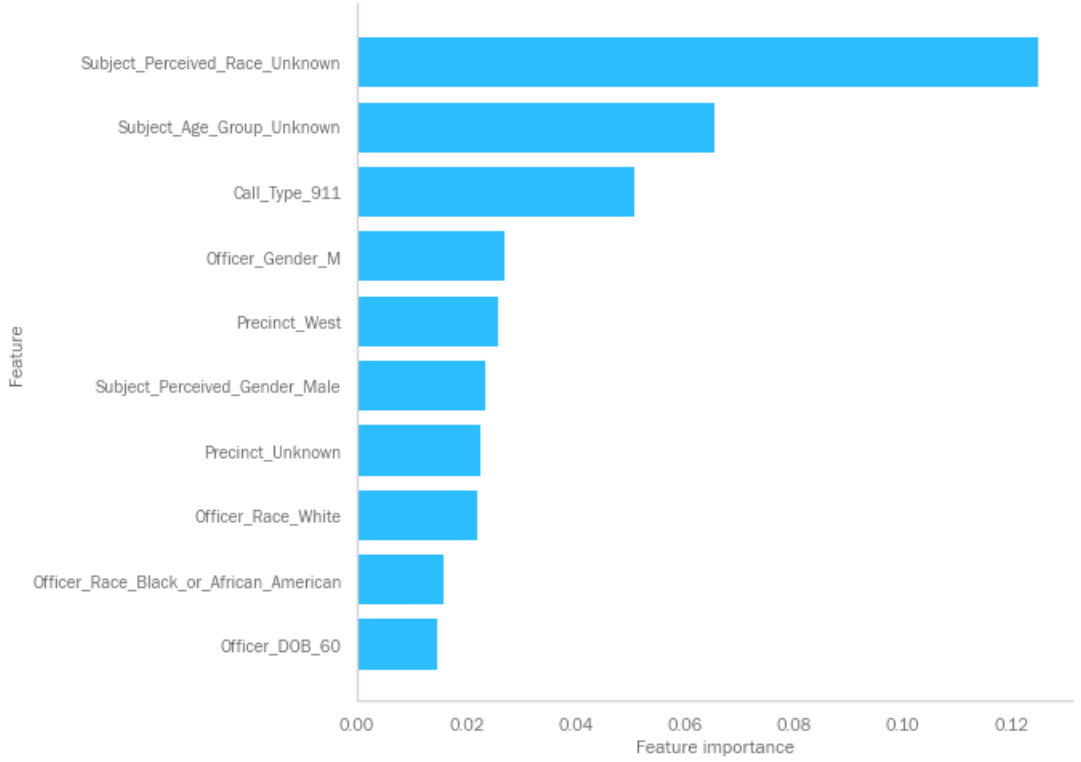

```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 911, percentage = 33.8788%
```



```
---------------
              precision    recall  f1-score   support

           0       0.79      0.74      0.76      1981
           1       0.38      0.44      0.41       708

    accuracy                           0.66      2689
   macro avg       0.58      0.59      0.59      2689
weighted avg       0.68      0.66      0.67      2689
```

In [209]:
```
plot_ten_feature_importances(xgb_clf, X_train_encoded_cleaned, 'xgb_clf')
```

```
Images/feature_importancexgb_clf.png
```



XGBoost with grid search

Grid Search found the following optimal parameters:
learning_rate: 0.2
max_depth: 6
min_child_weight: 1
n_estimators: 100
subsample: 0.7

```
Fit time:        38.56180787086487
Prediction time: 0.049866437911987305
Precision Score: Train 0.54782, Test 0.36758
Recall Score:    Train 0.69185, Test 0.45480
Accuracy Score:  Train 0.78071, Test 0.65043
F1 Score:        Train 0.61146, Test 0.40657
```

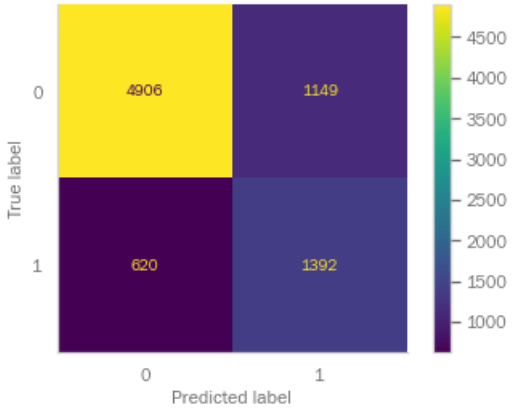
In [210]:
```
xgb_param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [6],
    'min_child_weight': [1, 2],
    'subsample': [0.5, 0.7],
    'n_estimators': [100],
}
```

In [211]:
```
grid_xgb = GridSearchCV(xgb_clf, xgb_param_grid, scoring='accuracy', cv=None, n_jobs=1)
grid_xgb.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)

xgb_best_parameters = grid_xgb.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(xgb_best_parameters.keys()):
    print('%s: %r' % (param_name, xgb_best_parameters[param_name]))

grid_xgb_train_preds = grid_xgb.predict(X_train_encoded_cleaned)
grid_xgb_test_preds = grid_xgb.predict(X_test_encoded_cleaned)
grid_xgb_training_accuracy = accuracy_score(y_train, grid_xgb_train_preds)
grid_xgb_test_accuracy = accuracy_score(y_test, grid_xgb_test_preds)

print('')
print('Training Accuracy: {:.4}%'.format(grid_xgb_training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(grid_xgb_test_accuracy * 100))
```

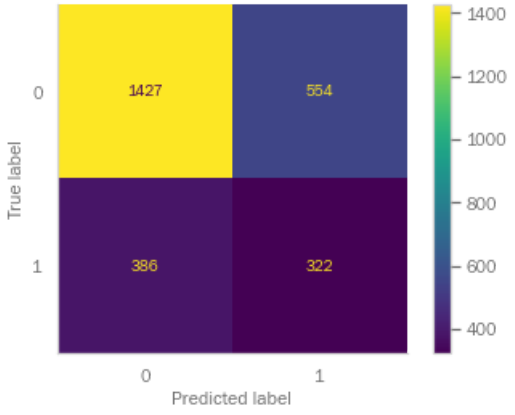
```
Grid Search found the following optimal parameters:
learning_rate: 0.2
max_depth: 6
min_child_weight: 1
n_estimators: 100
subsample: 0.7

Training Accuracy: 78.07%
Validation accuracy: 65.04%
```

In [212]:
```
metrics_df = run_model(grid_xgb, 'grid_xgb', X_train_encoded_cleaned, X_test_encoded_cleaned,
                       y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  44.42711281776428
Prediction time:  0.04388236999511719
Precision Score: Train 0.54782, Test 0.36758
Recall Score:    Train 0.69185, Test 0.45480
Accuracy Score:  Train 0.78071, Test 0.65043
F1 Score:        Train 0.61146, Test 0.40657
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 1769, percentage = 21.9288%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 940, percentage = 34.9572%
```




```
----------------
              precision    recall  f1-score   support

           0       0.79      0.72      0.75      1981
           1       0.37      0.45      0.41       708

    accuracy                           0.65      2689
   macro avg       0.58      0.59      0.58      2689
weighted avg       0.68      0.65      0.66      2689
```

In [213]:
```
# plot_ten_feature_importances(grid_xgb, X_train_encoded_cleaned)
```

Support Vector Classification [...]

```
Fit time:        46.65893816947937
Prediction time: 5.1941304206848145
Precision Score: Train 0.97655, Test 0.34218
Recall Score:    Train 0.99354, Test 0.34605
Accuracy Score:  Train 0.99244, Test 0.65266
F1 Score:        Train 0.98497, Test 0.34410
```

Review Top Performers

In [216]:
```
metrics_df[['Model','Train_Accuracy', 'Test_Accuracy', 'Train_F1', 'Test_F1']].sort_values('Test_Accu
```

Out[216]:

	Model	Train_Accuracy	Test_Accuracy	Train_F1	Test_F1
9	gnb1	0.358869	0.369654	0.409049	0.417726
12	dt4	0.402256	0.406843	0.428673	0.441331
4	dummy1	0.497707	0.507996	0.334210	0.351153
16	dt8	0.526218	0.521383	0.410185	0.412060
14	dt6	0.598364	0.587579	0.423898	0.416623
13	dt5	0.598364	0.587579	0.423898	0.416623
6	knn1	0.795215	0.614727	0.661059	0.390588
10	bnb1	0.640635	0.617330	0.439373	0.410315
2	dt1	0.695674	0.620677	0.520601	0.417143
0	logreg1	0.648320	0.621792	0.442304	0.415181
7	knn2	0.724557	0.622164	0.582017	0.428571
5	logreg3	0.648940	0.622536	0.442739	0.416331
1	logreg2	0.648444	0.622536	0.442391	0.416331
21	adaboost_clf	0.645717	0.624396	0.438507	0.418203
18	forest	0.648816	0.624768	0.454458	0.421777
8	knn3	0.993554	0.628858	0.987110	0.434240
3	dt2	0.993554	0.634065	0.987110	0.376426
19	gs_tree	0.644601	0.634437	0.417395	0.407474
11	dt3	0.993554	0.637040	0.987110	0.375959
15	dt7	0.993554	0.638527	0.987110	0.373711
22	gbt_clf	0.672617	0.641131	0.467003	0.429332
17	bagged_tree	0.653527	0.641874	0.417344	0.406654
24	grid_xgb	0.780712	0.650428	0.611465	0.406566
25	SVC	0.992438	0.652659	0.984972	0.344101
23	xgb_clf	0.809471	0.661212	0.659126	0.408057
20	forest2	0.826206	0.667906	0.663789	0.387937

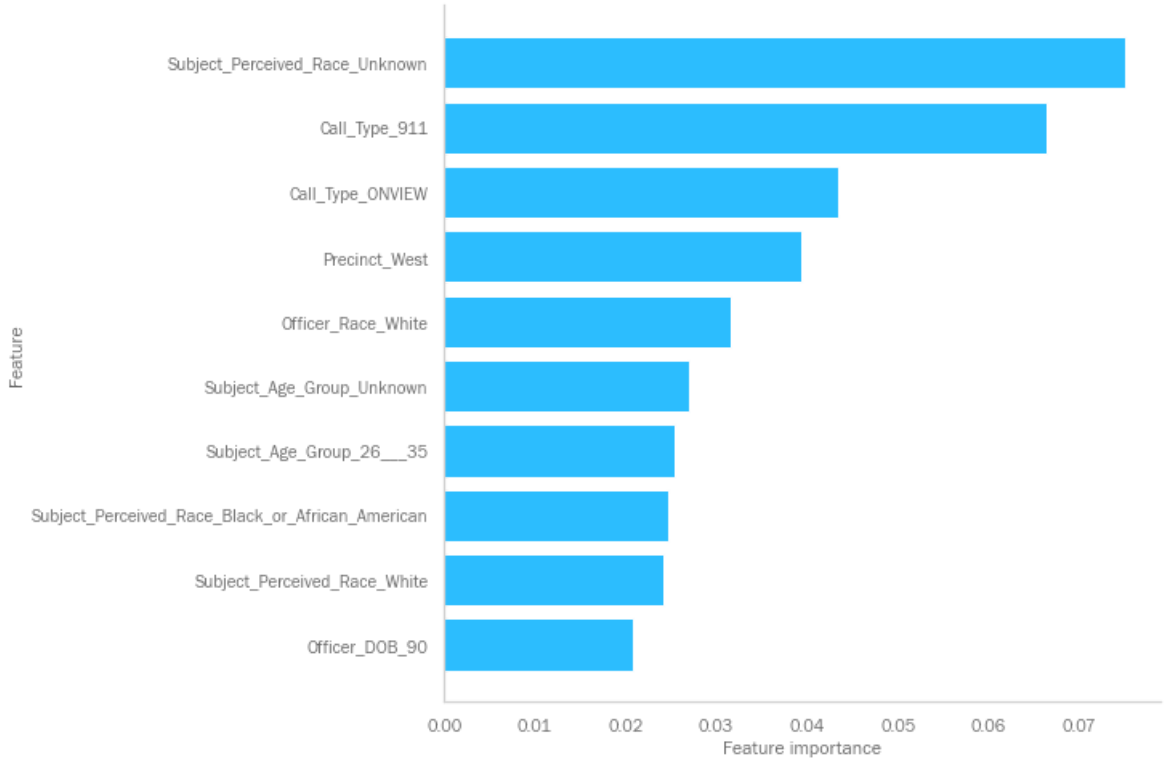
Random Forest

In [217]:
```
forest2 # This came out of the grid search.
# I had already added extra parameters to the grid so I think this is the best I can get from Random
```

Out[217]:
```
RandomForestClassifier(min_samples_leaf=3, min_samples_split=5,
                       n_estimators=150)
```

In [218]:
```
plot_ten_feature_importances(forest2, X_train_encoded_cleaned, 'forest2')
```

Images/feature_importanceforest2.png

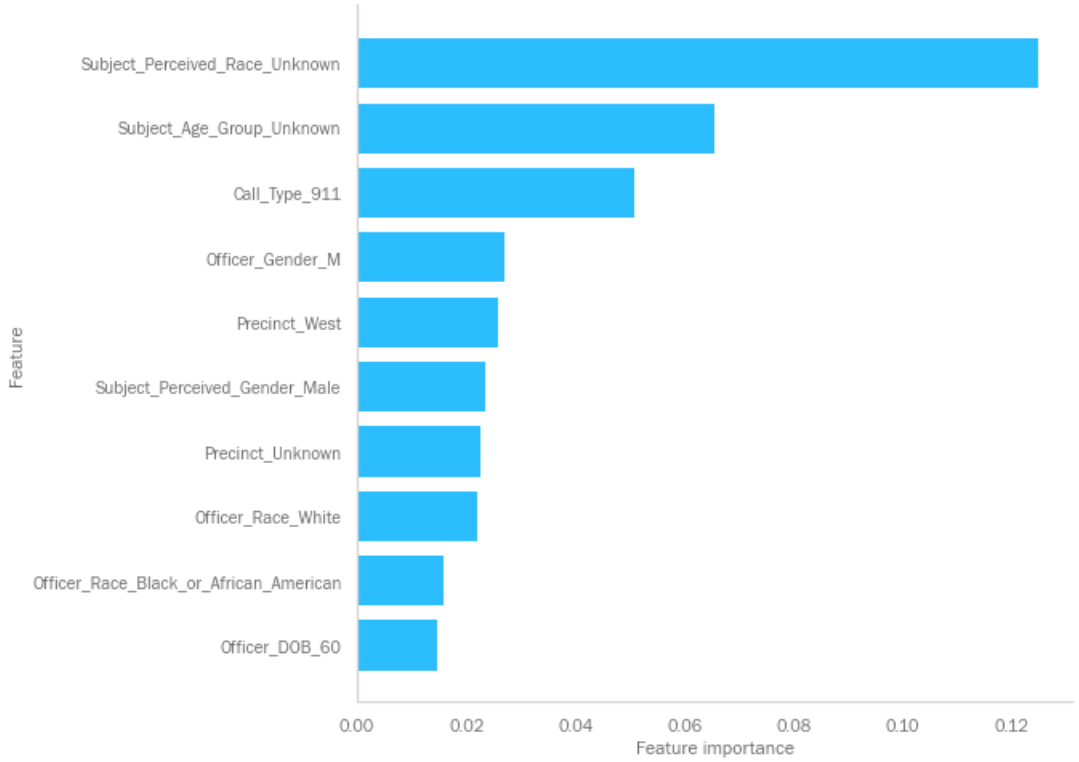


In [219]:
```
xgb_clf # Done with all default parameters.  Grid search should do better than this.
```

Out[219]:
```
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_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [220]:
```
plot_ten_feature_importances(xgb_clf, X_train_encoded_cleaned, 'xgb_clf')
```

```
Images/feature_importancexgb_clf.png
```



XGBoost

In [221]:
```
grid_xgb #Grid Search found the following optimal parameters:
# learning_rate: 0.2
# max_depth: 6
# min_child_weight: 1
# n_estimators: 100
# subsample: 0.7

# The param_grid should have included the default parameters so this model coud do no worse than the
# I will run again with extra parameters.
```

Out[221]:
```
GridSearchCV(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_parallel_tree=1, random_state=0,
                                     reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, subsample=1,
                                     tree_method='exact', validate_parameters=1,
                                     verbosity=None),
             n_jobs=1,
             param_grid={'learning_rate': [0.1, 0.2], 'max_depth': [6],
                         'min_child_weight': [1, 2], 'n_estimators': [100],
                         'subsample': [0.5, 0.7]},
             scoring='accuracy')
```

In [222]:
```
xgb_param_grid2 = {
    'learning_rate': [0.3, 0.4],
    'max_depth': [6, 7],
    'min_child_weight': [1, 2, 3],
    'subsample': [0.5, 0.7, 1.0],
    'n_estimators': [50, 100, 150],
}
```

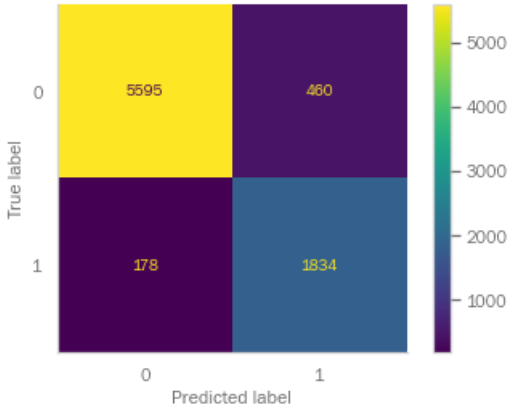
In [223]:

```
import time

grid_xgb2 = GridSearchCV(xgb_clf, xgb_param_grid2, scoring='accuracy', cv=None, n_jobs=1)

toc = time.time()
grid_xgb2.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)
fittime = time.time() - toc
print(fittime)

xgb2_best_parameters = grid_xgb2.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(xgb2_best_parameters.keys()):
    print('%s: %r' % (param_name, xgb2_best_parameters[param_name]))

grid_xgb2_train_preds = grid_xgb2.predict(X_train_encoded_cleaned)
grid_xgb2_test_preds = grid_xgb2.predict(X_test_encoded_cleaned)
grid_xgb2_training_accuracy = accuracy_score(y_train, grid_xgb2_train_preds)
grid_xgb2_test_accuracy = accuracy_score(y_test, grid_xgb2_test_preds)

print('')
print('Training Accuracy: {:.4}%'.format(grid_xgb2_training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(grid_xgb2_test_accuracy * 100))
```

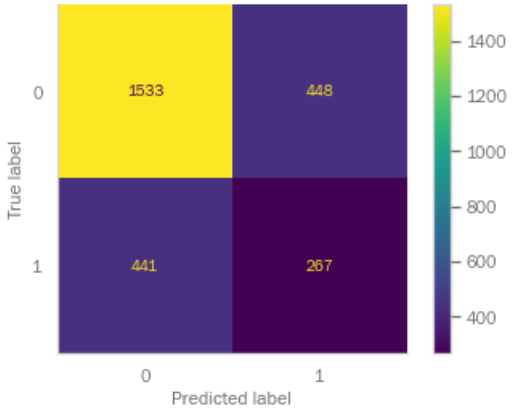
```
592.6325242519379
Grid Search found the following optimal parameters:
learning_rate: 0.4
max_depth: 7
min_child_weight: 1
n_estimators: 150
subsample: 1.0

Training Accuracy: 92.09%
Validation accuracy: 66.94%
```

In [224]:
```
# In order to get feature importances I need an XGB model instead of the gridsearch model
xgb_clf2 = XGBClassifier(learning_rate = 0.4, max_depth = 7, min_child_weight = 1,
                         subsample = 1.0, n_estimators = 150)

metrics_df = run_model(xgb_clf2, 'xgb_clf2', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  1.5787744522094727
Prediction time:  0.0468745231628418
Precision Score: Train 0.79948, Test 0.37343
Recall Score:    Train 0.91153, Test 0.37712
Accuracy Score:  Train 0.92091, Test 0.66939
F1 Score:        Train 0.85183, Test 0.37526
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 638, percentage = 7.9088%
```



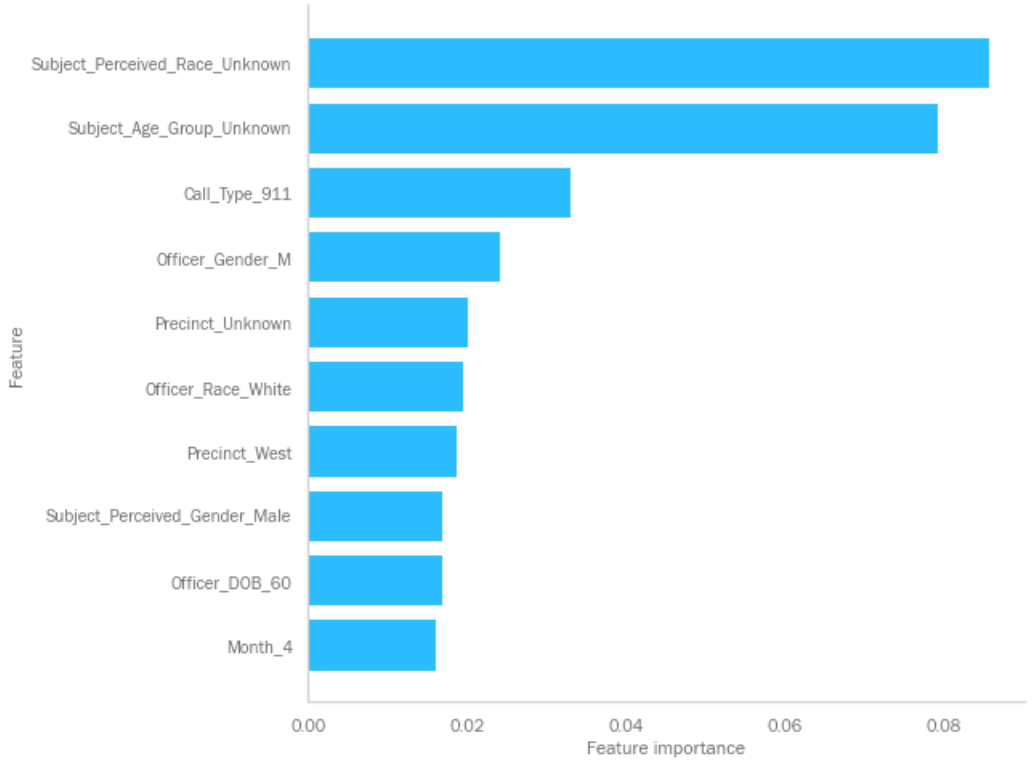

```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 889, percentage = 33.0606%
```




```
----------------
              precision    recall  f1-score   support

           0       0.78      0.77      0.78      1981
           1       0.37      0.38      0.38       708

    accuracy                           0.67      2689
   macro avg       0.58      0.58      0.58      2689
weighted avg       0.67      0.67      0.67      2689
```

In [225]:
```
plot_ten_feature_importances(xgb_clf2, X_train_encoded_cleaned, 'xgb_clf2')
```

```
Images/feature_importancexgb_clf2.png
```



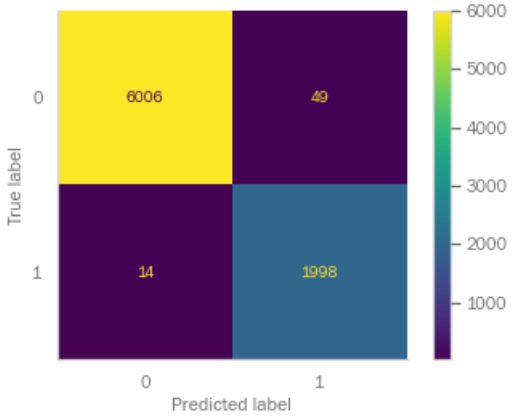
In [226]:
```
xgb_param_grid3 = {
    'learning_rate': [0.4, 0.5],
    'max_depth': [7, 10, 15],
    'min_child_weight': [1],
    'subsample': [1.0],
    'n_estimators': [150, 200],
}
```

In [227]:
```
grid_xgb3 = GridSearchCV(xgb_clf, xgb_param_grid3, scoring='accuracy', cv=None, n_jobs=1)

toc = time.time()
grid_xgb3.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)
fittime = time.time() - toc
print(fittime)

xgb3_best_parameters = grid_xgb3.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(xgb3_best_parameters.keys()):
    print('%s: %r' % (param_name, xgb3_best_parameters[param_name]))
```

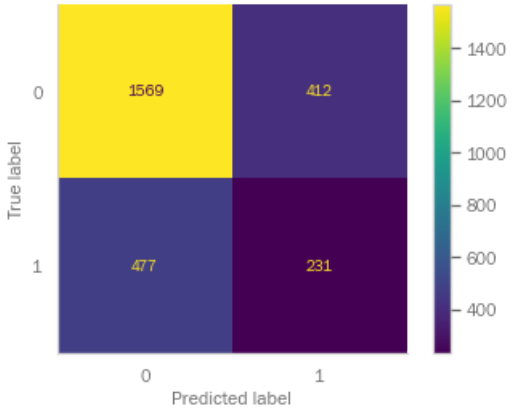
```
140.02306962013245
Grid Search found the following optimal parameters:
learning_rate: 0.5
max_depth: 15
min_child_weight: 1
n_estimators: 150
subsample: 1.0
```

In [228]:
```
# In order to get feature importances I need an XGB model instead of the gridsearch model
xgb_clf3 = XGBClassifier(learning_rate = 0.5, max_depth = 15, min_child_weight = 1,
                         subsample = 1.0, n_estimators = 150)

metrics_df = run_model(xgb_clf3, 'xgb_clf3', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  3.44378399848938
Prediction time:  0.06682109832763672
Precision Score: Train 0.97606, Test 0.35925
Recall Score:    Train 0.99304, Test 0.32627
Accuracy Score:  Train 0.99219, Test 0.66939
F1 Score:        Train 0.98448, Test 0.34197
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 63, percentage = 0.7810%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 889, percentage = 33.0606%
```

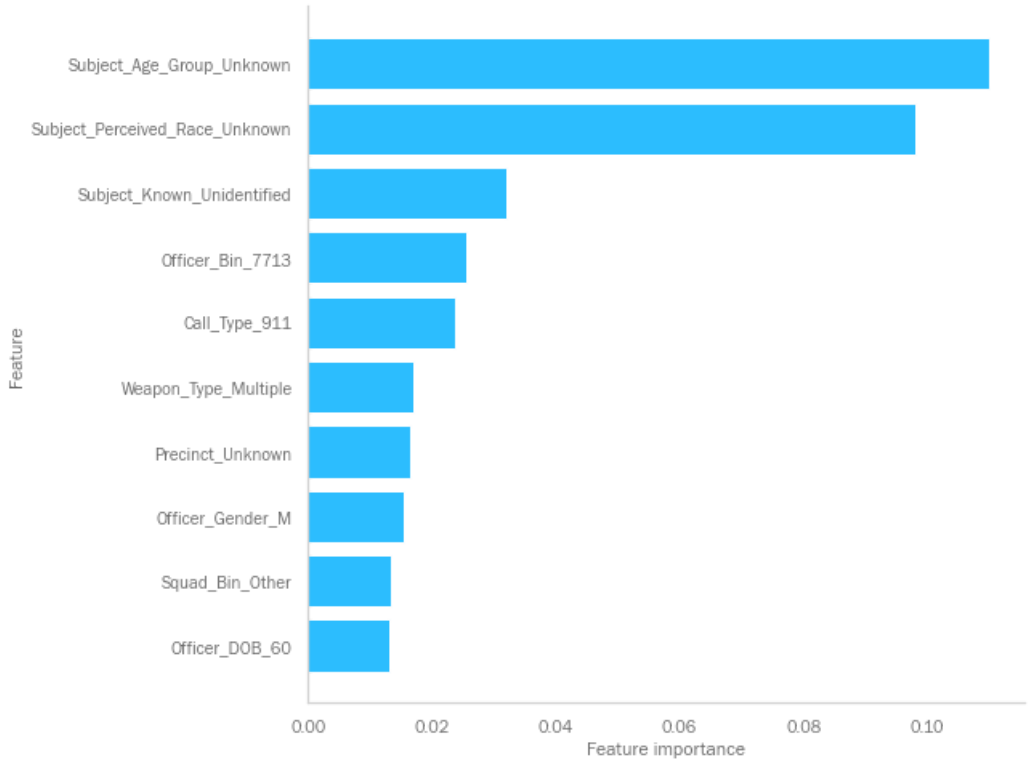



```
----------------
              precision    recall  f1-score   support

           0       0.77      0.79      0.78      1981
           1       0.36      0.33      0.34       708

    accuracy                           0.67      2689
   macro avg       0.56      0.56      0.56      2689
```

```
weighted avg       0.66      0.67      0.66      2689
```

In [229]:
```
plot_ten_feature_importances(xgb_clf3, X_train_encoded_cleaned, 'xgb_clf3')
```

```
Images/feature_importancexgb_clf3.png
```



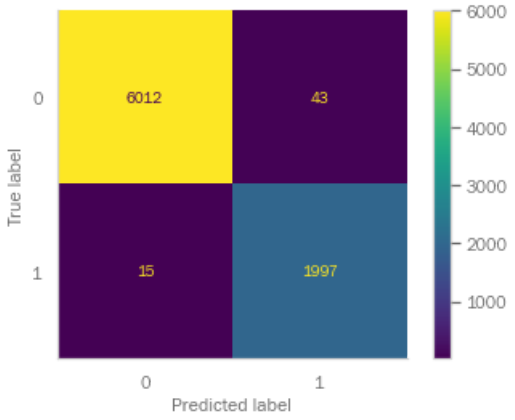
In [230]:
```
xgb_param_grid4 = {
    'learning_rate': [0.5, 0.7, 0.9],
    'max_depth': [15, 20, 25],
    'min_child_weight': [1],
    'subsample': [1.0],
    'n_estimators': [150, 200],
}
```

In [231]:
```
grid_xgb4 = GridSearchCV(xgb_clf, xgb_param_grid4, scoring='accuracy', cv=None, n_jobs=1)

toc = time.time()
grid_xgb4.fit(X_train_encoded_cleaned_SMOTE, y_train_SMOTE)
fittime = time.time() - toc
print(fittime)

xgb4_best_parameters = grid_xgb4.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(xgb4_best_parameters.keys()):
    print('%s: %r' % (param_name, xgb4_best_parameters[param_name]))
```

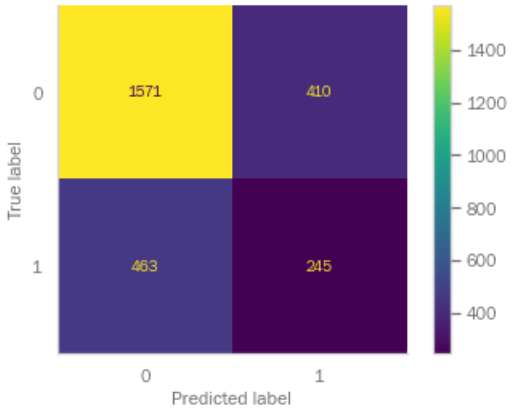
```
339.172155380249
Grid Search found the following optimal parameters:
learning_rate: 0.7
max_depth: 20
min_child_weight: 1
n_estimators: 200
subsample: 1.0
```

In [232]:
```
# In order to get feature importances I need an XGB model instead of the gridsearch model
xgb_clf4 = XGBClassifier(learning_rate = 0.7, max_depth = 20, min_child_weight = 1,
                         subsample = 1.0, n_estimators = 200)

metrics_df = run_model(xgb_clf4, 'xgb_clf4', X_train_encoded_cleaned, X_test_encoded_cleaned,
                    y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned_SMOTE, fit_y=y_train_S
```

```
Fit time:  5.632946968078613
Prediction time:  0.10571718215942383
Precision Score: Train 0.97892, Test 0.37405
Recall Score:    Train 0.99254, Test 0.34605
Accuracy Score:  Train 0.99281, Test 0.67534
F1 Score:        Train 0.98569, Test 0.35950
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 58, percentage = 0.7190%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 873, percentage = 32.4656%
```

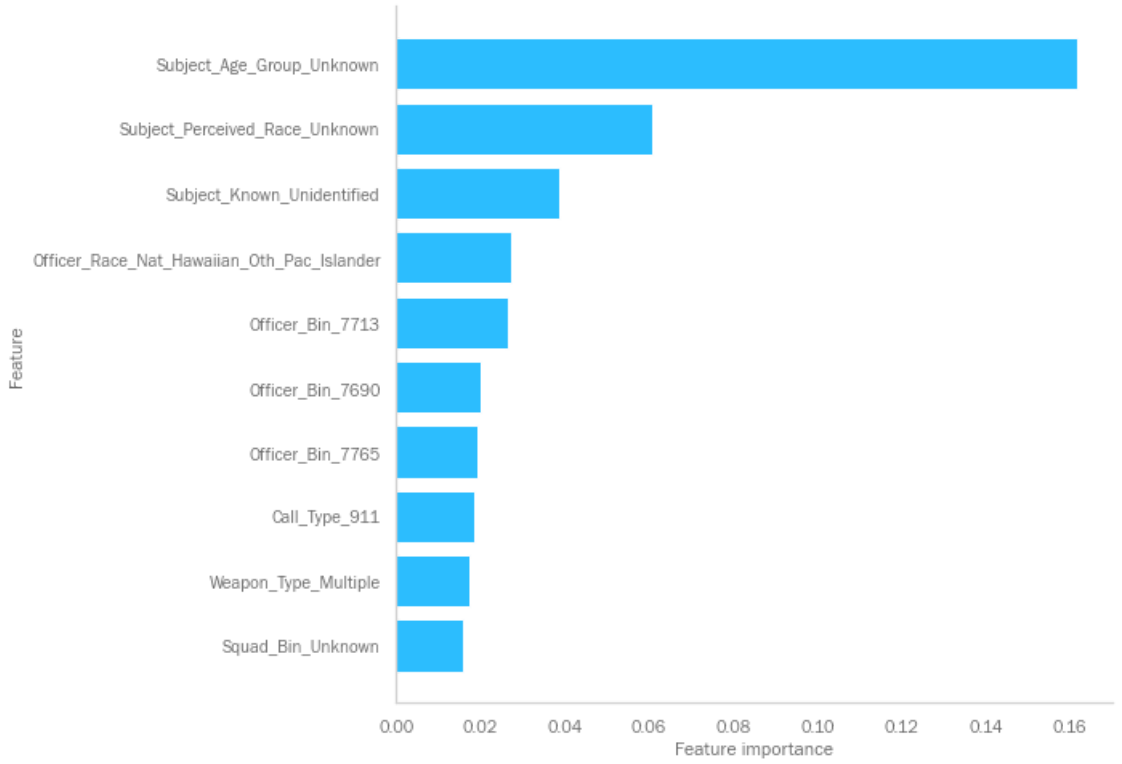



```
----------------
              precision    recall  f1-score   support

           0       0.77      0.79      0.78      1981
           1       0.37      0.35      0.36       708

    accuracy                           0.68      2689
   macro avg       0.57      0.57      0.57      2689
```

```
weighted avg       0.67      0.68      0.67      2689
```

In [233]:
```
plot_ten_feature_importances(xgb_clf4, X_train_encoded_cleaned, 'xgb_clf4')
```

```
Images/feature_importancexgb_clf4.png
```



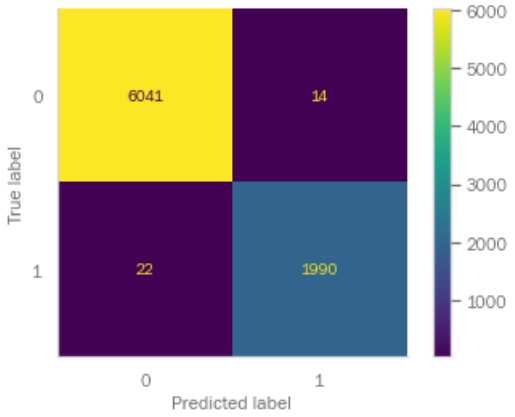
In [234]:
```
metrics_df[['Model','Train_Accuracy', 'Test_Accuracy', 'Train_F1', 'Test_F1']].sort_values('Test_Accu
```

Out[234]:

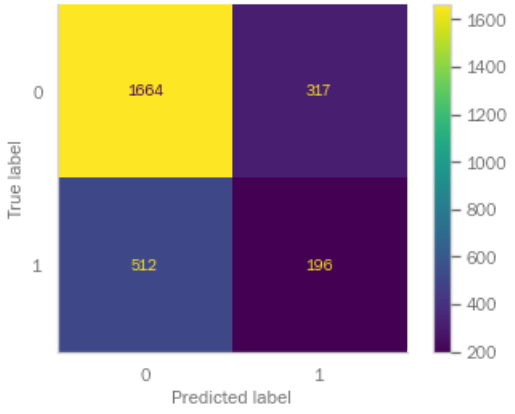
	Model	Train_Accuracy	Test_Accuracy	Train_F1	Test_F1
28	xgb_clf4	0.992810	0.675344	0.985686	0.359501
27	xgb_clf3	0.992190	0.669394	0.984479	0.341969
26	xgb_clf2	0.920912	0.669394	0.851835	0.375264
20	forest2	0.826206	0.667906	0.663789	0.387937
23	xgb_clf	0.809471	0.661212	0.659126	0.408057
25	SVC	0.992438	0.652659	0.984972	0.344101
24	grid_xgb	0.780712	0.650428	0.611465	0.406566
17	bagged_tree	0.653527	0.641874	0.417344	0.406654
22	gbt_clf	0.672617	0.641131	0.467003	0.429332
15	dt7	0.993554	0.638527	0.987110	0.373711
11	dt3	0.993554	0.637040	0.987110	0.375959
19	gs_tree	0.644601	0.634437	0.417395	0.407474
3	dt2	0.993554	0.634065	0.987110	0.376426
8	knn3	0.993554	0.628858	0.987110	0.434240
18	forest	0.648816	0.624768	0.454458	0.421777
21	adaboost_clf	0.645717	0.624396	0.438507	0.418203
5	logreg3	0.648940	0.622536	0.442739	0.416331
1	logreg2	0.648444	0.622536	0.442391	0.416331
7	knn2	0.724557	0.622164	0.582017	0.428571
0	logreg1	0.648320	0.621792	0.442304	0.415181
2	dt1	0.695674	0.620677	0.520601	0.417143
10	bnb1	0.640635	0.617330	0.439373	0.410315
6	knn1	0.795215	0.614727	0.661059	0.390588
13	dt5	0.598364	0.587579	0.423898	0.416623
14	dt6	0.598364	0.587579	0.423898	0.416623
16	dt8	0.526218	0.521383	0.410185	0.412060
4	dummy1	0.497707	0.507996	0.334210	0.351153
12	dt4	0.402256	0.406843	0.428673	0.441331
9	gnb1	0.358869	0.369654	0.409049	0.417726

In [235]:
```
#Curiosity, would my best model perform better if fit on unSMOTEd data?
xgb_clf5 = XGBClassifier(learning_rate = 0.7, max_depth = 20, min_child_weight = 1,
                         subsample = 1.0, n_estimators = 200)

metrics_df = run_model(xgb_clf5, 'xgb_clf5', X_train_encoded_cleaned, X_test_encoded_cleaned,
                   y_train, y_test, metrics_df, fit_X=X_train_encoded_cleaned, fit_y=y_train)
```

```
Fit time:  4.2970709800720215
Prediction time:  0.08876276016235352
Precision Score: Train 0.99301, Test 0.38207
Recall Score:    Train 0.98907, Test 0.27684
Accuracy Score:  Train 0.99554, Test 0.69171
F1 Score:        Train 0.99104, Test 0.32105
----------------

TRAIN Confusion Matrix
----------------
Number of mislabeled training points out of a total 8067 points : 36, percentage = 0.4463%
```




```
TEST Confusion Matrix
----------------
Number of mislabeled test points out of a total 2689 points : 829, percentage = 30.8293%
```




```
----------------
              precision    recall  f1-score   support

           0       0.76      0.84      0.80      1981
           1       0.38      0.28      0.32       708

    accuracy                           0.69      2689
   macro avg       0.57      0.56      0.56      2689
weighted avg       0.66      0.69      0.67      2689
```

In [236]:
```
metrics_df[['Model','Train_Accuracy', 'Test_Accuracy', 'Train_F1', 'Test_F1']].sort_values('Test_Accu
```

Out[236]:

	Model	Train_Accuracy	Test_Accuracy	Train_F1	Test_F1
29	xgb_clf5	0.995537	0.691707	0.991036	0.321048
28	xgb_clf4	0.992810	0.675344	0.985686	0.359501
27	xgb_clf3	0.992190	0.669394	0.984479	0.341969
26	xgb_clf2	0.920912	0.669394	0.851835	0.375264
20	forest2	0.826206	0.667906	0.663789	0.387937
23	xgb_clf	0.809471	0.661212	0.659126	0.408057
25	SVC	0.992438	0.652659	0.984972	0.344101
24	grid_xgb	0.780712	0.650428	0.611465	0.406566
17	bagged_tree	0.653527	0.641874	0.417344	0.406654
22	gbt_clf	0.672617	0.641131	0.467003	0.429332
15	dt7	0.993554	0.638527	0.987110	0.373711
11	dt3	0.993554	0.637040	0.987110	0.375959
19	gs_tree	0.644601	0.634437	0.417395	0.407474
3	dt2	0.993554	0.634065	0.987110	0.376426
8	knn3	0.993554	0.628858	0.987110	0.434240
18	forest	0.648816	0.624768	0.454458	0.421777
21	adaboost_clf	0.645717	0.624396	0.438507	0.418203
5	logreg3	0.648940	0.622536	0.442739	0.416331
1	logreg2	0.648444	0.622536	0.442391	0.416331
7	knn2	0.724557	0.622164	0.582017	0.428571
0	logreg1	0.648320	0.621792	0.442304	0.415181
2	dt1	0.695674	0.620677	0.520601	0.417143
10	bnb1	0.640635	0.617330	0.439373	0.410315
6	knn1	0.795215	0.614727	0.661059	0.390588
14	dt6	0.598364	0.587579	0.423898	0.416623
13	dt5	0.598364	0.587579	0.423898	0.416623
16	dt8	0.526218	0.521383	0.410185	0.412060
4	dummy1	0.497707	0.507996	0.334210	0.351153
12	dt4	0.402256	0.406843	0.428673	0.441331
9	gnb1	0.358869	0.369654	0.409049	0.417726

Conclusions

XGBoost appears to be our best performing model. However, there are so many questions about the validity of the missing data placeholders and the arrest flag prior to 2019, I think the process may need to be begun again.