NYPD Allegations

- **See the main project notebook for instructions to be sure you satisfy the rubric!**
- See Project 03 for information on the dataset.
- A few example prediction questions to pursue are listed below. However, don't limit yourself to them!
  - Predict the outcome of an allegation (might need to feature engineer your output column).
  - Predict the complainant or officer ethnicity.
  - Predict the amount of time between the month received vs month closed (difference of the two columns).
  - Predict the rank of the officer.

Be careful to justify what information you would know at the "time of prediction" and train your model using only those features.

Summary of Findings

Introduction

The problem we are trying to predict with this NYPD dataset is to find out whether a complaintant is going to get a successful outcome from their allegation. A successful outcome is defined as a complaintant being successful in getting a substantiated result from the governing board, which accepts that the officer did have misconduct. We will use a Decision Tree Classifier to classify the data and use the F1 score as the evaluation metric. Additionally, we will use our cleaned dataset from Project 3 to get more consistent data.

Baseline Model

The Baseline Model that we created utilized categorical columns relating only to the alledged officer because the outcome of an allegation might depend on the type of officer involved in a case. Four of these columns consist of nominal data while none are ordinal or quantitative. We One-Hot Encoded these columns into a Column Transformer and then passed them into the default Decision Tree Classifier model. After training this model and testing it using seperate testing data, the calculated F1 score ended up being `0.24049151550614395` which is not a very good score as F1 score is a value between zero and one where 1 is a good score and 0 is a bad score. The reason why this score is bad might be because we did not use any columns relating to the alleged victim or any numerical data.

Final Model

In the Final Model that we created, we feature-engineered two columns; `is_white` and `binned_age`. `is_white` is a binarizer that checks if an officer is white or not and sets values as 1 or 0 respectively. This is useful because in Project 3 we determined that there is a difference between the ethnicity of an officer and the time it takes to get a response for an allegation. Therefore, we assumed that there would also be differences between the ethnicity of an officer and the chance of getting a successful outcome from an allegation. `binned_age` is a transformer which bins the ages into sets of 5 for more accurate average age representation by generalizing the noise in data. This would be a good way of incorporating numerical data to the model. We also included categorical columns about the alleged victim because certain factors such as specific ethnicities or more severe allegations would be more likely to lead to a successful outcome. Additionally, we used a grid search to find the best parameters for the Decision Tree Classifier model. The best parameters ended up being `max_depth=550`, `min_samples_split=10`, and `min_samples_leaf=1 (default)`. We decided to stick with using a Decision Tree Classifier because the dataset we are dealing with uses mostly categorical data, which is better for classification than regression. The F1 score of the Final Model ended up being `0.3938630638882528`. Although this Final Model did not improve the F1 score by a massive amount compared to the Baseline Model, it shows a significant improvement in the effectiveness of the Final Model, which shows how impactful optimizing the features and adding relevant features can be.

Fairness Evaluation

In order to judge the "fairness" of our model, we focused on whether or not the model was fair based on the race of the officer (we narrowed this down to two categories: white and nonwhite. We tried to determine if our model is equally accurate for predicting the outcome of complaints filed against white officers and nonwhite officers. In order to justify this, we used the accuracy parity. If the model is biased towards one of these, that could suggest some sort of racial implications that would make our model "unfair", such as a potential discrepancy in our sample population or other forms of racial bias.

Null Hypothesis: Our biased because it successfully predicts the outcome of a complaint filed against white officers and nonwhite officers at an equal rate.

Alternative Hypothesis: Our model successfully predicts the outcome of a complaint filed against white officers at a higher rate than the outcome of complaints filed against nonwhite officers

Threshold: 0.05

According to the pvalue, which is 0.521, our result is not statistically significant. We cannot Reject the Null Hypothesis. This means that our model is at least fair based on our metric of accuracy parity. As well as that our model successfully predicts the outcome of a complaint filed against officers regardless of whether they are white or not.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pdz
import seaborn as sns
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

In [2]:
```
complaints = pd.read_csv('CCRB-Complaint-Data_202007271729/allegations_202007271729.csv')
pd.set_option('display.max_columns', None)
```

Cleaning and EDA (From Project 3)

Changing "Unknown", and "Refused" to np.nan in 'complainant_ethnicity' column and changing "Not described" to np.nan in 'complainant_gender' column. Were changing "Refused" to np.nan because the questons we are asking do not use the refusal information, so it is equivalent to being NaN for us.

In [3]:
```
complaints['complainant_gender'] = complaints['complainant_gender'].replace({'Not described': np.nan})
complaints['complainant_ethnicity'] = complaints['complainant_ethnicity'].replace({'Unknown':np.nan, 'Refused': np.nan})
```

Combining the year_received and year_closed date columns into two approximate datetime columns. Since we are not provided a day for these complaints, we are assuming that they all start on the first of the month.

In [4]:
```
days = np.ones(complaints.shape[0])
approx_start = complaints[['year_received', 'month_received']].assign(day_received = days)
approx_start = pd.to_datetime(dict(year=approx_start.year_received,
                                   month=approx_start.month_received,
```

```
                                day=approx_start.day_received))

approx_end = complaints[['year_closed', 'month_closed']].assign(day_closed = days)
approx_end = pd.to_datetime(dict(year=approx_end.year_closed,
                                 month=approx_end.month_closed,
                                 day=approx_end.day_closed))

complaints = complaints.drop(columns = ['year_closed', 'month_closed', 'year_received'])
complaints = complaints.assign(approx_start = approx_start, approx_end = approx_end)
```

Creating an approximate duration column which is the time delta between the date received and date closed.

In [5]:
```
approx_duration = complaints['approx_end'] - complaints['approx_start']
complaints = complaints.assign(approx_duration = approx_duration.dt.days)
```

Creating a column where the value is True if the complaintant was successful in getting a substantiated result from the governing board and False otherwise.

In [6]:
```
complaints['complaint_successful'] = complaints['board_disposition'].apply(lambda x: True if x.split(' ')[0] == 'Substantiated' else False)
```

Change precinct column values to string because it is a categorical variable

Creating a column where the value is True if the Police Officer's Ethnicity in the complaint is White and False otherwise.

In [7]:
```
complaints['officer_is_white'] = complaints['mos_ethnicity'].apply(lambda x: 'Yes' if x == 'White' else 'No')
```

In [7]:
```
complaints['precinct'] = complaints['precinct'].astype(str)
```

Dropping all unnessesary columns such as names, and id's that are not relevant to our questions

In [8]:
```
complaints = complaints.drop(columns = ['first_name', 'last_name',
                         'rank_abbrev_now', 'rank_now', 'board_disposition',
                         'rank_abbrev_incident', 'command_now',
                         'unique_mos_id', 'complaint_id',
                         'approx_start', 'approx_end'])
complaints.head(5)
```

Out[8]:

	shield_no	month_received	command_at_incident	rank_incident	mos_ethnicity	mos_gender	mos_age_incident	complainant_ethnicity	complainant_gender	complainant_age_incident	fado
0	8409	7	078 PCT	Police Officer	Hispanic	M	32	Black	Female	38.0	Ab Aut
1	5952	11	PBBS	Police Officer	White	M	24	Black	Male	26.0	Disco
2	5952	11	PBBS	Police Officer	White	M	24	Black	Male	26.0	Offe Lan
3	5952	7	PBBS	Police Officer	White	M	25	Black	Male	45.0	Ab Aut
4	24058	8	078 PCT	Police Officer	Hispanic	F	39	NaN	NaN	16.0	

Baseline Model

In [11]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics


#Get Data and Target Value
X = complaints.drop('complaint_successful', axis=1)
y = complaints['complaint_successful']

#generate train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=420)

#OneHotEncode Pipeline
baseline_cats = ["mos_ethnicity", "rank_incident", "shield_no", "fado_type"]
onehotpl = Pipeline(steps=[
    ("One-hot", OneHotEncoder(handle_unknown='ignore'))
])

#Column Transformer to add OneHotEncoder to specified categorical columns
ct = ColumnTransformer([
    ("onehot", onehotpl, baseline_cats)
],remainder='drop')

#General pipeline to utilize ColumnTransformer and pass it to a DecisionTreeClassifier
baselinepl = Pipeline(steps=[
    ("transform", ct),("dtc", DecisionTreeClassifier())
])

#Fit and Predict on General Pipeline
baselinepl.fit(X_train,y_train)
preds = baselinepl.predict(X_test)

#F1 Score
metrics.f1_score(y_test,preds)
```

Out[11]:
```
0.24049151550614395
```

Final Model

In [12]:
```
#feature engineering an is_white column that checks if an officer is white or not
complaints['is_white'] = complaints['mos_ethnicity'].apply(lambda x: 1 if x == 'White' else 0)
```

In [13]:
```
#feature engineering a binned_age column which bins the ages into sets of 5 for more accurate modeling
complaints['binned_age'] = complaints['mos_age_incident'].apply(lambda x: 5 * (x//5))
```

In [14]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import LabelBinarizer

#Get Data and Target Value
X = complaints.drop('complaint_successful', axis=1)
y = complaints['complaint_successful']

#generate train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=420)

#range of values to perform grid search on
grid_search_params = {
    'max_depth': np.arange(500, 600, 10),
    'min_samples_split': np.arange(10, 20, 2),
    'min_samples_leaf': [1, 2, 6, 10]
}

#grid search on DecisionTreeClassifier with 5 cross folds and F1 score as metric
grid_search_clf = GridSearchCV(DecisionTreeClassifier(), grid_search_params, cv = 5, scoring = 'f1')

final_cats = ["rank_incident", "shield_no",
                "allegation","contact_reason",
                "mos_gender",'command_at_incident',
                'complainant_ethnicity',
                'precinct']

pass_col = ['is_white', 'binned_age']


#OneHotEncode Pipeline
onehotpl = Pipeline(steps=[
    ("One-hot", OneHotEncoder(handle_unknown='ignore'))
])

#Column Transformer to apply OneHotEncoder to categorical columns and have feature engineered columns as a passthrough
ct = ColumnTransformer([
    ("onehot", onehotpl, final_cats),
    ('pass', FunctionTransformer(lambda x: x), pass_col)
], remainder='drop')


#General pipeline to utilize ColumnTransformer and pass it to a DecisionTreeClassifier optimized with Grid Search
final_pl = Pipeline(steps=[
    ("transform", ct),
    ("classify", grid_search_clf)
])

#Fit and Predict on General Pipeline
final_pl.fit(X_train,y_train)
preds = final_pl.predict(X_test)

#F1 Score
metrics.f1_score(y_test,preds)
```

Out[14]:
```
0.3938630638882528
```

In [15]:
```
#best estimators for DecisionTreeClassifier
grid_search_clf.best_estimator_
```

Out[15]:
```
DecisionTreeClassifier(max_depth=550, min_samples_split=10)
```

Fairness Evaluation

In [30]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.options.mode.chained_assignment = None


#### Below is a permutation test for accuracy pairty
#Create test,training set and observed value
X = complaints.drop('complaint_successful', axis=1)
y = complaints.complaint_successful
X_tr, X_ts, y_tr, y_ts = train_test_split(X,y, test_size=0.3)
baselinepl.fit(X_tr,y_tr)
preds = final_pl.predict(X_ts)
results = X_ts
results['prediction'] = preds
results['outcome'] = y_ts
results.groupby('is_white').prediction.mean().to_frame()

#observed accuracy for permutation test
obs = (
    results
    .groupby('is_white')
    .apply(lambda x: metrics.accuracy_score(x.outcome, x.prediction))
    .rename('accuracy')
    .to_frame()
    .diff()
    .iloc[-1][0]
)
metrs = []
n = 1000
for _ in range(n):
    #Generate test Statistic (Absolute difference in  means between accuracy when officer is white vs not white)
    s = (
```

```
        results[['is_white', 'prediction', 'outcome']].assign(is_white=results
                                                              .is_white.sample(frac=1.0, replace=False)
                                                              .reset_index(drop=True))
        .groupby('is_white')
        .apply(lambda x: metrics.accuracy_score(x.outcome, x.prediction))
        .diff()
        .iloc[-1]
    )

    metrs.append(s)


#Calculate the p_value
p_value = (metrs <= obs).mean()
p_value
```

Out[30]: 0.521

In [ ]: