Study of Fairness of Board Decisions in NYPD Complaint Data

This project was originally completed for DSC 80 at University of Californoa, San Diego, during Spring 2021 quarter with Professor Justin Eldridge.

Summary of Findings

Introduction

In Project 3, the focus of my work was to determine whether the NYPD complaint data contains any degree of racial bias, in particular researching whether the board would favor the White officers/complaintants in cases against the Black complaitants/officers. My findings suggested that there was a strong indication of racial bias in the dataset, but I lacked some extra information that would help me confirm such a hypothesis with a high degree of certainty.

In this project, my task will be to create a prediction model on the same dataset of complaints to the New York Police Department. This time, I am curious to determine whether there is a more general degree of bias that is evident from the dataset by **predicting the board disposition based on the set of features such as ethnicity, age and gender of both the officer and the complainant, as well the allegation against the officer and the reason the complainant was detained in the first place**. This is a classification model, since it will be predicting one of the two possible outcomes (board favored the complainant or not) based on a set of features. The choice of variable reflects the factors that would introduce the degree of bias into the board's decision (age, gender and ethnicity) and also considers the non-biased factors that led to the decision in order to not create a model that is biased in itself and fails to reflect the more important factors that play into the board's disposition.

Baseline Model

For my baseline model, I picked a basic set of features that related to the personal information of the complainants and the officers. I had four ordinal features (complainant and officer ages, as well as `fado_type` and `outcome_description`; the first two were passed in as is, while the latter two had to be engineered using two `FunctionTransformers`) and four categorical features (relating to gender and ethnicity, these were encoded using `OneHotEncoder`). My model used `DecisionTreeClassifier` in order to predict `board_disposition` (which was binarized using `FunctionTransformer`, 1 for "Substantiated", 0 for "Not Substantiated"); `DecisionTreeClassifier` is a reasonable classifier option due to the problem being a classification problem with a set of features that could determine the board outcome. The evaluation metric is the accuracy score, which is reasonable as I am trying to determine how accurate my predictions are compared to the actual outcomes, as I want to see if my model is good at predicting the board disposition given the provided features.

The baseline model produced great results on the NYPD dataset, with the accuracy scores being equal to ~0.92 and ~0.72 for training and testing datasets respectively. It seems that the model overfits to the training dataset quite a lot, while also being able to generalize quite well. Still, there is room for improvement, which is the goal of my final model.

Final Model

In the final model, I engineered several new features: I binned the ages in order to consolidate age groups and thus decrease fragmentation. The idea behind this is that some age groups might be more favored by the board, therefore I decided to attempt to improve the model by binning the age groups. Another new feature was the binarizing of the `complainant_gender`. In the baseline model, complainant gender is One Hot Encoded because it featured categories outside of Male and Female (it included data on the transgender and non-binary people). I decided to add the binarizing with 1 for Male and 0 for Not Male because of the assumption that the board might be more inclined to substantiate the claims done by non-males rather than males, so I thought that binarization would make the model more accurate. In addition to engineering these two new features, I added the `outcome_description` column to this model, with it being made ordinal with a `FunctionTransformer`.

I stuck with a `DecisionTreeClassifier`, and I ran a Grid Search in order to determine the best parameters for the model. The best parameters were calculated as follows:

{'max_depth': 5, 'min_samples_leaf': 10, 'min_samples_split': 5}

However, after calculating the accuracy score for this model, it turned out that it performed even worse than my baseline model, so I decided to work with the default parameters in order to produce the accuracy scores of ~0.82 and ~0.73 for training and test datasets respectively. The depth of the tree achieved was 51, and the number of leaves was 4260.

I was able to improve the model slightly. While the accuracy on the training set dropped significantly, the accuracy on the test set improved by ~0.01. I think this model is a bit better at generalizing to the never-before-seen data, but of course there are still ways to improve the model. I think that me binarizing the gender column and also binning the ages contributed to the improvement of the model.

Fairness Evaluation

As an aftermath of Project 3, I am concerned with the racial bias in the dataset. Therefore, I was set out to see if my model is based on the basis of race of the complainants. I binarized the ethnicity data for the complainants (1 for White, 0 for Non White) in order to make the groupings. My metric was using the recall value, as I measured the fairness based on how the proportion of correctly predicted substantiated outcomes for each ethnic group. In other words, I used true positive parity, which is justified by the notion that in a fair model, the board would be as likely to substantiated claims for White and Non-White complainants, so the "Equality of Opportunity" parity is reasonable to check whether the model reflects this notion.

For the permutation test, I picked 0.01 level of significance due to the relatively small size of the dataset, 500 repeptitions, and the test statistic is the absolute difference in recall values between the groups (of which there are only two); Null Hypothesis stated that there is no difference for White and Non-White complainants, and the Alternative Hypothesis suggested that there is a difference and the model is biased. The observed difference was ~0.021, and the

p-value produced by the permutation test was 0.706, meaning that I fail to reject my Null Hypothesis. I cannot confirm that there is no difference in recall values for the two groups and that there is no bias involved, but considering the mathematical computations of sklearn, I have reasons to believe that my model might be fair, despite my findings in Project 3 that suggested a degree of racial bias.

Conclusion

This was an interesting project to explore, and a great follow up to the analysis in Project 3. I definitely could improve my final model even further, but I was greatly limited with the amount of features I could engineer from the provided dataset. Still, I believe I was able to fine tune my model to an acceptable degree, and going through a process of preparing, training and evaluating a model and its fairness is a great experience that will be helpful in my future career in Data Science.

Code

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import sklearn
from sklearn import preprocessing
from sklearn import pipeline
from sklearn import compose
import re
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

Baseline Model

First, I need to load in the NYPD complaints dataset, which I have in the same directory. I will also make a copy of the DataFrame with the original dataset to have it as a reference, in case `nypd` DataFrames gets overwritten (which does happen later in the code).

```
nypd = pd.read_csv("nypd_data.csv")
original_nypd = nypd.copy()
```

Here is what the DataFrame looks like -- `original_nypd` looks exactly the same.

```
nypd.head()
```

```
   unique_mos_id first_name last_name command_now  shield_no
complaint_id  \
0          10004   Jonathan      Ruiz     078 PCT       8409
```

```
42835
1          10007       John     Sears     078 PCT       5952
24601
2          10007       John     Sears     078 PCT       5952
24601
3          10007       John     Sears     078 PCT       5952
26146
4          10009      Noemi    Sierra     078 PCT      24058
40253

   month_received  year_received  month_closed  year_closed  ...  \
0               7           2019             5         2020  ...
1              11           2011             8         2012  ...
2              11           2011             8         2012  ...
3               7           2012             9         2013  ...
4               8           2018             2         2019  ...

  mos_age_incident complainant_ethnicity complainant_gender  \
0               32                 Black             Female
1               24                 Black               Male
2               24                 Black               Male
3               25                 Black               Male
4               39                   NaN                NaN

  complainant_age_incident           fado_type
allegation  \
0                     38.0  Abuse of Authority  Failure to provide
RTKA card
1                     26.0         Discourtesy
Action
2                     26.0  Offensive Language
Race
3                     45.0  Abuse of Authority
Question
4                     16.0               Force                Physical
force

  precinct                                 contact_reason  \
0     78.0                      Report-domestic dispute
1     67.0                             Moving violation
2     67.0                             Moving violation
3     67.0  PD suspected C/V of violation/crime - street
4     67.0                               Report-dispute

                outcome_description
board_disposition
0  No arrest made or summons issued  Substantiated (Command Lvl
Instructions)
1   Moving violation summons issued                   Substantiated
(Charges)
```

```
2   Moving violation summons issued                    Substantiated
(Charges)
3  No arrest made or summons issued                    Substantiated
(Charges)
4    Arrest - other violation/crime      Substantiated (Command
Discipline A)

[5 rows x 27 columns]
```

Here is the list of all the columns that are in the DataFrame. For my baseline model, I want to look at the basic info that might affect the board decisions: gender of both the officer and the complainant, their ethnicity, age, the type of complaint, the result of the interaction, etc. I listed these columns in the next cell, storing them in the `req_columns` list.

```
nypd.columns
```

```
Index(['unique_mos_id', 'first_name', 'last_name', 'command_now',
'shield_no',
       'complaint_id', 'month_received', 'year_received',
'month_closed',
       'year_closed', 'command_at_incident', 'rank_abbrev_incident',
       'rank_abbrev_now', 'rank_now', 'rank_incident',
'mos_ethnicity',
       'mos_gender', 'mos_age_incident', 'complainant_ethnicity',
       'complainant_gender', 'complainant_age_incident', 'fado_type',
       'allegation', 'precinct', 'contact_reason',
'outcome_description',
       'board_disposition'],
      dtype='object')
```

```
req_columns = [
        'mos_ethnicity', 'mos_gender', 'mos_age_incident',
        'complainant_ethnicity', 'complainant_gender',
        'complainant_age_incident', 'fado_type',
        'outcome_description', 'board_disposition'
    ]
```

Slicing the DataFrame to only get the required columns.

```
nypd = nypd[req_columns]
nypd.head()
```

```
  mos_ethnicity mos_gender  mos_age_incident complainant_ethnicity  \
0      Hispanic          M                32                 Black
1         White          M                24                 Black
2         White          M                24                 Black
3         White          M                25                 Black
4      Hispanic          F                39                   NaN
```

```
  complainant_gender  complainant_age_incident           fado_type  \
0             Female                      38.0  Abuse of Authority
1               Male                      26.0         Discourtesy
2               Male                      26.0  Offensive Language
3               Male                      45.0  Abuse of Authority
4                NaN                      16.0               Force

                outcome_description
board_disposition
0  No arrest made or summons issued  Substantiated (Command Lvl
Instructions)
1   Moving violation summons issued                  Substantiated
(Charges)
2   Moving violation summons issued                  Substantiated
(Charges)
3  No arrest made or summons issued                  Substantiated
(Charges)
4    Arrest - other violation/crime      Substantiated (Command
Discipline A)
```

Out of the columns that I have gotten, I will be making features for my baseline model. There will be eight features engineered, one per column, out of which:

- Two are ordinal and correspond to ages. They are not modified in any way (for now).
- One more is also ordinal, and corresponds to `fado_type`. It will be transformed in order to convert the string to categories to numeric values based on the type: discourtesy (lowest), offensive language, abuse of authority and use of force (the largest).
- Another ordinal feature based on `outcome_description`: 0 for no outcome, 1 for summons and juvenile reports, 2 for arrests.
- Categorical features related to gender and ethnicity, which will be transformed using One Hot Encoding (gender is binary for officers, but has more than two categories for complainants).

First, I needed to set up a function that would create the ordinal encoding for `fado_type`. The next several cells fetch the info about the column by exploring the unique entries and defines a function that would process a column containing these values.

```
nypd['fado_type'].unique()
```

```
array(['Abuse of Authority', 'Discourtesy', 'Offensive Language',
'Force'],
      dtype=object)
```

```
fado_types = [
    'Discourtesy',
    'Offensive Language',
    'Abuse of Authority',
    'Force'
]
```

```
def transform_board(disp):
    if isinstance(disp, pd.DataFrame):
        return
pd.DataFrame(pd.Series(disp.values[:,0]).apply(transform_board))
    return int("Substantiated" in disp)
```

Similar process is done for `outcome_description`, which will also require grouping into types of outcome: arrest, summons/report, or neither. The function is defined here and will perform the needed type of grouping and encoding, and then will be passed to the function transformer later in the notebook.

```
nypd['outcome_description'].unique()
```

```
array(['No arrest made or summons issued',
       'Moving violation summons issued',
       'Arrest - other violation/crime',
       'Summons - other violation/crime', 'Arrest - OGA',
       'Other VTL violation summons issued', 'Arrest - resisting
arrest',
       'Arrest - disorderly conduct', 'Arrest - assault (against a
PO)',
       'Summons - disorderly conduct', 'Juvenile Report',
       'Parking summons issued', 'Disorderly-Conduct/Arr/Summons',
       'Assault/Arrested', 'Other Summons Claimed or Issued', nan,
       'Arrest - harrassment (against a PO)', 'Arrest on Other
Charge',
       'Obstruct-Govt-Admin/Arrested',
       'Traffic Summons Claimed or Issued', 'Resisting
Arrest/Arrested',
       'Harrassment/Arrested/Summons', 'Summons - OGA',
       'Summons - harrassment (against a PO)'], dtype=object)
```

```
def transform_outcome(out):
    if isinstance(out, pd.DataFrame):
        return
pd.DataFrame(pd.Series(out.values[:,0]).apply(transform_outcome))
    if ('No arrest') in out:
        return 0
    elif ('Arrest') in out:
        return 2
    elif (('Summons') in out) or ('Juvenile' in out) or (('summons')
in out and not (('arrest') in out)):
        return 1
```

This cell looks at the values for Board Disposition: these would require grouping into "Favor the complainant" and "Favor the officer" values.

```
pd.Series(nypd[['board_disposition']].values[:,0])
```

```
0        Substantiated (Command Lvl Instructions)
1                          Substantiated (Charges)
2                          Substantiated (Charges)
3                          Substantiated (Charges)
4              Substantiated (Command Discipline A)
                           ...
33353                             Unsubstantiated
33354                             Unsubstantiated
33355          Substantiated (Formalized Training)
33356          Substantiated (Formalized Training)
33357          Substantiated (Formalized Training)
Length: 33358, dtype: object
```

This cell stores the names of the columns that would undergo OneHotEncoding: these are gender and ethnicity values.

```
ohe_columns = ['mos_gender', 'complainant_gender', 'mos_ethnicity',
'complainant_ethnicity']
```

Using the functions defined, I created a `transformer`, which is an sklearn ColumnTransformer. It will be performing the preprocessing for the pipeline of my baseline model.

```
ohe = sklearn.preprocessing.OneHotEncoder(handle_unknown = 'ignore')

ordinal_fado = sklearn.preprocessing.OrdinalEncoder(categories =
[fado_types])

# the board transformer is defined here, but will come in later
board = sklearn.preprocessing.FunctionTransformer(transform_board)

outcome = sklearn.preprocessing.FunctionTransformer(transform_outcome)

transformer = sklearn.compose.ColumnTransformer(
    transformers = [
        ('ohe', ohe, ohe_columns),
        ('fado', ordinal_fado, ['fado_type']),
        ('outcome', outcome, ['outcome_description']),
    ], remainder = "passthrough"
)
```

This cell creates a copy of the dataset that doesn't contain any `NaN` values (as they would break the model). The choice of dropping the missing values is reasonable, since of most of the data missing from the dataset is complainant's gender, age or ethnicity. These are some of the key features in my model, and filling them with arbitrary values could make the model biased. A reasonable approach would be to perform data imputation, however for the sake of simplicity for this baseline model, I decided to perform classification without the missing values altogether.

```
# this cell demonstrates which columns have the most missing values --

# indeed, it's the personal information of the complainants
nypd.isna().sum()
```

```
mos_ethnicity                  0
mos_gender                     0
mos_age_incident               0
complainant_ethnicity       4464
complainant_gender          4195
complainant_age_incident    4812
fado_type                      0
outcome_description           56
board_disposition              0
dtype: int64
```

```
clean_nypd = nypd.dropna()[req_columns]
```

Fitting the transformer and transforming the `clean_nypd` dataset in order to ensure that the preprocesor works as intended. It looks like the result is what I need.

```
transformer.fit(clean_nypd)
```

```
ColumnTransformer(remainder='passthrough',
                  transformers=[('ohe',
OneHotEncoder(handle_unknown='ignore'),
                                 ['mos_gender', 'complainant_gender',
                                  'mos_ethnicity',
'complainant_ethnicity']),
                                ('fado',

OrdinalEncoder(categories=[['Discourtesy',

'Offensive '

'Language',
                                                                    'Abuse of
'

'Authority',

'Force']]),
                                 ['fado_type']),
                                ('outcome',
                                 FunctionTransformer(func=<function
transform_outcome at 0x000001A468A238B0>),
                                 ['outcome_description'])])
```

```
(transformer.transform(clean_nypd))
```

```
array([[0.0, 1.0, 1.0, ..., 32, 38.0,
        'Substantiated (Command Lvl Instructions)'],
       [0.0, 1.0, 0.0, ..., 24, 26.0, 'Substantiated (Charges)'],
       [0.0, 1.0, 0.0, ..., 24, 26.0, 'Substantiated (Charges)'],
       ...,
       [0.0, 1.0, 0.0, ..., 36, 21.0,
        'Substantiated (Formalized Training)'],
       [0.0, 1.0, 0.0, ..., 36, 21.0,
        'Substantiated (Formalized Training)'],
       [0.0, 1.0, 0.0, ..., 36, 21.0,
        'Substantiated (Formalized Training)']], dtype=object)
```

Time to create the Pipeline. For my model, I decided to use the DecisionTreeClassifier, since my model is a classification model with two options (board favors the complaint or not).

```
from sklearn.tree import DecisionTreeClassifier

pl = sklearn.pipeline.Pipeline(
    steps = [
        ('preprocessor', transformer),
        ('classifier', DecisionTreeClassifier())
    ]
)
```

This cell splits the dataset into the training subset and the test subset. Aditionally, it transforms the `board_disposition` column, which will be the target (`y`) that the model will be trying to predict. This is where the `board` FunctionTransformer is used that was defined earlier.

```
from sklearn.model_selection import train_test_split

X = clean_nypd.drop('board_disposition', axis = 1)
# y = transform_board(clean_nypd[['board_disposition']])
y = board.fit_transform(clean_nypd[['board_disposition']])

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Fitting the data, and then predicting the values. The cell will be outputing the accuracy score that evaluates how close my predictions are. On the training set, the model predicts expected 0.91 (pretty close), while on the training set the accuracy is somewhere around 0.72: this is pretty good, but could be better.

I stored the accuracy scores I got from this baseline model in order to compare them to the improved model that I created later.

```
pl.fit(X_train, y_train)
original_model_train = pl.score(X_train, y_train)
original_model_test = pl.score(X_test, y_test)
original_model_train, original_model_test
```

```
# the left value is on the training set, the right one is on the
testing set.

(0.9195679977263038, 0.7212276214833759)
```

Final Model

The model could be improved. To see how I could make it better, I decided to pull up the list of all the columns in hopes of finding some that might make my model even better.

```
original_nypd.columns

Index(['unique_mos_id', 'first_name', 'last_name', 'command_now',
'shield_no',
       'complaint_id', 'month_received', 'year_received',
'month_closed',
       'year_closed', 'command_at_incident', 'rank_abbrev_incident',
       'rank_abbrev_now', 'rank_now', 'rank_incident',
'mos_ethnicity',
       'mos_gender', 'mos_age_incident', 'complainant_ethnicity',
       'complainant_gender', 'complainant_age_incident', 'fado_type',
       'allegation', 'precinct', 'contact_reason',
'outcome_description',
       'board_disposition'],
      dtype='object')
```

I have some ideas on how I can improve my model:

- The age is a bit too fragmented in its current form -- it would make more sense to try to predict the board disposition using age bins, rather than try to calculate it year by year. For example, teens might be more likely to be favored by the board compared to adults, same with senior citizens. Therefore, binning makes total sense and might yield more accurate results.
- The gender column is way too fragmented for the complainants, with values specifying whether the complainant is transgender or non-binary. For officers, the data simply states whether they are male or female. To make the model more precise and to make it more unified, I decided to binarize the gender feature: now it is simply checking whether the person, both officer and the complainant, are male or not (cis and transgender men are grouped together). The issue with this grouping is that non-binary people are grouped with women, which has the potential of introducing bias. However, the board might base the their decision on gender, and men historically have been more likely to get harsher treatment by law enforcement, so I believe it is reasonable to train the model to check whether the person is male or not. For now, in efforts to improve the model, I will create this binarization and will evaluate the resulting fairness later.
- I added the `contact_reason` column, which specifies the reason for the officer/complainant contact (i.e. the crime that the complainant allegedly commited). This, in my opinion, will be main the feature that will lead to major improvement of my model, since it will contain information about why the complainant interacted with the

officer in the first place, and harsher crimes might lead to the board not favoring the complaint.

I expect that these new features (binned age and binarized gender) will improve my model by decreasing fragmentation and avoiding overfitting.

This cell creates several transformers that will engineer the new features that I outlined above.

```
age_bins = sklearn.preprocessing.KBinsDiscretizer(strategy =
'uniform', encode = 'ordinal')
ohe_columns.append('contact_reason')

def is_male(x):
    return pd.DataFrame(
        pd.Series(x.values[:,0])
        .str.contains('(?:M|Male|FTM)$', regex=True)
        .astype('int')
    )

is_male = sklearn.preprocessing.FunctionTransformer(is_male)
```

Preprocessor for the final model, using some of the features from the baseline model and including the newly engineered features.

```
final_transformer = sklearn.compose.ColumnTransformer(
    transformers = [
        ('ohe', ohe, ohe_columns),
        ('fado', ordinal_fado, ['fado_type']),
        ('outcome', outcome, ['outcome_description']),
        ('male', is_male, ['mos_gender', 'complainant_gender']),
        ('age_bins', age_bins, ['mos_age_incident',
'complainant_age_incident'])
    ], remainder = "passthrough"
)
```

In order to make my model more precise and to generalize it better, I also decided to perform the GridSearchCV in order to determine the best parameters for the DecisionTreeClassifier. It uses five cross validation folds and tests four different options for `min_samples_leaf` and `min_samples_split`, and six for `max_depth`. After GridSearchCV runs, I will use these parameters in my final pipeline.

```
if 'contact_reason' not in req_columns:
    req_columns.append('contact_reason')

from sklearn.model_selection import GridSearchCV

parameters = {
    'max_depth': [5, 8, 10, 12, 15, None],
    'min_samples_split':[5, 10, 15, 20],
    'min_samples_leaf':[5, 10, 15, 20]
```

```
}

X_grid =
pd.DataFrame(final_transformer.fit_transform(clean_nypd.drop('board_di
sposition', axis = 1)).toarray())

X_train, X_test, y_train, y_test = train_test_split(X_grid,
board.fit_transform(clean_nypd[['board_disposition']]))

clf = GridSearchCV(DecisionTreeClassifier(), parameters, cv = 5)
clf.fit(X_train, y_train)
clf.best_params_
```

```
{'max_depth': 5, 'min_samples_leaf': 10, 'min_samples_split': 5}
```

This cell defines the pipeline for the final model.

```
final_pl = sklearn.pipeline.Pipeline(
    steps = [
        ('preprocessing', final_transformer),
        ('classifier', DecisionTreeClassifier(
            max_depth = 5,
            min_samples_leaf= 10,
            min_samples_split= 5
        ))
    ]
)
```

Time to fit and run the model, storing the accuracy scores for both the training and testing datasets.

```
# clean_nypd = clean_nypd.drop('contact_reason', axis = 1)

clean_nypd = original_nypd[req_columns].dropna()
X = clean_nypd.drop('board_disposition', axis = 1)
y = transform_board(clean_nypd[['board_disposition']])

X_train, X_test, y_train, y_test = train_test_split(X, y)

final_pl.fit(X_train, y_train)
final_pl.score(X_train, y_train), final_pl.score(X_test, y_test)
```

```
(0.7546776561981905, 0.751207729468599)
```

It seems like that this model turned out to be even worse than my original model. After some testing, I concluded this is due to the parameters that I used in my DecisionTreeClassifier (the ones determined by the grid search). Therefore, for my final model, I will be sticking with the 'as-is' parameters, as they seem to work just fine on this dataset. I will be keeping the newly engineered features, however, as I still believe that binned ages and binarized gender should improve my model.

```
final_pl = sklearn.pipeline.Pipeline(
    steps = [
        ('preprocessing', final_transformer),
        ('classifier', DecisionTreeClassifier())
    ]
)

clean_nypd = original_nypd[req_columns].dropna()
X = clean_nypd.drop('board_disposition', axis = 1)
y = transform_board(clean_nypd[['board_disposition']])

X_train, X_test, y_train, y_test = train_test_split(X, y)

final_pl.fit(X_train, y_train)
final_pl.score(X_train, y_train), final_pl.score(X_test, y_test)
```

```
(0.8236464402444224, 0.7328786587098608)
```

```
# baseline model accuracy scores, for reference
original_model_train, original_model_test
```

```
(0.9195679977263038, 0.7212276214833759)
```

This is better. The model is still behaving less accurately on the training dataset, but it seems to be doing slightly better on the test dataset, suggesting a slight improvement in terms of generalization. This is still far from a perfect model, but at least I am able to see some improvement: the final model produces the accuracy score that is higher by 0.01, which is noticeable. For now, I am satistifed with how I was able to improve my model by engineering new featuers (binned ages and binarized genders) and determining which DecisionTreeClassifier parameters work best (as it turns out, the default ones work the best). Now, I will move on to assess the fairness of my model.

Here are the parameters that my model used for this evaluation:

- Depth = 51
- Number of leaves = 4260

```
final_pl.named_steps['classifier'].get_depth(),
final_pl.named_steps['classifier'].get_n_leaves()
```

```
(51, 4260)
```

Fairness Evaluation

My original analysis in Project 3 focused on the racial bias in the NYPD data. Now, I am curious to find out whether the model that I build carries out unfair predictions based on the race. I will use a binary system in order to determine that: I will group the ethnicities of the complainants into two groups, White and Non-White. The reasoning behind that is because, as I found out in Project 3, White complainants demonstrated to be favored by the board when submitting claims against Black officers. Now, the idea is to see whether the model is more likely to predict the

substantiated board disposition for White complaintants compared to complaints submitted by ethnic minorities.

To do my evaluation, I would first need to fetch the predictions from my model.

```
preds = final_pl.predict(X)
```

I will be using `sklearn.metrics` in order to evaluate the **recall** of my data. The reason I am using recall is because I want to see the proportion of true positives for each ethnic category (White or Non-White), as positives correspond to the favorable board disposition. So, by comparing the correct predictions, I will be able to see which group is more likely to be correctly predicted to be favorably prefered by the board.

```
from sklearn import metrics

metrics.confusion_matrix(y[0].values, preds)
```

```
array([[20207,  1005],
       [ 4071,  2866]], dtype=int64)
```

This might pose a problem: it seems that my predictions mostly predicted negative outcomes (only 14% of the predictions are favorable). This might be the direct outcome of how I grouped board dispositions: in the original dataset, Substantiated/Unsubstantiated/Exonerated each had around a third of the dataset, while in my model I grouped the latter two together, creating a 33/66 split. This is something I need to consider in my evaluation, and I believe that it shouldn't pose a problem since the difference in the recall should still be evident if there is unfairness.

```
np.mean(preds)
```

```
0.1375182066858503
```

I will collect the counts of True Negatives, False Positives, False Negatives and True Positives from the confusion matrix in order to calculate the recall.

```
tn, fp, fn, tp = metrics.confusion_matrix(y.values, preds).ravel()

recall = tp / (tp + fn)
recall
```

```
0.4131468934697996
```

This is the recall for the entire dataset. As it seems, it is at 0.413, which is average for the entire dataset. What I will do is I will assign the prediction column and the actual categories (represented by `y`) to the `complainant_ethnciity` column, engineered in a way that only includes the info on whether the complainant is White or not. Then, I will perform the permutation test and will use a test statistic to determine whether there is a discrepancy.

As my test statistic, I want to find the difference between the recall values for each of the subsets. So, as a test statistic I will use **absolute distance**. This is a reasonable metric because all I need to do is determine how close are two values for each of the subsets. I will pick a

significance level of 0.01 (this is a small dataset, so I am more likely to observe extreme values) and I will run the test 500 times.

Null Hypothesis: White and Non-White groups have similar recall values; the model is fair.

Alternative Hypothesis: The two groups have noticeably different recall values, suggesting bias.

This cell creates a transformer that will be binarizing the ethnicity column.

```
is_white = sklearn.preprocessing.FunctionTransformer(
    lambda x: pd.DataFrame(
        pd.Series(x.values[:,0]).str.contains('White').astype('int')
    )
)
```

These cells build a new DataFrame that I will be using in my evaluation.

```
fairness_df =
original_nypd[['complainant_ethnicity']].dropna().reset_index(drop=Tru
e)
fairness_df.head()
```

```
  complainant_ethnicity
0                 Black
1                 Black
2                 Black
3                 Black
4                 White
```

```
fairness_df = fairness_df.assign(**{
    'complainant_ethnicity' : is_white.fit_transform(fairness_df),
    'predictions' : pd.Series(preds),
    'actual_values' : y
})
fairness_df.head()
```

```
   complainant_ethnicity  predictions  actual_values
0                      0          1.0            1.0
1                      0          0.0            1.0
2                      0          1.0            1.0
3                      0          0.0            1.0
4                      1          1.0            1.0
```

Some of the values turned out to be missing. I will simply drop them because it seems that the values that are missing are the predictions and the actual values. This is most likely caused by the fact that when I ran the model, I used `dropna` which dropped ethnicity information in addition to gender and age, so it is likely that the shape of the prediction array is different from the ethnicity information. I don't think it will affect my analysis.

```
fairness_df.isna().sum()
```

```
complainant_ethnicity      0
predictions              745
actual_values            745
dtype: int64
```

```
fairness_df = fairness_df.dropna()
fairness_df.isna().sum()
```

```
complainant_ethnicity    0
predictions              0
actual_values            0
dtype: int64
```

This function calculates recall given a DataFrame of the format similar to `fairness_df`.

```
def find_recall(X):
    preds_arr = X['predictions'].values
    vals_arr = X['actual_values'].values
    tn, fp, fn, tp = metrics.confusion_matrix(vals_arr,
preds_arr).ravel()
    return tp / (tp + fn)

find_recall(fairness_df)
```

```
0.4131468934697996
```

Now, I will define a function `find_recall_diff` that will calculate the difference between recalls across two groups. The result that is outputed by me calling the function on the `fairness_df` is my **observed test statistic**.

```
def find_recall_diff(X):
    slices = X['complainant_ethnicity'] == 1
    white = X[slices]
    non_white = X[~slices]
    return np.abs(find_recall(white) - find_recall(non_white))

observed = find_recall_diff(fairness_df)
observed
```

```
0.020524866712295697
```

Time for the permutation test.

```
results = []
for i in range(500):
    shuffled = fairness_df.assign(**{
        'complainant_ethnicity': fairness_df.sample(replace=False,
frac=1).reset_index(drop=True)
    })
```

```
    results.append(find_recall_diff(shuffled))

p_val = np.mean(observed >= np.array(results))
p_val
```

```
0.706
```

My test produced a p-value of 0.706, which is not statistically significant. This means that I fail to reject my Null Hypothesis. This is not something that I expected, as I expected a certain degree of racial bias to show up especially after the findings in Project 3. However, perhaps the usage of different metrics and approaches allowed for a more comprehensive look into the dataset. The sklearn model performed mathematical calculations that weighted all the features and made the best use of them, so I have reasons to assume that the model reflects the actual dataset quite well. Threfore, based on that, I can assume that the model is not biased, and, by extension, I have the reason to believe that the dataset of my features also doesn't demonstrate a degree of racial bias, at least with the features engineered for my model.

```

```