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

This model will try to classify the column `board_disposition`. That is, the disposition of the complained officer, which should have three categories of results (Substantiated, Unsubstantiated, Exonerated) after the simplification.
Basing on the `CCRB Data Layout Table.xlsx`, I choose some of the features that may help to predict the disposition of the officer. Those features will be introduced columnwise later.

Baseline Model

My Baseline Model is built with the basic one-hot encoded categorical attributes, with the numerical left as it is. The accuracy is about 0.52. And the model is quite overfitted as you can see later.

Final Model

I conduct the Grid Search and the K-Fold cross validation to improve my model. Also, I tried some preprocessor to deal with the numerical data like Binomizer, StandardScaler, and Nomalnizer. However, there is no obvious improvement on the models. Although the model is not overfitted after changing the parameters, the performance is not improved.

At present I have no idea on what happens here. I need more knwoledge on the algorithms of preprocessing and the classifiers that I am using to improve my model.

Fairness Evaluation

After the R^2 evaluation of the models, I will conduct the *demographic parity* test with the permutation test to check whether the model can work fairly when a complain on the officer (Black race or not) will be substantiated (punished).

Code

In [122…

```
# Import Modules...
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures

# Show all dataframe columns
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 10)
```

In [2]:

```
# Read the csv to dataframe
```

```python
DF = pd.read_csv('allegations_202007271729.csv')
DF.head()
```

Out[2]:

	unique_mos_id	first_name	last_name	command_now	shield_no	complaint_id	month_received	year_received	month_closed	year_closed	c
0	10004	Jonathan	Ruiz	078 PCT	8409	42835	7	2019	5	2020	
1	10007	John	Sears	078 PCT	5952	24601	11	2011	8	2012	
2	10007	John	Sears	078 PCT	5952	24601	11	2011	8	2012	
3	10007	John	Sears	078 PCT	5952	26146	7	2012	9	2013	
4	10009	Noemi	Sierra	078 PCT	24058	40253	8	2018	2	2019	

In [3]:

```python
DF['complainant_ethnicity'].value_counts()
```

Out[3]:

```
Black              17114
Hispanic            6424
White               2783
Unknown             1041
Other Race           677
Asian                532
Refused              259
American Indian       64
Name: complainant_ethnicity, dtype: int64
```

Basic Information on the Model

- 1. This model will try to classify the "board_disposition". That is, the disposition of the complained officer, which should have three categories of results (Substantiated, Unsubstantiated, Exonerated). **I need to preprocess this column, merge the result into 3 categories metioned above, and change them into ordinal data (Unsubstantiated: 0, Exonerated: 1, Substantiated: 2)**
- 1. We will use the following possible features to make the classification

- year_received: Numerical Data, no need to do the extra pre-process.
- mos_age_incident: Numerical Data, no need to do extra pre-process.
- mos_ethnicity: Categorical Data, one-hot encode it.
- complaint_age_incident: Numerical Data, no need to do extra pre-process. (Change some rows whose age are out of range to NaN) Also, there are about 5000 rows missing this attributes. The missingness of this column is most likely to be NMAR, I will conduct the probablistic imputation to impute them.
- outcome_description: I will simplify some of its content.Then, I need to conduct the one-hot encoding. (56 empty rows, it is fine do just remove them).
- fado_type: Categorical Data, one-hot encode it.
- complainant_ethnicity: Categorical Data, one-hot encode it. Also, there are about 5000 rows missing this attributes. (There are some races called `Unknown` and `Refused`, I will change them into NaN before the imputation) The missingness of this column is most likely to be NMAR, I will conduct the probablistic imputation to impute them.

Data Cleaning, Imputation and Preparation

Here we only need to preprocess the `board_disposition` column, which should be used as the results to train and test. Also, I will drop the records whose `outcome_description` is empty, which affects only 56 rows.

As to the processing of missingness, I will do the probablistic imputation to impute them and save as much data as possible.

I will leave the preprocessing of other columns to the `pipeline` and `preprocessing` sklearn modules.

Finally, we need to split our data here. I will leave 30% of the data used to test our model, and the remaining 70% to train the model.

In [4]:
```
'''
Preprocess the board_disposition column
Simplify the results into 3 categories and enocde them into ordinal data.
(Unsubstantiated: 0, Exonerated: 1, Substantiated: 2)
'''
DF['board_disposition']= DF['board_disposition'].str.replace(r'Substantiated .+', 'Substantiated',
                                                              regex=True)
DF['board_disposition'].value_counts()
```

Out[4]:
```
Unsubstantiated    15448
Exonerated          9609
Substantiated       8301
Name: board_disposition, dtype: int64
```

In [5]:
```
DF['board_disposition'].replace({'Unsubstantiated':0, 'Substantiated':2, 'Exonerated': 1},
                                inplace=True)
DF['board_disposition'].value_counts()
```

Out[5]:
```
0    15448
1     9609
2     8301
Name: board_disposition, dtype: int64
```

In [6]:
```
'''
Data Imputation
Related Columns: complaint_age_incident, complainant_ethnicity
'''
# Change the unappropriate data into np.NaN
DF.loc[((DF['complainant_ethnicity'] == 'Unknown') | (DF['complainant_ethnicity'] == 'Refused')),
                                                 'complainant_ethnicity'] = np.NaN
DF['complainant_ethnicity'].value_counts()
```

Out[6]:
```
Black              17114
Hispanic            6424
White               2783
Other Race           677
Asian                532
American Indian       64
Name: complainant_ethnicity, dtype: int64
```

In [7]:
```
DF.loc[((DF['complainant_age_incident'] > 100 )|
       (DF['complainant_age_incident'] < 0)), 'complainant_age_incident'] = np.NaN
DF['complainant_age_incident'].value_counts()
```

Out[7]:
```
26.0    1150
24.0    1127
30.0    1056
25.0    1031
23.0    1026
        ...
6.0        1
7.0        1
3.0        1
90.0       1
84.0       1
Name: complainant_age_incident, Length: 86, dtype: int64
```

In [8]:
```
def prob_impute_missing(Series):
```

```
    num_null = Series.isna().sum()
    fill_values = Series.dropna().sample(num_null, replace=True)
    fill_values.index = Series.loc[Series.isna()].index
    return Series.fillna(fill_values.to_dict())
DF['complainant_ethnicity'] = prob_impute_missing(DF['complainant_ethnicity'])
DF['complainant_age_incident'] = prob_impute_missing(DF['complainant_age_incident'])
```

In [9]:

```
print(DF['complainant_ethnicity'].value_counts())
print(DF['complainant_age_incident'].value_counts())
```

```
Black              20626
Hispanic            7787
White               3378
Other Race           838
Asian                651
American Indian       78
Name: complainant_ethnicity, dtype: int64
26.0    1344
24.0    1320
30.0    1246
23.0    1209
25.0    1205
        ...
88.0       1
7.0        1
83.0       1
90.0       1
84.0       1
Name: complainant_age_incident, Length: 86, dtype: int64
```

In [10]:

```
'''
Remove the records whose outcome_description column is null.
'''
DF.dropna(subset=['outcome_description'], inplace=True)
DF['outcome_description'].isna().sum()
```

Out[10]: 0

The imputation and the basic cleaning is done. We can split the dataset now.

I will leave 30% of the data used to test our model, and the remaining 70% to train the model.

In [11]:

```
'''
Split the whole data set into training and testing data sets
```

```
The proportion is 70% Training + 30% Testing


'''

needed_properties = ['year_received', 'outcome_description', 'mos_age_incident',
                     'mos_ethnicity', 'complainant_age_incident', 'complainant_ethnicity',
                     'fado_type']
# 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(DF[needed_properties], DF['board_disposition'],
                                                    test_size=0.3)
```

In [12]:

```
X_train.shape
```

Out[12]:

```
(23311, 7)
```

In [13]:

```
X_test.shape
```

Out[13]:

```
(9991, 7)
```

Now we have 23311 records for training and 9991 records for testing our set

Baseline Model

As what the introduction says, we will use the pipeline to preprocess the data, and predict the result. Here are the details of the preprocessing for each attributes:

- year_received: Numerical Data, no need to do the extra pre-process.
- mos_age_incident: Numerical Data, no need to do extra pre-process.
- mos_ethnicity: Categorical Data, one-hot encode it.
- complaint_age_incident: Numerical Data, no need to do extra pre-process.
- fado_type: Categorical Data, one-hot encode it.
- complainant_ethnicity: Categorical Data, one-hot encode it.
- outcome_description: I will simplify some of its content.Then, I need to conduct the one-hot encoding. There are only two results that we are interested in, arrested or not.

In this baseline model, I will use the random forest classifier to do the classification, which is a very popular classifier with relatively large space for us to improve it by changing the parameters.
Also, it measures the relative importance of each feature on the prediciton, which can inspire us to conduct further study. Also, random frest classifier do not need us to scale the numerical data, I will verify this in the Final Model section.

In [14]:

```
'''
Use ColumnTransformer to preprocess the categorical columns and numerical columns separately.
'''
# This function will simplify this column to only two results that we are interested in, arrested or not.
def simplify_outcome(Series):
    Series[(~(Series == 'No arrest made or summons issued'))] = 'Arrested or Summoned'
    return Series

# Different Columns need Different Preprocess
catcols_needSimp = ['outcome_description'] # This column need to be simplified
catcols = ['complainant_ethnicity', 'mos_ethnicity', 'fado_type']
nums = ['year_received', 'mos_age_incident', 'complainant_age_incident']


simplify = FunctionTransformer(simplify_outcome)
pipeline_cat = Pipeline([
    ('first', simplify),
    ('second', OneHotEncoder(handle_unknown='ignore'))
])
ct = ColumnTransformer([
    ('cat_withSimp', pipeline_cat, catcols_needSimp),
    ('cat',OneHotEncoder(handle_unknown='ignore'), catcols),
    ('num', FunctionTransformer(lambda x:x), nums) # No need to do anything here.
])

# Final Pipeline
pl = Pipeline([
    ('features', ct),
    ('classify', RandomForestClassifier())
])
# Fit the model
pl.fit(X_train, y_train)
```

Out[14]:

```
Pipeline(memory=None,
         steps=[('features',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('cat_withSimp',
```

```
Pipeline(memory=None,
         steps=[('first',
                 FunctionTransformer(accept_sparse=False,
                                     check_inverse=True,
                                     func=<function simplify_outcome at
0x1845D268>,

                                     inv_kw_args=None,
                                     inverse_func=None,
                                     kw_args=None...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=None, max_features='auto',
                                        max_leaf_nodes=None, max_samples=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=100, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
```

Now we have get the fitted pipeline, we can try to evaluate our model with the test data. Here I will check the accuracy of the whole model. Also, I will conduct the analysis **similar** to the confusion matrix. Since we only have three categories result, we can still use the same idea to check the precision for those three categorical results separately.

- Accuracy of the training dataset: 0.916
- Accuracy of the whole testing dataset: 0.531
- Correct prediction of `Substantiated` : 0.40
- Correct prediciton of `Unsubstantiated` : 0.647
- Correct prediciton of `Exonerated` : 0.468

Apparently, our model is overfitted, with the far better performance on the training dataset than the test data set.
Also, the prediction of `Unsubstantiated` seems to be relatively better.

In [15]:

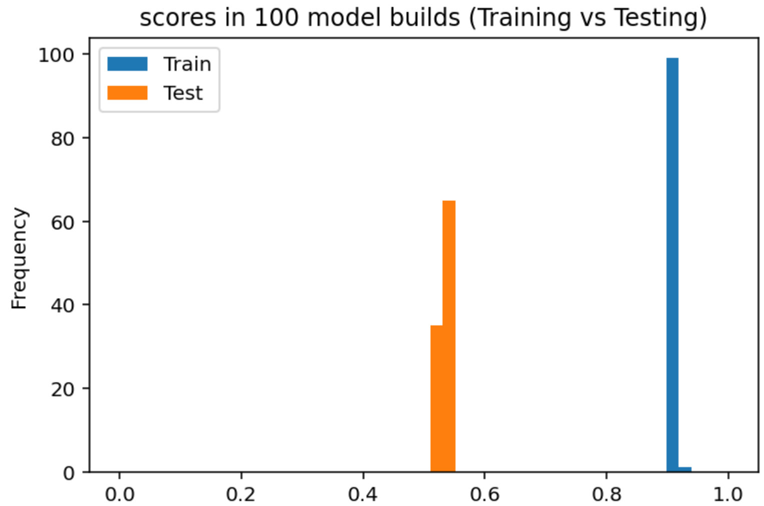
```python
out_train = []
out_test = []
for _ in range(100):
    X_tr, X_ts, y_tr, y_ts = train_test_split(DF[needed_properties], DF['board_disposition'],
                                              test_size=0.3)
    pl.fit(X_tr, y_tr)
    out_train.append(pl.score(X_tr, y_tr))
    out_test.append(pl.score(X_ts, y_ts))
```

In [16]:

```
result = pd.concat([pd.Series(out_train), pd.Series(out_test)], axis=1)
result.columns = ['Train', 'Test']
result.plot(kind='hist', bins =np.linspace(0, 1, 50), alpha=1, title='scores in 100 model builds (Training vs Testing)')
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x62f190>
```



In [17]:

```
# Accuracy of the whole data set
pl.score(X_train, y_train)
```

Out[17]:

```
0.8010381365020806
```

In [18]:

```
# Accuracy of the whole data set
pl.score(X_test, y_test)
```

Out[18]:

```
0.798919027124412
```

In [20]:

```
## Correct prediction of 'Substantiated'
temp_test_correct = y_test.loc[y_test == 2]
temp_test_attrs = X_test.loc[temp_test_correct.index]
pl.score(temp_test_attrs, temp_test_correct)
```

Out[20]:

```
0.7485053806297329
```

In [21]:
```
## Correct prediction of 'Unsubstantiated'
temp_test_correct = y_test.loc[y_test == 0]
temp_test_attrs = X_test.loc[temp_test_correct.index]
pl.score(temp_test_attrs, temp_test_correct)
```

Out[21]: 0.8419347873029583

In [22]:
```
## Correct prediction of 'Exonerated'
temp_test_correct = y_test.loc[y_test == 1]
temp_test_attrs = X_test.loc[temp_test_correct.index]
pl.score(temp_test_attrs, temp_test_correct)
```

Out[22]: 0.7734128376008418

Final Model

Here we can change the classifier or its parameters, and try to optimize the preprocessing procedure. We have the following possible models to improve the performance

- 1. RandomForestClassifier, with the PCA of the categorical data, and the standard scale of the numerical data.
- 1. We can try another classifer Decision Tree. We conduct the Grid Search to get the best parameter of the classifier.
- 1. Add more preprocesser to improve the numerical features and see whether it works.

In [129…
```
'''
Model 1 More Preprocessing Model
'''

# This function will simplify this column to only two results that we are interested in, arrested or not.
def simplify_outcome(Series):
    Series[(~(Series == 'No arrest made or summons issued'))] = 'Arrested or Summoned'
    return Series

# Different Columns need Different Preprocess
catcols_needSimp = ['outcome_description'] # This column need to be simplified
catcols = ['complainant_ethnicity', 'mos_ethnicity', 'fado_type']
nums = ['year_received', 'mos_age_incident', 'complainant_age_incident']


simplify = FunctionTransformer(simplify_outcome)
pipeline_simp_cat = Pipeline([
    ('first', simplify),
```

```python
    ('second', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    ('third', PCA(svd_solver='full'))
])

pipeline_cat = Pipeline([
    ('first', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    ('second', PCA(svd_solver='full'))
])

ct = ColumnTransformer([
    ('cat_withSimp', pipeline_simp_cat, catcols_needSimp),
    ('cat',pipeline_cat, catcols),
    ('num', StandardScaler(), nums) # No need to do anything here.
])

# Final Pipeline
pl = Pipeline([
    ('features', ct),
    ('classify', RandomForestClassifier())
])
# Fit the model
pl.fit(X_train, y_train)
```

Out[129…

```
Pipeline(memory=None,
         steps=[('features',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('cat_withSimp',
                                                  Pipeline(memory=None,
                                                           steps=[('first',
                                                                   FunctionTransformer(accept_sparse=False,
                                                                                       check_inverse=True,
                                                                                       func=<function simplify_outcome at
0x049DBD18>,

                                                                                       inv_kw_args=None,
                                                                                       inverse_func=None,
                                                                                       kw_args=None...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=None, max_features='auto',
                                        max_leaf_nodes=None, max_samples=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=100, n_jobs=None,
```

```
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
```

In [133…

```
pl.score(X_train, y_train)
```

Out[133… 0.9141606966668097

In [131…

```
pl.score(X_test, y_test)
```

Out[131… 0.530677609848864

As what we can forsee, the standard scaler should have make any difference to the random forest classifier. As to the PCA analysis , our categorical data do not have high dimensions, which means the number of categories is not that big. As a result, the PCA analysis may also not improve the performance of this model.

As a result, to improve our model, we may try to use some other classifiers here to see whether it is possible to imrpove our model. Also, it is possible to use Grid Search to find better parameters of a model. Also, we can use the K-Fold cross validation score to check the performance of our model, which will clearly shows us whether the model is overfitted.

In [135…

```
'''
Model 2 Decision Tree Classifier with Grid Search
'''
parameters = {
    'max_depth': [2,3,4,5,7,10,13,15,18,20, 22,None],
    'min_samples_split':[2,3,5,7,10,15,20],
    'min_samples_leaf':[2,3,5,7,10,15,20],
    'max_leaf_nodes': np.arange(2,100,20)
}
# This function will simplify this column to only two results that we are interested in, arrested or not.
def simplify_outcome(Series):
    Series[(~(Series == 'No arrest made or summons issued'))] = 'Arrested or Summoned'
    return Series

# Different Columns need Different Preprocess
catcols_needSimp = ['outcome_description'] # This column need to be simplified
catcols = ['complainant_ethnicity', 'mos_ethnicity', 'fado_type']
nums = ['year_received', 'mos_age_incident', 'complainant_age_incident']
```

```
simplify = FunctionTransformer(simplify_outcome)
pipeline_simp_cat = Pipeline([
    ('first', simplify),
    ('second', OneHotEncoder(handle_unknown='ignore')),
])

pipeline_cat = Pipeline([
    ('first', OneHotEncoder(handle_unknown='ignore')),
])

ct = ColumnTransformer([
    ('cat_withSimp', pipeline_simp_cat, catcols_needSimp),
    ('cat',pipeline_cat, catcols),
    ('num', StandardScaler(), nums) # No need to do anything here.
])

# Final Pipeline
pl = Pipeline([
    ('features', ct),
    ('classify', GridSearchCV(DecisionTreeClassifier(), parameters, cv=5))
])
# Fit the model
pl.fit(X_train, y_train)
```

Out[135…

```
Pipeline(memory=None,
         steps=[('features',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('cat_withSimp',
                                                  Pipeline(memory=None,
                                                           steps=[('first',
                                                                   FunctionTransformer(accept_sparse=False,
                                                                                       check_inverse=True,
                                                                                       func=<function simplify_outcome at
0x287888E0>,

                                                                                       inv_kw_args=None,
                                                                                       inverse_func=None,
                                                                                       kw_args=None...
                                                              presort='deprecated',
                                                              random_state=None,
                                                              splitter='best'),
                              iid='deprecated', n_jobs=None,
                              param_grid={'max_depth': [2, 3, 4, 5, 7, 10, 13,
                                                        15, 18, 20, 22, None],
                                          'max_leaf_nodes': array([ 2, 22, 42, 62, 82]),
```

```
                                      'min_samples_leaf': [2, 3, 5, 7, 10,
                                                           15, 20],
                                      'min_samples_split': [2, 3, 5, 7, 10,
                                                            15, 20]},
                        pre_dispatch='2*n_jobs', refit=True,
                        return_train_score=False, scoring=None,
                        verbose=0))],
         verbose=False)
```

In [136…

```
pl.named_steps['classify'].best_params_
```

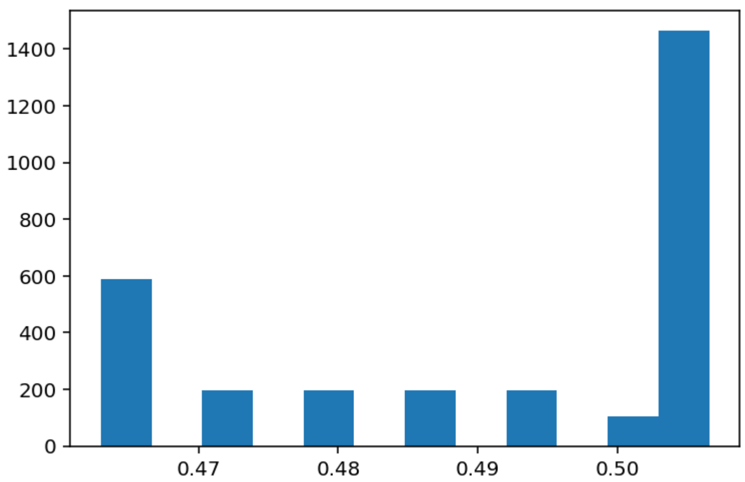
Out[136…

```
{'max_depth': 13,
 'max_leaf_nodes': 82,
 'min_samples_leaf': 7,
 'min_samples_split': 2}
```

In [142…

```
plt.hist(pl.named_steps['classify'].cv_results_['mean_test_score'], bins=12)
plt.suptitle('accuracies on validation set for CV');
```

accuracies on validation set for CV



In [140…

```
pl.score(X_test, y_test)
```

Out[140…

```
0.5051546391752577
```

Still, there is no improvements on the model after changing the classifier and the parameters. Here, I am going to check whether the process of features can be improved.

In [180…

```python
'''
Model 3 Decision Tree Classifier with preprocessing of numerical data
'''
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Binarizer
# This function will simplify this column to only two results that we are interested in, arrested or not.
def simplify_outcome(Series):
    Series[(~(Series == 'No arrest made or summons issued'))] = 'Arrested or Summoned'
    return Series

# Different Columns need Different Preprocess
catcols_needSimp = ['outcome_description'] # This column need to be simplified
catcols = ['complainant_ethnicity', 'mos_ethnicity', 'fado_type']


simplify = FunctionTransformer(simplify_outcome)
pipeline_simp_cat = Pipeline([
    ('first', simplify),
    ('second', OneHotEncoder(handle_unknown='ignore')),
])

pipeline_cat = Pipeline([
    ('first', OneHotEncoder(handle_unknown='ignore')),
])

ct = ColumnTransformer([
    ('cat_withSimp', pipeline_simp_cat, catcols_needSimp),
    ('cat',pipeline_cat, catcols),
    ('num_year', MinMaxScaler(), ['year_received']),
    ('num',StandardScaler(),
     ['mos_age_incident', 'complainant_age_incident']) # No need to do anything here.
])

# Final Pipeline
pl = Pipeline([
    ('features', ct),
    ('classify', DecisionTreeClassifier(max_depth = 13,
                                        max_leaf_nodes = 82,
                                        min_samples_leaf = 7,
                                        min_samples_split = 2))
])
# Fit the model
pl.fit(X_train, y_train)
```

Out[180…

```
Pipeline(memory=None,
         steps=[('features',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('cat_withSimp',
                                                  Pipeline(memory=None,
                                                           steps=[('first',
                                                                   FunctionTransformer(accept_sparse=False,
                                                                                       check_inverse=True,
                                                                                       func=<function simplify_outcome at
0x29CD6898>,

                                                                                       inv_kw_args=None,
                                                                                       inverse_func=None,
                                                                                       kw_args=None...
                                                   'complainant_age_incident'])],
                                   verbose=False)),
                ('classify',
                 DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                        criterion='gini', max_depth=13,
                                        max_features=None, max_leaf_nodes=82,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=7, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        presort='deprecated', random_state=None,
                                        splitter='best'))],
         verbose=False)
```

In [181…

```
cross_val_score(pl, X_train, y_train, cv=5).mean()
```

Out[181…

```
0.5065422243419242
```

Actually, I have no idea on what is happenning here, and I do try lots of preprocessors and classifiers here. However, it does not improve my performance here. Deeper understanding of the modeling is needed, and I will figure it out in my further study.

Fairness Evaluation

In this part, I will focus on the fairness of our data. Here our attribute of interest is the ethnicity of the officier.
I will conduct the demographic parity like this $\mathbb{P}(C=2|A=Black) = \mathbb{P}(C=2|A\neq Black)$ where $(C = 2) \Rightarrow $ "The complains on this officer is substantiated" and
$(A = Black) \Rightarrow $ "The officer's race is black". The model used here is the final model that we get above.

Here is the permutation test:

Null Hypothesis: The model is fair. The proportion to conclude that one complain on a officer is substantiated for `Black Race` officer and `Not Black Race` officers are roughly same.

Alternative Hypothesis: The model is unfair. The proportion to conclude that one complain on a officer is substantiated for `Black Race` officers is higher\lower than the `Not Black Race` officers.

Significance Level: 0.01

Conclusion: The p_value is 0.74. We fail to reject the null hypothesis. And we can confidently say this model is fair for both the black race officers and the non-black race officers, when we are predicting whether the complains about the officers will be substantiated (i.e. the officer will be punished).

In [229...

```
# Get the relative testing data set
results = pd.DataFrame(pl.predict(X_test)) # The first column is prediction
results['corrcect'] = y_test.values # The second column is the correct data
# Set the interesting set:
results['is_black'] = X_train['mos_ethnicity']
results.loc[~(results['is_black'] == 'Black'), 'is_black'] = 'Not Black'

# Get the Demographic Parity
obs_demographic = results.groupby('is_black').apply(lambda x:(x.iloc[:,0] == 2).mean()).diff().iloc[-1]
```

In [232...

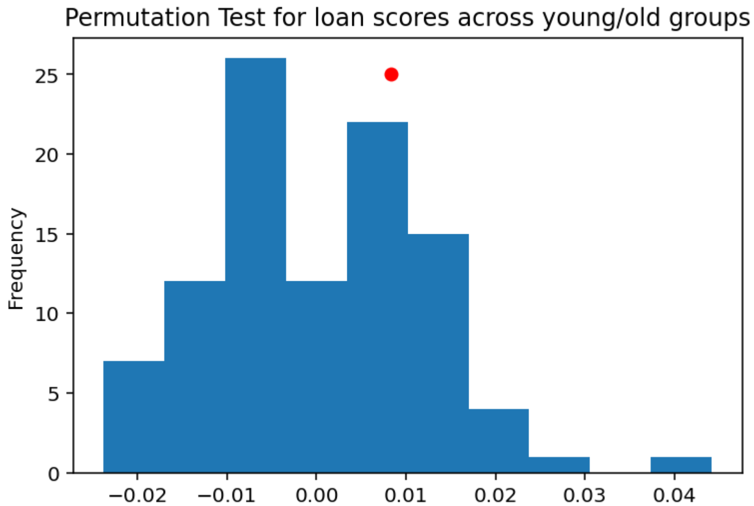
```
# Permutation Test
metrs = []
for _ in range(100):
    results['sampled'] = results.is_black.sample(frac=1.0, replace=False).reset_index(drop=True)
    s = (
        results.groupby('sampled')
        .apply(lambda x: (x.iloc[:,0] == 2).mean())
        .diff()
        .iloc[-1]
    )

    metrs.append(s)
```

In [243...

```
print(pd.Series(metrs <= obs_demographic).mean())
pd.Series(metrs).plot(kind='hist', title='Permutation Test for loan scores across young/old groups')
plt.scatter(obs_demographic, 25, c='r');
```

```
0.74
```


Permutation Test for loan scores across young/old groups
Frequency
0
5
10
15
20
25
−0.02
−0.01
0.00
0.01
0.02
0.03
0.04