# Final Project

In this project, I will be looking into the NYPD data set and strive to extract values out of the data. On the first part of this project, statistical and analytical approaches will be applied to the dataset,aiming at mining some useful information. On the second part of this project, a predictive model will be developed to address a specific task, which could be useful in reality. I will explain more on that as you read through this project.

## Section 1: Data Analysis and Data Mining

The dataset is from ProPublica and records more than 12,000 civilian complaints filed against New York City police officers. The records span decades, from September 1985 to January 2020, and contains the information including complaint date, complaint type, age, ethnicity, and gender of both police officers and complainants.

The objective/research question of this project is to examine: Are the complaints made agaisnt the police officers more easy to be resolved for Women than to be resolved for Men ?

I will answer this question by creating some exploratory data analysis (analyzing the statistics would help explain our questions) and conducting a few permutaiton tests for each category of complaint.

Based on our research setting, the most directly factors affecting the outcome (complain's disposition: Substantiated/ Unsubstantiated/ Exonerated) of a complaint is gender of complainants. On the other hand, we also recognize that there are other variables that might affect the substantiated rate, for example race of complainants, race of the police officer; however, we would only focused on the relationship between gender and complainant substantiated rate in this project.

I also regnozied that the result, inevitably, will come with certain bias as we ignore other genders beside female and male in the research question. We offer a way of thinking here but hopefully it will also provide more insights to the readers on the concerned matter.

I have labelled each finding with a tag in a pair of square brackets so that you can easily find the corresponding code :)

## Cleaning and EDA

#### *Brief discription about our data cleaning processes*

Fisrt of all, I did some very basic cleaning. I removed all the ages that are bellow 0 . Also,I took care of some wrong sheild numbers. [CLEAN I]

I also clean the "board_disposition" column. Since I am only interested in three main categories "exonerated", "substantiated", and "unsubstantiated"， I have decided to aggregate small sub-categories into one large categories. This greatly simplify our analysis while bear no undesirable repurcussion on the final coutcome. [CLEAN II]

[This cleaning step comes with ethical concern.so I only implement once ] Since my analysis mainly focus on the gender inequality and racial discrimination, and also because of the fact that I do not have enough sample representing the minority gender here. I have decided to remove all minority gender for the sake of our analysis.( I will implement this step right before hypothesis and I will not do it at the begining so that every minority gender get represented fully during the EDA.)

#### *let's ananlyze the demographic of our sample first through some diagrams and graphs*

As you can observe from this diagram, the race composition of the officers who are being accused actually suggests what is opposite to a commonly held perception that White officer are more probable of being accused from the alleged wrongdoing.[Diagram 1]

Secondly, let's look at the ethinities of people who file the complaints against the officers. The mojority of the complainant are black people here, which mean black people may be more susceptible to injustice or unfair treatment than any other races are. [Diagram 2]

In my third analysis, I found that male are more likely to submit a complaint than female are. I need to take this factor into consideration. Because this might potentially bias my analysis regarding genders since men women or other genders are not equally represented in the given sample. [Diagram3]

Then I also look at the "Age Distribution among Complainant Genders". The majority of people who filed a complaint are round 25 to 45 regardless of their genders. [Diagram4] [Diagram 5]

#### *Here is the interesting finding [Chart1 and Diagram 6]*

Also, it worth noticing that among all the disposition of the filed complaints, the proportions that complaints are disposed as "substantiated" between men and women differ in a magnitude that I could not neglect, which is about 5 percent of difference. That is being said, the complaint brought by women might less likely to be "substantiated" than men are. Aside from that, data also shows that proportion of women whose complaints are unsubstantiated is higher than the corresponding portion of their counterpart.Proportion which offciers who are substantiated of wrongdoing got away with those charges is much higher for the women than for the man. All those differences in our statistics suggest that there might be discrimination against general women population when comes to the disposition of a filed complaint. For that, we will conduct a hypothesis test to further investigate this finding.

This is a fllow-up investigation on the issue mentioned aboved. But this time, I will process the imformation at another level of granularity. Take an example to illustrate this, Say, what will the distribution of dispositions be like among the complaints that are about "Abuse of Power"?

## Assessment of Missingness

#### *Handle NI (non-ignorable) missingness [ Additional Info needed]*

I firmly believe that the variable "gender" have non-ignorable missingness. The reason is that sometimes it is hard tell someone's gender just by his or her appearence or you can hardly acquire this information without confronting any subjective issues. There might be a ambiguity in deciding what real gender is or someone just do not want their gender to be known by other people, which leads to the missing value In short, the missingness here actually depends on the variable itself. If we can have some additional information such as the complainant's pronounce ( provided by the complainant themself), we might be able to decisively figure out what's missing here.

***Using permuation test to assess the missingness***

***case 1 (complainant_ethnicity vs mos_age_incident) ( NO Dependence)(significance level 0.05)***

I found that the missingness of the "complainant_ethnicity" is actually independent from variable "mos_age_incident". Graphically speaking, the distributions of missingness against variable "mos_age_incident" are quite similar, which suggests that those two distributions may be the same.(i.e they come from the same data generating process) And the p-value that I get from our permutation test also suggests that two distribution are likely to be identical.

***(complainant_ethnicity vs year_received) ( Strong Dependence) (significance level 0.05)***

The missingness of "complainant_ethnicity" is strongly dependent on the variable "year_received". Graphically speaking, the distributions of missingness against variable "year_received" are quite different. And the small p-value (0) that we get from our permutation test also suggests that two distribution are likely to be different.

## Hypothesis Test (Permutation test)

Because I want to examine if gender plays an important role in disposition substantiated rate (for the same allegation), thus I conduct a permutation test and shuffle the complainant_gender to compare the percentage of complaint made by male and female that are substantiated for each allegation categories. Since there are four allegation categories (Abuse of Authority, Discourtesy, Force, Offensive Language), I will perform four permutation tests in total and compare their outcomes.

One of the appropriate test statistics to use is difference in proportion. More explicitly, the difference in male substantiated rate and women substantiated rate. And the significance level we choose for the permutation tests is 0.05.

**Null hypothesis: In the population, disposition substantiated rate of women complainant and men complainant have the same distribution.**

- p1=p2 or p1-p2=0
  - p1 stands for male complainant substantiated rate
  - p2 stands for female complainant substantiated rate

**Alternative hypothesis: In the population, women usually have less complainant substantiated rate than men have. p1-p2 > 0**

Results: p-values for Abuse of Authority, Discourtesy, Force and Offensive Language are 0.0, 0.012, 0.005, and 0.465

Conclusion:

- Because the p-values for Abuse of Authority, Discourtesy and Force are all much smaller than the 0.05, the statistical test gives us sufficient evidence to reject the null, which states disposition substantiated rate of women complainant and men complainant have the same distribution.
- However, since the p-value for Offensive Language (0.4715) is greater than 0.05, the test failed to reject the null hypothesis test for this category of complaint and agree that disposition substantiated rate of women complainant and men complainant have the same distribution.

# Code

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

In [2]:
```
# Before doing anyting, let's import the data
path=os.path.join("data","allegations_202007271729.csv")
allegations=pd.read_csv(path)
```

## Cleaning and EDA

we cleaned the following variables:

- complainant_age_incident: replace ages that are under 12 to np.nan to exclude negative ages and unrealistic ages.
- board_disposition: keep general information, only including Substantiated, Exonerated, Unsubstantiated for further analysis.
- shield_no: replace invaild shield_no (<=0) to np.nan

EDA Graphs:

- Univariate Analysis:
  - distribution of officers ethnicity
  - distribution of ethnicity of complainant
  - distribution of complainant gender
  - Distribution of complainant age
- Bivariate Analysis:
  - age distribution among Complainant Genders
  - Disposition result over famle and male
- Bivariate Analysis & Interesting Aggregates
  - Stacked barplot of substantiated rate among different fado reasons (category of complaint)
  - Chart1

In [3]:
```
allegations[["complainant_gender","fado_type","allegation","board_dispo
sition"]]
```

Out[3]:

| | complainant_gender | fado_type | allegation | board_disposition |
|---|---|---|---|---|

| | complainant_gender | fado_type | allegation | board_disposition |
|---|---|---|---|---|
| **0** | Female | Abuse of Authority | Failure to provide RTKA card | Substantiated (Command Lvl Instructions) |
| **1** | Male | Discourtesy | Action | Substantiated (Charges) |
| **2** | Male | Offensive Language | Race | Substantiated (Charges) |
| **3** | Male | Abuse of Authority | Question | Substantiated (Charges) |
| **4** | NaN | Force | Physical force | Substantiated (Command Discipline A) |
| **...** | ... | ... | ... | ... |
| **33353** | Male | Discourtesy | Word | Unsubstantiated |
| **33354** | Male | Abuse of Authority | Interference with recording | Unsubstantiated |
| **33355** | Male | Abuse of Authority | Search (of person) | Substantiated (Formalized Training) |
| **33356** | Male | Abuse of Authority | Vehicle search | Substantiated (Formalized Training) |
| **33357** | Male | Abuse of Authority | Frisk | Substantiated (Formalized Training) |

33358 rows × 4 columns

In [4]:

```
# CLEAN 1
# Some cleaning work are necessary
# remove the invalid age first. Remove all ages that are under 12
allegations['complainant_age_incident'] = allegations['complainant_age_
incident'].apply(lambda x: x if x > 11 else np.nan)
```

In [5]:

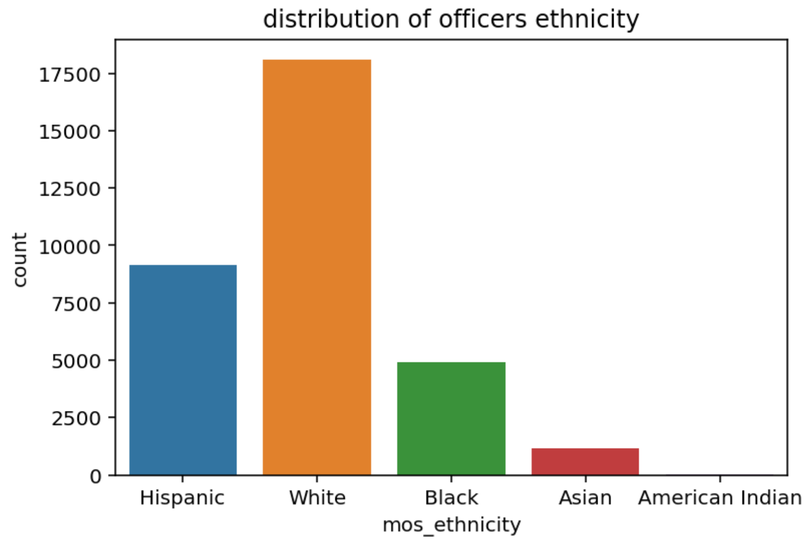
```
# CLEAN II
# cleaning disposition: keep general disposition -- Substantiated, Exon
erated, Unsubstantiated
```

```
allegations['board_disposition'] = allegations['board_disposition'].app
ly(lambda x: x.split(' ')[0])
```

In [6]:
```
# ClEAN III
# replace the shield_no which is equal or less than 0 to np.nan
allegations['shield_no'] = allegations['shield_no'].apply(lambda x: x i
f x > 0 else np.nan)
```

In [7]:
```
# Diagram 1 - Univariate Analysis
# ethnicity composition of complainted officers in NYPD

ax = sns.countplot(x = "mos_ethnicity", data = allegations)
ax.set_title('distribution of officers ethnicity')
```

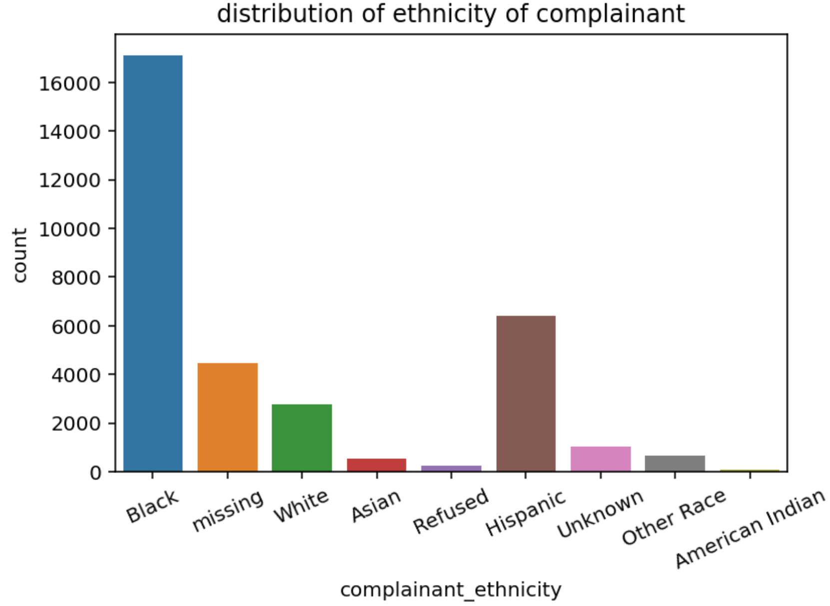
Out[7]:
```
Text(0.5, 1.0, 'distribution of officers ethnicity')
```




In [8]:
```
# Diagram 2 - Univariate Analysis
# the ethnicity of complainant
temp = allegations.copy()
temp['complainant_ethnicity'] = temp["complainant_ethnicity"].replace(n
p.nan,"missing")
```

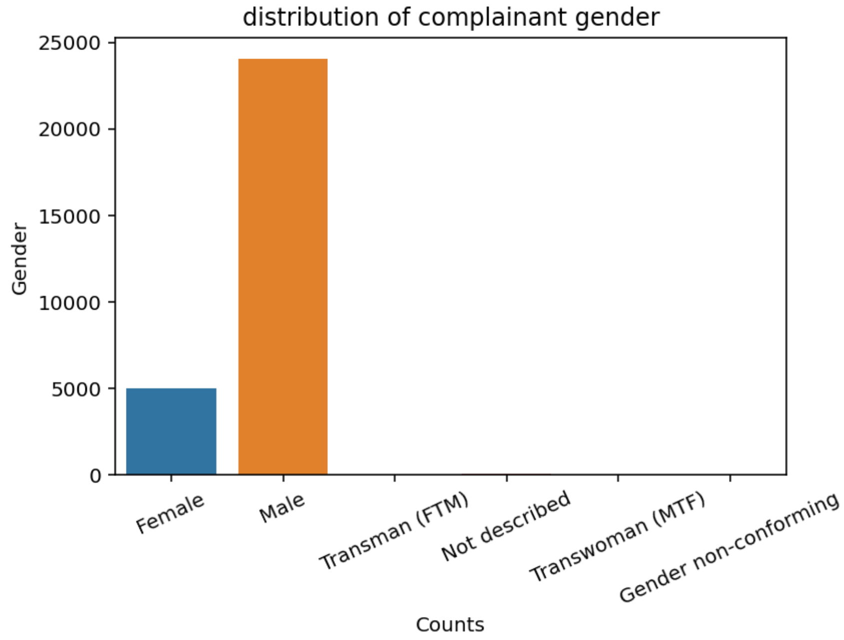
```
ax = sns.countplot(x = "complainant_ethnicity", data = temp)
plt.xticks(rotation=25)
ax.set_title('distribution of ethnicity of complainant')
```

Out[8]: Text(0.5, 1.0, 'distribution of ethnicity of complainant')




In [9]:
```
# disgram 3 - Univariate Analysis
# Let's look at the gender distribution of complainants
ax = sns.countplot(x='complainant_gender', data=allegations);
ax.set_title('distribution of complainant gender')
plt.xticks(rotation=25)
ax.set_ylabel('Gender')
ax.set_xlabel('Counts')
```

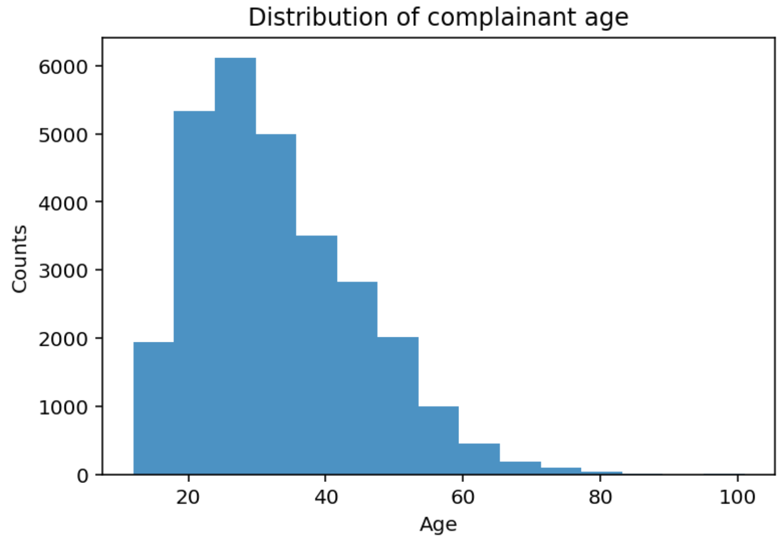
Out[9]: Text(0.5, 0, 'Counts')




In [10]:
```
# disgram 4  - Univariate Analysis
# distribution of age in total
plt.hist(allegations['complainant_age_incident'], bins=15, alpha=0.8)
plt.title('Distribution of complainant age')
plt.xlabel('Age')
plt.ylabel('Counts')
```

```
C:\Users\RUI\anaconda3\lib\site-packages\numpy\lib\histograms.py:839: R
untimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\Users\RUI\anaconda3\lib\site-packages\numpy\lib\histograms.py:840: R
untimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

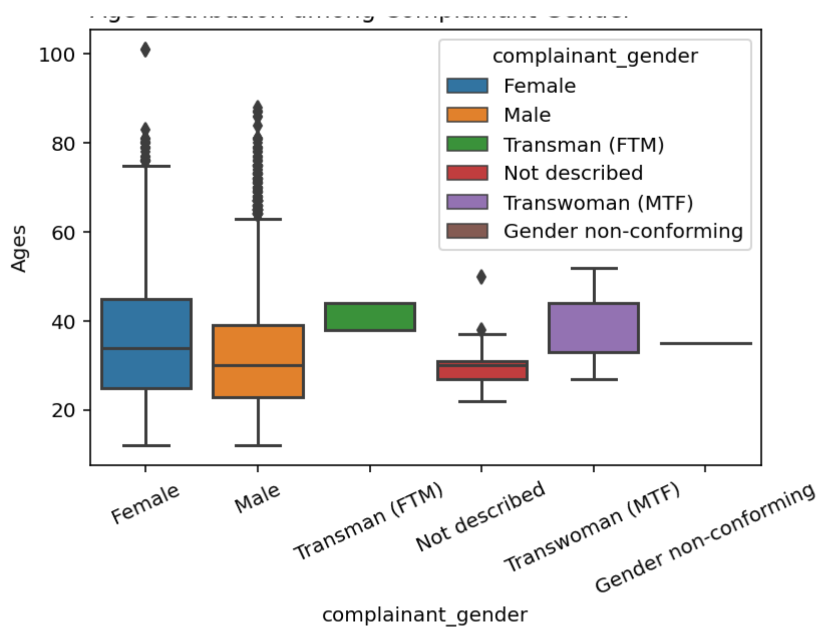
Out[10]:
```
Text(0, 0.5, 'Counts')
```




In [11]:
```
# diagram 5 - Bivariate Analysis
# Age Distribution among Complainant Genders - Bivariate Analysis
ax = sns.boxplot(data = allegations,x = 'complainant_gender', y = 'comp
lainant_age_incident',
                 hue = 'complainant_gender',dodge=False)
plt.xticks(rotation=25)
ax.set_title('Age Distribution among Complainant Gender', loc='left')
ax.set_ylabel('Ages')
```

Out[11]:
```
Text(0, 0.5, 'Ages')
```

Age Distribution among Complainant Gender




In [12]:

```
# Chart1
# get a table representing of the results regarding those filed complai
nts
# normalize the counts
# compare those data of men and women
data = allegations.pivot_table(
        values="unique_mos_id",
        index="complainant_gender",
        columns="board_disposition",
        aggfunc="count"
    )
cleaned_data=data.dropna()
cleaned_data.iloc[[0,1]].T.apply(lambda x:x/sum(x)).T
```

Out[12]:

| board_disposition | Exonerated | Substantiated | Unsubstantiated |
|---|---|---|---|
| complainant_gender | | | |
| Female | 0.281816 | 0.205537 | 0.512647 |

| board_disposition | Exonerated | Substantiated | Unsubstantiated |
|---|---|---|---|
| complainant_gender | | | |
| **Male** | 0.273132 | 0.255674 | 0.471195 |

**Graphs directly related to our hypothesis:**

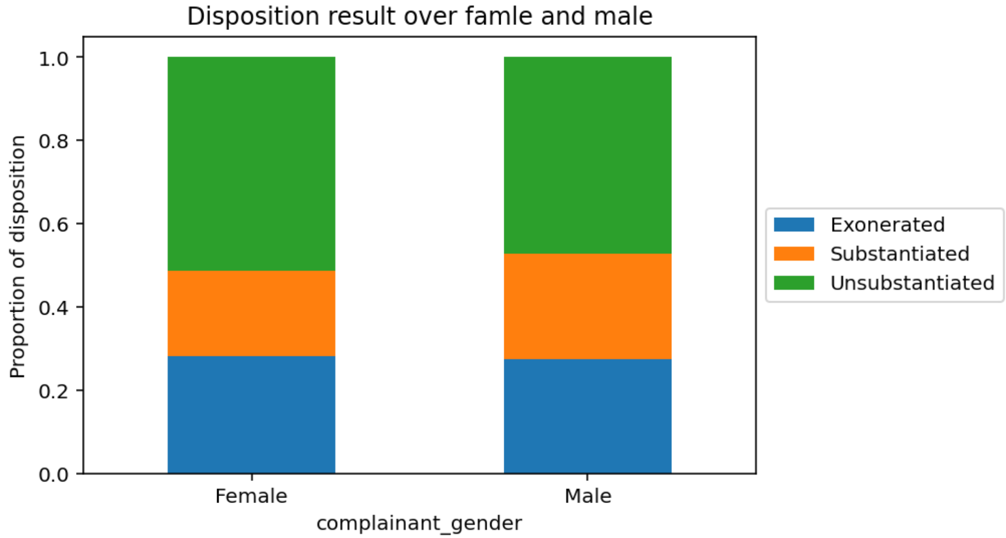
Because our research question (Are the complaints of women more succesful than men (for the same allegations?)) only focus on women and men, in our complainant_gender category, we would also just focus on the female and male to exam.

- Graphs to include:
  - Bar graph of count of female and male complains over final disposition
  - Bar graph of normalized female and male complains over final disposition

In [13]:
```
# only keep the rows there complainant_gender is female or male
gender = allegations[(allegations['complainant_gender'] == 'Female')
                    | (allegations['complainant_gender'] == 'Male')]
```

In [14]:
```
# generate the data
data = gender.pivot_table(
        values="unique_mos_id",
        index="board_disposition",
        columns="complainant_gender",
        aggfunc="count"
    )
data = (data/data.sum()).T

# generate plot
ax = data.plot(kind='bar', stacked=True, rot=0,
                 title='Disposition result over famle and male')

# customize plot
ax.legend(('Exonerated','Substantiated','Unsubstantiated'), loc='center
```

```
left', bbox_to_anchor=(1.0, 0.5))
ax.set_ylabel("Proportion of disposition");
```



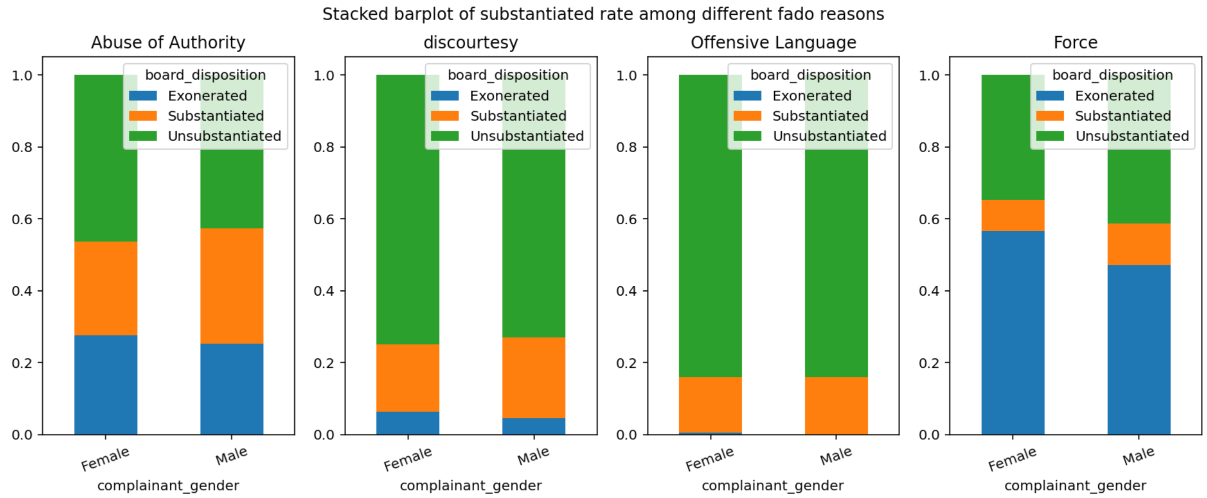

In [15]:
```
# diagram 6 - Bivariate Analysis & Interesting Aggregates
# multivariate analysis. does our statistics looks different at differe
nt level of granularity
# generate the dataset among four different fado types(category of comp
laint)
abuse_data = gender[gender['fado_type'] == 'Abuse of Authority']
abuse = abuse_data.pivot_table(
        values="unique_mos_id",
        index="board_disposition",
        columns="complainant_gender",
        aggfunc="count"
    )
abuse = (abuse/abuse.sum()).T


discourtesy_data = gender[gender['fado_type'] == 'Discourtesy']
discourtesy = discourtesy_data.pivot_table(
        values="unique_mos_id",
        index="board_disposition",
```

```
        columns="complainant_gender",
        aggfunc="count"
    )
discourtesy = (discourtesy/discourtesy.sum()).T
discourtesy

offensive_data = gender[gender['fado_type'] == 'Offensive Language']
offensive = offensive_data.pivot_table(
        values="unique_mos_id",
        index="board_disposition",
        columns="complainant_gender",
        aggfunc="count"
    )
offensive = (offensive/offensive.sum()).T
offensive

force_data= gender[gender['fado_type'] == 'Force']
force = force_data .pivot_table(
        values="unique_mos_id",
        index="board_disposition",
        columns="complainant_gender",
        aggfunc="count"
    )
force = (force/force.sum()).T
force

# generate the plot
fig, axes = plt.subplots(1, 4)
abuse.plot(ax = axes[0],kind='bar', stacked=True, rot=20,title = 'Abuse
of Authority',figsize = (15,5))
discourtesy.plot(ax = axes[1],kind='bar', stacked=True, rot=20, title =
'discourtesy')
offensive.plot(ax = axes[2],kind='bar', stacked=True, rot=20, title =
'Offensive Language')
force.plot(ax = axes[3],kind='bar', stacked=True, rot=20, title = 'Forc
e')
fig.suptitle('Stacked barplot of substantiated rate among different fad
o reasons')
```

Out[15]:

```
Text(0.5, 0.98, 'Stacked barplot of substantiated rate among different
```

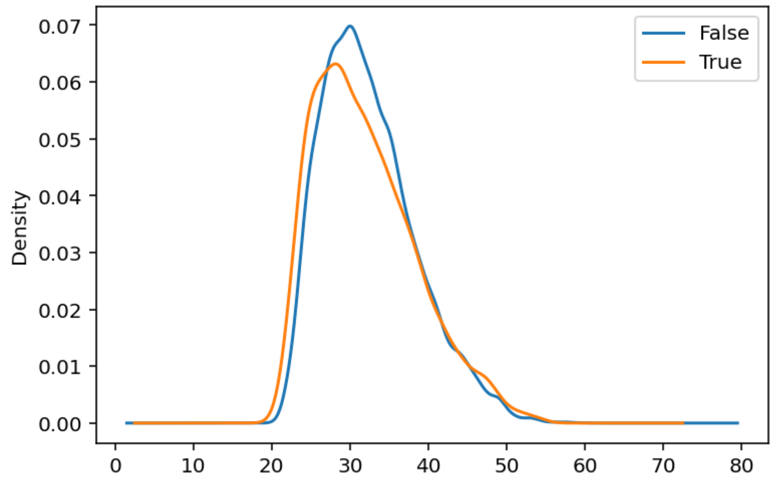
fado reasons')




## Assessment of Missingness

In [16]:
```
# permutation test [complainant_ethnicity vs. mos_age_incident]
# Basic plan: plot the distribution of mos_age_incident vs complainant_
ethnicity

null_vector=allegations["complainant_ethnicity"].isnull()
test_df=allegations.assign(is_null=null_vector)

test_df.groupby("is_null").mos_age_incident.plot(kind="kde",legend=True
)
```

Out[16]:
```
is_null
False    AxesSubplot(0.125,0.125;0.775x0.755)
True     AxesSubplot(0.125,0.125;0.775x0.755)
Name: mos_age_incident, dtype: object
```




In [17]:
```
# since two means are so close, use KS statistic to assess whether two
 distributions are different
from scipy.stats import ks_2samp
```

In [18]:
```
group_A=test_df[test_df["is_null"]==True]["mos_age_incident"]
group_B=test_df[test_df["is_null"]==False]["mos_age_incident"]
ks_statistic=ks_2samp(group_A, group_B).statistic
ks_statistic
```

Out[18]:
```
0.06591802120369275
```

In [19]:
```
num_repetitions=1000
simulations2=[]
for i in range(num_repetitions):
    # first and foremost, we must shuffle the column of age
    shuffled_col=test_df["mos_age_incident"].sample(replace=False, frac
=1).reset_index(drop=True)
    # put them in a table
    shuffled_table=allegations.assign(**{"mos_age_incident":shuffled_co
l,"is_null":allegations['complainant_ethnicity'].isnull()})
    # compute a different statistic
    group_A_null=shuffled_table[shuffled_table["is_null"]==True]["mos_a
```

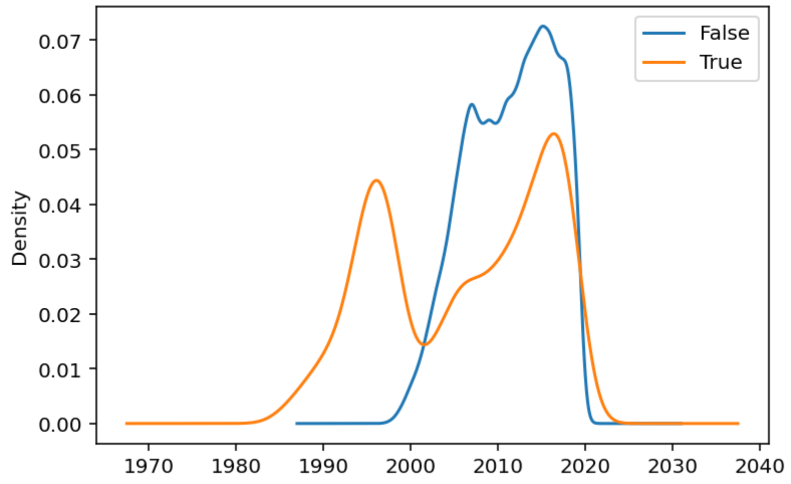
```
ge_incident"]
    group_B_null=shuffled_table[shuffled_table["is_null"]==False]["mos_
age_incident"]
    ks_statistic_null=shuffled_table.groupby("is_null")["mos_age_incide
nt"].mean().diff().iloc[-1]
    simulations2.append(ks_statistic_null)
```

In [20]:
```
# get the p_value
vectorized_simulations2=pd.Series(simulations2)
p_value=(ks_statistic<=vectorized_simulations2).mean()
p_value
```

Out[20]:
```
0.259
```

In [21]:
```
# permutation test [complainant_ethnicity vs. year_received]
# Basic plan: plot the distribution of mos_age_incident vs complainant_
ethnicity

null_vector=allegations["complainant_ethnicity"].isnull()
test_df=allegations.assign(is_null=null_vector)

test_df.groupby("is_null").year_received.plot(kind="kde",legend=True)
```

Out[21]:
```
is_null
False    AxesSubplot(0.125,0.125;0.775x0.755)
True     AxesSubplot(0.125,0.125;0.775x0.755)
Name: year_received, dtype: object
```




In [22]:
```
group_A=test_df[test_df["is_null"]==True]["year_received"]
group_B=test_df[test_df["is_null"]==False]["year_received"]
ks_statistic=ks_2samp(group_A, group_B).statistic
ks_statistic
```

Out[22]:
```
0.3466206227037251
```

In [23]:
```
num_repetitions=1000
simulations2=[]
for i in range(num_repetitions):
    # first and foremost, we must shuffle the column of age
    shuffled_col=test_df["year_received"].sample(replace=False, frac=1)
.reset_index(drop=True)
    # put them in a table
    shuffled_table=allegations.assign(**{"year_received":shuffled_col,
"is_null":allegations['complainant_ethnicity'].isnull()})
    # compute a different statistic
    group_A_null=shuffled_table[shuffled_table["is_null"]==True]["year_
received"]
    group_B_null=shuffled_table[shuffled_table["is_null"]==False]["year
_received"]
    ks_statistic_null=shuffled_table.groupby("is_null")["year_received"
```

```
].mean().diff().iloc[-1]
    simulations2.append(ks_statistic_null)
```

In [24]:
```
vectorized_simulations2=pd.Series(simulations2)
p_value=(ks_statistic<=vectorized_simulations2).mean()
p_value
```

Out[24]:
```
0.001
```

### Hypothesis/Permutation Test

Four permutaion tests for each disposition:

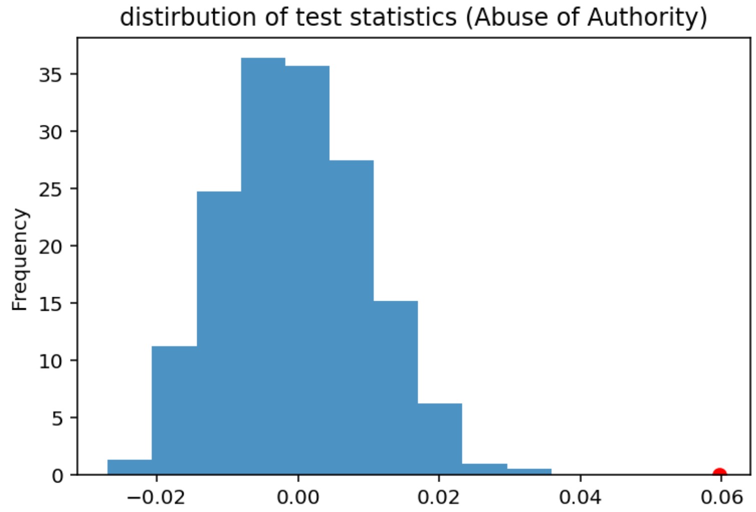
- Abuse of Authority
- Discourtesy
- Offensive Language
- Force

In [25]:
```
# allegation type: 'abuse'
display(abuse)
abuse_data = abuse_data[['fado_type','complainant_gender','board_dispos
ition']].reset_index()
# find test statistics: difference in proportion (male - female)
size = abuse_data.shape[0]
obs = abuse.diff().iloc[-1][1]
stats = []
for _ in range(1000):
    # shuffle the gender
    shuffled_gender = (
        abuse_data['complainant_gender']
        .sample(replace=False, frac=1)
        .reset_index(drop=True)
    )


    # put them in a table
```

```
    shuffled = (
        abuse_data
        .assign(**{'shuffled_gender': shuffled_gender})
    )

    group_counts =(shuffled.pivot_table(
        values="fado_type",
        index="board_disposition",
        columns="shuffled_gender",
        aggfunc="count"
    ))
    group_means = (group_counts/group_counts.sum()).T
    difference = group_means.diff().iloc[-1][1]

    stats.append(difference)

p_value = (pd.Series(stats) >= obs).mean()
p_value_abuse = p_value
print('p-value: %f' % p_value)
pd.Series(stats).plot(kind='hist', density=True, alpha=0.8, title = 'di
stirbution of test statistics (Abuse of Authority)')
plt.scatter(obs, 0, color='red', s=40);
```

| board_disposition | Exonerated | Substantiated | Unsubstantiated |
|---|---|---|---|
| **complainant_gender** | | | |
| **Female** | 0.275875 | 0.261297 | 0.462828 |
| **Male** | 0.252910 | 0.320964 | 0.426126 |

```
p-value: 0.000000
```

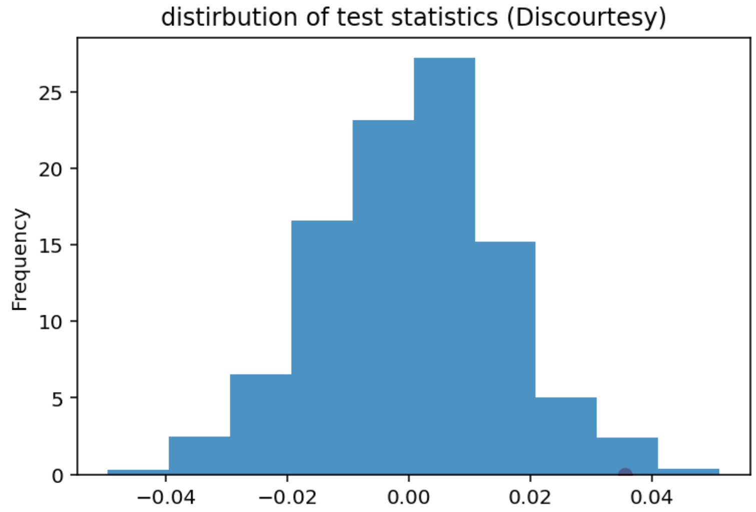



In [26]:
```
# allegation type: 'discourtesy'
display(discourtesy)
discourtesy_data = discourtesy_data[['fado_type','complainant_gender',
'board_disposition']].reset_index()
# find test statistics: difference in proportion (male - female)
size = discourtesy_data.shape[0]
obs = discourtesy.diff().iloc[-1][1]
stats = []
for _ in range(1000):
    # shuffle the gender
    shuffled_gender = (
        discourtesy_data['complainant_gender']
        .sample(replace=False, frac=1)
        .reset_index(drop=True)
    )

    # put them in a table
    shuffled = (
        discourtesy_data
        .assign(**{'shuffled_gender': shuffled_gender})
    )
```

```
    group_counts =(shuffled.pivot_table(
        values="fado_type",
        index="board_disposition",
        columns="shuffled_gender",
        aggfunc="count"
    ))
    group_means = (group_counts/group_counts.sum()).T
    difference = group_means.diff().iloc[-1][1]

    stats.append(difference)

p_value = (pd.Series(stats) >= obs).mean()
p_value_discourtesy = p_value
print('p-value: %f' % p_value)
pd.Series(stats).plot(kind='hist', density=True, alpha=0.8, title = 'di
stirbution of test statistics (Discourtesy)')
plt.scatter(obs, 0, color='red', s=40);
```

| board_disposition | Exonerated | Substantiated | Unsubstantiated |
|---|---|---|---|
| **complainant_gender** | | | |
| **Female** | 0.063851 | 0.187623 | 0.748527 |
| **Male** | 0.046144 | 0.223289 | 0.730567 |

```
p-value: 0.012000
```

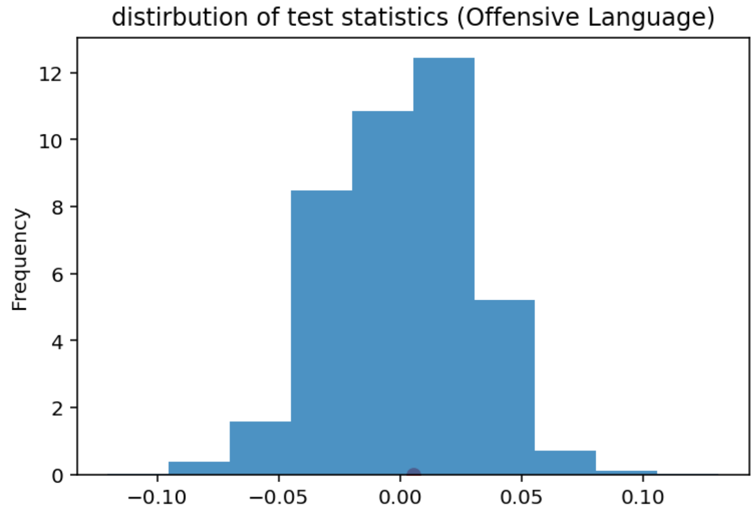



In [27]:
```
# allegation type: 'offensive'
display(offensive)
offensive_data = offensive_data[['fado_type','complainant_gender','boar
d_disposition']].reset_index()
# find test statistics: difference in proportion (male - female)
size = offensive_data.shape[0]
obs = offensive.diff().iloc[-1][1]
stats = []
for _ in range(2000):
    # shuffle the gender
    shuffled_gender = (
        offensive_data['complainant_gender']
        .sample(replace=False, frac=1)
        .reset_index(drop=True)
    )

    # put them in a table
    shuffled = (
        offensive_data
        .assign(**{'shuffled_gender': shuffled_gender})
    )
```

```
    group_counts =(shuffled.pivot_table(
        values="fado_type",
        index="board_disposition",
        columns="shuffled_gender",
        aggfunc="count"
    ))
    group_means = (group_counts/group_counts.sum()).T
    difference = group_means.diff().iloc[-1][1]

    stats.append(difference)

p_value = (pd.Series(stats) >= obs).mean()
p_value_offensive = p_value
print('p-value: %f' % p_value)
pd.Series(stats).plot(kind='hist', density=True, alpha=0.8, title = 'di
stirbution of test statistics (Offensive Language)')
plt.scatter(obs, 0, color='red', s=40);
```

| board_disposition | Exonerated | Substantiated | Unsubstantiated |
|---|---|---|---|
| **complainant_gender** | | | |
| **Female** | 0.004695 | 0.154930 | 0.840376 |
| **Male** | NaN | 0.160183 | 0.839817 |

```
p-value: 0.465000
```




In [28]:

```
# allegation type: 'Force'
display(force)
force_data = force_data[['fado_type','complainant_gender','board_dispos
ition']].reset_index()
# find test statistics: difference in proportion (male - female)
size = force_data.shape[0]
obs = force.diff().iloc[-1][1]
stats = []
for _ in range(2000):
    # shuffle the gender
    shuffled_gender = (
        force_data['complainant_gender']
        .sample(replace=False, frac=1)
        .reset_index(drop=True)
    )

    # put them in a table
    shuffled = (
        force_data
        .assign(**{'shuffled_gender': shuffled_gender})
    )
```

```
    group_counts =(shuffled.pivot_table(
        values="fado_type",
        index="board_disposition",
        columns="shuffled_gender",
        aggfunc="count"
    ))
    group_means = (group_counts/group_counts.sum()).T
    difference = group_means.diff().iloc[-1][1]

    stats.append(difference)

p_value = (pd.Series(stats) >= obs).mean()
p_value_force = p_value
print('p-value: %f' % p_value)
pd.Series(stats).plot(kind='hist', density=True, alpha=0.8, title = 'di
stirbution of test statistics (Force)')
plt.scatter(obs, 0, color='red', s=40);
```

| board_disposition | Exonerated | Substantiated | Unsubstantiated |
|---|---|---|---|
| **complainant_gender** | | | |
| **Female** | 0.565966 | 0.086998 | 0.347036 |
| **Male** | 0.471483 | 0.116142 | 0.412375 |

```
p-value: 0.005000
```


In [29]:

```
# generate data for dataframe
data = {'p_value':[p_value_abuse,p_value_discourtesy,p_value_force,p_va
lue_offensive]}
summary_pvalue = pd.DataFrame(data)
summary_pvalue = summary_pvalue.assign(disposition = ['Abuse of Authori
ty','Discourtesy',
                                                        'Force','Offensiv
e Language']).set_index('disposition')
# add column reject/ftr
summary_pvalue = summary_pvalue.assign(**{'Reject/FTR': summary_pvalue[
'p_value'].
                                            apply(lambda x: 'Reject' if x
< 0.05 else 'FTR')})
summary_pvalue
```

Out[29]:

| disposition | p_value | Reject/FTR |
|---|---|---|
| Abuse of Authority | 0.000 | Reject |
| Discourtesy | 0.012 | Reject |
| Force | 0.005 | Reject |

| | p_value | Reject/FTR |
|---|---|---|
| Offensive Language disposition | 0.465 | FTR |

# Section2 Predictive Model

## Introduction

- In this part, I am trying to **predict the outcome of the allegations that are sent to NYPD officers who are being accused of wrongdoing and misdemeanor**. The problem is going to be a **classification** problem with the goal to **classify whether a given allegation will be 1 (meaning substantiated) or 0 (meaning other outcomes besides substantiated)**.The model will saves time and energy for the invesitgators from police department when they are trying to figuring out if the accused wrongdoing actually occured. If the outcome of a prediction gives you 1 (substantiated), then you know that the allegation you have is likely to be true and with careful investigation, the allegation might be substantiated with the given charges.
- After buliding the model, I will carefully examine its effectiveness and moral implications. That is being said that a good model designed to solve attempted probelm should be good at accomplishing two goals: First, my model must reach certain amount of precision and accuracy with respect to different subpopulations with which my model is trying to predict. Second, it should be and must be fair enough. I will do pairty test on our model and see if the model is lack of fairness or bias against certain subgroups. Should the model turns out to be not fair enough, additional processes are needed to make sure that people whose welfare depends on the ourcome of this model, are being treated fairly and responsibly.
- To be more specific, the metrics to evaluate the performance of our model are accuracy,recall, specificity, precison as well as f1 score. On the other hand, in oder to evaluate the fairness of my model, applications of parity test and permutation test will play important roles in our assessment. The result will be demonstrated below.
- my objective for the final model is about 82 percent overall accuracy with high specificity. My final model will reach this accuracy.

## Baseline Model

- In baseline Model, I am using 8 features in total
- The **nominal features** I used for building our model are: `'rank_incident'`, `'mos_ethnicity'`, `'mos_gender'`, `'complainant_ethnicity'`, `'complainant_gender'`, and `'fado_type'`. Because those variables are nominal, I will use one-hot encoder to encode them in the pipeline.
- The **quantitative feature** I used on buliding the model are `'mos_age_incident'` and `complainant_age_incident'`. I will leave it as is and use them directly in our prediction.
- The mean accuracy score baseline model scores only about **69%**, which is ill-performed. Clearly, I need some improvements on the feature selection process and more useful feature engineering on my models, and in later part I plan to use grid search to find the best perameters combination for my model.
- *I get 5173 true negative, 1031 false positive, 1412 false negative, and 654 true positive (might change if re-run). **The outcome here is very concerning as both false positive and false negative count is high and true positive is small.**
- In **false positive case**, if the allegation outcome is not substantiated(0), but my model lebels the allegation as 1 (Substantiated), this would cause the officer to be punished when he or she did not do wrong.
- In **false negative case**, if the allegation outcome is substantiated(1), but my model lebels the allegation as 0 (not substatantiated), this would cause the wrongdoing officer not to be punished when he or she did do something wrong.
- *In conlusion, bad performance on predicting certain outcome and lack of overall accuracy tells us that this baseline model needs improvements. Below are the evaluation metrics being used: Accuracy Score: 0.704; Recall:0.3267; Specificity:0.8288 Precision:0.387. False Discovery Rate:0.613.

### Final Model

- **Feature Added**:
  - I would like to utilize more variables from our dataset; thus I choosed the feature `year_received` to add into the model. I believe that this features would capture some characteristics that influence the allegation outcome. In certain year, depending on the judge, would substantiated more allegations; whereas in other year, the judge is more strict and would substatiated less leegations.

- In the final Model, we **engineered** on two new features:
  - **time_elapsed**: calculate the time difference between case received and case closed in years using 'year_closed', 'month_closed','year_received','month_received'. I believe that the time between the case received and closed is an extremly important factor that influences the outcome. Substantiated cases might take less time to determine, whereas the cases that are failed to substantiated might take longer to examine the detail and make decision.
  - I also take logarithm on complainant_age_incident** to normalized complainants' age distribution, since the original data is skewed and might have some variation in range
- By try different model type and fit it into our pipeline, I conclude that **RandomForestClassifier** is the best model in the case as it ended up with the highest accuracy score among all the classifier I tested (*DecisionTreeClassifier: 0.750, RandomForestClassifier: 0.808, KNeighborsClassifier: 0.774,LogisticRegression:0.762,svc:0.765,GradientBoostingClassifier:0.772,LGBMClassifier:0
- *By doing **grid search**, the best parameters combination is `tree__max_depth': None,'tree__min_samples_leaf': 1, 'tree__min_samples_split': 2` , `tree_n_estimators=200` .
- The overall performance increases in my final model. Here are the statistics we used to assess our model: Accuracy score: 0.82; Specificity:0.953; Precision:0.741; FDR；0.258.

## Fairness Evaluation

- The prediction accuracies must be the same when you are trying to make a prediction for different subgroups of the entire populations
- With this key interest in mind, let's examine the fairness issue by looking at the below question: Does the accuracy depend on complainants' age ? I will solve this problem by doing a permutation tests. In this question, we define two age groups. One is called "Old" and the other is called "Young". The cut line between two group is 21 years old. If you are older than 21, then you are being regarded as "Old", but if you are younger than 21, then you are "Young".
- Null hypothesis: Complainants from different age groups will experience same model accuracies from its prediction. Alternative hypothesis: Complainants from different age groups will experience different model accuracies from its prediction.

- The P value of permutation test is 0.1446, which suggests that,no matter what group does certain allegation belong to, my model always gives roughly the same prediction accuarcy,meaning that my model is fair for the complainants came from different age groups.

### Demo

The demo part is at the end of my report, which, basically, is a demonstration of the appliactions of my predictive model with emphasis on its real world implications.

# Code

In [30]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import re

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
```

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution fig
ures
```

In [31]:

```
allegations
```

Out[31]:

| | unique_mos_id | first_name | last_name | command_now | shield_no | complaint_id | month_re |
|---|---|---|---|---|---|---|---|
| **0** | 10004 | Jonathan | Ruiz | 078 PCT | 8409.0 | 42835 | |
| **1** | 10007 | John | Sears | 078 PCT | 5952.0 | 24601 | |
| **2** | 10007 | John | Sears | 078 PCT | 5952.0 | 24601 | |
| **3** | 10007 | John | Sears | 078 PCT | 5952.0 | 26146 | |
| **4** | 10009 | Noemi | Sierra | 078 PCT | 24058.0 | 40253 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **33353** | 9992 | Tomasz | Pulawski | 078 PCT | 2642.0 | 35671 | |
| **33354** | 9992 | Tomasz | Pulawski | 078 PCT | 2642.0 | 35671 | |
| **33355** | 9992 | Tomasz | Pulawski | 078 PCT | 2642.0 | 35671 | |
| **33356** | 9992 | Tomasz | Pulawski | 078 PCT | 2642.0 | 35671 | |
| **33357** | 9992 | Tomasz | Pulawski | 078 PCT | 2642.0 | 35671 | |

33358 rows × 27 columns

### To construct the classifier, I have to impute the missingness:

- Drop the null values if the number of missingness is relative small (under 1000):
  - allegation
  - precinct
  - contact_reason
  - outcome_description
- categorical variable (command_at_incident, complainant_ethnicity, complainant_gender):
  - making another category ('missing') to fill the missing values
- quantative variables (complainant_age_incident):
  - probabilistic Imputation to perserve the original mean and variance

In [32]:
```
allegations = allegations.dropna(subset = ['allegation', 'precinct','contact_reason','outcome_description'])
```

In [33]:
```
# categorical variables fillna with missing
allegations = allegations.assign(**{"command_at_incident": allegations['command_at_incident'].fillna('missing'),
                                 'complainant_ethnicity': allegations['complainant_ethnicity'].fillna('missing'),
                                 'complainant_gender': allegations['complainant_gender'].fillna('missing')})
```

In [34]:
```
allegations['complainant_age_incident'] = allegations['complainant_age_incident'].apply(lambda x: x if x > 0 else np.nan)
```

In [35]:
```
# use probabilistic to impute quantative variables
# number of nulls
num_null = allegations['complainant_age_incident'].isnull().sum()
# draw fill vals from distribution
fill_values = allegations['complainant_age_incident'].dropna().sample(num_null, replace=True)
# align the index, which is missing?
```

```
fill_values.index = allegations.loc[allegations['complainant_age_incide
nt'].isnull()].index
allegations = allegations.fillna({'complainant_age_incident': fill_valu
es.to_dict()})  # fill the vals
```

In [36]:
```
# Fill nan value of the remaining categorical variables
allegations=allegations.fillna("Missing")
```

To predict wheather the case is substanstiated or not, we replace board_disposition by 1 if that case is substnatitated, else 0.

In [37]:
```
allegations['board_disposition'] = allegations['board_disposition'].app
ly(lambda x: 1 if x == 'Substantiated' else 0)
```

### Baseline Model

In [38]:
```
# one-hot encode
one_hot_feat = ['rank_incident', 'mos_ethnicity','mos_gender',\
                'complainant_ethnicity','complainant_gender','fado_typ
e']
age_feat = ['mos_age_incident','complainant_age_incident']
preproc = ColumnTransformer(transformers=[('one-hot', OneHotEncoder(spa
rse=False,handle_unknown='ignore'), one_hot_feat),
                                          ('age', FunctionTransform
er(lambda x:x), age_feat)])
pl = Pipeline([
            ('preprocessor', preproc),
            ('tree', DecisionTreeClassifier())
])

# features
X = allegations.drop('board_disposition', axis=1)
# outcome
y = allegations.board_disposition
```

```
# split train/test set
X_train, X_test, y_train, y_test = train_test_split(X, y)
pl.fit(X_train, y_train)
display(pl.score(X_train, y_train))
display(pl.score(X_test, y_test))
```

```
0.912652666371075
```

```
0.7045949214026602
```

In [39]:

```
from sklearn import metrics
```

In [40]:

```
# evaluate the model
preds = pl.predict(X_test)

display('Proportion of predictions that are correct: {}'.format(metrics
.accuracy_score(y_test, preds)))
print('Confusion_matrix(Counts of TN/FP/FN/TP): \n{}'.format(metrics.co
nfusion_matrix(y_test, preds)))
display('Recall score (TP/P): {}'.format(metrics.recall_score(y_test, p
reds)))
print('Specificity score (TN/N): {}'.format(metrics.recall_score(y_test
, preds, pos_label=0)))
display('Precision: {}'.format(metrics.precision_score(y_test, preds)))
print('FDR: {}'.format(1 - metrics.precision_score(y_test, preds)))
recall=metrics.recall_score(y_test, preds)
precision=metrics.precision_score(y_test, preds)
F1=2*((recall)*(precision))/(recall+precision)
print('F1 score: {}'.format(F1))
```

```
'Proportion of predictions that are correct: 0.7045949214026602'
```

```
Confusion_matrix(Counts of TN/FP/FN/TP):
[[5173 1031]
 [1412  654]]
```

```
'Recall score (TP/P): 0.3165537270087125'
```

```
Specificity score (TN/N): 0.8338168923275306
```

```
'Precision: 0.3881305637982196'
```

Precision: 0.3881305637982196

FDR: 0.6118694362017805
F1 score: 0.3487070114636097

### Final Model-- Selections and investigations

Feature Engineering:

- **time_elapsed**: calculate the time difference between case received and case closed in years using 'year_closed', 'month_closed','year_received','month_received'.
- take logarithm on both **mos_age_incident and complainant_age_incident** to normalized both the officers and complainants' age distribution.

In [41]:
```
# First try
def final_model(allegations, model):

    # feature_engineering one
    time_feat = ['year_closed', 'month_closed','year_received','month_r
eceived']
    def time_elapsed(df):
        return (df['year_closed']+df['month_closed']/12 - df['year_rece
ived']-df['month_received']/12).to_frame()

    # one hot features
    one_hot_feat = ['mos_ethnicity','complainant_ethnicity','fado_type'
,\
                    'outcome_description','precinct','year_closed']
    # taking log
    log_feat = ['mos_age_incident','complainant_age_incident']

    # column transform
    # with feature_engineering two, logging officers' and complainants'
ages to normalized the data
    preproc = ColumnTransformer(transformers=[('one-hot', OneHotEncoder
(sparse=False,handle_unknown='ignore'), one_hot_feat),
                                              ('log', FunctionTransform
```

```
er(lambda x:np.log(x)), log_feat),
                                                    ('time_elapsed',FunctionTr
ansformer(time_elapsed), time_feat)])
    pl = Pipeline([
            ('preprocessor', preproc),
            ('tree', model)])

    # features
    X = allegations.drop({'board_disposition'}, axis=1)
    # outcome
    y = allegations.board_disposition

    # split train/test set
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    pl.fit(X_train, y_train)

    #.score Return the mean accuracy on the given test data and labels.
    display('----------{}----------'.format(str(model)))
    display('Mean accuracy in train data : {:.3f}'.format(pl.score(X_tr
ain, y_train)))
    display('Mean accuracy in test data : {:.3f}'.format(pl.score(X_tes
t, y_test)))

final_model(allegations,RandomForestClassifier())
```

```
'----------RandomForestClassifier()----------'

'Mean accuracy in train data : 0.951'

'Mean accuracy in test data : 0.790'
```

Below is the most optimal combinations of features. Believe me, I have been tried an hour to get this combination. I deleted features that are not so useful in improving the accuracy of my model and add some more one-hot encoded features, like allegations and contact reasons.

In [178]:

```
# Best combination
def final_model(allegations, model):

    # feature_engineering one
```

```
    time_feat = ['year_closed', 'month_closed','year_received','month_r
eceived']
    def time_elapsed(df):
        return (df['year_closed']+df['month_closed']/12 - df['year_rece
ived']-df['month_received']/12).to_frame()

    # one hot features
    one_hot_feat = ['fado_type',\
                    'outcome_description','precinct','year_closed','con
tact_reason',"allegation"]
    # taking log
    log_feat = ['complainant_age_incident']

    # column transform
    # with feature_engineering two, logging officers' and complainants'
ages to normalized the data
    preproc = ColumnTransformer(transformers=[('one-hot', OneHotEncoder
(sparse=False,handle_unknown='ignore'), one_hot_feat),
                                              ('log', FunctionTransform
er(lambda x:np.log(x)), log_feat),
                                             ('time_elapsed',FunctionTr
ansformer(time_elapsed), time_feat)])
    pl = Pipeline([
            ('preprocessor', preproc),
            ('tree', model)])

    # features
    X = allegations.drop({'board_disposition'}, axis=1)
    # outcome
    y = allegations.board_disposition

    # split train/test set
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    pl.fit(X_train, y_train)

    #.score Return the mean accuracy on the given test data and labels.
    display('----------{}----------'.format(str(model)))
    display('Mean accuracy in train data : {:.3f}'.format(pl.score(X_tr
ain, y_train)))
    display('Mean accuracy in test data : {:.3f}'.format(pl.score(X_tes
```

```
t, y_test)))

final_model(allegations,RandomForestClassifier(n_jobs=-1))
```

```
'----------RandomForestClassifier(n_jobs=-1)----------'

'Mean accuracy in train data : 0.992'

'Mean accuracy in test data : 0.808'
```

**Search for the best model and parameters using the pipeline.**

- DecisionTreeClassifier
- RandomForestClassifier
- KNeighborsClassifier
- LogisticRegression
- SupportVectorMachine
- XGBbooster
- lightGBM

In [139]:

```
models = [DecisionTreeClassifier(),RandomForestClassifier(),KNeighborsC
lassifier(),LogisticRegression(max_iter=2000),svm.SVC(),GradientBoostin
gClassifier(),LGBMClassifier()]
for i in models:
    final_model(allegations, i)
```

```
'----------DecisionTreeClassifier()----------'

'Mean accuracy in train data : 0.991'

'Mean accuracy in test data : 0.750'

'----------RandomForestClassifier()----------'

'Mean accuracy in train data : 0.991'

'Mean accuracy in test data : 0.817'

'----------KNeighborsClassifier()----------'
```

```
'Mean accuracy in train data : 0.855'

'Mean accuracy in test data : 0.774'

'----------LogisticRegression(max_iter=2000)----------'

'Mean accuracy in train data : 0.768'

'Mean accuracy in test data : 0.762'

'----------SVC()----------'

'Mean accuracy in train data : 0.767'

'Mean accuracy in test data : 0.765'

'----------GradientBoostingClassifier()----------'

'Mean accuracy in train data : 0.772'

'Mean accuracy in test data : 0.772'

'----------LGBMClassifier()----------'

'Mean accuracy in train data : 0.803'

'Mean accuracy in test data : 0.774'
```

By comparing the mean accuracy (pl.score) in test data, we found that **Random forest classifier** has the hightest mean accuracy among seven classifiers, while the DecisionTreeClassifier has lowerest mean accuracy scores.

#### Search for the best model and parameters using the pipeline and using Random forest classifier.

- Trying every combination ('grid search'):
  - `'tree__max_depth'` : [20, None]
  - `'tree__min_samples_leaf'` : [1, 2, 3]
  - `'tree__min_samples_split'` : [1,2,3]

- `"tree__n_estimators"`:[100,200]

In [147]:
```
from sklearn.model_selection import GridSearchCV
# feature_engineering one
time_feat = ['year_closed', 'month_closed','year_received','month_recei
ved']
def time_elapsed(df):
    return (df['year_closed']+df['month_closed']/12 - df['year_receive
d']-df['month_received']/12).to_frame()

# one hot features
one_hot_feat = ['fado_type',\
                    'outcome_description','precinct','year_closed','con
tact_reason',"allegation"]
# taking log
log_feat = ['complainant_age_incident']

# column transform
# with feature_engineering two, logging officers' and complainants' age
s to normalized the data
preproc = ColumnTransformer(transformers=[('one-hot', OneHotEncoder(spa
rse=False,handle_unknown='ignore'), one_hot_feat),
                                            ('log', FunctionTransformer
(lambda x:np.log(x)), log_feat),
                                            ('time_elapsed',FunctionTra
nsformer(time_elapsed), time_feat)])
pl = Pipeline([
        ('preprocessor', preproc),
        ('tree', RandomForestClassifier())])

# features
X = allegations.drop({'board_disposition'}, axis=1)
# outcome
y = allegations.board_disposition

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

# different combinations
```

```
parameters = {
    'tree__max_depth': [20,None],
    'tree__min_samples_leaf': [1,2,3],
    'tree__min_samples_split': [1,2,3],
    "tree__n_estimators":[100,200]
}


clf = GridSearchCV(pl, parameters, cv=3,n_jobs=-1)
clf.fit(X_train, y_train)

# shows the best parameters combinations
display('best parameters')
clf.best_params_
```

```
'best parameters'
```

Out[147]:
```
{'tree__max_depth': None,
 'tree__min_samples_leaf': 1,
 'tree__min_samples_split': 2,
 'tree__n_estimators': 200}
```

Afetr obtaining the best parameters, let's evaluate the performance.

In [148]:
```
# calculating the new
print(clf.score(X_train,y_train))
print(clf.score(X_test,y_test))
```

```
0.9915756378733525
0.820677146311971
```

In [149]:
```
# evaluate the model
preds = clf.predict(X_test)

display('Proportion of predictions that are correct: {}'.format(metrics
.accuracy_score(y_test, preds)))
print('Confusion_matrix(Counts of TN/FP/FN/TP): \n{}'.format(metrics.co
nfusion_matrix(y_test, preds)))
display('Recall score (TP/P): {}'.format(metrics.recall_score(y_test, p
```

```
reds)))
print('Specificity score (TN/N): {}'.format(metrics.recall_score(y_test
, preds, pos_label=0)))
display('Precision: {}'.format(metrics.precision_score(y_test, preds)))
print('FDR: {}'.format(1 - metrics.precision_score(y_test, preds)))
recall=metrics.recall_score(y_test, preds)
precision=metrics.precision_score(y_test, preds)
F1=2*((recall)*(precision))/(recall+precision)
print('F1 score: {}'.format(F1))
```

```
'Proportion of predictions that are correct: 0.820677146311971'

Confusion_matrix(Counts of TN/FP/FN/TP):
[[5946  293]
 [1190  841]]

'Recall score (TP/P): 0.414081733136386'

Specificity score (TN/N): 0.9530373457284821

'Precision: 0.7416225749559083'

FDR: 0.25837742504409167
F1 score: 0.5314375987361769
```

### Comments on the final Evaluation of my final model

The specificity of my model is high, which is good. Because the main purpose of the model is to tell the judges whether he or she could have reason to believe that the charged officers is innocent or could eventually be exonerated. If the model tells that the charge is likely to be substantiated, the judge should not believe the outcome right away. Instead, a careful investigation should be conducted to given compliant. The NYPD and court should also take police officers' interest into consideration rather than believe the model blindly. After all, it only serve a suggestive purpose rather than telling the judge what to do.

## Fairness Evaluation

In [150]:
```python
# Visualize the demongraphic parity
X_test=X_test.reset_index(drop=True)
X_test["is_young"]=(allegations["mos_age_incident"]<= 21).replace({True
:'young', False:'old'})
X_test["prediction"]=preds
y= y_test.reset_index(drop=True)
X_test['tag'] = y
```

In [151]:
```python
# Accuracy Parity

(
    X_test
    .groupby('is_young')
    .apply(lambda x: metrics.accuracy_score(x.tag, x.prediction))
    .rename('accuracy')
    .to_frame()
)
```

Out[151]:

| | accuracy |
|---|---|
| **is_young** | |
| **old** | 0.821063 |
| **young** | 0.875000 |

In [152]:
```python
obs = X_test.groupby('is_young').apply(lambda x: metrics.accuracy_score
(x.tag, x.prediction)).diff().iloc[-1]

metrs = []
for _ in range(10000):
    s = (
        X_test[['is_young', 'prediction', 'tag']]
        .assign(is_young=X_test.is_young.sample(frac=1.0, replace=False
).reset_index(drop=True))
        .groupby('is_young')
        .apply(lambda x: metrics.accuracy_score(x.tag, x.prediction))
        .diff()
```

```
        .iloc[-1]
    )
    metrs.append(s)
```

In [153]:
```
print(pd.Series(metrs <= obs).mean())
```

```
0.4858
```

# Working Demo

Now, we can actually build a predictive model that can "guess" whether the officer who are charged with wrongdoing can actually be exnonerated or not. This is useful because each month, the police department in NY city is very likely to receive large amount of compliants, and therefore it is impossible to carefully investigate each and every filed cases. Sometimes the lack of evidence or any form of injustices could possibly sabotage or somehow affect the outcome of the investigation, which eventually could be against either complainants' or officer's interest.

Think about it. We do not want the innocent officers being punished by fabricated or at least exagerrated allegations but also we want the officers who actually harm the citizens to be procecuted to full extent of the law. What if there is an algorithm that can help the judges or investigators of misdemeanors to determine the possible outcome of the certain complaint? Since such algorithm is not 100 correct but it is fairly accurate enough, the output of the model could therefore serve as a reliable advisor telling the possibilities of certain outcome based on empirical observations.

This could save the investigators time and energy, allowing them spend more time on more imperative matters.All they have do is enter some key info of the complaint, which is a quite esay thing to do.

In [174]:
```
# Application
def allegations_judge(info):
    outcome=clf.predict(info)
    if outcome==1:
        return "More investigation is required !"
```

```
    else:
        return "Likely a false allegation.Treat it with less prioity"
```