NYPD Allegations

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

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

Summary of Findings

Introduction

*The **classification model** I will be focusing on in this project is "Predicting the Officer ethnicity using multiple vairables". The chosen target variable is complainant_ethnicity and the evaluation martic I will be using in this question is **macro-F1 Score**, given that our data is unbalanced in terms of complainant ethnicity, I will not use micro-F1 score. In fact, in multi-categorical problems, **accuracy** will always equal to micro-F1 and will be influenced by inbalanced data, which is why I chose **macro-F1 Score** instead of **accuracy**. However, I will also include the result using **accuracy** for comparison.

(The contents below are the description of dataset, similar to project 3)

The dataset consists of more than 12,000 civilian complaints filed against New York City police officers. The New York City's Civilian Complaint Review Board provided with records about closed cases for every police officer still on the force as of late June 2020 who had at least one substantiated allegation against them. The records span decades, from September 1985 to January 2020.

Each record in the data lists the name, rank, shield number, and precinct of each officer as of today and at the time of the incident; the age, race and gender of the complainant and the officer; a category describing the alleged misconduct; and whether the CCRB concluded the officers' conduct violated NYPD rules.

observations: 33358

number of variables: 27

variable names: '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'

some describtion of variables:

1. fado_type: Top-level category of complaint, consists with 4 categories (Offensive Language, Discourtesy, Abuse of Authority, Force) and more specific complaints within each category.
2. allegation: Specific category of complaint in each fado_type
3. board_disposition: corresponds to the outcome_description, it is the final investigation result by the CCRB, which consists with 3 outcomes(Substantiated, Exonerated, Unsubstantiated)

In this project, I will focus on predicting the ethnicity of officer. In this case, I will use the findings and adapt some of the cleaning approaches from project 3.

Cleaning and EDA

The general cleaning of this project is to create a dataset that is suitable for our prediction model, therefore, I will attach some of the findings from project 3:

1. Even though there are fluctuation, **white officer** are the majority across all years, and **black complainant** are the majority. There are more **Asian officers** while very few complainant. Among both groups, **Native American** are the fewest.
2. There is still a lot **NaN values in complainants' ethnicity** in each year that we need to deal with.
3. The amount of cases across years are different, namely, more recent years have more cases, which reach its peak around 2015 - 2016.
4. **All complainant ethnicity are missing for year_closed < 2000**

Therefore, the cleaning in this project consist of the following:

1. Drop all rows with missingness in complainants' ethnicity, given that it is our target variable, which is not allowed to have NaN or Unknown ethnicity.(Additionally, all data for year_closed < 2000 will and refused, unknown ethnicity will be dropped, which will become an inevitable flaw for this project)
2. ["board_disposition"] will be summarized to 3 types (Substantiated, Exonerated, Unsubstantiated)
3. ["unique_mos_id"], ["first_name"]， ["last_name"], ["shield_no"], ["complaint_id"] will also be dropped, given that they may improve our model, but will certainly decrease the ability to generalize.
4. Keep ["rank_abbrev_now"] instead of ["rank_now"]; ["rank_abbrev_incident"] instead of ["rank_incident"].

Baseline Model

In this section, I will chose DecisionTreeClassifier and macro-f1 score as evaluation metric. The reason why chosing macro-f1 score is that my data is unbalanced in terms of complainant ethnicity, while micro-f1 score is essentially equal to accuracy, in this case macro-f1 score is my best choice.

number of features: 17 number of quantitative, ordinal, and nominal features: 4, 11, 3

Perfomance: My conclusion is that my baseline model performed badly, F1 score is around 0.85 and accuracy is around 0.87. In general, my baseline model is not accuracy and having too many incorrectly classified cases and there is a bigger space to improve.

Final Model

(Same contents included at the end of section) In this section, I trained my model using the following variables, categorical and numerical specifically:

['command_now', 'command_at_incident', 'rank_abbrev_incident','rank_abbrev_now']

['year_received', 'year_closed', 'time', 'time_binariz']

The two new features I constructed is ['time'] and ['time_binariz']. ['time'] variable is constructed by calculating the duration of executing a case based on day,month,year of received and closed. ['time_binariz'] variable is constructed based on binarizing time variable with the thershold of 365. For categorical variables, I used one-hot encoding to transform them, and for numerical variables, I used standardize to transform them. My assumption is that time of dealing with cases can associated with officers' ethnicity.

The reason why I have chosen the above variables is based on the attribution of each after I trained my models, I found out that the variables above are more important than others.

The model I chose for the final is RandomForestClassifier, given that it provides me better predicion and acceptable training time for classification problem. After finishing a lot of grid search work, I finally come out of the best parameters: PCA components of 0.9, RandomForestClassifier with 200 n_estimators. The final model achieved roughly 0.07 improvement in terms of F1 macro score and 0.05 of accuracy. However, the generalization ability of both my baseline model and final model are weak, achieving bad scores in cross_validation.

The graph results of scores will be included in the section.

Fairness Evaluation

In this section, I will perform permutaion test on variable time. Given that I have already binarized time using threshold 365, I will directly adapt this result column for permutation test.

Null Hypothesis: my model is fair; the precision for my two subsets(time below 365 and above) are roughly the same

Alternative Hypothesis: my model is unfair; the f1 macro for the above 365 subset is higher than the below 365 subset

I constructed the permutation test on np.permutation to generate shuffled time_binariz, then I combine each row of it to my df. I then seperate each groups (time_binariz = 0, 1) and do prediction, f1_score of each groups, and finally calculate the differences between them. I saved the difference between two f1 scores in a list for the purpose of generating distribution graph as well as calculating p-value.

The result for my permutation test is p-value = 0.096, which is not significant comparing to 0.01, therefore I cannot reject the null hypothesis to conclude that my model is unfair in terms of time_binariz

Code

In [46]:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import warnings
import time
warnings.filterwarnings("ignore")
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

In [26…

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, p

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer, StandardScaler, Binarizer
from sklearn.impute import SimpleImputer
from sklearn.metrics import f1_score, make_scorer, accuracy_score
from sklearn import metrics

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
```

Cleaning

In [12…

```
original = pd.read_csv('data/allegations_202007271729.csv')
df = original[[
    'command_now', 'month_received', 'year_received', 'month_closed',
    'year_closed', 'command_at_incident', 'rank_abbrev_incident',
    'rank_abbrev_now', 'mos_ethnicity', 'mos_gender', 'mos_age_incident',
    'complainant_ethnicity', 'complainant_gender', 'complainant_age_incident',
    'fado_type', 'allegation', 'precinct', 'contact_reason',
```

```
    'outcome_description', 'board_disposition'
]]
```

In [13…

```
def clean_race(string):
    if (string == 'Unknown') | (string == 'Other Race') | (string == 'Refused'):
        return np.nan
    else:
        return string

def clean_disposition(string):
    if ('Unsubstantiated' in string):
        return 'Unsubstantiated'
    elif ('Exonerated' in string):
        return 'Exonerated'
    else:
        return 'Substantiated'

def pivoting(col, row):
    out = df.groupby([col, row]).size().reset_index().pivot(columns=col,
                                                              index=row,
                                                              values=0)
    return out
```

In [13…

```
df.loc[:, 'complainant_ethnicity'] = df['complainant_ethnicity'].apply(clean_race)
df.loc[:, 'board_disposition'] = df['board_disposition'].apply(clean_disposition)
df = df.dropna(subset=['complainant_ethnicity']).reset_index().drop('index', axis = 1)
# comparison_column = pd.Series(np.where(df['mos_ethnicity'] == df['complainant_ethnic
# df['compare'] = comparison_column
df.head()
```

Out[131]:

	command_now	month_received	year_received	month_closed	year_closed	command_at_incident
0	078 PCT	7	2019	5	2020	078 PCT
1	078 PCT	11	2011	8	2012	PBBS
2	078 PCT	11	2011	8	2012	PBBS
3	078 PCT	7	2012	9	2013	PBBS
4	078 PCT	5	2017	10	2017	078 PCT

Baseline Model

In [13…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26917 entries, 0 to 26916
Data columns (total 20 columns):
```

```
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   command_now               26917 non-null  object 
 1   month_received            26917 non-null  int64  
 2   year_received             26917 non-null  int64  
 3   month_closed              26917 non-null  int64  
 4   year_closed               26917 non-null  int64  
 5   command_at_incident       26789 non-null  object 
 6   rank_abbrev_incident      26917 non-null  object 
 7   rank_abbrev_now           26917 non-null  object 
 8   mos_ethnicity             26917 non-null  object 
 9   mos_gender                26917 non-null  object 
 10  mos_age_incident          26917 non-null  int64  
 11  complainant_ethnicity     26917 non-null  object 
 12  complainant_gender        26899 non-null  object 
 13  complainant_age_incident  26574 non-null  float64
 14  fado_type                 26917 non-null  object 
 15  allegation                26917 non-null  object 
 16  precinct                  26903 non-null  float64
 17  contact_reason            26788 non-null  object 
 18  outcome_description       26884 non-null  object 
 19  board_disposition         26917 non-null  object 
dtypes: float64(2), int64(5), object(13)
memory usage: 4.1+ MB
```

In [13…

```
df = df.astype({"precinct": object}) # for precinct should be categorical
df["precinct"]= df["precinct"].apply(str)
df["precinct"] = df["precinct"].replace('nan','NULL')
df = df.replace(float('nan'), np.nan)
```

In [13…

```
X = df.drop(['month_closed', 'month_received','mos_ethnicity'], axis=1)
y = df['mos_ethnicity']
```

In [13…

```
types = X.dtypes
catcols = types.loc[types == np.object].index
numcols = types.loc[types != np.object].index
```

In [13…

```
catcols, numcols
```

Out[136]:

```
(Index(['command_now', 'command_at_incident', 'rank_abbrev_incident',
        'rank_abbrev_now', 'mos_gender', 'complainant_ethnicity',
        'complainant_gender', 'fado_type', 'allegation', 'precinct',
        'contact_reason', 'outcome_description', 'board_disposition'],
       dtype='object'),
 Index(['year_received', 'year_closed', 'mos_age_incident',
        'complainant_age_incident'],
       dtype='object'))
```

In [13…

```
len(catcols)
```

Out[137]:

```
13
```

In [13…

```
cats = Pipeline([
    ('imp', SimpleImputer(strategy='constant', fill_value = 'NULL')),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
#    ('pca', PCA(svd_solver='full', n_components=0.99))
])

nums = Pipeline(steps=[
```

```
#     ('scaler', pp.StandardScaler())
    ('impute', SimpleImputer(strategy='constant', fill_value = 0))
])

ct = ColumnTransformer([
    ('numcols', nums, numcols),
    ('catcols', cats, catcols)
])

pl = Pipeline([('preprocessor', ct), ('clf', DecisionTreeClassifier())])
```

In [13…

```
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
```

In [14…

```
pl.fit(X_tr, y_tr)
pl.score(X_ts, y_ts) # this will calculate accuracy for my baseline model, same as acc
```

Out[140]: 0.8845468053491827

In [14…

```
preds = pl.predict(X_ts)
f1_score(y_ts, preds, average='macro')  # this will calculate macro-F1 score
```

Out[141]: 0.8517904506921973

In [14…

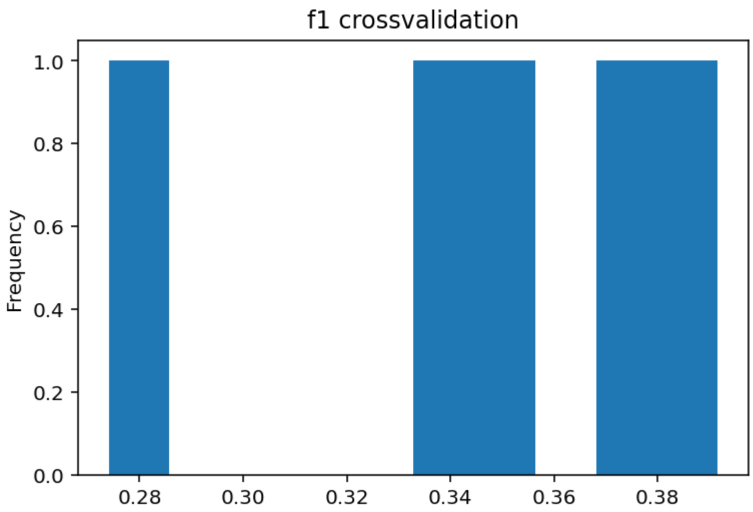
```
cross_f1_score = cross_val_score(pl, X, y, cv=5, scoring='f1_macro')
cross_accuracy_score = cross_val_score(pl, X, y, cv=5)
cross_f1_score, cross_accuracy_score
```

Out[142]:

```
(array([0.35277404, 0.33610582, 0.37275036, 0.39160766, 0.27402965]),
 array([0.47901189, 0.35586924, 0.28515698, 0.37934237, 0.40627903]))
```

In [14…

```
pd.Series(cross_f1_score).plot(kind='hist', title='f1 crossvalidation');
```



In [ ]:

```
pd.Series(cross_accuracy_score).plot(kind='hist', title='accuracy crossvalidation');
```

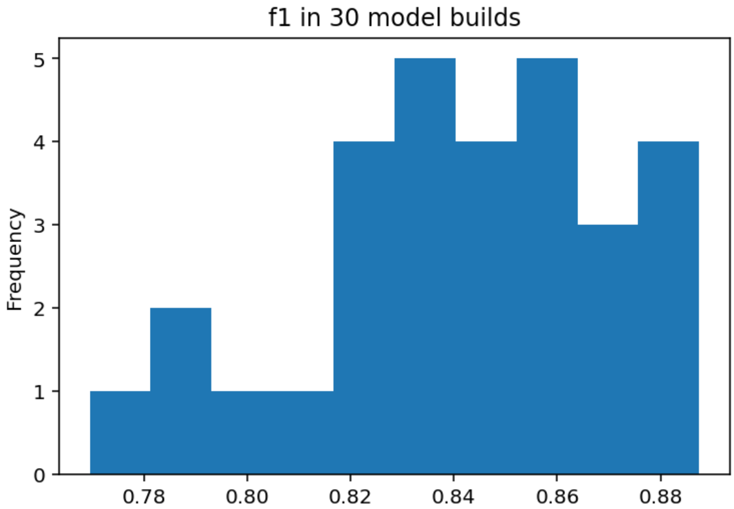
In [14…

```
accuracy = []
f1 = []
for _ in range(30):
    X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
```

```
    pl.fit(X_tr, y_tr)
    preds = pl.predict(X_ts)
    accuracy.append(pl.score(X_ts, y_ts))
    f1.append(f1_score(y_ts, preds, average='macro'))
```

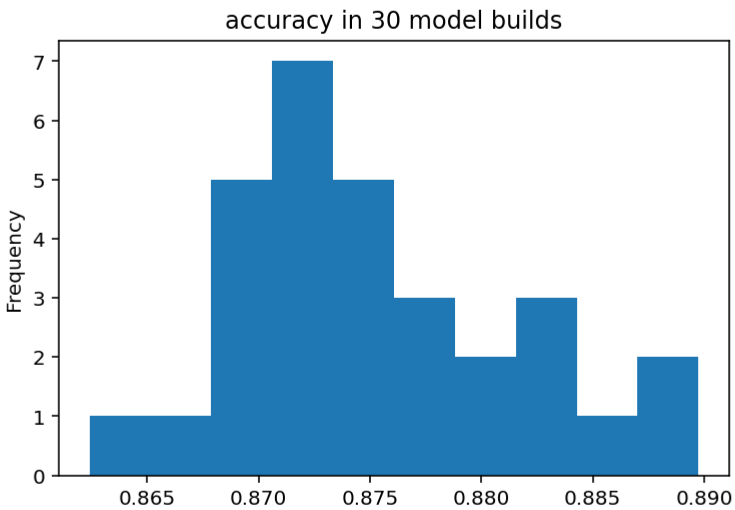
In [14…

```
pd.Series(f1).plot(kind='hist', title='f1 in 30 model builds');
```



In [14…

```
pd.Series(accuracy).plot(kind='hist', title='accuracy in 30 model builds');
```



In [14…

```
dict(zip(X.columns, pl.steps[1][1].feature_importances_))
```

Out[146]:

```
{'command_now': 0.042556919636000785,
 'year_received': 0.035626766898184024,
 'year_closed': 0.06419408935265851,
 'command_at_incident': 0.04006438804114215,
 'rank_abbrev_incident': 0.0002675426613749244,
 'rank_abbrev_now': 0.001339254738567179,
 'mos_gender': 0.00014658173671803956,
 'mos_age_incident': 0.00041638532329781616,
 'complainant_ethnicity': 0.0011818638260805321,
 'complainant_gender': 0.0,
 'complainant_age_incident': 0.0008106039873672822,
 'fado_type': 0.0011161520248649127,
 'allegation': 0.002931301287964808,
 'precinct': 0.0,
 'contact_reason': 0.0,
```

```
 'outcome_description': 0.00034127818304219277,
 'board_disposition': 0.0003240508763168184}
```

Final Model

First I creat a column recording the length of deal time for each case.

In [16…

```
day = pd.Series(np.ones(len(df)), dtype = 'int').astype('string')
```

In [17…

```
df['date_received'] = pd.to_datetime(df["year_received"].astype("string") + "/" +
               df["month_received"].astype("string") + "/" + day)
df['date_closed'] = pd.to_datetime(df["year_closed"].astype("string") + "/" +
               df["month_closed"].astype("string") + "/" + day)
df['time'] = df['date_closed'] - df['date_received']
df['time'] = df['time'].dt.days
# df['time'] = df['time'].dt.days.apply(str)
binarizer = Binarizer(threshold=365)
t = np.array(df['time'])
df['time_binariz'] = binarizer.transform(t.reshape(-1, 1))
```

In [17…

```
X = df.drop(
    [
        'month_closed',
        'month_received',
        #     'year_closed', 'year_received',
        'date_received',
        'date_closed',
        'complainant_age_incident',
        'complainant_gender',
        'complainant_ethnicity',
        'mos_age_incident',
        'mos_gender',
        'precinct',
        'contact_reason',
        'outcome_description',
        'allegation',
        'fado_type',
        'board_disposition',
#          'time',
#          'time_binariz',
        'mos_ethnicity'
    ],
    axis=1)
y = df['mos_ethnicity']
```

In [17…

```
types = X.dtypes
catcols = types.loc[types == np.object].index
numcols = types.loc[types != np.object].index
```

In [17…

```
catcols, numcols
```

Out[173]:

```
(Index(['command_now', 'command_at_incident', 'rank_abbrev_incident',
        'rank_abbrev_now'],
       dtype='object'),
 Index(['year_received', 'year_closed', 'time', 'time_binariz'], dtype='object'))
```

In [17…

```
cats = Pipeline([
```

```
    ('imp', SimpleImputer(strategy='constant', fill_value='NULL')),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    #     ('pca', PCA(svd_solver='full', n_components=0.99))
])

nums = Pipeline(steps=[
    ('impute', SimpleImputer(strategy='constant', fill_value=0)),
    ('scaler', StandardScaler()),
    ('pca', PCA(svd_solver='full', n_components=0.90))
])

ct = ColumnTransformer([('numcols', nums, numcols),
                        ('catcols', cats, catcols)])

pl = Pipeline([
    ('preprocessor', ct),
    (
        'clf',
        RandomForestClassifier(
            n_estimators=200,
#             #random_state=90,
#             max_depth=95,
#             max_features=7
        ))])
```

In [17…

```
pl.get_params().keys()
```

Out[175]:

```
dict_keys(['memory', 'steps', 'verbose', 'preprocessor', 'clf', 'preprocessor__n_jobs', 'preprocessor__remainder', 'preprocessor__sparse_threshold', 'preprocessor__transformer_weights', 'preprocessor__transformers', 'preprocessor__verbose', 'preprocessor__numcols', 'preprocessor__catcols', 'preprocessor__numcols__memory', 'preprocessor__numcols__steps', 'preprocessor__numcols__verbose', 'preprocessor__numcols__impute', 'preprocessor__numcols__scaler', 'preprocessor__numcols__pca', 'preprocessor__numcols__impute__add_indicator', 'preprocessor__numcols__impute__copy', 'preprocessor__numcols__impute__fill_value', 'preprocessor__numcols__impute__missing_values', 'preprocessor__numcols__impute__strategy', 'preprocessor__numcols__impute__verbose', 'preprocessor__numcols__scaler__copy', 'preprocessor__numcols__scaler__with_mean', 'preprocessor__numcols__scaler__with_std', 'preprocessor__numcols__pca__copy', 'preprocessor__numcols__pca__iterated_power', 'preprocessor__numcols__pca__n_components', 'preprocessor__numcols__pca__random_state', 'preprocessor__numcols__pca__svd_solver', 'preprocessor__numcols__pca__tol', 'preprocessor__numcols__pca__whiten', 'preprocessor__catcols__memory', 'preprocessor__catcols__steps', 'preprocessor__catcols__verbose', 'preprocessor__catcols__imp', 'preprocessor__catcols__ohe', 'preprocessor__catcols__imp__add_indicator', 'preprocessor__catcols__imp__copy', 'preprocessor__catcols__imp__fill_value', 'preprocessor__catcols__imp__missing_values', 'preprocessor__catcols__imp__strategy', 'preprocessor__catcols__imp__verbose', 'preprocessor__catcols__ohe__categories', 'preprocessor__catcols__ohe__drop', 'preprocessor__catcols__ohe__dtype', 'preprocessor__catcols__ohe__handle_unknown', 'preprocessor__catcols__ohe__sparse', 'clf__bootstrap', 'clf__ccp_alpha', 'clf__class_weight', 'clf__criterion', 'clf__max_depth', 'clf__max_features', 'clf__max_leaf_nodes', 'clf__max_samples', 'clf__min_impurity_decrease', 'clf__min_impurity_split', 'clf__min_samples_leaf', 'clf__min_samples_split', 'clf__min_weight_fraction_leaf', 'clf__n_estimators', 'clf__n_jobs', 'clf__oob_score', 'clf__random_state', 'clf__verbose', 'clf__warm_start'])
```

In [17…

```
%%script echo skipping
grid = {
#     'clf__n_estimators': np.arange(0, 100, 5),
    #     'clf__n_estimators': np.arange(0, 100, 5)
    #     'clf__n_estimators': np.arange(170, 180, 1)
    'clf__max_depth': np.arange(0, 200, 10),
#     'clf__max_features': np.arange(3, 8, 1),
    #     'clf__max_depth': [140, 145, 150, 155, 160, None],
    #     'clf__min_samples_split':[2,3,5,7,10,15,20],
```

```
    #     'clf__min_samples_leaf':[5,10,20,30,40,50]
}

gridsearch = GridSearchCV(estimator=pl,
                          param_grid=grid,
                          n_jobs=-1,
                          cv=3,
                          scoring=make_scorer(f1_score, average='macro'),
                          return_train_score=True)

# now perform full fit on whole pipeline
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
gridsearch.fit(X, y)
# print("Best Model from gridsearch: {}".format(gridsearch.best_estimator_))
print("Best parameters from gridsearch: {}".format(gridsearch.best_params_))
print("CV score=%0.3f" % gridsearch.best_score_)
cv_results = gridsearch.cv_results_
# print(cv_results)
```

Couldn't find program: 'echo'

In [17…

```
%%script echo skipping
gridsearch.cv_results_.keys()
```

Couldn't find program: 'echo'

In [17…

```
%%script echo skipping
# index = gridsearch.param_grid['clf__n_estimators']
# index = gridsearch.param_grid['clf__max_depth']
index = gridsearch.param_grid['clf__max_features']

test = gridsearch.cv_results_['split0_test_score']
train = gridsearch.cv_results_['split0_train_score']
pd.DataFrame({'test': test, 'train': train}, index=index).plot()
```

Couldn't find program: 'echo'

In [17…

```
T1 = time.time()
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
pl.fit(X_tr, y_tr)
preds = pl.predict(X_ts)
print('F1: ' + str(f1_score(y_ts, preds, average='macro')))
print('Accuracy: ' + str(accuracy_score(y_ts, preds)))
T2 = time.time()
print('Time: ' + str(T2-T1))
```

```
F1: 0.9206619702532238
Accuracy: 0.9213967310549777
Time: 46.962180614471436
```

In [18…

```
cross_f1_score = cross_val_score(pl, X, y, cv=5, scoring='f1_macro')
cross_accuracy_score = cross_val_score(pl, X, y, cv=5)
```

In [18…

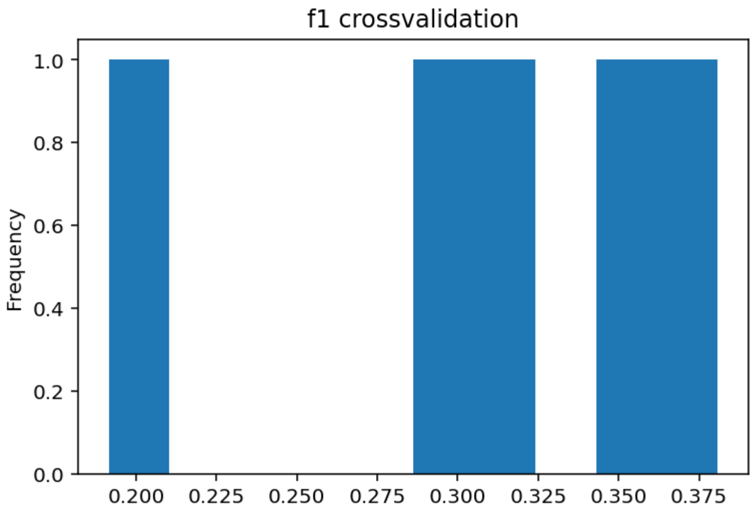
```
cross_f1_score, cross_accuracy_score
```

Out[181]:

```
(array([0.38079513, 0.29606642, 0.34526834, 0.32065439, 0.19153575]),
 array([0.5230312 , 0.38447251, 0.26769459, 0.34664685, 0.45309307]))
```

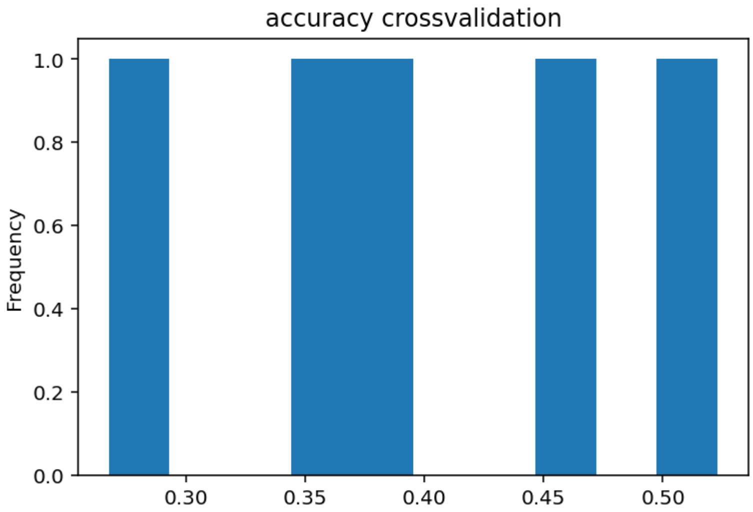
In [19…

```
pd.Series(cross_f1_score).plot(kind='hist', title='f1 crossvalidation');
```



In [19…

```
pd.Series(cross_accuracy_score).plot(kind='hist', title='accuracy crossvalidation');
```

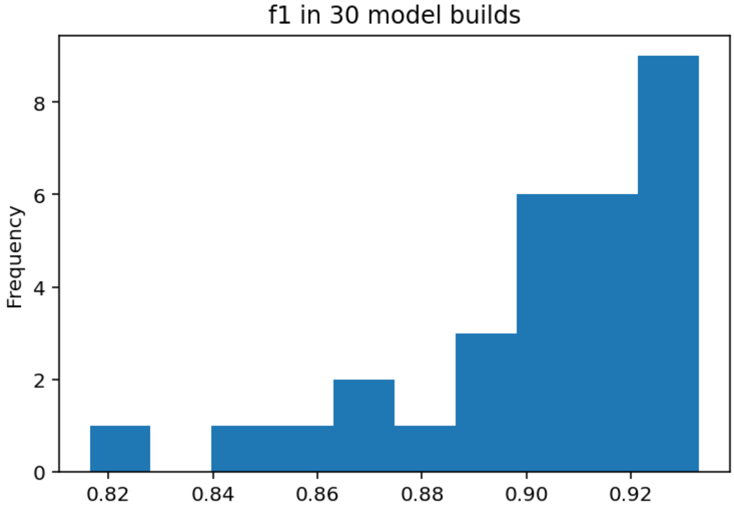


In [18…

```
accuracy = []
f1 = []
for _ in range(30):
    X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
    pl.fit(X_tr, y_tr)
    preds = pl.predict(X_ts)
    accuracy.append(pl.score(X_ts, y_ts))
    f1.append(f1_score(y_ts, preds, average='macro'))
```

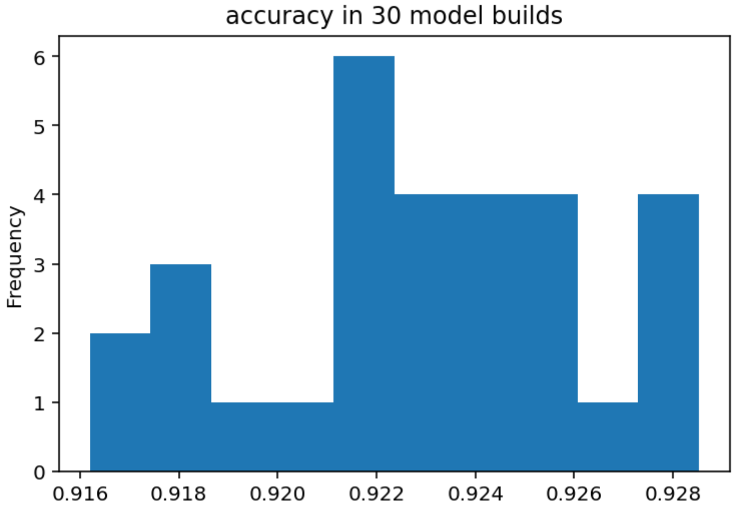
In [18…

```
pd.Series(f1).plot(kind='hist', title='f1 in 30 model builds');
```



In [18…

```
pd.Series(accuracy).plot(kind='hist', title='accuracy in 30 model builds');
```



In [18…

```
pl['clf'].get_params()
```

Out[186]:

```
{'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': 200,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [18…

```
dict(zip(X.columns, pl.steps[1][1].feature_importances_))
```

Out[187]:

```
{'command_now': 0.13285851613825075,
 'year_received': 0.13001709154584404,
```

```
 'year_closed': 0.0007685406672565792,
 'command_at_incident': 0.001083752014236308,
 'rank_abbrev_incident': 0.0006397769554489079,
 'rank_abbrev_now': 0.0003235080810626737,
 'time': 0.0006877415631164772,
 'time_binariz': 9.509996242256721e-06}
```

Summary

In this section, I trained my model using the following variables, categorical and numerical specifically:

['command_now', 'command_at_incident', 'rank_abbrev_incident', 'rank_abbrev_now']

['year_received', 'year_closed', 'time']

The two new features I constructed is ['time'] and ['time_binariz']. ['time'] variable is constructed by calculating the duration of executing a case based on day,month,year of received and closed. ['time_binariz'] variable is constructed based on binarizing time variable with the thershold of 365. For categorical variables, I used one-hot encoding to transform them, and for numerical variables, I used standardize to transform them. My assumption is that time of dealing with cases can associated with officers' ethnicity.

The reason why I have chosen the above variables is based on the attribution of each after I trained my models, I found out that the variables above are more important than others.

The model I chose for the final is RandomForestClassifier, given that it provides me better predicion and acceptable training time for classification problem. After finishing a lot of grid search work, I finally come out of the best parameters: PCA components of 0.9, RandomForestClassifier with 200 n_estimators. The final model achieved roughly 0.07 improvement in terms of F1 macro score and 0.05 of accuracy. However, the generalization ability of both my baseline model and final model are weak, achieving bad scores in cross_validation.

Fairness Evaluation

In this section, I will perform permutaion test on variable time. Given that I have already binarized time using threshold 365, I will directly adapt this result column for permutation test.

Null Hypothesis: my model is fair; the precision for my two subsets(time below 365 and above) are roughly the same

Alternative Hypothesis: my model is unfair; the f1 macro for the above 365 subset is higher than the below 365 subset

I constructed the permutation test on np.permutation to generate shuffled time_binariz, then I combine each row of it to my df. I then seperate each groups (time_binariz = 0, 1) and do prediction, f1_score of each groups, and finally calculate the differences between them. I saved the difference between two f1 scores in a list for the purpose of generating distribution graph as well as calculating p-value.

The result for my permutation test is p-value = 0.096, which is not significant comparing to 0.01, therefore I cannot reject the null hypothesis to conclude that my model is unfair in terms of time_binariz

In [29…

```
col = [
    'command_now', 'year_received', 'year_closed', 'command_at_incident',
    'rank_abbrev_incident', 'rank_abbrev_now', 'time', 'time_binariz',
    'mos_ethnicity'
]

X = df[df['time_binariz'] == 0]
Y = df[df['time_binariz'] == 1]
X = X[col]
Y = Y[col]

predict_X = pl.predict(X.drop('mos_ethnicity', axis=1))
predict_Y = pl.predict(Y.drop('mos_ethnicity', axis=1))

f1_X = f1_score(X['mos_ethnicity'], predict_X, average='macro')
f1_Y = f1_score(Y['mos_ethnicity'], predict_Y, average='macro')
f1_diff = f1_Y - f1_X

time_binariz = df['time_binariz']
```

In [28…

```
n = 1000
permutations = np.column_stack([
    np.random.permutation(time_binariz)
    for _ in range(n)
]).T.astype('bool')
```

In [29…

```
df_c = df.copy()
df_c = df_c[col]
f1_diffs = []
for i in permutations:
    X_c = df_c[i == 0]
    Y_c = df_c[i == 1]
    f1_X_c = f1_score(X_c['mos_ethnicity'],
                      pl.predict(X_c.drop('mos_ethnicity', axis=1)),
                      average='macro')
    f1_Y_c = f1_score(Y_c['mos_ethnicity'],
                      pl.predict(Y_c.drop('mos_ethnicity', axis=1)),
                      average='macro')
    f1_diffs.append(f1_X_c - f1_Y_c)
```

In [29…

```
np.count_nonzero(f1_diffs >= f1_diff) / n
```

Out[294]:

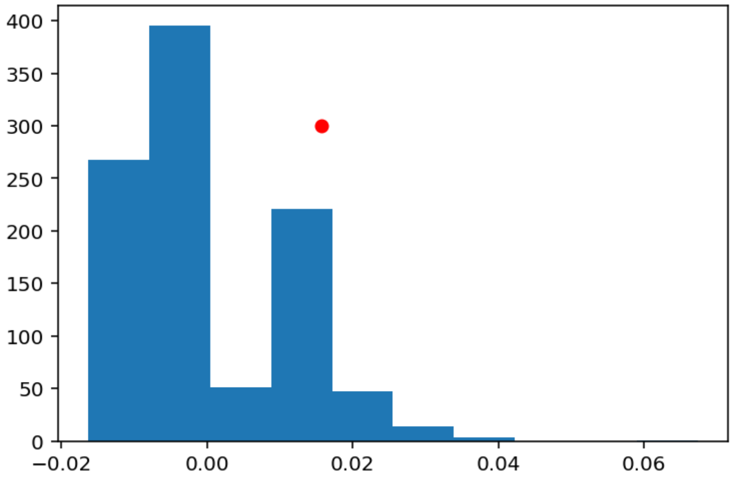
```
0.096
```

In [29…

```
plt.hist(f1_diffs)
plt.scatter(f1_diff, 300, color = 'red')
```

Out[296]:

```
<matplotlib.collections.PathCollection at 0x22b052bceb0>
```



In [ ]: