Snapchat Political Ads

- **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 reach (number of views) of an ad.
  - Predict how much was spent on an ad.
  - Predict the target group of an ad. (For example, predict the target gender.)
  - Predict the (type of) organization/advertiser behind an ad.

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

1 Introduction

1.1 Prediction problem we are attempting

We want to predict the reach(i.e Impressions) of an ad.

1.2 Classification or Regression?

This is a `regression` problem because the label is quantitative and continuous.

1.3 Choice of target variable and evaluation metric

We choose the variable `Impressions` as the target variable.

Our evaluation metric is R2. The reason we don't choose MSE/MAE/RMSE is that, it's hard to find a proper threshold to decide whether the model is good or not. They are better choice when doing comparison. However, for R2, it's easy to say it's a good model when R2 is larger than 0.6.

---

2 Baseline Model

2.1 Feature description

- The number of features (27)
  - nominal: 20
    (['ADID', 'CreativeUrl', 'OrganizationName', 'BillingAddress', 'CandidateBallotInformation', 'PayingAdvertiserName', 'Gender', 'AgeBracket', 'CountryCode', 'RegionID', 'ElectoralDistrictID', 'MetroID', 'Interests', 'OsType', 'Segments', 'LocationType', 'Language', 'AdvancedDemographics', 'Targeting Geo - Postal Code', 'CreativeProperties'])
  - quantitative: 5
    (['Spend', 'LatLongRad', 'Targeting Connection Type', 'Targeting Carrier (ISP)', 'Year'])
  - ordinal: 2
    (['StartDate', 'EndDate'])

2.2 Performance of model

R2 for the baseline model is 0.4391987093594003
I think it's a bad model. Because the larger R2 is, the better. And usually if R2 is bigger than 0.6, the model can be viewed as a good model. For this model, it has only achieved about 0.43 R2, so I think it's a bad one.

---

3 Final Model

3.1 New features added and their advantages

- **time, spend:** We have drawn a conclusion from project 3 that time duration and spend are important characteristics affecting the number of views of an advertisement.

- **CreativeUrl:** This feature is actually a new one. We get the form of file on the website from the url. We believe that the differences between forms(video, picture) will have a large effect on the Impressions.
- **age_start, age_end:** These two features are derived from AgeBracket. We create these 2 new features because the data entries in the column AgeBracket is chaotic(containing age+, age-, age-age), which we believe is meaningless. So we split entries in this column into two columns. to do this, we set the largest age to be 100 and the smallest age to be 0.
- **CreativeProperties:** This feature is actually a new one. We get domain name of url from the origin column.

Actually, using dataset with new features, we get the R2 promoted to about 0.45751476625330056 for the same model of LinearRegression.

3.2 Model type

- XGBRegressor model
- LinearRegression model
- SVR model
- DecisionTreeRegressor model
- RandomForestRegressor model
- ExtraTreesRegressor model
- GradientBoostingRegressor model
- KNeighborsRegressor model

3.3 Best parameters

> XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, importance_type='gain', learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, n_estimators=100, n_jobs=1, nthread=None, objective='reg:linear', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1)

3.4 Model selection method

We use the 5-fold cross validation method to do model selection.

---

4 Fairness Evaluation

We evaluate our model using variables we create: age_start & age_end.
We will use the mean of age_start and age_end, and the mean smaller than 40 is viewed as young and bigger then 40 is viewed as old.
After doing the permutation, we get p-value of about **0.4022**, which means that our model doesn't have a bias between the young and the old. Hence, the data is fair.

Extra(Inference)

This is the weight of every feature in the LinearRegression. We can find that some attributes contribute more than others.(Like PayingAdvertiserName contributes most to the target variable):

```
PayingAdvertiserName           1.694779e+06
CountryCode                    4.346214e+05
OsType                         3.966712e+05
Year                           3.074031e+05
AgeBracket                     1.516741e+05
Targeting Geo - Postal Code    1.360645e+05
RegionID                       1.291749e+05
CreativeUrl                    9.882583e+04
ElectoralDistrictID            5.397202e+04
Language                       4.961746e+04
Spend                          2.000108e+04
LatLongRad                    -2.085556e+04
age_end                       -2.808062e+04
ADID                          -3.633852e+04
Targeting Carrier (ISP)       -4.428166e+04
StartDate                     -4.981333e+04
Interests                     -6.378904e+04
Gender                        -1.099415e+05
time                          -1.614656e+05
CreativeProperties            -1.653596e+05
LocationType                  -2.273732e+05
Genre                         -2.905208e+05
CandidateBallotInformation    -2.992644e+05
Segments                      -3.017640e+05
OrganizationName              -3.548954e+05
AdvancedDemographics          -5.541756e+05
Type                          -7.046631e+05
MetroID                       -7.986371e+05
BillingAddress                -9.854999e+05
age_start                     -1.212433e+06
```

Code

In [45]:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import re
from sklearn.preprocessing import *

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import *

from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import *
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVR
from sklearn.tree import *
import xgboost as xgb

%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
import matplotlib
matplotlib.style.use('ggplot')
import warnings
warnings.filterwarnings('ignore')
```

In [8]:

```
# we downloaded the dataset from proj3.
#it is a combination of 2018 and 2019 datasets of political ads on snapchat.
fp = os.path.join('data', 'snapchat_ads_1819.csv')
ads = pd.read_csv(fp, encoding = "utf-8")
ads = ads.drop(['Unnamed: 0'], axis=1)

# Use datetime object to transform time
ads['StartDate'] = pd.to_datetime(ads['StartDate'])
ads['EndDate'] = pd.to_datetime(ads['EndDate'])

# Create label and data
label = ads['Impressions']
data = ads.drop('Impressions', axis=1)

data.head()
```

Out[8]:

	ADID	CreativeUrl	Spend	
0	2ac103bc69cce2d24b198e6a6d052dbff2c25ae9b6bb9e...	https://www.snap.com/political-ads/asset/69afd...	165	2
1	40ee7e900be9357ae88181f5c8a56baf6d5aab0e8d0f51...	https://www.snap.com/political-ads/asset/0885d...	17	1
2	c80ca50681d552551ceaf625981c0202589ca710d51925...	https://www.snap.com/political-ads/asset/a36b7...	60	2
3	a3106af2289b62f57f63f4fb89753bdf94e2fadede0478...	https://www.snap.com/political-ads/asset/46819...	2492	1
4	7afda4224482eb70315797966b4dcdeb856df916df5bdc...	https://www.snap.com/political-ads/asset/ee833...	5795	0

5 rows × 27 columns

Baseline Model

In [9]:

```
class rankTime(BaseEstimator, TransformerMixin):

    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        X[:, 0] = np.array(pd.Series(X[:, 0]).rank(method='first'))
        X[:, 1] = np.array(pd.Series(X[:, 1]).rank(method='first'))
        return X
```

In [10]:

```
obj_ads = data.select_dtypes(include=['object']).copy()

# Nominal
nom_ = Pipeline([
    ('imp', SimpleImputer(strategy="most_frequent")),
    ('ohe', OneHotEncoder(handle_unknown='ignore'))
])
nom = obj_ads.columns.tolist()

# Quantitative
qua_ = Pipeline([
    ('imp', SimpleImputer(strategy="most_frequent")),
    ('nor', Normalizer())
])
qua = [feat for feat in data.columns if feat not in nom and feat != 'StartDate'
and feat != 'EndDate']

# Ordinal
ordin_ = Pipeline([
    ('imp', SimpleImputer(strategy="most_frequent")),
    ('ord', rankTime()),
    ('nor', Normalizer())
])
ordin = ['StartDate', 'EndDate']

X_train, X_test, y_train, y_test = train_test_split(data, label, random_state=1)

ct = ColumnTransformer([('nom', nom_, nom), ('qua', qua_, qua), ('ordin', ordin_
, ordin)])
pl = Pipeline([('feats', ct), ('rgr', LinearRegression())])
pl.fit(X_train, y_train)
preds = pl.predict(X_test)
rmse = np.sqrt(((preds - y_test)**2).mean())
print('RMSE for predicting Impressions is ' + str(rmse))
print('R2 for predicting Impressions is ', r2_score(preds, y_test))
```

```
RMSE for predicting Impressions is 2037232.439917216
R2 for predicting Impressions is  0.4391987476707492
```

In [11]:

```
print('Number of all features:', len(data.columns))
print('Number of nominal features:', len(nom))
print('Number of quantitative features:', len(qua))
print('Number of ordinal features:', len(ordin))
```

```
Number of all features: 27
Number of nominal features: 20
Number of quantitative features: 5
Number of ordinal features: 2
```

Final Model

In [12]:

```
data.head()
```

Out[12]:

	ADID	CreativeUrl	Spend	
0	2ac103bc69cce2d24b198e6a6d052dbff2c25ae9b6bb9e...	https://www.snap.com/political-ads/asset/69afd...	165	2
1	40ee7e900be9357ae88181f5c8a56baf6d5aab0e8d0f51...	https://www.snap.com/political-ads/asset/0885d...	17	1
2	c80ca50681d552551ceaf625981c0202589ca710d51925...	https://www.snap.com/political-ads/asset/a36b7...	60	2
3	a3106af2289b62f57f63f4fb89753bdf94e2fadede0478...	https://www.snap.com/political-ads/asset/46819...	2492	1
4	7afda4224482eb70315797966b4dcdeb856df916df5bdc...	https://www.snap.com/political-ads/asset/ee833...	5795	0

5 rows × 27 columns

In [13]:

```
# Drop useless columns

# Features added
# Creative form(png/mp4/jpg)
data['CreativeUrl'] = data['CreativeUrl'].apply(lambda x: x.split('=')[-1])
# Transform agebracket to two new features: age_start & age_end
def age(age):
    if age != age:
        return '0-100'
    elif '-' in age and len(age) < 5:
        return '0-' + age.replace('-', '')
    elif '+' in age and len(age) < 5:
        return age.replace('+', '') + '-100'
    else:
        return age
age = data['AgeBracket'].apply(lambda x: age(x))
data['age_start'] = age.apply(lambda x: float(x.split('-')[0]))
data['age_end'] = age.apply(lambda x: float(x.split('-')[1]))
# Use the domain name of the url of Creative Property
def transformCreativeProp(x):
    x += '/'
    reg1 = '\.\w+\/'
    pat1_ls = re.findall(reg1, x)
    if len(pat1_ls) == 0:
        return np.nan
    pat1 = pat1_ls[0]
    reg2 = '\w+' + pat1
    pat2 = re.findall(reg2, x)[0][:-1]
    return pat2
data['CreativeProperties'] = data['CreativeProperties'].apply(lambda x: transfor
mCreativeProp(x) if not pd.isna(x) else np.nan)
# time is the time duration in hours between the start time and the end time
duration = data['EndDate'] - data['StartDate']
data['time'] = duration.apply(lambda x: x.total_seconds() // 3600)
# genre is the genre of the ads, it could be Arts, Advocates, Basketable, Advent
ure...
data['Genre'] = data['Interests'].apply(lambda x: str(x).split(' ')[0])
# type is the type of the ads, it could be New, College, Occupation, Educatio
n...
data['Type'] = data['AdvancedDemographics'].apply(lambda x: str(x).split(' ')[0
])

data.head()
```

Out[13]:

	ADID	CreativeUrl	Spend	StartDate	
0	2ac103bc69cce2d24b198e6a6d052dbff2c25ae9b6bb9e...	mp4	165	2018-11-01 22:42:22+00:00	23
1	40ee7e900be9357ae88181f5c8a56baf6d5aab0e8d0f51...	mp4	17	2018-11-15 15:52:06+00:00	15
2	c80ca50681d552551ceaf625981c0202589ca710d51925...	png	60	2018-09-28 23:10:14+00:00	02
3	a3106af2289b62f57f63f4fb89753bdf94e2fadede0478...	mp4	2492	2018-10-27 19:23:19+00:00	23
4	7afda4224482eb70315797966b4dcdeb856df916df5bdc...	mp4	5795	2018-10-25 04:00:00+00:00	23

5 rows × 32 columns

In [14]:

```
#For the baseline model, we were using one-hot encoding to collect all the categ
orical or object data
obj_ads = data.select_dtypes(include=['object']).copy()

# Nominal
nom_ = Pipeline([
    ('imp', SimpleImputer(strategy="most_frequent")),
    ('ohe', OneHotEncoder(handle_unknown='ignore'))
])
nom = obj_ads.columns.tolist()

# Quantitative
qua_ = Pipeline([
    ('imp', SimpleImputer(strategy="most_frequent")),
    ('nor', Normalizer())
])
qua = [feat for feat in data.columns if feat not in nom and feat != 'StartDate'
and feat != 'EndDate']

# Ordinal
ordin_ = Pipeline([
    ('imp', SimpleImputer(strategy="most_frequent")),
    ('ord', rankTime()),
    ('nor', Normalizer())
])
ordin = ['StartDate', 'EndDate']

X_train, X_test, y_train, y_test = train_test_split(data, label, random_state=1)

ct = ColumnTransformer([('nom', nom_, nom), ('qua', qua_, qua), ('ordin', ordin_
, ordin)])


regressors = []
regressors.append(xgb.XGBRegressor())
regressors.append(LinearRegression())
regressors.append(SVR())
regressors.append(DecisionTreeRegressor())
regressors.append(RandomForestRegressor())
regressors.append(ExtraTreesRegressor())
regressors.append(GradientBoostingRegressor())
regressors.append(KNeighborsRegressor())




r2_dict = {}
for regressor in regressors:
    pl = Pipeline([('feats', ct), ('rgr', regressor)])
    parameters = {}
    if type(regressor).__name__ == 'SVR':
        print(type(regressor).__name__)
        parameters = {
            'rgr__C': [.5, 1, 2, 3, 7, 10, 100, 1000]
        }
    elif type(regressor).__name__ == 'DecisionTreeRegressor':
        print(type(regressor).__name__)
        parameters = {
            'rgr__max_depth': [10, 20, 30, 40, 50, 100, 200, None]
```

```python
        }
    elif type(regressor).__name__ == 'RandomForestRegressor':
        print(type(regressor).__name__)
        parameters = {
            'rgr__max_depth': [10, 20, 30, 40, 50, 100, 200, None]
        }
    elif type(regressor).__name__ == 'ExtraTreesRegressor':
        print(type(regressor).__name__)
        parameters = {
            'rgr__max_depth': [10, 20, 30, 40, 50, 100, 200, None]
        }
    elif type(regressor).__name__ == 'GradientBoostingRegressor':
        print(type(regressor).__name__)
        parameters = {
            'rgr__n_estimators': [80, 90, 100, 120, 150, 200]
        }
    elif type(regressor).__name__ == 'KNeighborsRegressor':
        print(type(regressor).__name__)
        parameters = {
            'rgr__n_neighbors': [1, 3, 5, 7, 10, 20, 40, 50]
        }
    elif type(regressor).__name__ == 'XGBRegressor':
        print(type(regressor).__name__)
        parameters = {

        }
    elif type(regressor).__name__ == 'LinearRegression':
        print(type(regressor).__name__)
        parameters = {

        }
    search = GridSearchCV(pl, parameters, cv=5)
    search.fit(X_train, y_train)
    preds = search.predict(X_test)
    r2_dict[search] = r2_score(preds, y_test)
```

```
XGBRegressor
[15:30:43] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
[15:30:44] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
[15:30:46] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
[15:30:47] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
[15:30:48] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
[15:30:49] WARNING: /workspace/src/objective/regression_obj.cu:152:
reg:linear is now deprecated in favor of reg:squarederror.
LinearRegression
SVR
DecisionTreeRegressor
RandomForestRegressor
ExtraTreesRegressor
GradientBoostingRegressor
KNeighborsRegressor
```

In [15]:

```
search_best = max(r2_dict, key=r2_dict.get)
preds = search_best.predict(X_test)
print('R2 for predicting Impressions is ', r2_score(preds, y_test))
```

```
R2 for predicting Impressions is  0.8517658777143433
```

In [16]:

```
print('Best Estimator:')
print(search_best.best_estimator_.steps[1][1])
```

```
Best Estimator:
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=1
00,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=
0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
```

In [17]:

```
# R2 for linear regression
r2_dict[list(r2_dict.keys())[1]]
```

Out[17]:

```
0.45751476625330056
```

In [18]:

```
# Inference
rgr = list(r2_dict.keys())[1].best_estimator_.steps[1][1]
dict_coef = dict(zip(data.columns, rgr.coef_))
pd.Series(dict(sorted(dict_coef.items(), key=lambda d: d[1], reverse=True)))
```

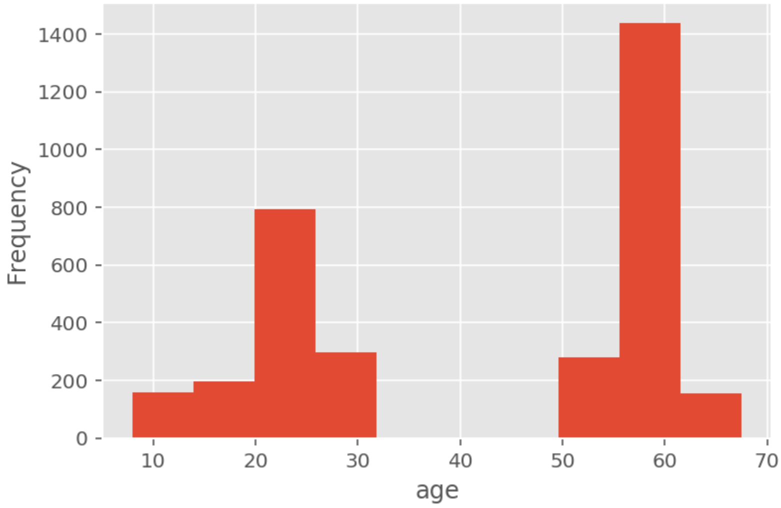
Out[18]:

```
PayingAdvertiserName            1.694779e+06
CountryCode                     4.346214e+05
OsType                          3.966712e+05
Year                            3.074031e+05
AgeBracket                      1.516741e+05
Targeting Geo - Postal Code     1.360645e+05
RegionID                        1.291749e+05
CreativeUrl                     9.882583e+04
ElectoralDistrictID             5.397202e+04
Language                        4.961746e+04
Spend                           2.000108e+04
LatLongRad                     -2.085556e+04
age_end                        -2.808062e+04
ADID                           -3.633852e+04
Targeting Carrier (ISP)        -4.428166e+04
StartDate                      -4.981333e+04
Interests                      -6.378904e+04
Gender                         -1.099415e+05
time                           -1.614656e+05
CreativeProperties             -1.653596e+05
LocationType                   -2.273732e+05
Genre                          -2.905208e+05
CandidateBallotInformation     -2.992644e+05
Segments                       -3.017640e+05
OrganizationName               -3.548954e+05
AdvancedDemographics           -5.541756e+05
Type                           -7.046631e+05
MetroID                        -7.986371e+05
BillingAddress                 -9.854999e+05
age_start                      -1.212433e+06
EndDate                        -2.819404e+06
Targeting Connection Type      -3.298042e+06
dtype: float64
```

Fairness Evaluation

In [46]:

```
# We want to see is the data has a bias on the agebracket. To do this, we will u
se the mean of
# age_start and age_end, and the mean smaller than 40 is viewed as young and big
ger then 40 is viewed as old
age_mean = (data['age_start'] + data['age_end']) / 2
(age_mean).plot(kind='hist');
plt.xlabel('age');
```



In [42]:

```
results = X_test
results['is_young'] = (age_mean <= 40).replace({True:'young', False:'old'})
results['prediction'] = preds
results['y'] = y_test

obs = results.groupby('is_young').apply(lambda x: mean_squared_error(x.y, x.pred
iction)).diff().iloc[-1]
metrs = []
for _ in range(5000):
    s = (
        results[['is_young', 'prediction', 'y']]
        .assign(is_young=results.is_young.sample(frac=1.0, replace=False).reset_
index(drop=True))
        .groupby('is_young')
        .apply(lambda x: mean_squared_error(x.y, x.prediction))
        .diff()
        .iloc[-1]
    )

    metrs.append(s)
```

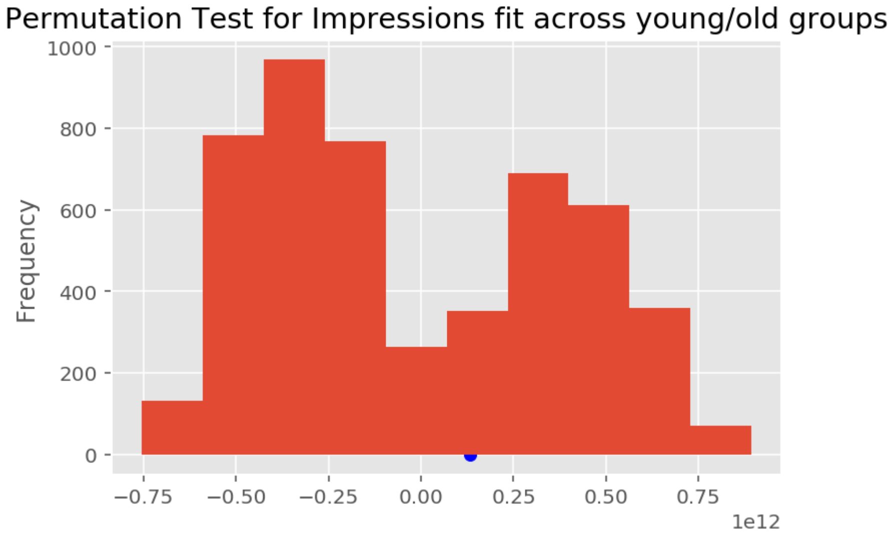
In [55]:

```
print('P-value: ', pd.Series(metrs >= obs).mean())
plt.scatter(obs, 0, c='b');
pd.Series(metrs).plot(kind='hist', title='Permutation Test for Impressions fit a
cross young/old groups')
```

```
P-value:  0.4022
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3ff53730f0>
```



So the data is unbiased.