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

Introduction

Thoughout this project, we are attempting to predict the target gender of the Political Ads. This is a classification problem because a Snapchat Ad could only target men, women, or all genders.

The reason why we chose the target variable is that we assumed the gender an AD is targeting might be strongly associated with some other features of the AD, such as the money spent on the AD, etc.

The model would be evaluated on whether it could predict the target gender accurately, given some other features.Due to class imbalance of targeted gender in our dataset (thus accuracy is not a good measure), and the fact that we have trinary outcomes (we cant use scikit package to find recall and precision), we decided to look at 6 parameters that is very similar to true positive, false positive, true negative and false negative: True Female, False not Female, True Male, False not Male,True All Gender, False not All Gender. Take True Female and False not Female as an example, True Female is the rate of correctly identified ads targetting female, False not Female is the rate of ads that targets female but not identified as targetting female. We can not compute precision and recall directly, but we believe these 6 measures are better metrics for model performance compare to accuracy.

Baseline Model

There are five features we picked in total.Two of them are quantitative, two are nominal and one is ordinal.'Spend': Amount spent by advertiser (USD), is quantitative; 'Impressions': Number of times the AD was viewed, is quantitative; 'Segments',segments targeting criteria used in AD, is ordinal; 'Language',language used in the AD, is ordinal; 'AgeBracket': Targeting ages of the AD, is nominal.

The model overal gives an accuracy around 91%. We originally thought this means shows that our model is sufficient enough to make predictions. However, when we looked into the dataset in details, we find out that we have imbalanced class problem in our outcome - Gender. More than 90% in the gender column of the dataset is " All Genders. Thus, the accuracy score of our model does not really mean anything. We can not compute precision and recall as the target variable we are trying to predict is trinary: FEMALE, MALE, All Gender. After consideration, we decide to use 6 variables that are very similar to precision and recall. (as described in the second paragraph of introduction). We have very low rates for correctly identified genders rate:

true_Fe(True Female):0.36363636363636365 true_ma(True Male):0.13043478260869565 true_all((True All Gender):0.9774520856820744 (the most likely reason for this being high is that the majority of data is all genders)

and very high values for incorrectly identified genders rate:

false_not_Fe (actually female but not identified as female): 0.6363636363636364 false_not_ma (actually male but not identified as male): 0.8695652173913043 false_not_all((actually all but not identifies all): 0.022547914317925577 (the most likely reason for this being low is that the majority of data is low)

Therefore, there are a lot of rooms to improve our model.We need to explore our dataset further to improve our model, by exploring new features and look back to the original dataset.

Final Model

The first feature we add the duration of ads in seconds, obtained by substracting enddate from startdate. The second is we add the impression/dollar, obtained by deviding spend from impressoions. The reason why we think they are good for our data, to potentially improve our model accuracy, is that we noticed that the average duration of an AD targeting males, is higher than the average of those targeting females and all genders we assume maybe there's correlation between these terms. We also included Organization Data as a feature, because when we do univariate data analysis, we find out that there are a considerable amount of organizations who only make ads targets all gender, and some make more ads targetting females than males.

When we were evaluating our base models, we find out that the outcome column in our dataset is heavily imbalanced. As result, we dont think KNeighborsClassifier to be an approriate model for our classification problem because the default value for param weight is Uniform. With such an imbalanced dataset, we dont consider Uniform to be a approriate param for weight. Therefore, we changed our model to DecisiontreeClassifier. It indeed showed better performance. We first run this classifier (overfitted) 100 times to have an idea what is the max_depth and leafnodes an overfitted decision model may have. We then pick the minimum value among all the values of max_depth as a ceiling for our choice of possible parameters(using GridSearchCV). We also tried other combination of parameters close to the number given by GridSearchCV, and used the one which gives the best results. We used cross validation method to test the accuracy of the new model and find out the accuracy of the final model is about 3% higher than our baseline model. We then looked at out our own evaluation metrics:

true_Fe(True Female):0.04225352112676056 true_ma(True Male):0.5652173913043478 true_all((True All Gender):0.9842180774748924

false_not_Fe (actually female but not identified as female): 0.9577464788732395 false_not_ma (actually male but not identified as male): 0.4347826086956522 false_not_all((actually all but not identifies all): 0.015781922525107572

Our model has clearly improved.

Fairness Evaluation

We wanted to look at if the ads will be more gender specified if segments is provided by the advertisor. Therefore, We took a closer look at the segment and the prediction column. The result is different from what is ecpected, that the distribution of targets add being gender specified is actually the same whether or not segment is provided by advertisor.

We also looked at accuracy polarity,trying to see if the distributions of gender being the same for ads in which segment is provided by advertiser or not provided by advertiser. We find out that the distribution is the same, and the model is pretty fair in this aspect. However, as indicated throughout the entire write up, we have imbalanced class issue in our dataset for our target variable "Gender". Accuracy may not be a good parity used to measure fairness. We think a true positive parity will give us more information about proportion of gender across ads with segments provided by advisor or not. However, the trinary nature of our target outcome prevented us from doing this.

Code

In [13]:

```
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 figures
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from tqdm import tqdm_notebook as tqdm
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
```

Baseline Model

Read the data and concate the dataset.

In [285]:

```python
first = pd.read_csv('PoliticalAds2018.csv')
second = pd.read_csv('PoliticalAds2019.csv')
df = pd.concat([first,second])
index = range(len(df))
df['index'] = index
df = df.set_index('index')
df.head()
```

Out[285]:

	ADID	CreativeUrl	Spend
index			
0	68189125240c16e3dcec398a7b4e040e74621ccf11df25...	https://www.snap.com/political-ads/asset/b2d0c...	2
1	3542d40ba9fb0f0aa52889b03ea4a7db64ee4a3687aa6e...	https://www.snap.com/political-ads/asset/bc5a0...	143
2	6d1eb1af8c01a43def7b695fd19f2c43fa6625d126a063...	https://www.snap.com/political-ads/asset/d9367...	1913
3	64d906646b616c034c91b69b9e7851944844eb456dd203...	https://www.snap.com/political-ads/asset/e56c0...	56
4	4a090b72334ceabe7779ffe261a518b6f182e3fde4337e...	https://www.snap.com/political-ads/asset/8a8cb...	255

5 rows × 27 columns

Now, we would want to determine and setup the features to be used to make the prediction.

In [286]:

```
1  df.columns
```

Out[286]:

```
Index(['ADID', 'CreativeUrl', 'Spend', 'Impressions', 'StartDate', 'En
dDate',
       'OrganizationName', 'BillingAddress', 'CandidateBallotInformati
on',
       'PayingAdvertiserName', 'Gender', 'AgeBracket', 'CountryCode',
       'RegionID', 'ElectoralDistrictID', 'LatLongRad', 'MetroID', 'In
terests',
       'OsType', 'Segments', 'LocationType', 'Language',
       'AdvancedDemographics', 'Targeting Connection Type',
       'Targeting Carrier (ISP)', 'Targeting Geo - Postal Code',
       'CreativeProperties'],
      dtype='object')
```

We have 27 columns in total - 26 potential features but we might not need all of them.In this case, whether we should use the certain feature, and if so, whether we should use them directly or do some tranformation?

We noticed that some of the features are definitely useless - such as ADID and CreativeUrl for our problem. After investigation, we have decided to use the following features and fill NA values according to the text file given.

In [287]:

```
useful = ['Spend','Impressions','Segments','Language','AgeBracket','Gender']
data = df[useful].copy()
data['Segments'] = data['Segments'].fillna('Not Provided by advertiser')
data['Language'] = data['Language'].fillna('Agnostic Langauge')
data['AgeBracket'] = data['AgeBracket'].fillna('All Ages')
data['Gender'] = data['Gender'].fillna('All Genders')
cleaned = data.dropna()
cleaned.head()
```

Out[287]:

	Spend	Impressions	Segments	Language	AgeBracket	Gender
index						
0	2	1301	Provided by Advertiser	sv	18-24	All Genders
1	143	49094	Provided by Advertiser	Agnostic Langauge	18+	All Genders
2	1913	886571	Not Provided by advertiser	Agnostic Langauge	18-23	All Genders
3	56	11770	Provided by Advertiser	Agnostic Langauge	18+	FEMALE
4	255	142929	Provided by Advertiser	Agnostic Langauge	All Ages	All Genders

Now we have a cleaned dataset, and ready to build our models on it

In [288]:

```
y = cleaned.Gender
X = cleaned.drop(['Gender'],axis = 1)
```

In [289]:

```
nums = Pipeline([('ohe',OneHotEncoder(sparse=False))])
numcol =  ['Segments','Language','AgeBracket']
std = Pipeline([('std',StandardScaler())])
stdcol = ['Spend','Impressions']
ct = ColumnTransformer([('num',nums,numcol),('std',std,stdcol)],remainder = 'pas
```

Here, to avoid the mismatch of unique values in train and test set caused by one-hot encoding, we do the column transformation prior test_split.

In [290]:

```python
transformed = pd.DataFrame(ct.fit_transform(X))
transformed.head()
```

Out[290]:

	0	1	2	3	4	5	6	7	8	9	...	115	116	117	118	119	120	121	122	
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0
1	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0
2	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0
4	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-0

5 rows × 125 columns

In [291]:

```python
pl_base = Pipeline([('r',KNeighborsClassifier(n_neighbors = 3))])
```

In [292]:

```python
x_train,x_test,y_train,y_test = train_test_split(transformed,y,test_size=0.3)
pl_base.fit(x_train,y_train)
pred = pl_base.predict(x_test)
pred
```

Out[292]:

```
array(['All Genders', 'All Genders', 'All Genders', 'All Genders',
       'FEMALE', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'FEMALE', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'FEMALE', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'FEMALE', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
```

In [293]:

```
y_expect = pl_base.predict(x_test)
accuracy = metrics.accuracy_score(y_test,y_expect)
accuracy
```

Out[293]:

```
0.9098277608915907
```

In [294]:

```
scores = cross_val_score(pl_base, x_train, y_train, cv=5)
scores
```

Out[294]:

```
array([0.90889371, 0.91106291, 0.90672451, 0.91956522, 0.92374728])
```

The model seems good enough with accuracy? However, we need to check further.

In [295]:

```
cleaned['Gender'].value_counts()
```

Out[295]:

```
All Genders    2967
FEMALE          250
MALE             72
Name: Gender, dtype: int64
```

and we have observed that we have an class imbalance issue with our dataset, as result, accuracy is no longer a model performance metric. We are also not able to compute precision and recall using sklearn as our outcomes are not binary. We decide to compute something similar in order to take a closer look at our model performance

In [296]:

```
use = y_test.reset_index().drop(['index'],axis = 1)
use
```

Out[296]:

	Gender
0	All Genders
1	All Genders
2	All Genders
3	All Genders
4	FEMALE
...	...
982	All Genders
983	All Genders
984	All Genders
985	All Genders
986	MALE

987 rows × 1 columns

In [ ]:

```

```

In [297]:

```
#cmopute true female and false not female
act_fe = use[use=='FEMALE'].dropna().index
pred_fe = pd.Series(y_expect)
is_all_fe_hitted = pred_fe[act_fe].value_counts()
is_all_fe_hitted
```

Out[297]:

```
All Genders    42
FEMALE         28
MALE            7
dtype: int64
```

In [298]:

```python
# we can use this to compute the true female and false female
true_Fe = is_all_fe_hitted['FEMALE']/len(act_fe)
false_not_Fe = 1 - is_all_fe_hitted['FEMALE']/len(act_fe)
true_Fe,false_not_Fe
```

Out[298]:

```
(0.36363636363636365, 0.6363636363636364)
```

In [299]:

```python
act_ma = use[use=='MALE'].dropna().index
pred_ma = pd.Series(y_expect)
is_all_ma_hitted = pred_ma[act_ma].value_counts()
is_all_ma_hitted
```

Out[299]:

```
All Genders    17
MALE            3
FEMALE          3
dtype: int64
```

In [300]:

```python
true_ma = is_all_ma_hitted['MALE']/len(act_ma)
false_not_ma = 1 - is_all_ma_hitted['MALE']/len(act_ma)
true_ma,false_not_ma
```

Out[300]:

```
(0.13043478260869565, 0.8695652173913043)
```

In [301]:

```python
act_all = use[use=='All Genders'].dropna().index
pred_all = pd.Series(y_expect)
is_all_all_hitted = pred_all[act_all].value_counts()
is_all_all_hitted
```

Out[301]:

```
All Genders    867
FEMALE          17
MALE             3
dtype: int64
```

In [302]:

```
# we can use above to compute the true all and false not all
true_all = is_all_all_hitted['All Genders']/len(act_all)
false_not_all = 1 - is_all_all_hitted['All Genders']/len(act_all)
true_all,false_not_all
```

Out[302]:

```
(0.9774520856820744, 0.022547914317925577)
```

In [303]:

```
# The above stats show us that our baseline identify 'All Genders' pretty well b
# at identifying both of the specific genders.
```

Final Model

Adding more features:

First, we add the duration of ads, obtained by substracting enddate from startdate. Second, we add the impression/dollar, obtained by deviding spend from impressoions

In [304]:

```
useful = ['Spend','Impressions','Segments','Language','AgeBracket','Gender','Org
data = df[useful].copy()
data['Segments'] = data['Segments'].fillna('Not Provided by advertiser')
data['Language'] = data['Language'].fillna('Agnostic Langauge')
data['AgeBracket'] = data['AgeBracket'].fillna('All Ages')

#We added 2 features into our model, duration and unit_spend
duration = (pd.to_datetime(df.EndDate) - pd.to_datetime(df.StartDate)).apply(lam
unit_spend = data['Impressions']/data['Spend']
data['Duration'] = duration
data['impression/dollar'] = unit_spend.replace(np.inf,0)

data['Gender'] = data['Gender'].fillna('All Genders')
data = data.dropna()
nums = Pipeline([('ohe',OneHotEncoder(sparse=False))])
numcol =  ['Segments','Language','AgeBracket','OrganizationName']

std = Pipeline([('std',StandardScaler())])
stdcol = ['Duration','impression/dollar']
ct = ColumnTransformer([('num',nums,numcol),('std',std,stdcol)],remainder = 'pas
transformed = pd.DataFrame(ct.fit_transform(data))
transformed.head()
```

Out[304]:

	0	1	2	3	4	5	6	7	8	9	...	405	406	407	408	409	410	411	412	41
0	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	-0.225354	0.430039	2	130
1	0	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	-0.628843	-0.307611	143	4909
2	1	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	-0.304471	-0.019139	1913	88657
3	0	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	-0.219196	-0.627314	56	1177
4	0	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	-0.554936	0.213934	255	14292

5 rows × 415 columns

As discovered with the baseline model, the Gender column(outcome) are heavily imbalanced. We, therefore, consider k neighbours to be an unappropriate model to use. Because the default parameter for weights in this KNN is uniform, and the dataset clearly shows that the outcome is imbalanced. As result, we considered DecisionTreeClassifier to be a better model.

In [305]:

```
#we will use an overfitted model to see the possible max_depth and node_count fi
rez = []
depth = []
node_count = []
for i in tqdm(range(100)):
    x_train,x_test,y_train,y_test = train_test_split(transformed.drop([414],axis
    pl_final = Pipeline([('r',DecisionTreeClassifier())])
    pl_final.fit(x_train,y_train)
    y_expect = pl_final.predict(x_test)

    depth.append(pl_final['r'].tree_.max_depth)
    node_count.append(pl_final['r'].tree_.node_count)

    accuracy = metrics.accuracy_score(y_test,y_expect)
    rez.append(accuracy)

np.mean(rez)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: TqdmD
eprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """
```

100% 100/100 [01:19<00:00, 1.26it/s]

Out[305]:

```
0.9440707964601769
```

In [306]:

```
depths = np.array(depth)
depths
```

Out[306]:

```
array([37, 39, 37, 43, 41, 40, 38, 38, 41, 42, 42, 39, 40, 40, 40, 40,
39,
       42, 40, 40, 36, 40, 40, 36, 41, 39, 40, 42, 39, 41, 40, 40, 41,
39,
       39, 39, 42, 39, 36, 40, 43, 37, 39, 36, 39, 39, 41, 36, 38, 41,
41,
       44, 41, 42, 40, 39, 39, 41, 43, 39, 39, 38, 41, 38, 41, 39, 36,
41,
       39, 41, 42, 40, 39, 39, 39, 41, 37, 37, 38, 41, 38, 39, 42, 41,
35,
       36, 38, 40, 41, 39, 41, 36, 38, 39, 38, 39, 38, 39, 39, 38])
```

In [307]:

```
nodes = np.array(node_count) #overfitted
nodes
```

Out[307]:

```
array([225, 245, 233, 247, 235, 235, 209, 243, 241, 267, 247, 245, 243
,
       235, 229, 239, 241, 243, 231, 225, 225, 267, 245, 229, 239, 223
,
       227, 249, 247, 237, 239, 237, 231, 231, 227, 261, 233, 227, 229
,
       229, 263, 211, 251, 235, 237, 229, 261, 217, 245, 241, 229, 253
,
       245, 265, 261, 229, 243, 233, 227, 245, 243, 227, 235, 237, 233
,
       245, 221, 255, 237, 241, 245, 243, 231, 221, 233, 235, 215, 209
,
       247, 237, 237, 247, 249, 247, 227, 247, 235, 245, 229, 243, 243
,
       227, 223, 243, 215, 241, 241, 233, 235, 225])
```

In [308]:

```
len(x_train)
```

Out[308]:

```
1844
```

In [309]:

```
min(depths) #the max_depth param in a good model should be smaller than 34 to pr
```

Out[309]:

```
35
```

In [310]:

```
min(nodes) # we can have much less leaf nodes
```

Out[310]:

```
209
```

In [311]:

```
a = []
for i in range(2,35,2):
    a.append(i)
a.append(None)
b = []
for i in range(2,35,2):
    b.append(i)
np.array(b)
```

Out[311]:

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32,
34])
```

In [325]:

```
parameters = {
    'max_depth': a,
    'min_samples_split':b,
    'min_samples_leaf':[2,3,5,7,10,15,20,25,30,35,40,45,50]
}
```

In [326]:

```
clf = GridSearchCV(DecisionTreeClassifier(), parameters, cv=5)
```

In [411]:

```
clf.fit(x_train, y_train)
```

Out[411]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_de
pth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,

min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,

min_weight_fraction_leaf=0.0,
                                              presort=False, random_st
ate=None,
                                              splitter='best'),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18,
20, 22,
                                       24, 26, 28, 30, 32, 34, None],
                         'min_samples_leaf': [2, 3, 5, 7, 10, 15, 20,
25, 30,
                                              35, 40, 45, 50],
                         'min_samples_split': [2, 4, 6, 8, 10, 12, 14,
16, 18,
                                               20, 22, 24, 26, 28, 30,
32,
                                               34]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring=None, verbose=0)
```

In [412]:

```
clf.best_params_ # in my first run i got 22, but i have tried every number from
                    # this is the only param that changes
```

Out[412]:

```
{'max_depth': 34, 'min_samples_leaf': 2, 'min_samples_split': 4}
```

In [245]:

```
clf.score(x_test, y_test) #higher than baseline model
```

Out[245]:

```
0.9494310998735778
```

In [528]:

```
pl_final_final = Pipeline([('r',DecisionTreeClassifier(max_depth = 22,min_sample
```

In [529]:

```
xf_train,xf_test,yf_train,yf_test = train_test_split(transformed.drop([414],axis
```

In [530]:

```
pl_final_final.fit(xf_train,yf_train)
```

Out[530]:

```
Pipeline(memory=None,
         steps=[('r',
                 DecisionTreeClassifier(class_weight=None, criterion='
gini',
                                        max_depth=22, max_features=Non
e,
                                        max_leaf_nodes=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=2, min_sample
s_split=4,
                                        min_weight_fraction_leaf=0.0,
                                        presort=False, random_state=No
ne,
                                        splitter='best'))],
         verbose=False)
```

In [531]:

```
scores = cross_val_score(pl_final_final, xf_train, yf_train, cv=5)
scores #accuracy is better than in baseline
```

Out[531]:

```
array([0.92682927, 0.91327913, 0.93495935, 0.91056911, 0.94293478])
```

In [539]:

```
predf = pl_final_final.predict(xf_test)
predf
```

Out[539]:

```
array(['FEMALE', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'FEMALE', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'FEMALE', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'FEMALE', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
       'All Genders', 'All Genders', 'All Genders', 'All Genders', 'MA
LE',
       'All Genders', 'All Genders', 'All Genders', 'All Genders',
```

In [540]:

```
usef =yf_test.reset_index().drop(['index'],axis = 1)
usef.head()
```

Out[540]:

	414
0	All Genders
1	All Genders
2	All Genders
3	All Genders
4	All Genders

In [541]:

```
pred = pd.Series(predf)
pred.head()
```

Out[541]:

```
0          FEMALE
1     All Genders
2     All Genders
3     All Genders
4     All Genders
dtype: object
```

In [542]:

```
#cmopute true female and false not female
act_fe = usef[usef=='FEMALE'].dropna().index
is_all_fe_hitted = pred_fe[act_fe].value_counts()
# we can use this to compute the true female and false female
true_Fe = is_all_fe_hitted['FEMALE']/len(act_fe)
false_not_Fe = 1 - is_all_fe_hitted['FEMALE']/len(act_fe)
true_Fe,false_not_Fe
```

Out[542]:

```
(0.04225352112676056, 0.9577464788732395)
```

In [543]:

```
act_ma = usef[usef=='MALE'].dropna().index
is_all_ma_hitted = pred[act_ma].value_counts()
true_ma = is_all_ma_hitted['MALE']/len(act_ma)
false_not_ma = 1 - is_all_ma_hitted['MALE']/len(act_ma)
true_ma,false_not_ma
```

Out[543]:

```
(0.5652173913043478, 0.4347826086956522)
```

In [ ]:

```

```

In [544]:

```
act_all = usef[usef=='All Genders'].dropna().index
is_all_all_hitted = pred[act_all].value_counts()
is_all_all_hitted
# we can use above to compute the true all and false not all
true_all = is_all_all_hitted['All Genders']/len(act_all)
false_not_all = 1 - is_all_all_hitted['All Genders']/len(act_all)
true_all,false_not_all
```

Out[544]:

```
(0.9842180774748924, 0.015781922525107572)
```

the above three cells shows that the final model improved a lot compare to the first model, as the correcly identified gender rate: True Fe, true_ma, true_all increased and unidentified gender rate: false_not_Fe,false_not_ma,false_not_all decreased

In [545]:

```
cleaned
```

Out[545]:

	Spend	Impressions	Segments	Language	AgeBracket	Gender
index						
0	2	1301	Provided by Advertiser	sv	18-24	All Genders
1	143	49094	Provided by Advertiser	Agnostic Langauge	18+	All Genders
2	1913	886571	Not Provided by advertiser	Agnostic Langauge	18-23	All Genders
3	56	11770	Provided by Advertiser	Agnostic Langauge	18+	FEMALE
4	255	142929	Provided by Advertiser	Agnostic Langauge	All Ages	All Genders
...	...	...	...	...	...	...
3284	7	1696	Provided by Advertiser	Agnostic Langauge	18+	All Genders
3285	0	146	Provided by Advertiser	ar	21+	All Genders
3286	729	224435	Provided by Advertiser	nb	18-30	All Genders
3287	155	46058	Not Provided by advertiser	Agnostic Langauge	18+	All Genders
3288	44	5177	Not Provided by advertiser	en	18+	All Genders

3289 rows × 6 columns

Fairness Evaluation

In [546]:

```
results = xf_test
result = cleaned.iloc[results.index,]
result['provided'] = result['Segments'].replace({True:'Provided by Advertiser',F
result['prediction'] = predf
result['actual'] = yf_test
result.head()
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy)
  This is separate from the ipykernel package so we can avoid doing im
ports until
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy)
  after removing the cwd from sys.path.
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy)
  """
```

Out[546]:

index	Spend	Impressions	Segments	Language	AgeBracket	Gender	provided	prediction	
520	1571	142335	Not Provided by advertiser	Agnostic Langauge	18-34	All Genders	Not Provided by advertiser	FEMALE	G
666	14	1641	Not Provided by advertiser	Agnostic Langauge	35++	All Genders	Not Provided by advertiser	All Genders	G
739	548	126731	Provided by Advertiser	nb	18-34	All Genders	Provided by Advertiser	All Genders	G
714	58	19094	Provided by Advertiser	Agnostic Langauge	18+	All Genders	Provided by Advertiser	All Genders	G
1601	777	316118	Not Provided by advertiser	en	18+	All Genders	Not Provided by advertiser	All Genders	G

In [547]:

```python
provided_by_adver = result[result['provided'] == 'Provided by Advertiser']
not_provided_by_adver =result[result['provided'] == 'Not Provided by advertiser'
```

In [548]:

```python
prov = provided_by_adver.groupby('prediction').count()
prov_gen = (prov.iloc[:,0]/prov.iloc[:,0].sum()).to_frame()
noprov = not_provided_by_adver.groupby('prediction').count()
noprov_gen = (noprov.iloc[:,0]/noprov.iloc[:,0].sum()).to_frame()
prov_gen
```

Out[548]:

prediction	Spend
All Genders	0.900778
FEMALE	0.079767
MALE	0.019455

In [549]:

```python
noprov_gen #the demographic parity shows the target gender distribution is very
#the target segment is provided by the advisor
```

Out[549]:

prediction	Spend
All Genders	0.916968
FEMALE	0.046931
MALE	0.036101

In [369]:

```python
(
    result
    .groupby('provided')
    .apply(lambda x: metrics.accuracy_score(x.actual, x.prediction))
    .rename('accuracy')
    .to_frame()
)
```

Out[369]:

	accuracy
provided	
Not Provided by advertiser	0.944649
Provided by Advertiser	0.940385

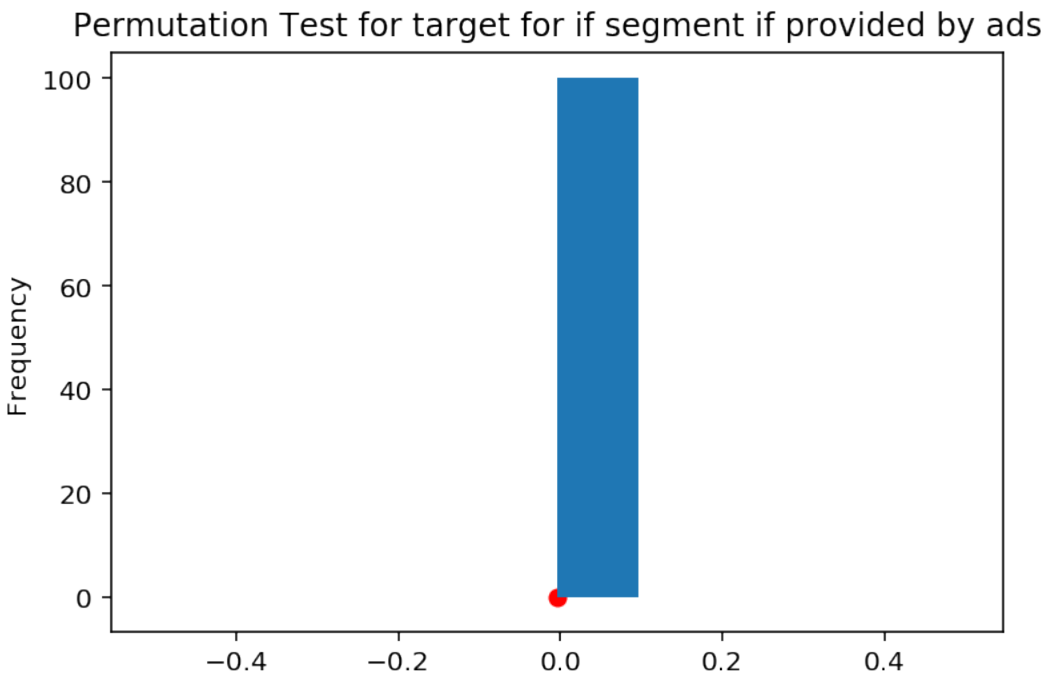
In [370]:

```python
#accuracy parity
obs = result.groupby('provided').apply(lambda x: metrics.accuracy_score(x.actual

metrs = []
for _ in range(100):
    s = (
        result[['provided', 'prediction', 'actual']]
        .assign(is_provided=result.provided.sample(frac=1.0, replace=False).rese
        .groupby('provided')
        .apply(lambda x: metrics.accuracy_score(x.actual, x.prediction))
        .diff()
        .iloc[-1]
    )

    metrs.append(s)
```

In [372]:

```
print(pd.Series(metrs <= obs).mean())
pd.Series(metrs).plot(kind='hist', title='Permutation Test for target for if seg
plt.scatter(obs, 0, c='r');
#The accuracy parity tell us that the distributions of targeted gender is "the s
#by adviser and not provided by adviser. However, again accuracy doesnt tell us
# is heavily imbalanced in gender.
```

```
1.0
```



If our target variable is binary, i wil try to compute the True Positive Parity of the model. However,we can not do it here since our target variable is trinary