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

We will try to predict the number of impressions made by an ad. This is because the number of impressions the only measure of success we have for each advertisement. Since the number of impressions is a numerical value, we will be using a regression model for this problem. Our goal is to create a model that best predicts the number of impressions an ad will receive based on its other data, so we will try to maximize our model's score (R^2). This is a measure of how well our model predicts impressions.

Baseline Model

For our baseline model, we dropped name/label that should have no effect on the number of impressions, and date columns that our model would have a hard time processing. We split the columns into two categories: categorical and numerical.

We first imputed the categorical columns with string 'NULL's; this is justifiable because most of the null values were intentionally null, so they can be treated as their own category. We then one hot encoded these columns and used PCA (principle component analysis) to reduce the dimensionality of our data and remove excess columns that are highly correlated.

We then imputed the numerical columns with zeroes; this is justifiable because intentionally null values in numerical columns most commonly represent zeroes.

From looking at the data and its columns, we predicted that the number of impressions would be positively and linearly correlated with the amount spent on the advertisement. Hence, we chose to use a linear regression model.

In 100 train test splits, baseline model had a median score of 0.52. While this is not good, it is not too bad considering the mininmal feature engineering we had done. In addition, much of the provided data is null, and many columns seem irrelevant to how many impressions an ad received.

Final Model

For our final model, we dropped some additional columns that were entirely null or represented geographical data. Our reason for removing the latter was because the data contained many columns describing geographical location. To avoid overfitting, we removed some of these columns and kept the simplest one with no null values: 'CountryCode'.
We then altered many of the columns into boolean columns. We did this because many columns were filled with mostly null values (intentionally) and there were very few instances of each non-null value, which made treating those columns as categorical prone to overfitting. To fix this, we altered the data so null values were True and non-null values were False. To reduce the number of categories in 'Language', we grouped languages specified less than 50 times into its own category 'other'. We also imputed null values with 'any'. To reduce the number of categories in 'AgeBracket', we kept only the lower bound of each age bracket. This also gave us the option of treating 'AgeBracket' as a numerical column.

To select a model, we used a pipeline to test many different models on the same data. We tested:

- LinearRegression
- KNeighborsRegressor
- SVR
- NuSVR
- DecisionTreeRegressor
- RandomForestRegressor
- AdaBoostRegressor
- GradientBoostingRegressor

We found that linear regression had the highest score and still provided the best model. For the PCA, we found that when n_components = 0.95, the model performed the best.

Fairness Evaluation

Because our model is a regression model, we decided to test the fairness of our model using squared error. We created a new column for squared error and grouped each categorical column. We then calculated the mean squared error and plotted our results. One of the plots that stood out was the segments plot because the Segments is a boolean column with a fair number of ads in both categories.

Thus, we performed a permutation test on to see if the difference in mean squared error between the ads that did and did not provide segments could be explained by chance. From our test, our observed test statistic (absolute difference in mean squared error) has a p-value of 0.644. Our significance level was set to 0.01, so we failed to reject our null hypothesis. Therefore, we assumed that our model was not biased towards ads that did or did not specify segments.

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 figures
```

In [182]:
```
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import FunctionTransformer
from sklearn.impute import SimpleImputer

from sklearn.svm import SVR
from sklearn.svm import NuSVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

We will begin by performing minimal cleaning of the data:

- Join the 2018 and 2019 data
- Drop columns with irrelevant data (names, ids, labels, etc.)

In [152]:
```
# load the csv files for 2018 and 2019
fp_18 = os.path.join('data', 'ads_2018.csv')
fp_19 = os.path.join('data', 'ads_2019.csv')

rel_cols = ['Spend', 'Impressions']


# read files into dataframes
df_18 = pd.read_csv(fp_18)#, #usecols=rel_cols)
df_19 = pd.read_csv(fp_19)#, #usecols=rel_cols)
ad_data = pd.concat([df_18, df_19], ignore_index=True)


# if planning to use dates, convert to datetime objects

# convert StartDate and EndDate to date time objects (UTC)
# ad_data['StartDate'] = pd.to_datetime(ad_data['StartDate'])
# ad_data['EndDate'] = pd.to_datetime(ad_data['EndDate'])

# drop identification/name columns: these should have no effect on the number
 of impressions
# timezones will not be used in our analysis
ad_data = ad_data.drop(['ADID', 'CreativeUrl', 'OrganizationName', 'PayingAdve
rtiserName', 'CreativeProperties', 'StartDate', 'EndDate'], axis=1)
orig_data = ad_data.copy()
ad_data.head()
```

```
Out[152]:
```

	Spend	Impressions	BillingAddress	CandidateBallotInformation	Gender
0	1044	137185	3050 K Street,Washington,20007,US	NaN	NaN
1	279	94161	1730 Rhode Island Ave NW,Washington,20036,US	NaN	NaN
2	6743	3149886	US	NaN	NaN
3	3698	573475	435 E. Main,Greenwood,46143,US	NaN	NaN
4	445	232906	17-25 New Inn Yard,London,EC2A 3EA,GB	NaN	NaN

Baseline Model

In [5]:
```
# drop the target column; drop datetime objects
X = ad_data.drop(['Impressions'], axis=1)
y = ad_data.Impressions

types = X.dtypes
cat_cols = types.loc[types == np.object].index
num_cols = types.loc[types != np.object].index

# display
print('categorical columns:\n', cat_cols.values, '\n')
print('numerical columns:\n',num_cols.values)
```

```
categorical columns:
 ['BillingAddress' 'CandidateBallotInformation' 'Gender' 'AgeBracket'
 'CountryCode' 'RegionID' 'ElectoralDistrictID' 'MetroID' 'Interests'
 'OsType' 'Segments' 'LocationType' 'Language' 'AdvancedDemographics'
 'Targeting Geo - Postal Code']

numerical columns:
 ['Spend' 'LatLongRad' 'Targeting Connection Type'
 'Targeting Carrier (ISP)']
```

In [6]:
```
cats = Pipeline([
    ('impute', SimpleImputer(strategy='constant', fill_value='NULL')),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    ('pca', PCA(svd_solver='full', n_components=0.99))
])

ct = ColumnTransformer([
    ('cat_cols', cats, cat_cols),
    ('num_cols', SimpleImputer(strategy='constant', fill_value=0), num_cols)
])

pl = Pipeline([('feats', ct), ('reg', LinearRegression())])
# pl = Pipeline([('feats', ct), ('reg', RandomForestRegressor(n_estimators=10
0))])
```

In [7]:
```
# separate the data into training (70%) and testing (30%) sets
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3)

# fit the model to the training set
pl.fit(X_tr, y_tr)

# calculate the score using the test set
pl.score(X_ts, y_ts)
```

Out[7]:
```
0.6182396490316843
```

In [8]:
```
# predicted values using the model
preds = pl.predict(X_ts)

# calculate the root mean square error
np.sqrt(np.mean((preds - y_ts)**2))
```

Out[8]:
```
1822852.8233084541
```

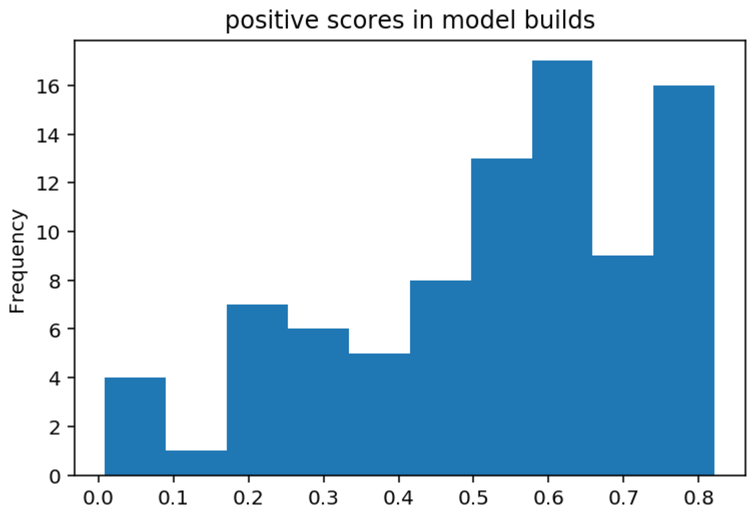
```
In [11]: output = []
         for _ in range(100):
             #
             X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3)

             pl.fit(X_tr, y_tr)
             output.append(pl.score(X_ts, y_ts))
```

```
In [ ]: scores = pd.Series(output)
        scores.plot(kind='hist', title='positive scores in model builds')
```

The existence of negative outliers makes this histogram rather useless, so we will re-plot only the positive scores. Since there are negative outliers, we know that the mean would be skewed to the left, so to measure center, we will use the median.

```
In [16]: scores[scores>0].plot(kind='hist', title='positive scores in model builds')
         print('median score:', scores.median())
```

```
median score: 0.5240352197578249
```



For our baseline model, in 100 train test splits, we got a median model score of about 0.52

Exploratory Data Analysis

Plot the number of impressions by spending, separating by color whether or not certain specifications were made.

In [153]:
```
# examine the data types
ad_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3303 entries, 0 to 3302
Data columns (total 20 columns):
Spend                          3303 non-null int64
Impressions                    3303 non-null int64
BillingAddress                 3303 non-null object
CandidateBallotInformation     225 non-null object
Gender                         322 non-null object
AgeBracket                     3029 non-null object
CountryCode                    3303 non-null object
RegionID                       1013 non-null object
ElectoralDistrictID            65 non-null object
LatLongRad                     0 non-null float64
MetroID                        180 non-null object
Interests                      786 non-null object
OsType                         21 non-null object
Segments                       2189 non-null object
LocationType                   18 non-null object
Language                       914 non-null object
AdvancedDemographics           96 non-null object
Targeting Connection Type      0 non-null float64
Targeting Carrier (ISP)        0 non-null float64
Targeting Geo - Postal Code    399 non-null object
dtypes: float64(3), int64(2), object(15)
memory usage: 516.2+ KB
```

In [154]:
```
# examine proportion of null data in each column
ad_data.isna().mean()
```

Out[154]:
```
Spend                          0.000000
Impressions                    0.000000
BillingAddress                 0.000000
CandidateBallotInformation     0.931880
Gender                         0.902513
AgeBracket                     0.082955
CountryCode                    0.000000
RegionID                       0.693309
ElectoralDistrictID            0.980321
LatLongRad                     1.000000
MetroID                        0.945504
Interests                      0.762035
OsType                         0.993642
Segments                       0.337269
LocationType                   0.994550
Language                       0.723282
AdvancedDemographics           0.970936
Targeting Connection Type      1.000000
Targeting Carrier (ISP)        1.000000
Targeting Geo - Postal Code    0.879201
dtype: float64
```

There are a few columns that only null values. We will remove these columns. We also see that many columns are about geographical location:

- BillingAddress: no null values, but fairly unique (there are 199 different addresses)
- CountryCode: no null values
- RegionID: some null values
- ElectoralDistrictID: very high null proportion
- MetroID: very high null proportion
- LocationType: very high null proportion
- Targeting Geo - Postal Code: high null proportion
  Since CountryCode has no null values, we will use only CountryCode for geographical data. All of these columns are being removed to improve efficiency and reduce the effects of overfitting.

```
In [155]: # drop columns with only null values
          ad_data = ad_data.drop(
              ad_data.columns[ad_data.isna().mean() == 1]
              , axis=1)
          ad_data.columns
```

```
Out[155]: Index(['Spend', 'Impressions', 'BillingAddress', 'CandidateBallotInformatio
          n',
                 'Gender', 'AgeBracket', 'CountryCode', 'RegionID',
                 'ElectoralDistrictID', 'MetroID', 'Interests', 'OsType', 'Segments',
                 'LocationType', 'Language', 'AdvancedDemographics',
                 'Targeting Geo - Postal Code'],
                dtype='object')
```

```
In [156]: # drop the remaining geographical data columns other than CountryCode
          ad_data = ad_data.drop(['BillingAddress', 'RegionID', 'Targeting Geo - Postal
           Code'], axis=1)
```

We will now examine the 'Segments' column.

```
In [157]: ad_data.Segments.value_counts()
```

```
Out[157]: Provided by Advertiser    2189
          Name: Segments, dtype: int64
```

We see that segments only has two types of entries: 'Provided by Advertiser' and NaN. Thus, it makes more sense to treat 'Segments' as a boolean column that is:

- True, if the advertiser did not provide user specific data (NaN)
- False, if the advertiser provided user specific data ('Provided by Advertiser')

We will now see the effect providing a segment has on the number of impressions.

In [158]:
```
# select impressions and segments
segments = ad_data[['Impressions', 'Segments']].copy()

# convert segments into boolean column
segments['Segments'] = segments['Segments'].isna()

segments.groupby('Segments').mean()
```

Out[158]:

	Impressions
Segments	
False	772037.610324
True	472416.135548

In [159]:
```
segments.groupby('Segments').count()
```

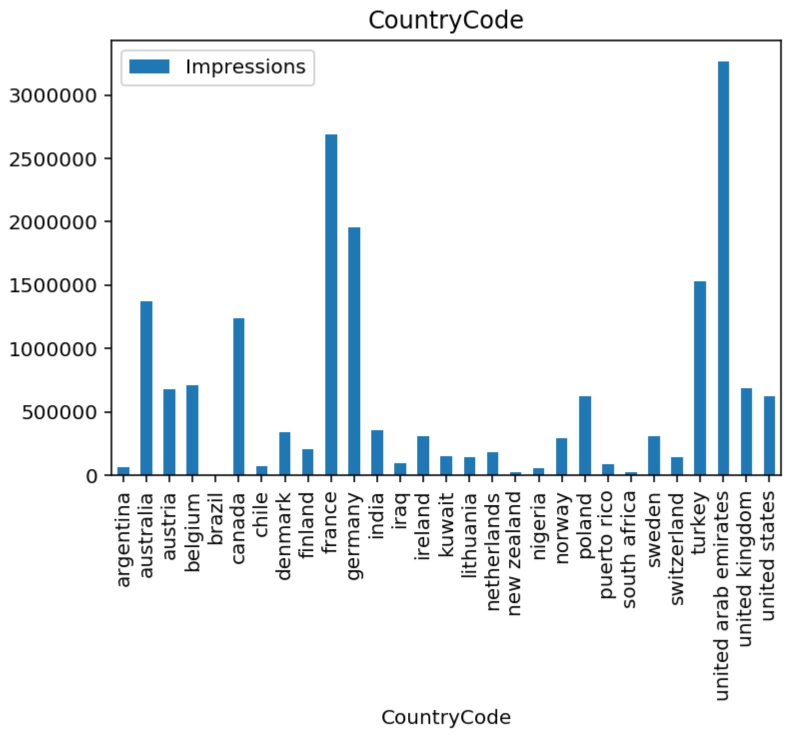
Out[159]:

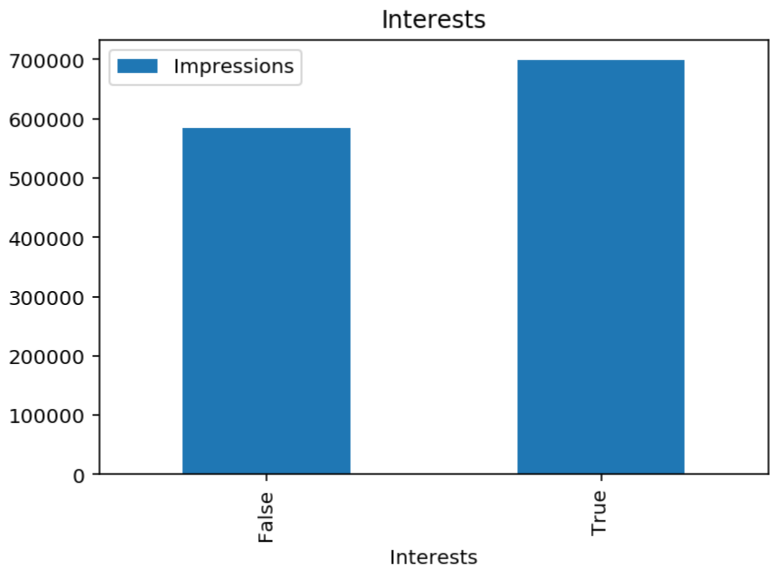
	Impressions
Segments	
False	2189
True	1114

It seems as if specifying a segment (False) corresponds to a higher impression count, however, there are likely other confounding variables affecting this. We will now examine other columns affects on impressions in a similar manner.

In [160]:
```
# CountryCode
ad_data[['Impressions', 'CountryCode']].groupby('CountryCode').mean().plot(kin
d='bar', title='CountryCode')#.min()

# Interests
# since interests are also rather unique, and a large portion of the data is n
ull,
# we will convert interests into a boolean column:
# True if null, False if interests were provided
interests = ad_data[['Impressions', 'Interests']].copy()
# convert segments into boolean column
interests['Interests'] = interests['Interests'].isna()
interests.groupby('Interests').mean().plot(kind='bar', title='Interests')
```

Out[160]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x256f9181be0>
```





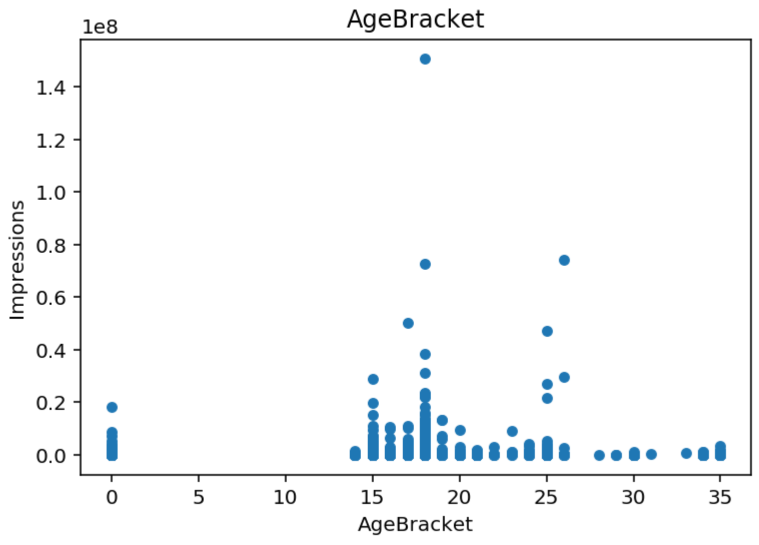
We want a way to convert AgeBracket into a numerical value. We can do this by keeping only the lower bound of the AgeBracket data.

```
In [161]: # Using regex, remove all symbols and any successive digits
          ad_data.AgeBracket = (
              ad_data.AgeBracket.str.replace('[^\d]+\d*', '')
              # impute null values with 0, because these ads are agnostic to age
              .fillna(0)
              # convert data type from object to integer
              .astype('int32')
          )

          ad_data.AgeBracket.head()
```

```
Out[161]: 0    35
          1    18
          2     0
          3    18
          4    25
          Name: AgeBracket, dtype: int32
```

```
In [162]: # AgeBracket
          ad_data.plot(kind='scatter', x='AgeBracket', y='Impressions', title='AgeBracke
          t')
```

```
Out[162]: <matplotlib.axes._subplots.AxesSubplot at 0x256f92646d8>
```



We will now look at the language category.

```
In [163]: ad_data['Language'].value_counts()
```

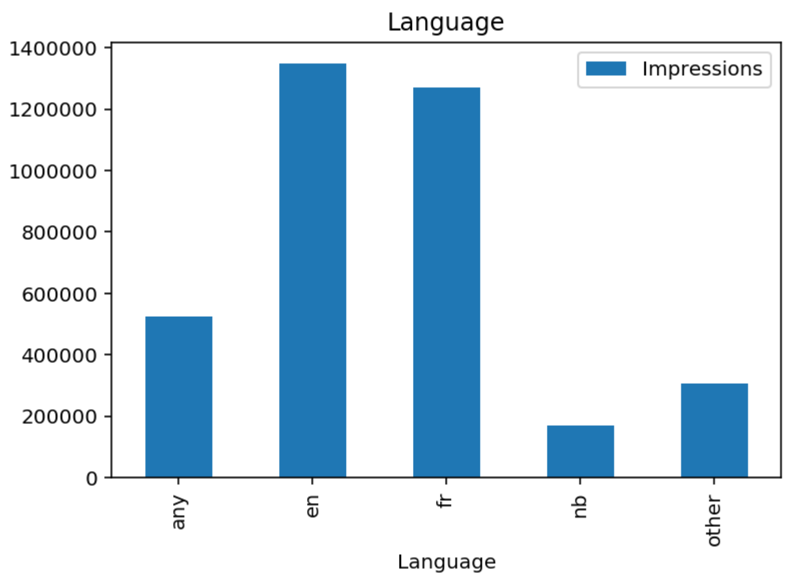
```
Out[163]: en        578
          fr         96
          nb         62
          nl         45
          da         33
          de         24
          en,es      23
          es         16
          ar         11
          nb,en       6
          nl,en       5
          sv          4
          fi          4
          ar,en       3
          de,en       2
          en,de       1
          es,en       1
          Name: Language, dtype: int64
```

We see that the most common language is English by far, and that there are some languages that are very rarely specified. We can combine the languages that appear less than 50 times under 'other.'

```
In [164]: # get value counts
          vc = ad_data['Language'].value_counts()
          # replace entries that occur < 50 times with 'other'
          ad_data.Language = ad_data.Language.replace(vc[vc < 50].index, 'other')

          # replace null entries with 'any'
          ad_data.Language = ad_data.Language.fillna('any')
```

```
In [165]: ad_data[['Impressions', 'Language']].groupby('Language').mean().plot(kind='ba
          r', title='Language')
```

```
Out[165]: <matplotlib.axes._subplots.AxesSubplot at 0x25681d51240>
```



In [166]:
```
# update Interests and Segments to be boolean columns
ad_data.Segments = segments.Segments
ad_data.Interests = interests.Interests
```

In [167]:
```
ad_data.head()
```

Out[167]:

	Spend	Impressions	CandidateBallotInformation	Gender	AgeBracket	CountryCode
0	1044	137185	NaN	NaN	35	united states
1	279	94161	NaN	NaN	18	united states
2	6743	3149886	NaN	NaN	0	united states
3	3698	573475	NaN	NaN	18	united states
4	445	232906	NaN	NaN	25	united kingdom

Get the remaining column names to select when building the pipeline.

In [168]:
```
rel_cols = ad_data.columns
rel_cols
```

Out[168]:
```
Index(['Spend', 'Impressions', 'CandidateBallotInformation', 'Gender',
       'AgeBracket', 'CountryCode', 'ElectoralDistrictID', 'MetroID',
       'Interests', 'OsType', 'Segments', 'LocationType', 'Language',
       'AdvancedDemographics'],
      dtype='object')
```

In [188]:
```
X = ad_data.drop(['Impressions'], axis=1)
y = ad_data.Impressions

cat_cols = ['CountryCode', 'AgeBracket', 'Language', 'CandidateBallotInformati
on', 'Gender', 'ElectoralDistrictID', 'MetroID',
             'OsType', 'LocationType', 'AdvancedDemographics', 'Interests', 'S
egments']
num_cols = ['Spend']
```

```
In [189]: cats = Pipeline([
              ('impute', SimpleImputer(strategy='constant', fill_value='NULL')),
              ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
              ('pca', PCA(svd_solver='full', n_components=0.95))
          ])

          ct = ColumnTransformer([
              ('cat_cols', cats, cat_cols),
          #     ('num_cols', StandardScaler(), num_cols),
               ('num_cols', SimpleImputer(strategy='constant', fill_value=0), num_cols)
          ], remainder='passthrough')



          reg = RandomForestRegressor(n_estimators=100)


          pl = Pipeline([('feats', ct), ('reg', reg)])
```

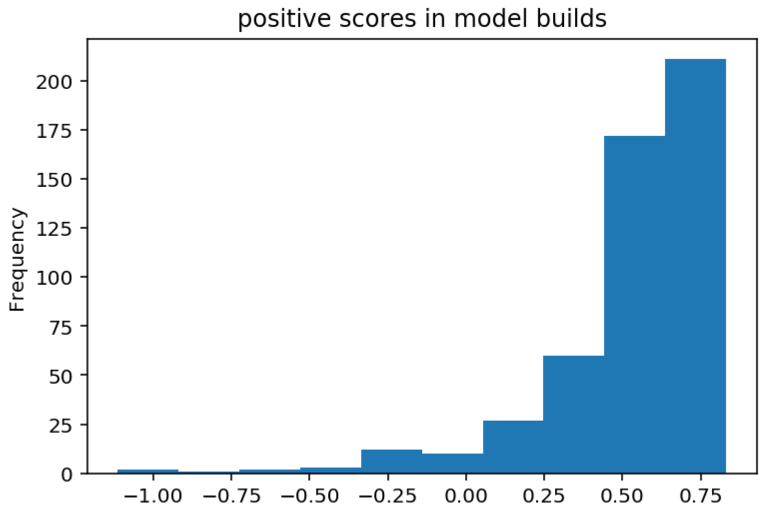
```
In [190]: # separate the data into training (70%) and testing (30%) sets
          X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3)

          # fit the model to the training set
          pl.fit(X_tr, y_tr)

          # calculate the score using the test set
          pl.score(X_ts, y_ts)
```

```
Out[190]: 0.45197041482595895
```

```
In [26]: output = []
         for _ in range(100):
             #
             X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3)

             pl.fit(X_tr, y_tr)
             output.append(pl.score(X_ts, y_ts))
```

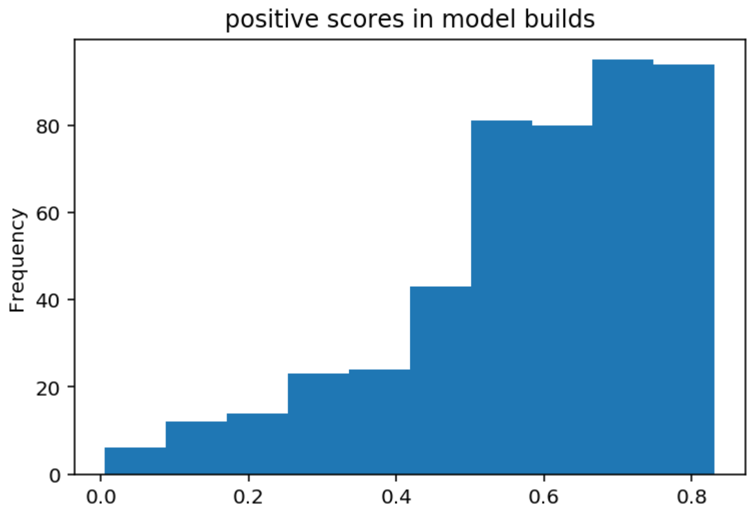
```
In [191]: scores = pd.Series(output)
          scores.plot(kind='hist', title='positive scores in model builds')
```

```
Out[191]: <matplotlib.axes._subplots.AxesSubplot at 0x25682a68dd8>
```



```
In [192]: scores[scores>0].plot(kind='hist', title='positive scores in model builds')
          print('median score:', scores.median())
```

```
median score: 0.6021655324327034
```



So far, we have tried using Linear Regression and Random Forest Regression. We will now test other regressors and compare to find the model with the best performance.

In [194]:
```
# list of regression models to test
regs = [LinearRegression(), KNeighborsRegressor(), SVR(kernel="rbf", C=0.025),
    NuSVR(), DecisionTreeRegressor(), RandomForestRegressor(), AdaBoostRegress
or(),
    GradientBoostingRegressor()
]

# test each model
for reg in regs:
    # use the column transformer we already defined
    pl = Pipeline(steps=[('column transformer', ct),
                         ('reg', reg)])
    pl.fit(X_tr, y_tr)
    print(reg)
    print("model score: %.3f" % pl.score(X_ts, y_ts))
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
model score: 0.804
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
model score: 0.418

C:\Users\kchin\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

SVR(C=0.025, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
    gamma='auto_deprecated', kernel='rbf', max_iter=-1, shrinking=True,
    tol=0.001, verbose=False)
model score: -0.016

C:\Users\kchin\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWar
ning: The default value of gamma will change from 'auto' to 'scale' in versio
n 0.22 to account better for unscaled features. Set gamma explicitly to 'aut
o' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

NuSVR(C=1.0, cache_size=200, coef0=0.0, degree=3, gamma='auto_deprecated',
      kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001,
      verbose=False)
model score: -0.012
DecisionTreeRegressor(criterion='mse', max_depth=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_state=None, splitter='best')
model score: 0.268

C:\Users\kchin\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: Fu
tureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', 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, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
model score: 0.489
AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear',
                  n_estimators=50, random_state=None)
model score: 0.436
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.1, loss='ls', max_depth=3,
                          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, n_estimators=100,
                          n_iter_no_change=None, presort='auto',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=Fals
e)
model score: 0.459
```

We can see that the model with the best performance is the Linear Regression model by far, with a score of .804.

Final Model

In [18]:
```
# select the relevant columns
data = orig_data[rel_cols]
data.head()
```

Out[18]:

	Spend	Impressions	CandidateBallotInformation	Gender	AgeBracket	CountryCode
0	1044	137185	NaN	NaN	35++	united states
1	279	94161	NaN	NaN	18+	united states
2	6743	3149886	NaN	NaN	NaN	united states
3	3698	573475	NaN	NaN	18+	united states
4	445	232906	NaN	NaN	25+	united kingdom

Building The Pipeline

In [19]:
```
# separate feature and target data
X = data.drop(['Impressions'], axis=1)
y = data.Impressions


# define columns

# columns to convert to booleans
bool_cols = ['CandidateBallotInformation', 'Gender', 'ElectoralDistrictID', 'M
etroID',
             'OsType', 'LocationType', 'AdvancedDemographics', 'Interests', 'S
egments']

cat_cols = ['CountryCode', 'AgeBracket', 'Language']
num_cols = ['Spend']
```

In [20]:
```
# adjust language
# get value counts
vc = X['Language'].value_counts()
# replace entries that occur < 50 times with 'other'
X.Language = X.Language.replace(vc[vc < 50].index, 'other')

# replace null entries with 'any'
X.Language = X.Language.fillna('any')
```

In [21]:
```
# convert certain columns to booleans
for col in bool_cols:
    X[col] = X[col].isna()
```

In [22]:
```
# adjust age bracket
X.AgeBracket = (
    X.AgeBracket.str.replace('[^\d]+\d*', '')
    # impute null values with 0, because these ads are agnostic to age
    .fillna(0)
    # convert data type from object to integer
    .astype('int32')
)
```

After some testing, the best model found was the linear regression model.

In [50]:
```
# create pipeline

cats = Pipeline([
    ('impute', SimpleImputer(strategy='constant', fill_value='NULL')),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    ('pca', PCA(svd_solver='full', n_components=.9))
])

ct = ColumnTransformer([
    ('cat_cols', cats, cat_cols+bool_cols),
    ('num_cols', SimpleImputer(strategy='constant', fill_value=0), num_cols)
], remainder='passthrough')

reg = LinearRegression()

pl = Pipeline([('feats', ct), ('reg', reg)])
```

In [51]:
```
# separate the data into training (70%) and testing (30%) sets
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3)

# fit the model to the training set
pl.fit(X_tr, y_tr)

# get predicted values
pred = pl.predict(X_ts)

# calculate the score using the test set
pl.score(X_ts, y_ts)
```

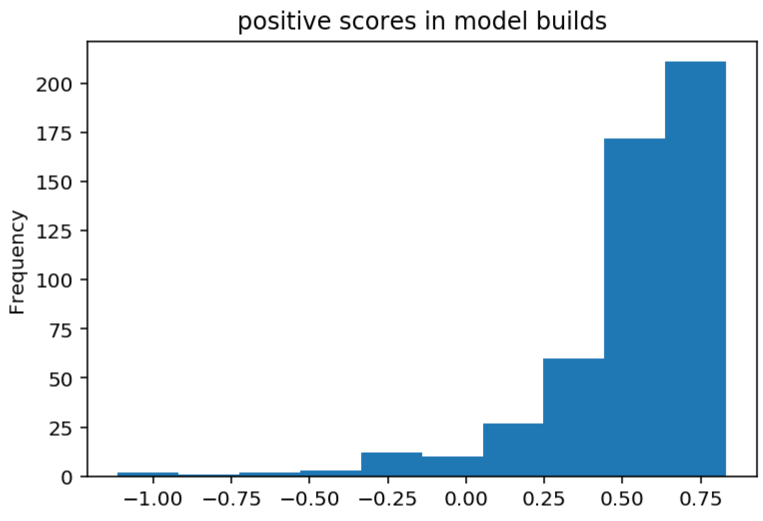
Out[51]:
```
0.6090304011960644
```

In [52]:
```
# calculate the rmse
np.sqrt(np.mean((pred - y_ts)**2))
```

Out[52]:
```
2600779.1436384353
```

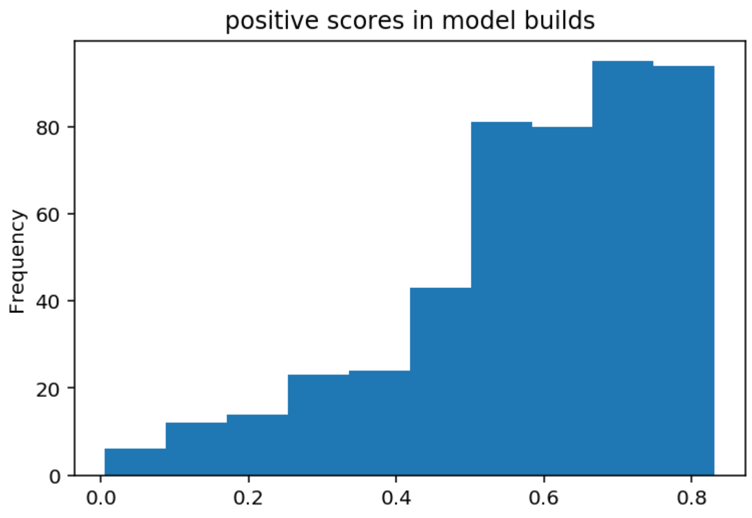
In [53]:
```
# lin reg
output = []
rmses = []
for _ in range(500):
    X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3)

    pl.fit(X_tr, y_tr)
    output.append(pl.score(X_ts, y_ts))

scores = pd.Series(output)
scores.plot(kind='hist', title='positive scores in model builds')
```

Out[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x25681aecba8>
```



In [54]:
```
scores[scores>0].plot(kind='hist', title='positive scores in model builds')
print('median score:', scores.median())
```

```
median score: 0.6021655324327034
```



From 500 tests, the median score is around .6, which is an improvement to the baseline model's score of about 0.52

Fairness Evaluation

To assess the fairness, we will examine the the test data set and the model's predictions.

In [55]:
```
# test set feature data
X_ts.head()
```

Out[55]:

	Spend	CandidateBallotInformation	Gender	AgeBracket	CountryCode	ElectoralD
3136	12	True	True	18	netherlands	True
2494	13	True	True	17	norway	True
2588	8	True	True	18	united states	False
3143	4706	True	True	23	france	True
350	196	True	True	18	united states	True

Combine the feature data with the target data and predictions

In [56]:
```
result_data = pd.concat([X_ts, pd.DataFrame({'Impressions':y_ts, 'pred':pred
})], sort=False, axis=1)
result_data.head()
```

Out[56]:

	Spend	CandidateBallotInformation	Gender	AgeBracket	CountryCode	ElectoralD
3136	12	True	True	18	netherlands	True
2494	13	True	True	17	norway	True
2588	8	True	True	18	united states	False
3143	4706	True	True	23	france	True
350	196	True	True	18	united states	True

To analyze the fairness of our regression model, we will look at the root mean squared error well specific demographics. To do this, it will be convenient to have a column of each ad's squared error.

In [57]:
```
result_data['sq_error'] = ((result_data['Impressions'] - result_data['pred'])*
*2)
result_data.head()
```

Out[57]:

	Spend	CandidateBallotInformation	Gender	AgeBracket	CountryCode	ElectoralD
3136	12	True	True	18	netherlands	True
2494	13	True	True	17	norway	True
2588	8	True	True	18	united states	False
3143	4706	True	True	23	france	True
350	196	True	True	18	united states	True

In [58]:
```
for col in bool_cols+cat_cols:
    print(result_data[[col, 'sq_error']].groupby(col).mean())
```

```
                                   sq_error
CandidateBallotInformation
False                          1.575894e+12
True                           1.154577e+13
            sq_error
Gender
False   1.508569e+12
True    1.174355e+13
                            sq_error
ElectoralDistrictID
False                   3.571664e+13
True                    1.022196e+13
             sq_error
MetroID
False    2.299004e+12
True     1.121456e+13
            sq_error
OsType
False   1.355699e+11
True    1.077517e+13
                  sq_error
LocationType
False         2.217597e+12
True          1.077118e+13
                          sq_error
AdvancedDemographics
False                 1.193084e+13
True                  1.066741e+13
               sq_error
Interests
False      6.702409e+12
True       1.192908e+13
              sq_error
Segments
False     1.491241e+13
True      2.217772e+12
                    sq_error
CountryCode
argentina       2.551251e+11
australia       3.372942e+13
austria         1.074433e+14
belgium         3.605998e+12
canada          2.185782e+13
denmark         1.135381e+12
finland         6.843116e+11
france          2.948166e+13
germany         1.648174e+12
india           7.305676e+11
ireland         4.370263e+11
kuwait          5.183968e+12
lithuania       4.587029e+10
netherlands     5.218414e+12
new zealand     7.416674e+08
nigeria         6.820553e+11
norway          2.334455e+12
poland          5.327193e+11
puerto rico     3.364707e+10
```

```
south africa    1.513583e+10
sweden          2.887477e+11
switzerland     8.325834e+10
turkey          1.625710e+13
united kingdom  2.308344e+12
united states   9.518389e+12
                sq_error
AgeBracket
0           5.661388e+12
14          3.987626e+11
15          2.435970e+13
16          6.558161e+13
17          4.111942e+13
18          7.884892e+12
19          5.366168e+11
20          3.334177e+12
21          1.213397e+12
22          3.290652e+10
23          3.749251e+13
24          2.462607e+12
25          3.920416e+12
26          1.590352e+12
28          1.289673e+11
30          1.062281e+12
31          1.669042e+10
33          2.273642e+10
34          1.619625e+11
35          1.080891e+12
              sq_error
Language
any       8.757534e+12
en        1.635547e+13
fr        4.557962e+12
nb        3.362125e+11
other     2.395859e+13
```

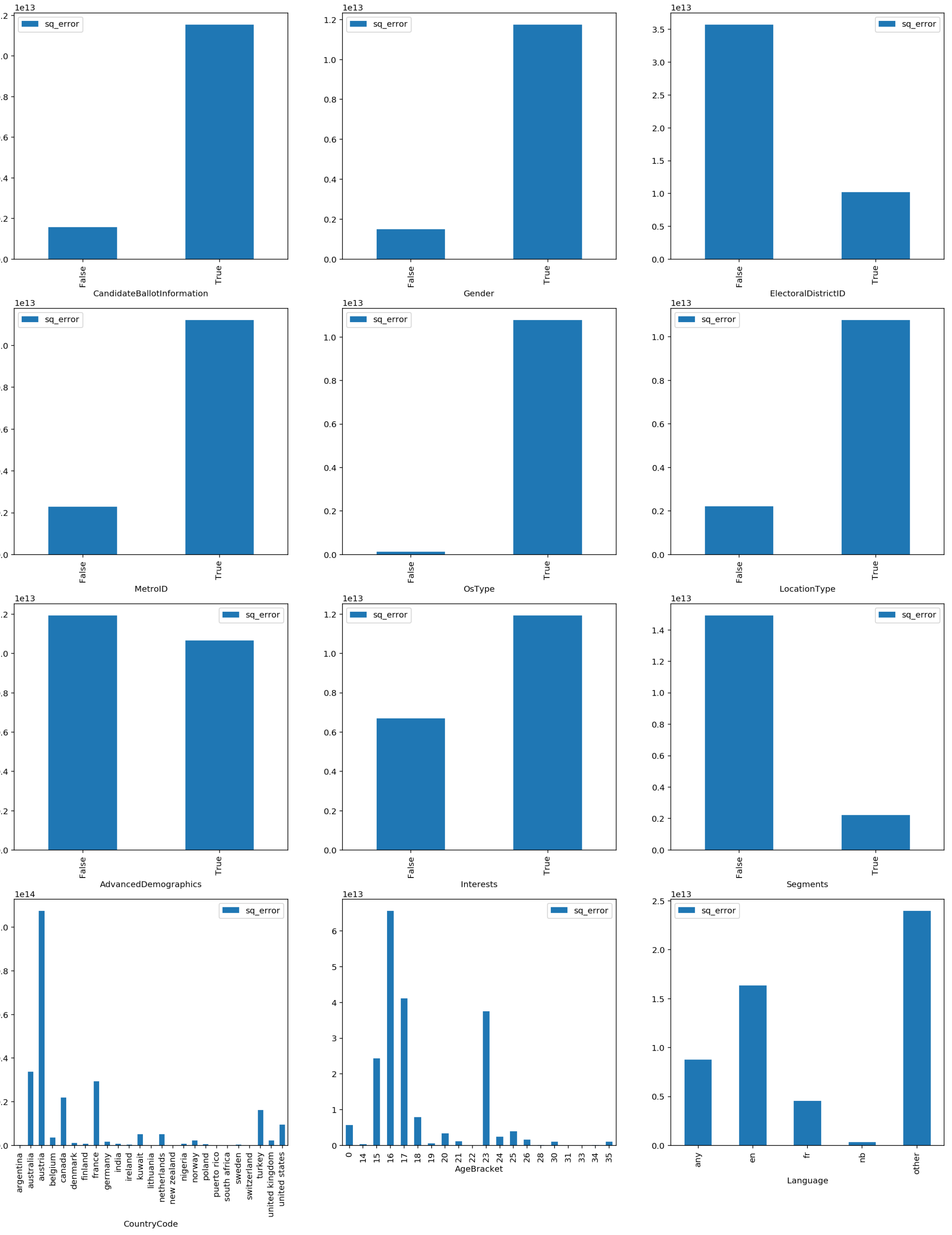
In [68]:
```
# for col in bool_cols+cat_cols:
#     result_data[[col, 'sq_error']].groupby(col).mean().plot(kind='bar')
```

In [69]:
```
# result_data[['Segments', 'sq_error']].groupby('Segments').mean()
```

In [61]:
```
# np.sqrt(np.mean((preds - y_ts)**2))
result_data['sq_error'].mean()
```

Out[61]:
```
10710756844270.656
```

In [67]:
```
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(20, 25))
for i, col in enumerate(bool_cols+cat_cols):
    result_data[[col, 'sq_error']].groupby(col).mean().plot(kind='bar', ax=axe
s[i//3, i%3])
```



From these plots, we can see that most of the boolean columns are biased, but this is not surprising since many of those columns are largely null/True (~90%). However, the segments column is more balanced.

```
In [87]: # display counts
         result_data.groupby('Segments').count().iloc[:, 0]

Out[87]: Segments
         False    663
         True     328
         Name: Spend, dtype: int64
```

Despite this, the ads that provide segments have much larger error than the ads that do not. To determine if this is actually bias and not chance, we will use a permutation test.

Use a permutation test to see if the difference in squared error can be explained by chance.

Null Hypothesis: The model's predictions, for the number of impressions an ad will receive, follow the same distribution regardless of whether or not the ad specified a segment.

Alternative Hypothesis: The model's predictions favor ads that did not specify a segment, predicting them more accurately.

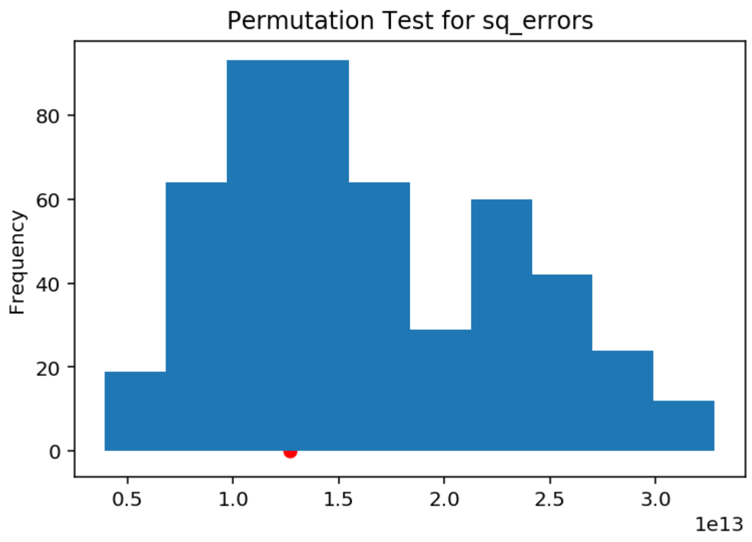
Test Statistic: Absolute difference in mean squared error.

Significance level: We set a significance level of 0.01

```
In [147]: # calculate the observed test statistic
          obs = result_data[['Segments', 'sq_error']].groupby('Segments').mean().diff().
          abs().iloc[-1]

          sq_errors = []
          for _ in range(500):
              # shuffle the values of the Segments column
              resample = (
                  result_data[['Segments', 'sq_error']]
                  .assign(Segments=result_data['Segments'].sample(frac=1.0, replace=Fals
          e).reset_index(drop=True))
              )

              # calculate test statistic
              ts = resample.groupby('Segments').mean().diff().abs().iloc[-1].values[0]

              sq_errors.append(ts)
```

```
In [148]: # calculate the percentage of test statistics equal to or more extreme than th
          e observed
          pd.Series(sq_errors >= obs[0]).mean()
          # this is the p-value

Out[148]: 0.644
```

In [150]:
```
# histogram plot of the absolute differences in squared error
pd.Series(sq_errors).plot(kind='hist', title='Permutation Test for sq_errors')
plt.scatter(obs, 0, c='r');
```



For our test, we got a p-value of 0.644, so the difference in squared error between ads that did and did not specify a segment can be explained by random chance. We now know that our model is not biased for whether or not an ad specified a segment; however, there may be other biases that exist within our model that we have yet to explore.