ads

December 8, 2019

1 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.

2 Summary of Findings

2.0.1 Introduction

The dataset consists of the information of all the political ads on Snapchat in 2018 and 2019. The dataset has info about the funding, the organization, the payer, number of audience reached, and specific range the organzation wants to reach. I observed that for usa ads, on average the `Spend` is higher. Through observation of what regions are targetted most frequently within the United States, I noticed that states like `Minnesota`, `Colorado`, `Florida`, `Virginia` are the most freqeuntly targetted states. I noticed that these are typical "Swing" states. Continuing on my question from project 3, I am interested in predicting the `Spend` based on the `CountryCode`, `RegionID`, `Gender`, `Impressions` (obviously) and other features.

The target variable is `Spend`, a quantitative value, so I will be using regressor for the prediction. The evaluation metric I am using is R squared score. The higher the R score, the better the model performs.

The application of this prediction model can be useful because for companies that have a goal in mind (5000 views), they can use the model to predict the amount of money they will spend on this ad, and be able to calculate marketing cost of the product and whether the ad is worth it.

2.0.2 Baseline Model

- I plan to create a decision tree regressor which will be trained on data columns: `CountryCode`(nominal), `RegionID`(nominal), `Gender`(nominal), and `Impressions` (quantitative).
- Preprocessing of the data include accessing the missingness of `CountryCode`, `RegionID`, `Gender`. Missingness to these columns, according to `README` means that the ad is targetted to all the regions or all the geners. So, I first replaced the `NaN` values in these two columns with `ALL`.
- `Impressions` is stardardized by using `StandardScaler` from sklearn library.
- `RegionID` ,`Gender`,`Countrycode` are OneHotEncoded using sklearn library.
- Then I decided to use a linear regression model to predict the `Spend`.
- I used train-test-split method to split my data into training set and test set. After fitting the model with training data, I use the model to predict on testing set. The R^2 score is calulated using r2_score method from sklearn.metrics library.
- The score fluctuates a lot, ranging from .9 to even negative value. Thus, the model overfits the data. A better model with additional features is needed to predict the `Spend`.

2.0.3 Final Model

- Additional Features:
  - `Length` is the time from the `StartDate` to `EndDate`. If the `EndDate` is `NaN`, the `Length` is replaced by the time length from the start date to today's date.
  - `Year` was which crv file the data came from (2018 or 2019). Maybe there is a price change from snapchat from 2018 to 2019.
- Current Features:
  - `Length`, quantitative, standardized using `StandardScaler`.
  - `Year`, nominal/ordinal, but I considered it to be nominal,so I one-hot-encoded it.
  - `Impressions`, quantitative, standardized using `StandardScaler`
  - Other nominal features: `Gender`, `CountryCode`, `RegionID`, all one-hot-encoded.
- Regressor Selection: The baseline model was a Linear Regression model. To improve that, I tried multiple other regressosr: `RandomForestRegressor`, `DecisionTreeRegressor` and `KNeighborsRegressor` in this sequence.
- Within in each regressor, I performed `GridSearch` to cross-validate, in order to find the best parameters such as `max_depth` and `n_estimators` and so on.
- After finding the best parameters, I used the best parameters to create a pipeline and test the R2_score.
- The `KNeighborsRegressor` performed the best out of the tree of them. It flucuates the least and has the highest R-score for both training and testing datasets.

2.0.4 Fairness Evaluation

- I will use a permutation test to evaluate the fairness of my model on a subset of the data. I choose to do the permutation test on whether the ad is in the United States or not. It might be more expensive to buy ads in the United States because there are many snapchat users in the United States. Indeed, non-usa ads tend to have less `Spend` than usa ads. However, does the model has a bias against non-usa ads in terms of accuracy (R2 score)?
- During observation, I noticed that there is a difference between the R2 score of usa ads and non-usa ads.
- To investigate whether this difference is significant or not, I performed a permutation test.
  - Null Hypothesis: The model predicts usa ads and non-usa ads equally well in terms of R2 score.
  - Alternative Hypothesis: The model doesn't predict the non-usa ads as well as it does with the usa ads, in terms of R2 score.
- The p-value was more than 0.05. Thus, we fail to reject the Null Hypothesis.

[364]:
```
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns
import os
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
import sklearn.preprocessing as pp
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import FunctionTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import r2_score
from datetime import datetime, date
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn import metrics
```

3 Code

[340]:
```
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
fp2018 = os.path.join('Data', '2018.csv')
fp2019 = os.path.join("Data", "2019.csv")
df2018 = pd.read_csv(fp2018)
```

```
df2019 = pd.read_csv(fp2019)

#then create a column "Year"
df2018['Year'] = "2018"
df2019["Year"] = "2019"
#concat two dfs into a singled df
df = pd.concat([df2018, df2019], ignore_index =True)
df.head()
df["Ended"] = ~df.EndDate.isna()
df.head()
df.Gender = df["Gender"].replace(np.nan, "BOTH")

us = df[["Spend", "Impressions", "RegionID", "Gender"]]
us.head()

df["StartDate"] = pd.to_datetime(df["StartDate"]).dt.date
df["EndDate"] = pd.to_datetime(df["EndDate"]).dt.date
df.head()
df["Length"] = df["EndDate"] - df["StartDate"]
df["TimeSinceStart"] = datetime.now().date() - df["StartDate"]
df["Length"].fillna(df["TimeSinceStart"], inplace = True)
df.loc[0]
df = df.fillna("ALL")
df.head()
```

[340]:
```
                                                ADID  \
0  2ac103bc69cce2d24b198e6a6d052dbff2c25ae9b6bb9e…
1  40ee7e900be9357ae88181f5c8a56baf6d5aab0e8d0f51…
2  c80ca50681d552551ceaf625981c0202589ca710d51925…
3  a3106af2289b62f57f63f4fb89753bdf94e2fadede0478…
4  7afda4224482eb70315797966b4dcdeb856df916df5bdc…

                                         CreativeUrl  Spend  Impressions  \
0  https://www.snap.com/political-ads/asset/69afd…    165        49446
1  https://www.snap.com/political-ads/asset/0885d…     17        23805
2  https://www.snap.com/political-ads/asset/a36b7…     60        12883
3  https://www.snap.com/political-ads/asset/46819…   2492       377236
4  https://www.snap.com/political-ads/asset/ee833…   5795       467760

    StartDate     EndDate                  OrganizationName  \
0  2018-11-01  2018-11-06          Bully Pulpit Interactive
1  2018-11-15  2018-11-24  Amnesty International Switzerland
2  2018-09-28  2018-10-10                   Chong and Koster
3  2018-10-27  2018-11-06       Middle Seat Consulting, LLC
4  2018-10-25  2018-11-06       Middle Seat Consulting, LLC

                                      BillingAddress  \
```

```
0  1140 Connecticut Ave NW, Suite 800,Washington,…
1                                                 CH
2  1640 Rhode Island Ave. NW, Suite 600,Washingto…
3                    Po Box 21600,Washington,20009,US
4                    Po Box 21600,Washington,20009,US

  CandidateBallotInformation        PayingAdvertiserName      …        \
0                        ALL             NextGen America      …
1                        ALL       Amnesty International      …
2                        ALL  Voter Participation Center      …
3                        ALL              Beto for Texas      …
4                        ALL              Beto for Texas      …

  Language     AdvancedDemographics Targeting Connection Type  \
0      ALL                      ALL                       ALL
1       de                      ALL                       ALL
2      ALL  Marital Status (Single)                       ALL
3      ALL                      ALL                       ALL
4      ALL                      ALL                       ALL

  Targeting Carrier (ISP) Targeting Geo - Postal Code  \
0                     ALL                         ALL
1                     ALL                         ALL
2                     ALL                         ALL
3                     ALL                         ALL
4                     ALL                         ALL

                                  CreativeProperties  Year Ended  Length  \
0  web_view_url:https://nextgenamerica.org/lookup…  2018  True  5 days
1                                                ALL  2018  True  9 days
2  web_view_url:https://www.voterparticipation.or…  2018  True 12 days
3  web_view_url:https://betofortexas.com/vote/?ut…  2018  True 10 days
4                                                ALL  2018  True 12 days

  TimeSinceStart
0       402 days
1       388 days
2       436 days
3       407 days
4       409 days

[5 rows x 31 columns]
```

3.0.1 Baseline Model

[241]:
```
us = df[["Spend", "Impressions", "Gender", "RegionID", "CountryCode"]]
from sklearn.linear_model import LogisticRegression

num_feat = ['Impressions']
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# Categorical columns and associated transformers
cat_feat = ['Gender', "RegionID", "CountryCode"]
cat_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown = "ignore"))
])

# preprocessing pipeline (put them together)
preproc = ColumnTransformer(transformers=[('num', num_transformer, num_feat),␣
 ↪('cat', cat_transformer, cat_feat)])

pl = Pipeline(steps=[('preprocessor', preproc), ('regressor',␣
 ↪LinearRegression())])
X = us.drop("Spend", axis = 1)
y = us["Spend"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
pl.fit(X_train, y_train)
print("Training data score", end = "")
print(pl.score(X_train, y_train))
print("Testing data score:", end = "")
print(pl.score(X_test, y_test))
```

```
Training data score0.786155943455033
Testing data score:0.5289925534404853
```

[341]:
```
#display the baseline model mechanism
pl.fit(X_train, y_train)
```

[341]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('num',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
StandardScaler(copy=True,
  with_mean=True,
```

```
with_std=True))],
                                                                         verbose=False),
                                                              ['Impressions']),
                                                             ('cat',
                                                              Pipeline(memory=None,
                                                                       steps=[('onehot',
OneHotEncoder(categorical_features=None,
 categories=None,
 drop=None,
 dtype=<class 'numpy.float64'>,
 handle_unknown='ignore',
 n_values=None,
 sparse=True))],
                                                                         verbose=False),
                                                              ['Gender', 'RegionID',
                                                               'CountryCode'])],
                                               verbose=False)),
                          ('regressor',
                           LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                            normalize=False))],
         verbose=False)
```

3.0.2 Final Model

[270]:

```
#Create columntransformer for the categorical and numeric features
df2 = df[["Spend", "CountryCode","Impressions", "Gender", "RegionID", "Year",␣
 ↪"Length"]]
df2["Length"]= pd.to_numeric(df2['Length'].dt.days, downcast='integer')

num_feat = ['Impressions', "Length"]
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# Categorical columns and associated transformers
cat_feat = ['Gender', "Year", "CountryCode", "RegionID"]
cat_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown = "ignore"))
])

# preprocessing pipeline (put categorical and numerical transformer together)
preproc = ColumnTransformer(transformers=[('num', num_transformer, num_feat),␣
 ↪('cat', cat_transformer, cat_feat)])
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:3:
SettingWithCopyWarning:
```

```
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/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[314]:

```
#try RandomForestRegressor, use GridSearch to find best params
X = df2.drop("Spend", axis = 1)
y = df2["Spend"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
preproc.fit(X_train)
#preprocess the data first because the entire pipeline cannot be passed to␣
 ↪GridSearchCV
data = preproc.transform(X_train)
parameters = {
    'n_estimators': [7,10,13,15,18, 20, 22, 25],
    "max_depth":[5, 10, 15, 20, 25, 30, 35, 40]

}
clf = GridSearchCV(RandomForestRegressor(criterion = "mse"), parameters, cv=5)
clf.fit(data, y_train)
clf.best_params_
#The best parameters were given : max_depth = 5, n_estimators = 10
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/model_selection/_search.py:814:
DeprecationWarning: The default of the `iid` parameter will change from True to
False in version 0.22 and will be removed in 0.24. This will change numeric
results when test-set sizes are unequal.
  DeprecationWarning)
```

[314]:

```
{'max_depth': 5, 'n_estimators': 10}
```

[324]:

```
#Testing RandomRegressor Results
X = df2.drop("Spend", axis = 1)
y = df2["Spend"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
pl_randomforest = Pipeline(steps=[('preprocessor', preproc), ('regressor',␣
 ↪RandomForestRegressor(n_estimators = 10,
                                                                            ␣
 ↪                  max_depth = 5,
                                                                            ␣
 ↪                  criterion = "mse"))])
pl_randomforest.fit(X_train, y_train)
print("Training data score", end = "")
print(pl_randomforest.score(X_train, y_train))
```

```
print("Testing data score:", end = "")
print(pl_randomforest.score(X_test, y_test))
```

```
Training data score0.8853819444284218
Testing data score:0.7084054931358765
```

[325]:
```
#displaying the random forest regressor mechanism
pl_randomforest.fit(X_train, y_train)
```

[325]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('num',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
StandardScaler(copy=True,
  with_mean=True,
  with_std=True))],
                                                           verbose=False),
                                                  ['Impressions', 'Length']),
                                                 ('cat',
                                                  Pipeline(memory=None,
                                                           steps=[('onehot',
OneHotEncod…
                                   verbose=False)),
                ('regressor',
                 RandomForestRegressor(bootstrap=True, criterion='mse',
                                       max_depth=5, 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))],
         verbose=False)
```

[333]:
```
#Testing DecisionTreeRegressor GridSearch
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
preproc.fit(X_train)
#preprocess the data first because the entire pipeline cannot be passed to␣
 ↪GridSearchCV
data = preproc.transform(X_train)
data2 = preproc.transform(X_test)
```

```
parameters = {
    "max_depth":[5, 6, 7, 8, 9, 10, 15, 20,None],
    'min_samples_split':[2,3,5,7,10,15,20],
    'min_samples_leaf':[2,3,5,7,10,15,20]

}
clf = GridSearchCV(DecisionTreeRegressor(), parameters, cv=5)
clf.fit(data, y_train)
clf.best_params_
#the best_params were : max_depth = 7, min_sample_leaf = 2, min_samples_split =␣
↪3
```

[333]:
```
{'max_depth': 7, 'min_samples_leaf': 2, 'min_samples_split': 3}
```

[334]:
```
#Testing DecisionTreeRegressor R score
X = df2.drop("Spend", axis = 1)
y = df2["Spend"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
pl_tree = Pipeline(steps=[('preprocessor', preproc), ('regressor',␣
↪DecisionTreeRegressor(max_depth=  7,
                                                                               ␣
↪                 min_samples_leaf = 2,
                                                                               ␣
↪                 min_samples_split = 3))])
pl_tree.fit(X_train, y_train)
print("Training data score", end = "")
print(pl_tree.score(X_train, y_train))
print("Testing data score:", end = "")
print(pl_tree.score(X_test, y_test))
#This result is really bad! So I am not going to use DecisionTreeRegressor
```

```
Training data score0.7744766751008756
Testing data score:0.6466212782997857
```

[335]:
```
#displaying Decision Tree Regressor pipeline mechanism
pl_tree.fit(X_train, y_train)
```

[335]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('num',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
StandardScaler(copy=True,
  with_mean=True,
```

```
  with_std=True))],
                                                              verbose=False),
                                                     ['Impressions', 'Length']),
                                                    ('cat',
                                                     Pipeline(memory=None,
                                                              steps=[('onehot',
OneHotEncod…
                                                              verbose=False),
                                                     ['Gender', 'Year',
                                                      'CountryCode',
                                                      'RegionID'])],
                                       verbose=False)),
                    ('regressor',
                     DecisionTreeRegressor(criterion='mse', max_depth=7,
                                           max_features=None, max_leaf_nodes=None,
                                           min_impurity_decrease=0.0,
                                           min_impurity_split=None,
                                           min_samples_leaf=2, min_samples_split=3,
                                           min_weight_fraction_leaf=0.0,
                                           presort=False, random_state=None,
                                           splitter='best'))],
         verbose=False)
```

[304]:

```
#Testing KNeightborsRegressor GridSearch: find the best parameters
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
preproc.fit(X_train)
data = preproc.transform(X_train)
parameters = {
    "n_neighbors":[2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50,␣
 ↪55],
    "weights":["uniform", "distance"],
    "metric":["euclidean", "manhattan"]

}
clf = GridSearchCV(KNeighborsRegressor(), parameters, cv=5)
clf.fit(data, y_train)
clf.best_params_
```

[304]:

```
{'metric': 'manhattan', 'n_neighbors': 15, 'weights': 'distance'}
```

[313]:

```
#Testing KNeighborsRegressor R score! Whether the parameters we are using␣
 ↪actually works well.
X = df2.drop("Spend", axis = 1)
y = df2["Spend"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
pl_neighbors = Pipeline(steps=[('preprocessor', preproc), ('regressor',␣
 ↪KNeighborsRegressor(n_neighbors=  15,
```

```
↪                   weights = "distance",
↪                   metric = "manhattan"))])
pl_neighbors.fit(X_train, y_train)
print("Training data score", end = "")
print(pl_neighbors.score(X_train, y_train))
print("Testing data score:", end = "")
print(pl_neighbors.score(X_test, y_test))
#This is the most stable model I have encountered. There is less overfitting
↪than the other regressors
```

```
Training data score0.9999999999983322
Testing data score:0.8982344848153475
```

[336]:
```
#display KNeighborsRegresor mechanism
pl_neighbors.fit(X_train, y_train)
```

[336]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('num',
                                                  Pipeline(memory=None,
                                                           steps=[('scaler',
StandardScaler(copy=True,
  with_mean=True,
  with_std=True))],
                                                           verbose=False),
                                                  ['Impressions', 'Length']),
                                                 ('cat',
                                                  Pipeline(memory=None,
                                                           steps=[('onehot',
OneHotEncod…
 categories=None,
 drop=None,
 dtype=<class 'numpy.float64'>,
 handle_unknown='ignore',
 n_values=None,
 sparse=True))],
                                                           verbose=False),
                                                  ['Gender', 'Year',
                                                   'CountryCode',
                                                   'RegionID'])],
                                   verbose=False)),
                ('regressor',
```

```
                KNeighborsRegressor(algorithm='auto', leaf_size=30,
                                    metric='manhattan', metric_params=None,
                                    n_jobs=None, n_neighbors=15, p=2,
                                    weights='distance'))],
        verbose=False)
```

3.0.3 Fairness Evaluation

[447]:
```
df2 = df[["Spend", "CountryCode","Impressions", "Gender", "RegionID", "Year",␣
 ↪"Length"]]
df2["Length"]= pd.to_numeric(df2['Length'].dt.days, downcast='integer')
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
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/indexing.html#indexing-view-versus-copy
```

[461]:
```
#Predictions of USA ads on Spend are higher than predictions of non-usa ads.
X = df2.drop('Spend', axis=1)
y = df2.Spend
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.35)
pl_neighbors = Pipeline(steps=[('preprocessor', preproc), ('regressor',␣
 ↪KNeighborsRegressor(n_neighbors=  15,
                                                                                  ␣
 ↪                 weights = "distance",
                                                                                  ␣
 ↪                 metric = "manhattan"))])
pl_neighbors.fit(X_train, y_train)
pl.score(X_test,y_test)

results = X_test
preds = pl.predict(X_test)

results['is_usa'] = (results.CountryCode =="united states")
results['prediction'] = preds
results['tag'] = y_test
results.head()
results.groupby('is_usa').prediction.mean().to_frame()
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:14:
SettingWithCopyWarning:
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/indexing.html#indexing-view-versus-copy

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:15:
SettingWithCopyWarning:
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/indexing.html#indexing-view-versus-copy
  from ipykernel import kernelapp as app
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:16:
SettingWithCopyWarning:
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/indexing.html#indexing-view-versus-copy
  app.launch_new_instance()
```

[461]:

is_usa	prediction
False	1383.830424
True	2185.370130

[462]:

```
#Does usa ads indeed on average have higher spend in reality?
results.groupby('is_usa').tag.mean().to_frame()
```

[462]:

is_usa	tag
False	1534.109504
True	2295.809735

[463]:

```
#Does the model perform equally well on USA ads and non-USA ads in terms of R^2␣
↪score?
(
    results
    .groupby('is_usa')
    .apply(lambda x: r2_score(x.tag, x.prediction))
    .rename('R2')
    .to_frame()
)
#The model seems to perform really well on United States ads, and not so well␣
↪on non-usa ads.
```

[463]:

```
             R2
is_usa
False   0.664736
True    0.813116
```

[464]:

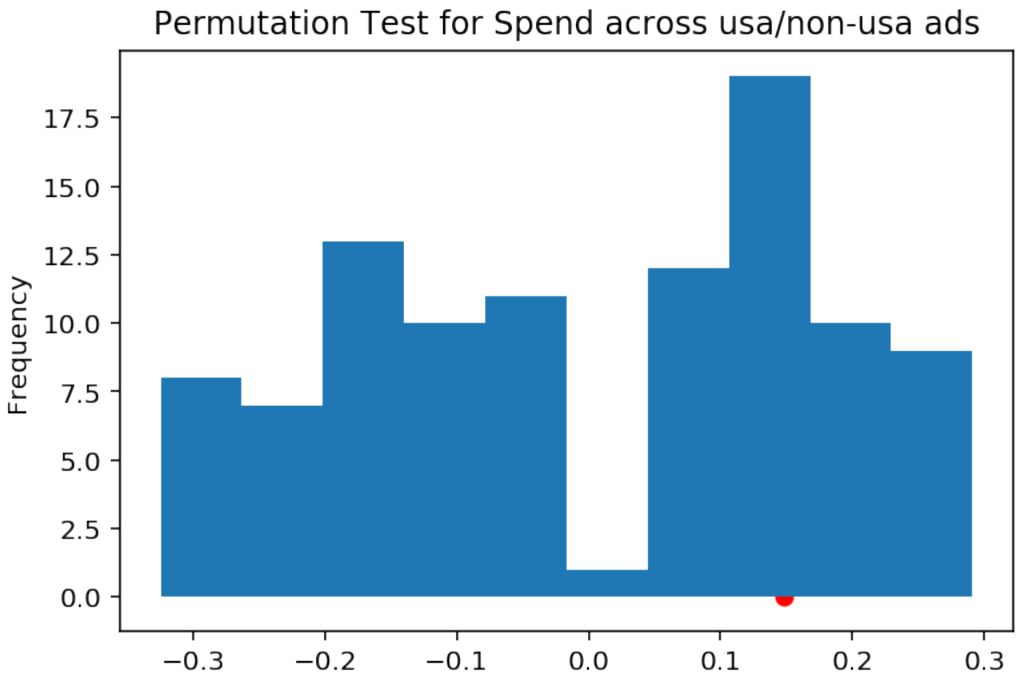
```
obs = results.groupby('is_usa').apply(lambda x: r2_score(x.tag, x.prediction)).
 ↪diff().iloc[-1]
print("The observation difference in R2 is: ", end = "")
print(obs)
metrs = []
for _ in range(100):
    s = (
        results[['is_usa', 'prediction', 'tag']]
        .assign(is_usa=results.is_usa.sample(frac=1.0, replace=False).
 ↪reset_index(drop=True))
        .groupby('is_usa')
        .apply(lambda x: r2_score(x.tag, x.prediction))
        .diff()
        .iloc[-1]
    )

    metrs.append(s)
```

```
The observation difference in R2 is: 0.14837963463320236
```

[465]:

```
print("The p-value of the permutation test is: ", end="")
print(pd.Series(obs >= metrs).mean())
pd.Series(metrs).plot(kind='hist', title='Permutation Test for Spend across usa/
 ↪non-usa ads')
plt.scatter(obs, 0, c='r');
#The p-value is larger than 0.05, thus we fail to reject the null hypothesis.
```

```
The p-value of the permutation test is: 0.78
```


Permutation Test for Spend across usa/non-usa ads
Frequency
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

```
[ ]:
```

```
[ ]:
```