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

For this project, I will be using the Snapchat Political Ads data from 2018 and 2019 to create a machine learning pipeline model. For the model, I will attempt to predict the money spent on an ad based on other features like number of views (Impressions) or other features like radius targeting. In order to predict the money spent on a given ad, I am planning to use a regression model as supposed to a classification model as I am not trying to classify the money spent and am focusing on just trying to predict the money spent. I chose to predict money spent because of the many factors about the ad and its influence on the money spent on the ad. For the evaluation metric of this model, I chose to use R2_scpre of the model. With the R2_score metric, we can see the difference between the predicted money spent accourding of the model and the money actual spent. An r score as close to 1 as possible is the main objective of this regression model.

Baseline Model

For the baseline model, I chose to use most of the columns excluding columns like Ad ID and urls which do not have any relationship with the money spent on the ad. I decided to not use the StartDate, EndDate, AdID, CreativeUrl, OrganizationName, BillingAddress, PayingAdvertiserName, CreativeProperties. For my features, all are nominal except for impressions which is quantitative. I decided to separate the rest of the columns of data into two groups which are either categorical or numerical data. For the categorical data, I decided to clean the data by replacing the missing values with what was recommended on the readme file which was similar to the procedure of the cleaning of data in the previous project. For the non missing data, I replaced this type of data with the value 'Some targeting' so that when you use one hot encoder, it does not results in hundreds of columns. For the categorical columns, I first used the SimpleImputer to fill all missing values with value 'NULL'. I then used OneHotEncoder ignoring unknown and with sparse set to False. For the numerical columns, I used SimpleImputer with strategy fill with the median to get rid of missing values. I applied both transformers within the pipeline model. I used train_test_split to split the data into data used to train the model and test data to check its accuracy with the pipeline score metric or r_2 score.

Final Model

For the final model, the first feature I engineered was creating a column that stated whether there were more than 3 kinds of targeting for each ad. The column contained the sum across the row of the label ‘Some targeting’ which I used to clean the data and also make it easier to do one-hot-encoding. For values less than 3, it was replaced with the label ‘2 or less’ and ‘3 or more’ for values more than 3. After creating the column, I did one hot encoding and primary component analysis to remove unnecessary components. I also created another feature by taking the months of the start date and used one-hot encoding and primary component analysis to see the influence that the time of the year affects how much money spent on the ad. For the currency code, I applied values based on currency with the strongest value and replace the currency code with the currency value with respect to others and then like the other features I did one-hot-encoding and primary component analysis. I also standardized the numerical columns to somewhat mitigate the effect on outliers in the model and imputed values with the median as it is less subject to outliers. For the categorical features, I replaced the missing values with the label ‘NULL’ and I also did one-hot-encoding in the baseline but I also added primary component analysis to get rid of unnecessary information. Instead of using the linear regression model and other linear models from the sklearn linear_model library, I decided to use the ensemble library and decided to use RandomForestRegressor

which significantly increased the r2_score of the model. To generate the training and testing data, I used train_test_split with the default 0.25 test size and random_state=0. In order to find the best model, I created a dictionary with all the models and and found the model with the highest cross validation score and used that as the best model in my grid search. In the grid search, I searched through different combination of parameters like n_estimators, max_features, and max_depth with 5 folds along the training data to find the best parameters for the best model. The r2_score went from around 0.83-0.87 with the RandomForestRegressor() with default parameters to around 0.87-0.9 with the best model which was the GradientBoostingRegressor() and the best parameters which were n_estimators being 200 and max_depth and max_features at 5.

Fairness Evaluation

For the fairness evaluation, I chose to explore whether the model is fair when applied to ads whether they have more or less than 100,000 impressions (views). For the null hypothesis, the models are fair and there is no difference in R2_score when it comes to ads with more or less than 100,000 views. For the alternate hypothesis, the model is unfair and there is a difference in R2_score when it comes to ads with more or less than 100,000 views. In order to test the null hypothesis, I am running a permutation test to test the fairness of the model with respect to number of views. I shuffled the views column and find the difference in R2_scores for ads with less than 100k and more than 100k views and conduct that for many repetitions and see how many of those differences are as extreme as the observed difference in R2_scores for ads with less than 100k and more than 100k views. After running the permutation test and getting a p_value of 0.0 which was the probability of getting a difference as extreme as the observed difference, the model supports the alternate hypothesis and there is a difference in R2_scores based on number of views with less R2_scores with more views.

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 [2]:
```
fp_2018 = os.path.join('data', 'PoliticalAds_2018.csv')
fp_2019 = os.path.join('data', 'PoliticalAds_2019.csv')
ads_2018 = pd.read_csv(fp_2018)
ads_2019 = pd.read_csv(fp_2019)
ads = pd.concat([ads_2018, ads_2019], ignore_index=True)
```

Convert Start and End Dates to Datetimes with time zone default UTC

In [3]:
```
ads['StartDate'] = pd.to_datetime(ads['StartDate'])
ads['EndDate'] = pd.to_datetime(ads['EndDate'])
ads.columns
```

Out[3]:
```
Index(['ADID', 'CreativeUrl', 'Currency Code', 'Spend', 'Impressions',
       'StartDate', 'EndDate', 'OrganizationName', 'BillingAddress',
       'CandidateBallotInformation', 'PayingAdvertiserName', 'Gender',
       'AgeBracket', 'CountryCode', 'Regions (Included)', 'Regions (Exclude
d)',
       'Electoral Districts (Included)', 'Electoral Districts (Excluded)',
       'Radius Targeting (Included)', 'Radius Targeting (Excluded)',
       'Metros (Included)', 'Metros (Excluded)', 'Postal Codes (Included)',
       'Postal Codes (Excluded)', 'Location Categories (Included)',
       'Location Categories (Excluded)', 'Interests', 'OsType', 'Segments',
       'Language', 'AdvancedDemographics', 'Targeting Connection Type',
       'Targeting Carrier (ISP)', 'CreativeProperties'],
      dtype='object')
```

Cleaning the Data

I used the readme file to fill missing values with appropriate values. For example if gender was empty, the read me stated that the targets all genders so I filled all the missing with all genders. I used the readme to fill in missing values because many of those missing values before cleaning actually have meaningful values so I filled those particular missing values so it would be what values are truly missing when assessing missingness.

In [4]:
```
ads['Gender'] = ads['Gender'].fillna('All genders')
ads['AgeBracket'] = ads['AgeBracket'].fillna('All ages')
ads['CountryCode'] = ads['CountryCode'].str.title()
ads['CandidateBallotInformation'] = ads['CandidateBallotInformation'].fillna(
'No information provided')
ads['Regions (Included)'] = ads['Regions (Included)'].fillna('None included')
ads['Regions (Excluded)'] = ads['Regions (Excluded)'].fillna('All regions')
ads['Electoral Districts (Included)'] = ads['Electoral Districts (Included)'].
fillna('No electoral districts targeting')
ads['Electoral Districts (Excluded)'] = ads['Electoral Districts (Excluded)'].
fillna('All electoral districts')
ads['Radius Targeting (Included)'] = ads['Radius Targeting (Included)'].fillna
('No radius targeting')
ads['Radius Targeting (Excluded)'] = ads['Radius Targeting (Excluded)'].fillna
('Full radius')
ads['Metros (Included)'] = ads['Metros (Included)'].fillna('No metros targetin
g')
ads['Metros (Excluded)'] = ads['Metros (Excluded)'].fillna('All metros')
ads['Postal Codes (Included)'] = ads['Postal Codes (Included)'].fillna('No pos
tal codes targeting')
ads['Postal Codes (Excluded)'] = ads['Postal Codes (Excluded)'].fillna('All po
stal codes')
ads['Location Categories (Included)'] = ads['Location Categories (Included)'].
fillna('No location categories targeting')
ads.rename({'Location Categories (Exlcuded)': 'Location Categories (Excluded)'
})
ads['Location Categories (Excluded)'] = ads['Location Categories (Excluded)'].
fillna('All location categories')
ads['Interests'] = ads['Interests'].fillna('No interest targeting')
ads['OsType'] = ads['OsType'].fillna('All operating systems')
ads['Language'] = ads['Language'].fillna('No language targeting')
ads['AdvancedDemographics'] = ads['AdvancedDemographics'].fillna('No 3rd party
data')
ads['Targeting Connection Type'] = ads['Targeting Connection Type'].fillna('No
internet connection targeting')
ads['Targeting Carrier (ISP)'] = ads['Targeting Carrier (ISP)'].fillna('All ca
rrier types')
```

Baseline Model

In [1]:
```
from sklearn.linear_model import LinearRegression, LogisticRegression, Lasso,
Ridge, LassoLars
from sklearn.preprocessing import FunctionTransformer, LabelEncoder, OrdinalEn
coder
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import GridSearchCV, KFold, cross_val_score
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.svm import SVR
from sklearn.metrics import r2_score

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, Gradien
tBoostingRegressor, ExtraTreesRegressor
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import Binarizer
```

Created a copy of the ads dataframe and saved the currencies and months as a series to be used later in the feature engineering for the final model. For the baseline model, I removed ADID, Start and End Date, Organization Name, Creative Url, BillingAddress, PayingAdvertiserName, and CreativeProperties because after applying unique to these columns it was clearly that there were too many unique values that would not neccessarily have any relationship with trying to predict how much money was spent on an ad.

In [6]:
```
ads_copy = ads.copy()
dates = ads_copy['StartDate']
curr = ads_copy['Currency Code']
months = ads_copy['StartDate'].dt.month
ads_copy = ads_copy.drop(['ADID', 'StartDate', 'EndDate', 'OrganizationName',
'CreativeUrl', 'BillingAddress', 'PayingAdvertiserName', 'CreativeProperties'
], axis=1)
```

Separated the remaining columns into categorical and numerical columns to apply transformers separately

In [7]:
```
types = ads_copy.drop(['Spend'], axis=1).dtypes
catcols = types.loc[types == np.object].index.to_list()
numcols = types.loc[types != np.object].index.to_list()
```

To make one-hot-encoding more efficient with the nominal data, I decided to replace all values that illustrated some type of targeting with the label 'Some targeting'. In the previous project, I filled the missing values with what was recommended on the readme file so all original non-missing values would have the label 'Some targeting.'

In [8]:
```
with_age = ads_copy.loc[ads_copy['AgeBracket'] != 'All ages'].index
ads_copy.loc[with_age, 'AgeBracket'] = 'Some targeting'
with_gender = ads_copy.loc[ads_copy['Gender'] != 'All genders'].index
ads_copy.loc[with_gender, 'Gender'] = 'Some targeting'
with_info = ads_copy.loc[ads_copy['CandidateBallotInformation'] != 'No informa
tion provided'].index
ads_copy.loc[with_info, 'CandidateBallotInformation'] = 'Some targeting'
with_region_incl = ads_copy.loc[ads_copy['Regions (Included)'] != 'None includ
ed'].index
ads_copy.loc[with_region_incl, 'Regions (Included)'] = 'Some targeting'
with_region_excl = ads_copy.loc[ads_copy['Regions (Excluded)'] != 'All region
s'].index
ads_copy.loc[with_region_excl, 'Regions (Excluded)'] = 'Some targeting'
with_elec_incl = ads_copy.loc[ads_copy['Electoral Districts (Included)'] != 'N
o electoral districts targeting'].index
ads_copy.loc[with_elec_incl, 'Electoral Districts (Included)'] = 'Some targeti
ng'
with_elec_excl = ads_copy.loc[ads_copy['Electoral Districts (Excluded)'] != 'A
ll electoral districts'].index
ads_copy.loc[with_elec_excl, 'Electoral Districts (Excluded)'] = 'Some targeti
ng'
with_rad_incl = ads_copy.loc[ads_copy['Radius Targeting (Included)'] != 'No ra
dius targeting'].index
ads_copy.loc[with_rad_incl, 'Radius Targeting (Included)'] = 'Some targeting'
with_rad_excl = ads_copy.loc[ads_copy['Radius Targeting (Excluded)'] != 'Full
 radius'].index
ads_copy.loc[with_rad_excl, 'Radius Targeting (Excluded)'] = 'Some targeting'
with_met_incl = ads_copy.loc[ads_copy['Metros (Included)'] != 'No metros targe
ting'].index
ads_copy.loc[with_met_incl, 'Metros (Included)'] = 'Some targeting'
with_met_excl = ads_copy.loc[ads_copy['Metros (Excluded)'] != 'All metros'].in
dex
ads_copy.loc[with_met_excl, 'Metros (Excluded)'] = 'Some targeting'
with_post_incl = ads_copy.loc[ads_copy['Postal Codes (Included)'] != 'No posta
l codes targeting'].index
ads_copy.loc[with_post_incl, 'Postal Codes (Included)'] = 'Some targeting'
with_post_excl = ads_copy.loc[ads_copy['Postal Codes (Excluded)'] != 'All post
al codes'].index
ads_copy.loc[with_post_excl, 'Postal Codes (Excluded)'] = 'Some targeting'
with_loc_incl = ads_copy.loc[ads_copy['Location Categories (Included)'] != 'No
location categories targeting'].index
ads_copy.loc[with_loc_incl, 'Location Categories (Included)'] = 'Some targetin
g'
with_loc_excl = ads_copy.loc[ads_copy['Location Categories (Excluded)'] != 'Al
l location categories'].index
ads_copy.loc[with_loc_excl, 'Location Categories (Excluded)'] = 'Some targetin
g'
with_interest = ads_copy.loc[ads_copy['Interests'] != 'No interest targeting']
.index
ads_copy.loc[with_interest, 'Interests'] = 'Some targeting'
with_os = ads_copy.loc[ads_copy['OsType'] != 'All operating systems'].index
ads_copy.loc[with_os, 'OsType'] = 'Some targeting'
with_lang = ads_copy.loc[ads_copy['Language'] != 'No language targeting'].inde
x
ads_copy.loc[with_lang, 'Language'] = 'Some targeting'
with_dem = ads_copy.loc[ads_copy['AdvancedDemographics'] != 'No 3rd party dat
```

```
a'].index
ads_copy.loc[with_dem, 'AdvancedDemographics'] = 'Some targeting'
with_target = ads_copy.loc[ads_copy['Targeting Connection Type'] != 'No intern
et connection targeting'].index
ads_copy.loc[with_target, 'Targeting Connection Type'] = 'Some targeting'
with_carr = ads_copy.loc[ads_copy['Targeting Carrier (ISP)'] != 'All carrier t
ypes'].index
ads_copy.loc[with_carr, 'Targeting Carrier (ISP)'] = 'Some targeting'
```

In [9]:
```
ads_copy.head()
```

Out[9]:

	Currency Code	Spend	Impressions	CandidateBallotInformation	Gender	AgeBracket	CountryCode
0	EUR	6000	2080852	No information provided	All genders	All ages	Denmark
1	USD	306	164497	No information provided	All genders	All ages	United States
2	GBP	445	232906	No information provided	All genders	Some targeting	United Kingdom
3	USD	60	12883	No information provided	Some targeting	Some targeting	United States
4	USD	3403	964607	No information provided	All genders	Some targeting	United States

5 rows × 26 columns

For the nominal data, I imputed the missing values if there were any with the value 'NULL' even though I imputed these missing values in the previous project but if there was missingness the SimpleImputer would take care of those issues. I also used OneHotEncoder to separate the data and see how the values of each column influenced money spent. For the quantative data, I simply imputed the missing values with the median because the median is more resistant to outliers that simply imputing with the mean.

In [10]:
```
cat_transformer = Pipeline(steps=[('imp', SimpleImputer(strategy='constant', f
ill_value='NULL')),
                                  ('ohe', OneHotEncoder(handle_unknown='ignore'
, sparse=False))])
num_transformer = Pipeline(steps=[('imp', SimpleImputer(strategy='median'))])
```

Created a column transformer to be applied in the pipeline for the baseline model

In [11]:
```
preproc = ColumnTransformer(transformers=[('cat', cat_transformer, catcols),
                                          ('num', num_transformer, numcols)])
```

Created a baseline model using the column transformer of the quantitative and nominal data and used linear regression as the regression model

In [12]:
```
pl_base = Pipeline(steps=[('preproc', preproc), ('reg', LinearRegression())])
```

Split the data into training and testing data

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(ads_copy.drop(['Spend'], a
xis=1), ads_copy.Spend, random_state=0)
```

In [14]:
```
pl_base.fit(X_train, y_train)
```

Out[14]:
```
Pipeline(memory=None,
         steps=[('preproc',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('cat',
                                                  Pipeline(memory=None,
                                                           steps=[('imp',
                                                                   SimpleImpu
ter(add_indicator=False,

copy=True,

fill_value='NULL',

missing_values=nan,

strategy='constant',

verbose=0)),
                                                                  ('ohe',
                                                                   OneHotEnco
der(categories='auto',

drop=None,...
                                                    'Targeting Connection Typ
e',
                                                    'Targeting Carrier (IS
P)']),
                                                 ('num',
                                                  Pipeline(memory=None,
                                                           steps=[('imp',
                                                                   SimpleImpu
ter(add_indicator=False,

copy=True,

fill_value=None,

missing_values=nan,

strategy='median',

verbose=0))],
                                                                  verbose=False),
                                                  ['Impressions'])],
                                   verbose=False)),
                ('reg',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=Non
e,
                                  normalize=False))],
         verbose=False)
```

Predictions based on the data

In [15]:
```
preds = pl_base.predict(ads_copy.drop('Spend', axis=1))
```

R2_score of the baseline model

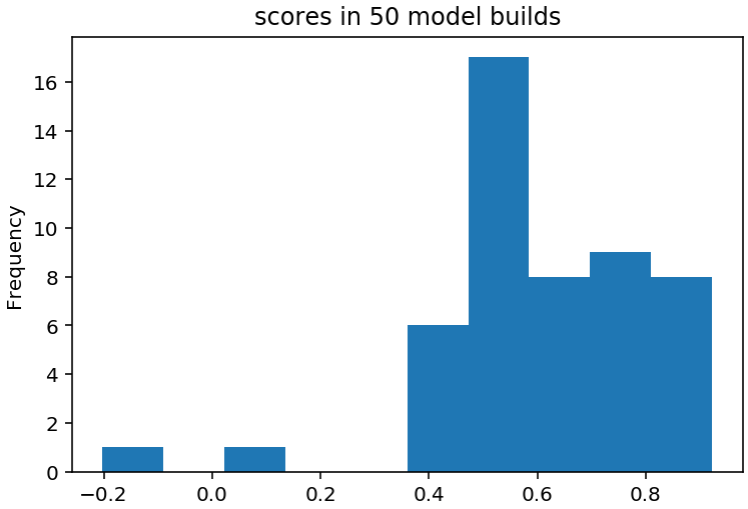
In [16]:
```
r_score = pl_base.score(X_test, y_test)
r_score
```

Out[16]: 0.5907277536558709

Histogram of the R2_scores produced by the baseline model

In [17]:
```
out = []
for _ in range(50):
    X_tr, X_ts, y_tr, y_ts = train_test_split(ads_copy.drop('Spend', axis=1),
ads_copy.Spend)
    pl_base.fit(X_tr, y_tr)
    score = pl_base.score(X_ts, y_ts)
    out.append(score)
```

In [18]:
```
pd.Series(out).plot(kind='hist', title='scores in 50 model builds');
```



Final Model

Created a new transformed to select the columns of the dataframe

In [19]:
```
class SelectColumns(BaseEstimator, TransformerMixin):
    def __init__(self, cols):
        self.cols = cols
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.cols]
```

First Engineered Feature

I summed the number of columns with the label Some targeting across each row and created a column of the dataframe stating whether an ad had at least 3 types of targeting or not. I would eventually use one-hot-encoding for this newly created column

In [20]:
```
cat_df = ads_copy[catcols].copy()
cat = cat_df.drop(['Currency Code'], axis=1)
targeting = (cat == 'Some targeting').astype(int).sum(axis=1)
more_than_three = targeting >= 3
more_than_three_repl = {
    False: '2 or less',
    True: '3 or more'
}
three_types_plus = more_than_three.replace(more_than_three_repl)
three_types_plus
ads_copy['3+ Types of Targeting'] = three_types_plus
catcols = list(cat.columns)
```

Creating a column for months of ad release

In [21]:
```
ads_copy['Month'] = months
```

Transformer that takes the newly created feature for whether an ad has at least 3 types of targeting and one-hot-encodes that newly created nominal column as well as uses primary component analysis to get rid of extra information

In [22]:
```
targeting_transformer = Pipeline(steps=[('ohe', OneHotEncoder(handle_unknown=
'ignore', sparse=False)),
                                        ('pca', PCA(svd_solver='full', n_compon
ents=0.99))])
```

Transformer that takes the newly created feature for month ad ad was released and one-hot-encodes that newly created nominal column as well as uses primary component analysis to get rid of extra information. In order to impute missing values I chose to fill it with the most frequent value because median and mean did not make sense when dealing with months as it is nominal and not quantative

In [23]:
```
month_transformer = Pipeline(steps=[('mon', SelectColumns(['Month'])),
                                    ('imp', SimpleImputer(strategy='most_frequ
ent')),
                                     ('ohe', OneHotEncoder(handle_unknown='ign
ore', sparse=False)),
                                     ('pca', PCA(svd_solver='full', n_componen
ts=0.99))])
```

Dictionary to replace all currency codes with its rating based on strength of currency. 0-4 for currencies with 0 being the currency with the lowest value

In [24]:
```
currency_dict = {
    'EUR': 3,
    'USD': 2,
    'CAD': 1,
    'AUD': 0,
    'GBP': 4
}
curr_repl = curr.replace(currency_dict)
ads_copy['Currency Code'] = curr_repl
```

Transformer to impute currencies with most frequent since it is an ordinal column and uses one-hot-encoding and primary component analysis to drop extra information

In [25]:
```
curr_transformer = Pipeline(steps=[('curr', SelectColumns(['Currency Code'])),
                                   ('imp', SimpleImputer(strategy='most_freque
nt')),
                                  ('ohe', OneHotEncoder(handle_unknown='ignor
e', sparse=False)),
                                  ('pca', PCA(svd_solver='full', n_components=
0.99))])
```

In [26]:
```
cat_transformer = Pipeline(steps=[('cat', SelectColumns(catcols)),
                                  ('imp', SimpleImputer(strategy='constant', f
ill_value='NULL')),
                                 ('ohe', OneHotEncoder(handle_unknown='ignore'
, sparse=False)),
                                  ('PCA', PCA(svd_solver='full', n_components=
0.99))])
```

Unlike the baseline model, I am using standard scaling on the quantative data to reduce the effect of outliers on predictions based on the pipeline

In [27]:
```
scale_transformer = Pipeline(steps=[('by', SelectColumns(numcols)),
                                    ('imp', SimpleImputer(strategy='median')),
                                    ('scale', StandardScaler())])
```

Column transformer to implement all the previous transformers on the dataframe

In [28]:
```
proc = ColumnTransformer(transformers=[('cat', cat_transformer, catcols),
                                       ('target', targeting_transformer, ['3+
 Types of Targeting']),
                                       ('curr', curr_transformer, ['Currency C
ode']),
                                       ('scale', scale_transformer, numcols),
                                      ('month', month_transformer, ['Month'])
                                      ])
```

Creating a new pipeline with the column transformer and instead using RandomForestRegressor as the regression model

In [29]:
```
pl = Pipeline(steps=[('proc', proc), ('reg', RandomForestRegressor())])
```

Split the data into training and testing data

In [30]:
```
X_train, X_test, y_train, y_test = train_test_split(ads_copy.drop(['Spend'], a
xis=1), ads_copy.Spend, random_state=0)
```

In [31]:
```
pl.fit(X_train, y_train)
```

Out[31]:
```
Pipeline(memory=None,
         steps=[('proc',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('cat',
                                                  Pipeline(memory=None,
                                                           steps=[('cat',
                                                                   SelectColu
mns(cols=['CandidateBallotInformation',

'Gender',

'AgeBracket',

'CountryCode',

'Regions '

'(Included)',

'Regions '

'(Excluded)',

'Electoral '

'Districts '

'(Included)',

'...
                 RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                       criterion='mse', max_depth=None,
                                       max_features='auto', max_leaf_nodes=No
ne,
                                       max_samples=None,
                                       min_impurity_decrease=0.0,
                                       min_impurity_split=None,
                                       min_samples_leaf=1, min_samples_split=
2,
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators=100, n_jobs=None,
                                       oob_score=False, random_state=None,
                                       verbose=0, warm_start=False))],
         verbose=False)
```

Predictions based on the training data

In [32]:
```
preds = pl.predict(ads_copy.drop('Spend', axis=1))
```

R2_score of the improved model

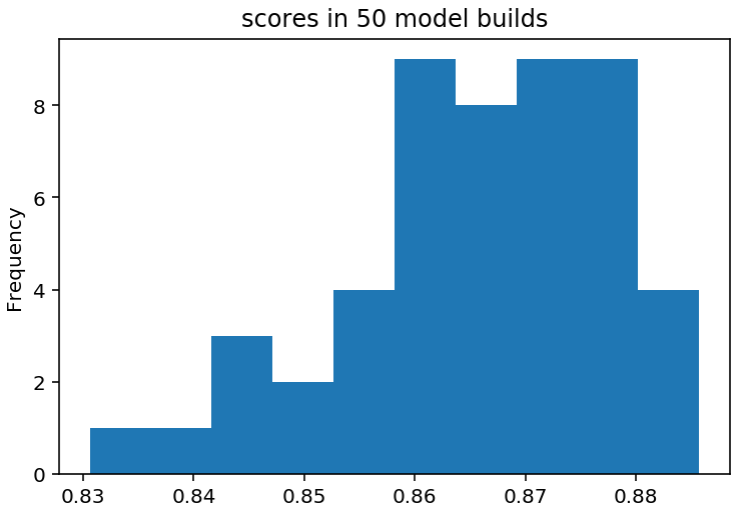
In [33]:
```
r_score = pl.score(X_test, y_test)
r_score
```

Out[33]: 0.8764331821811764

Histograms of the R2_scores produced by the improved model

In [34]:
```
out = []
for _ in range(50):
    X_tr, X_ts, y_tr, y_ts = train_test_split(ads_copy.drop(['Spend'], axis=1
), ads_copy.Spend, random_state=0)
    pl.fit(X_tr, y_tr)
    score = pl.score(X_ts, y_ts)
    out.append(score)
```

In [35]:
```
pd.Series(out).plot(kind='hist', title='scores in 50 model builds');
```



We want to find the best model so I created a dictionary of all potential models whether they come from the linear model library or the ensemble library

In [36]:
```
models = []
model_stds = {}
model_scores = {}
models.append(('LR', LinearRegression()))
models.append(('KNR', KNeighborsRegressor()))
models.append(('DTR', DecisionTreeRegressor()))
models.append(('AB', AdaBoostRegressor()))
models.append(('GBM', GradientBoostingRegressor()))
models.append(('RF', RandomForestRegressor()))
models.append(('ET', ExtraTreesRegressor()))
models = dict(models)
```

I created training and testing data and used each model in the previous dictionary. For each model, I created a pipeline applying the previous column transformer from the improved model and changed the model by looping through the dictionary of models. For each model I found the median of the cross validation scores with five folds across the training data.

In [37]:
```
from sklearn.model_selection import cross_validate
```

In [38]:
```
X_train = ads_copy.drop('Spend', axis=1)
y_train = ads_copy.Spend
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, random_s
tate=0)
for name, model in models.items():
    pl = Pipeline(steps=[('proc', proc), ('mod', model)])
    pl.fit(X_train, y_train)
    score = pl.score(X_test, y_test)
    cross_val = cross_val_score(pl, X_train, y_train, cv=5)
    model_scores[name] = np.median(cross_val)
```

Finding the name of the model with the highest cross validation score median

In [39]:
```
max_score = max(model_scores.values())
for name, score in model_scores.items():
    if score == max_score:
        model_name = name
```

Finding the best model to eventually search for best parameters

In [40]:
```
best_model = models[model_name]
best_model
```

Out[40]:
```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, 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='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=Fals
e)
```

Pipeline using the best regression model from the previous search

In [41]:
```
pl_best_model = Pipeline(steps=[('proc', proc), ('reg', best_model)])
```

In [42]:
```
#pl_best_model.get_params().keys()
```

Possible parameters to search through to find best parameters

In [43]:
```
params = {
    'reg__n_estimators': [2, 5, 10, 25, 100, 200],
    'reg__max_features': [5, 20, None],
    'reg__max_depth': [2, 5, 9, None]
}
```

Conduct a 5 fold grids search for the best parameters for the best model

In [44]:
```
grids = GridSearchCV(pl_best_model, param_grid=params, cv=5)
```

In [45]:
```
grids.fit(X_train, y_train)
```

Out[45]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('proc',
                                        ColumnTransformer(n_jobs=None,
                                                          remainder='drop',
                                                          sparse_threshold=0.
3,
                                                          transformer_weights
=None,
                                                          transformers=[('ca
t',
                                                                         Pipe
line(memory=None,

steps=[('cat',

SelectColumns(cols=['CandidateBallotInformation',

'Gender',

'AgeBracket',

'CountryCode',

'Regions '

'(Included)',

'Regions '

'(Exc...
                                                                        presort='de
precated',
                                                                        random_stat
e=None,
                                                                        subsample=
1.0,
                                                                        tol=0.0001,
                                                                        validation_
fraction=0.1,
                                                                        verbose=0,
                                                                        warm_start=
False))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'reg__max_depth': [2, 5, 9, None],
                         'reg__max_features': [5, 20, None],
                         'reg__n_estimators': [2, 5, 10, 25, 100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [46]:
```
grids.best_params_
```

Out[46]:
```
{'reg__max_depth': 5, 'reg__max_features': 5, 'reg__n_estimators': 200}
```

Finding the best and final model using the grid search best estimator

In [87]:
```
pl_best = grids.best_estimator_
```

In [88]:
```
pl_best.fit(X_train, y_train)
```

Out[88]:
```
Pipeline(memory=None,
         steps=[('proc',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('cat',
                                                  Pipeline(memory=None,
                                                           steps=[('cat',
                                                                   SelectColu
mns(cols=['CandidateBallotInformation',

'Gender',

'AgeBracket',

'CountryCode',

'Regions '

'(Included)',

'Regions '

'(Excluded)',

'Electoral '

'Districts '

'(Included)',

'...
                                           learning_rate=0.1, loss='ls',
                                           max_depth=5, max_features=5,
                                           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=200,
                                           n_iter_no_change=None,
                                           presort='deprecated',
                                           random_state=None, subsample=1.0,
                                           tol=0.0001, validation_fraction=0.
1,
                                           verbose=0, warm_start=False))],
         verbose=False)
```

Predictions based on the best and final model

In [89]:
```
preds = pl_best.predict(X_test)
```

R2_score using the best model and best parameters

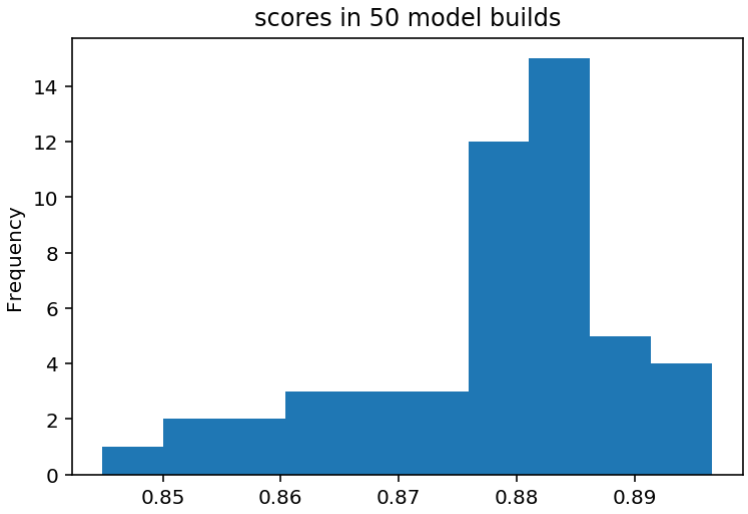
In [90]:
```
pl_best.score(X_test, y_test)
```

Out[90]:
```
0.8824603828789102
```

Histogram of the R2_score produced by the best model with the best parameters

In [91]:
```
out = []
for _ in range(50):
    X_tr, X_ts, y_tr, y_ts = train_test_split(ads_copy.drop(['Spend'], axis=1
), ads_copy.Spend, random_state=0)
    pl_best.fit(X_tr, y_tr)
    score = pl_best.score(X_ts, y_ts)
    out.append(score)
```

In [92]:
```
pd.Series(out).plot(kind='hist', title='scores in 50 model builds');
```



Fairness Evaluation

Null Hypothesis: My model is fair, the R2_scores for my two subsets are roughly the same

Alternate Hypothesis: My model is unfair, the R2_scores for my subsets are different as there are higher R2_scores for ads with less than 100k views

First to split the ads into subsets with more or less than 100,000 views, I used binarizer with the threshold at 100k and then selected indices based on the binarizer values to split the dataset into two subsets with one with only ads with less than 100k views and one with onyl ads with more than 100k views

In [75]:
```
binarize_views = Binarizer(threshold=100000)
new_impressions = binarize_views.fit_transform(ads_copy[['Impressions']])
df = pd.DataFrame(new_impressions, columns=['Views'])
less_than_hundred_indices = df[df['Views'] == 0].index
hundred_plus_indices = df[df['Views'] == 1].index
less_than_hundred = ads_copy.loc[less_than_hundred_indices]
more_than_hundred = ads_copy.loc[hundred_plus_indices]
```

I then conduct a permutation test by shuffling the views (Impressions) and finding the R2_scores of both subsets and then calculate the difference (less than 100k - more than 100k) and add that value to the list of differences. I also calculate the R2_score of the observed subsets and compare and see how many differences are as extreme as the observed difference (less-more)

In [93]:
```
n_repetitions = 50
diffs = []
rmses = []
for _ in range(n_repetitions):
    shuffled_less_views = less_than_hundred['Impressions'].sample(replace=Fals
e, frac=1).reset_index(drop=True)
    shuffled_less = less_than_hundred.assign(Impressions=shuffled_less_views)
    shuffled_more_views = more_than_hundred['Impressions'].sample(replace=Fals
e, frac=1).reset_index(drop=True)
    shuffled_more = less_than_hundred.assign(Impressions=shuffled_more_views)
    X_less_train, X_less_test, y_less_train, y_less_test = train_test_split(sh
uffled_less.drop('Spend', axis=1), shuffled_less.Spend, random_state=0)
    X_more_train, X_more_test, y_more_train, y_more_test = train_test_split(sh
uffled_more.drop('Spend', axis=1), shuffled_more.Spend, random_state=0)
    pl_best.fit(X_less_train, y_less_train)
    preds_less = pl_best.predict(X_less_test)
    pl_best.fit(X_more_train, y_more_train)
    preds_more = pl_best.predict(X_more_test)
    less_r_2 = r2_score(y_less_test, preds_less)
    more_r_2 = r2_score(y_more_test, preds_more)
    diff_score = less_r_2 - more_r_2
    diffs.append(diff_score)
X_obs_less_train, X_obs_less_test, y_obs_less_train, y_obs_less_test = train_t
est_split(less_than_hundred.drop('Spend', axis=1), less_than_hundred.Spend, ra
ndom_state=0)
X_obs_more_train, X_obs_more_test, y_obs_more_train, y_obs_more_test = train_t
est_split(more_than_hundred.drop('Spend', axis=1), more_than_hundred.Spend, ra
ndom_state=0)
pl_best.fit(X_obs_less_train, y_obs_less_train)
preds_obs_less = pl_best.predict(X_obs_less_test)
pl_best.fit(X_obs_more_train, y_obs_more_train)
preds_obs_more = pl_best.predict(X_obs_more_test)
less_r_2_obs = r2_score(y_obs_less_test, preds_obs_less)
more_r_2_obs = r2_score(y_obs_more_test, preds_obs_more)
```

I calculated the R2_score of the observed subsets and compare and see how many differences are as extreme as the observed difference (less-more)

In [97]:
```
obs_diff = less_r_2_obs - more_r_2_obs
```

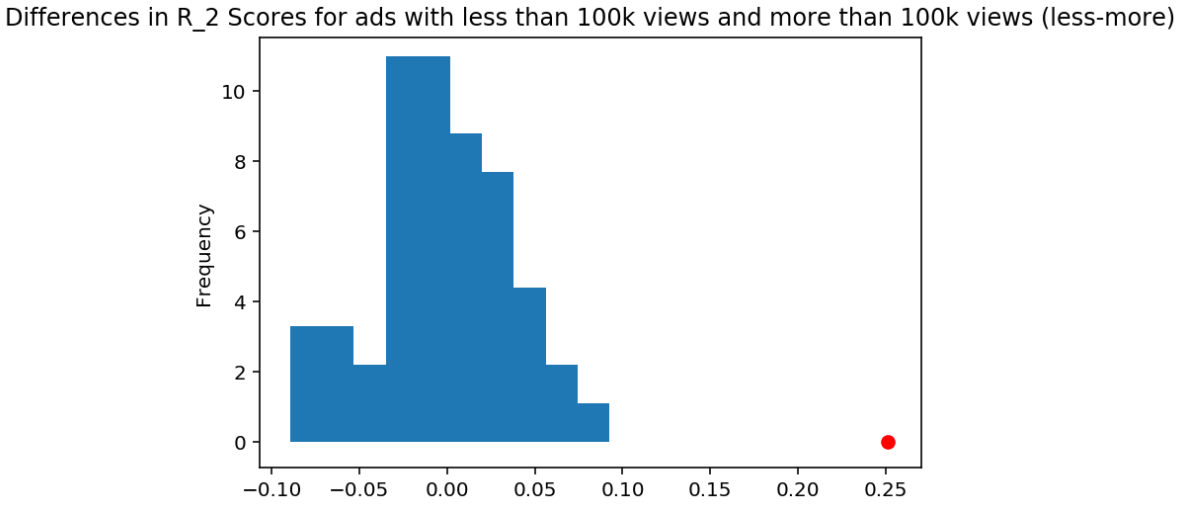
Finding p_value to see how extreme the observed difference is

In [98]:
```
p_val = np.count_nonzero(diffs > obs_diff) / n_repetitions
p_val
```

Out[98]: 0.0

Histogram of all the difference in R2_scores

In [99]:
```
title = 'Differences in R_2 Scores for ads with less than 100k views and more
 than 100k views (less-more)'
pd.Series(diffs).plot(kind='hist', density=True, title=title)
plt.scatter(obs_diff, 0, color='red', s=40);
```



Model seems to show increased R_2 scores for ads with less than 100k views which favors the alternate hypothesis that there are higher R2_scores for ads with less than 100k views. The model does not seem to be as accurate when it has to deal with large outliers