U.S. Housing Rent Prediction

Sean Tsung (3034159522) Vinson Chiu (3032731067)
Jianing Yu (3034036490) Zijun Zhang (3034071421)
Felicia Xu (3032719042)

University of California, Berkeley

IEOR 142
Fall 2021

December 17, 2021

Contents

1 Introduction **2**

2 Exploratory Data Analysis **2**

3 Analytic Modeling **3**

3.1 Linear Regression 3

3.2 Decision Tree Regression 3

3.3 Random Forest Regression 4

3.4 Gradient Boosted Regression 4

3.5 Neural Networks 4

3.6 Summary and Results 4

4 Impact and Future Improvements **5**

5 Appendix **6**

1 Introduction

We decided that an exploration of factors that might be predictive of rent prices would be highly educational and applicable to our current lives. People who are searching for a place to rent may find themselves overwhelmed by the vast range of rent prices and housing information available, and looking for a new home could be even more difficult after the pandemic. Ever since the emergence of COVID-19, the quantitative easing released by the Federal Reserve has led to low mortgage rates, which unprecedentedly attracted many more people looking to buy houses. Buyers challenged each other and drove up housing prices through competitive bidding. Therefore, the demand for rentals has increased as it is becoming increasingly difficult to buy a home. The pressure to rent the diminishing available homes may force some people to make quick and uninformed decisions. Consequently, it would be meaningful if we could enlighten home renters and allow them to make careful choices through our various forms of data analyses on rent prices. We hope to use a variety of techniques to not only predict housing prices across the United States based on features of the houses themselves, such as the number of bedrooms, but also on the basis of regional factors such as the crime rate in the area where the house is situated. Two key questions we hope to address include the following:

1. What factors might be important for predicting the rent price?
2. How might rent prices be related to regional factors like the crime rate, median income, and number of public schools in the area?

2 Exploratory Data Analysis

We combined five datasets:

1. A housing dataset with data for houses in the US listed on Craigslist, including their price, location, and several potentially useful features. [1]
2. A dataset with the median rent in different cities in the U.S. [2]
3. A dataset with the crime rate, population, and other crime-related features for each county in the U.S. [3]
4. A dataset with the list of public schools in each county in the U.S. [4]
5. A dataset with the median income for all states in the U.S. in 2020.[5]

We joined dataset 1 with dataset 2 using a combination of region/city and state to match rows, and further joined dataset 3, dataset 4, and dataset 5 with the rest of the data using a combination of county and state to match rows. In addition, we imputed the missing median income value with the mean income value of the state in the dataset. Furthermore, we conducted data cleaning to deal with NaN values and filter out extreme values. Specifically, we noticed that the maximum rent in our dataset was an astonishing sum of $2,768,307,249, which was probably an erroneous figure and was certainly

[1]https://www.kaggle.com/austinreese/usa-housing-listings
[2]https://www.kaggle.com/paultimothymooney/zillow-house-price-data
[3]https://www.kaggle.com/mikejohnsonjr/united-states-crime-rates-by-county
[4]https://www.kaggle.com/carlosaguayo/usa-public-schools
[5]https://data.world/uscensusbureau/acs-2015-5-e-income/workspace/file?filename=USA_All_States.csv

not reflective of ordinary market prices. Looking at the distribution of the rent values, we found that it was reasonable to only look at houses where the rent is less than $8000.

After our initial data cleaning, we had around 265,000 rows/data points. After merging and dropping irrelevant columns, including the ones we joined the tables on, we had 18 columns. We had 9 numerical features such as area in square feet, number of beds, bathrooms, and crime rate, etc., and also 9 categorical features such as type of household, parking options, dogs allowed, etc.

To better identify the patterns within the dataset, we created a choropleth map (Fig. 1) showing an estimate of the median rent price in each state. Since our median rent price dataset contained the median rent in cities across the United States, we estimated the median rent price in each state using the dataset by grouping the cities by the state they were in and taking the median of the data values for all the cities in a given state. This visualization shows that the median rent is higher in general on the East Coast and the West Coast, and highest of all in California.

3 Analytic Modeling

3.1 Linear Regression

We initially took 18 variables into consideration for the regression model before creating dummy variables and building a regression model.

We created dummy variables for the categorical variables in our data that were not booleans ($n - 1$ dummy variables for a feature with n categories).

We first built a linear regression model with 9 numerical variables along with our boolean categorical variables and the dummy variables that we created for the rest of the categorical variables.

We then calculated the Variance Inflation Factor (VIF) for our variables to check for multicollinearity between our features. We found that *cats_allowed* had a high VIF value, so we decided to remove it. We suspect that there might have been collinearity between the *cats_allowed* feature and the *dogs_allowed* feature, since they are quite similar features that both inform us about a house's policy on pets.

After removing *cats_allowed*, we built a new linear regression model and found that all of our features had very low p-values, except for two of the dummy variables for the type of house (Figure 2). We concluded that overall, the dummy variables for the type of house were still useful to us and so did not remove them.

3.2 Decision Tree Regression

We one-hot encoded our categorical variables. We ran 5-fold cross validation for our CART model with GridSearchCV and 51 values of *ccp_alpha*. We set *min_samples_leaf* to 5, and *min_samples_split* to 20. After running cross validation, our optimal *ccp_alpha* (cost complexity pruning value) is 0.00026.

3.3 Random Forest Regression

We ran 5-fold cross-validation for our Random Forest regressor model with GridSearchCV. In our cross-validation process, we tested using the square root of the number of features as our *max_features* versus using log_2 of the number of features. We chose reasonable values for our other hyperparameters. For instance, we set *min_samples_split* to 10 and *n_estimators* to 200.

3.4 Gradient Boosted Regression

It was not feasible for us to run cross-validation for our boosting model because it could not be run in a reasonable amount of time. We thus chose some reasonable hyperparameter values and directly ran the model without cross-validation. We set the *max_features* to 5, *min_samples_leaf* to 5, and *n_estimators*.

3.5 Neural Networks

Lastly, we ran a neural network with one hidden layer. We used Keras to implement the Sequential neural network. We add a dense layer with a value of 123 and *input_dim* of 36. We use Normal Initializer and ReLu activation. For the hidden layer, we add 2,670 neurons using ReLu activation. Lastly, we add $dense = 1$ using Linear Activation. We set the epoch number to be 100 and *batch_size* to be 150.

3.6 Summary and Results

The performance of our models are summarized in Table 1. Out of the five types of models we tested, linear regression performed the worst. Our best model was the random forest regressor, which had an out-of-sample R-squared value of 0.830 and also had the lowest out-of-sample RMSE and MAE out of all of our models, and our second best model was the decision tree regressor. We decided to run the bootstrap to see if we could conclude to a high degree of confidence that the random forest regressor model was the best model (by comparing it to the second best model to see if there was actually a significant difference between the two in terms of out-of-sample performance), and also create confidence intervals for the OSR^2 value of our best model.

	Linear Regression	Decision Tree Regressor	Random Forest Regressor	Gradient Boosted Regressor	Neural Network
OSR2	0.512	0.754	0.830	0.704	0.572
Out-of-sample RMSE	402.3966	285.8934	237.3954	313.4054	361.3051
Out-of-sample MAE	255.757	129.494	105.052	186.340	228.794

Table 1: Summary of the performance

After running the bootstrap with 500 samples for our random forest model and for our decision tree, we plotted the histograms (Figure 3) for the bootstrapped OSR^2 values for our random forest model and our bootstrap OSR^2 values for our decision tree values. From the histograms, we could see that the bootstrap OSR^2 values for the decision tree did not even overlap with the bootstrap OSR^2 values for the random forest model, i.e., all the decision tree bootstrap OSR^2 values are strictly lower than the random forest bootstrap OSR^2 values. Therefore, we can conclude with a high degree of confidence that the random forest model performs better. Our 95% confidence interval for the random forest model OSR2 was $[0.824, 0.837]$.

We also calculated an importance score (Figure 4, Table 2) for each feature in our Random Forest model to see which features were the most important in our model. We found that the median rent price in the city where the house was situated was the most important predictor by a large margin, followed by the area in square feet. Surprisingly, the number of public schools in the area was the third most important predictor variable. Median income in the area was also an important variable (more or less tied with the number of public schools), which was more in line with our expectations.

4 Impact and Future Improvements

Our analysis will potentially allow prospective renters to make a more informed decision about their renting decisions by calculating predicted rent prices based on a number of significant factors.

We have also identified some factors we believe are particularly important for predicting rent prices, such as the median rent price in the area, the area of the property in square feet, and (surprisingly) the number of public schools in the area. This will be impactful in that we believe that we have identified factors that people looking for a house to rent should particularly pay attention to, especially if they do not have the time and resources to do a full analysis involving many factors. These factors may also be of interest to homeowners hoping to rent out their homes if they are considering how much they should rent their place for (provided they have some control over the rent price).

In terms of future work and improvements, we could expand the scope of our analysis by extending it to consider a wider range of features, such as the number of hospitals in the area, and unemployment rates. This would potentially allow us to build a model with more predictive power, thus heightening its impact. We could have incorporated a time series analysis to examine how rent prices have been evolving over time. This would broaden the impact of our models by allowing us to examine trends in rent prices over time and the factors potentially shaping those trends. We could also look at the factors affecting the buying and selling of houses as compared to renting a house. It would be interesting to see if the most important factors potentially affecting house prices are the same as those which might affect rent prices.

5 Appendix

Feature	Importance score
median_rent_2019_12	35.4
sqfeet	16.5
Number_Of_Public_School	6.4
median_income	6.4
baths	5.8
laundry_options w/d in unit	5.1
crime_rate_per_1000000	5.0
parking_options_attached garage	3.8
bed	3.7
smoking_allowed	1.3
comes_furnished	1.1
dogs_allowed	1.0

Table 2: Importance score

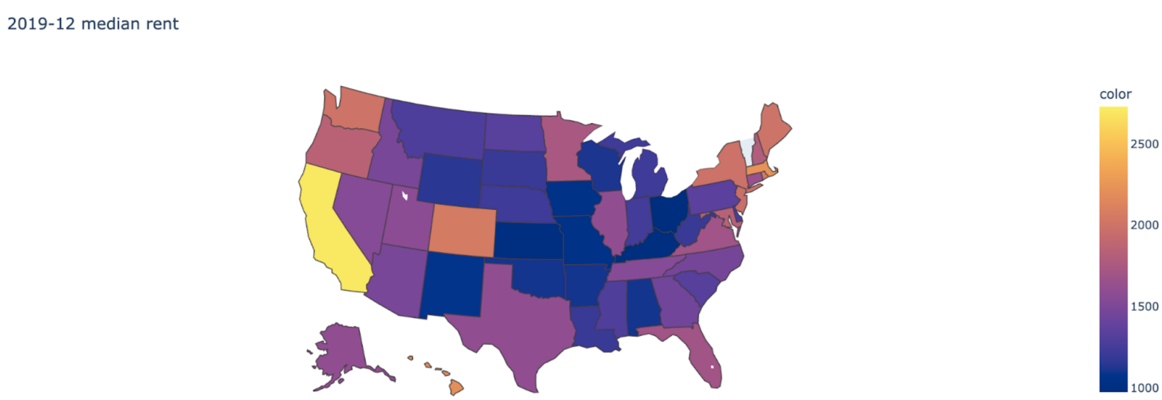


Figure 1: 2019 Median rent by state

```
In [382]: lr3 = smf.ols(formula="price ~ " + " + ".join(["Q('" + i +"')" for i in cols2]),
                        data= train).fit()
          print(lr3.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.521
Model:                            OLS   Adj. R-squared:                  0.521
Method:                 Least Squares   F-statistic:                     5988.
Date:                Fri, 17 Dec 2021   Prob (F-statistic):               0.00
Time:                        20:32:12   Log-Likelihood:            -1.3840e+06
No. Observations:              186924   AIC:                         2.768e+06
Df Residuals:                  186889   BIC:                         2.768e+06
Df Model:                          34
Covariance Type:            nonrobust
=====================================================================================================
                                          coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------------------
Intercept                             -86.9385      5.623    -15.462      0.000     -97.959     -75.918
Q('sqfeet')                             0.0042      0.000     17.385      0.000       0.004       0.005
Q('beds')                              -7.1392      0.389    -18.335      0.000      -7.902      -6.376
Q('baths')                            219.7905      1.821    120.728      0.000     216.222     223.359
Q('dogs_allowed')                     -14.2547      2.229     -6.396      0.000     -18.623      -9.886
Q('smoking_allowed')                  -86.7169      2.293    -37.826      0.000     -91.210     -82.224
Q('wheelchair_access')                -34.9831      3.596     -9.729      0.000     -42.031     -27.936
Q('electric_vehicle_charge')          188.4801      8.584     21.957      0.000     171.655     205.305
Q('comes_furnished')                  -92.4974      4.701    -19.675      0.000    -101.712     -83.283
Q('median_rent_2019_12')                0.6151      0.002    276.125      0.000       0.611       0.619
Q('crime_rate_per_100000')              0.1224      0.004     28.953      0.000       0.114       0.131
Q('population')                      2.906e-05   1.47e-06     19.751      0.000    2.62e-05    3.19e-05
Q('Number_Of_Public_School')           -0.0184      0.005     -3.997      0.000      -0.027      -0.009
Q('median_income')                   8.561e-05   1.91e-05      4.471      0.000    4.81e-05       0.000
Q('type_assisted living')            1953.0117    397.519      4.913      0.000    1173.884    2732.139
Q('type_condo')                       151.2010      7.478     20.220      0.000     136.545     165.857
Q('type_cottage/cabin')                76.0246     23.631      3.217      0.001      29.709     122.340
Q('type_duplex')                       63.2099      8.305      7.611      0.000      46.932      79.488
Q('type_flat')                        223.7320     24.332      9.195      0.000     176.042     271.422
Q('type_house')                       147.7324      3.561     41.484      0.000     140.753     154.712
Q('type_in-law')                      -59.5787     46.315     -1.286      0.198    -150.354      31.197
Q('type_land')                         10.4099    397.663      0.026      0.979    -769.000     789.819
Q('type_loft')                        139.2582     20.370      6.836      0.000      99.333     179.183
Q('type_manufactured')               -107.4166      9.321    -11.524      0.000    -125.686     -89.147
Q('type_townhouse')                    71.2607      5.066     14.068      0.000      61.332      81.189
Q('laundry_options_laundry on site')  -73.1620      3.062    -23.892      0.000     -79.164     -67.160
Q('laundry_options_no laundry on site') -140.4749   10.518    -13.356      0.000    -161.089    -119.861
Q('laundry_options_w/d hookups')        8.4785      2.912      2.912      0.004       2.771      14.186
Q('laundry_options_w/d in unit')      175.2748      2.668     65.686      0.000     170.045     180.505
Q('parking_options_carport')          -80.2680      3.206    -25.039      0.000     -86.551     -73.985
Q('parking_options_detached garage')   38.7407      4.781      8.104      0.000      29.371      48.111
Q('parking_options_no parking')        93.1629      9.872      9.437      0.000      73.813     112.513
Q('parking_options_off-street parking') -56.7496     2.270    -24.999      0.000     -61.199     -52.300
Q('parking_options_street parking')   -60.3856      4.806    -12.565      0.000     -69.805     -50.966
Q('parking_options_valet parking')    624.7074     45.080     13.858      0.000     536.352     713.063
==============================================================================
```

Figure 2: OLS Output

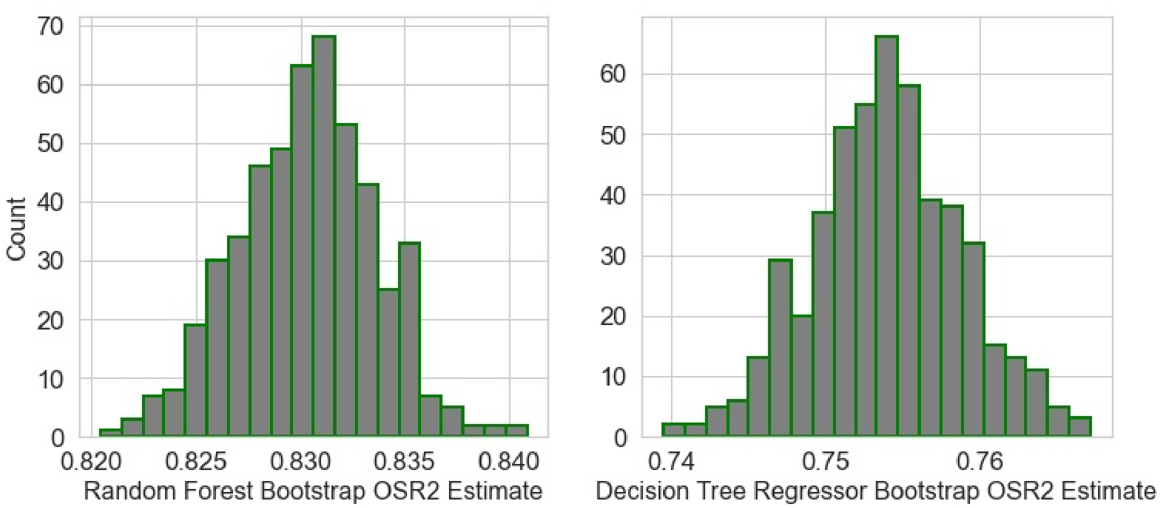


Figure 3: Comparison of bootstrapped OSR2 between random forest and decision tree

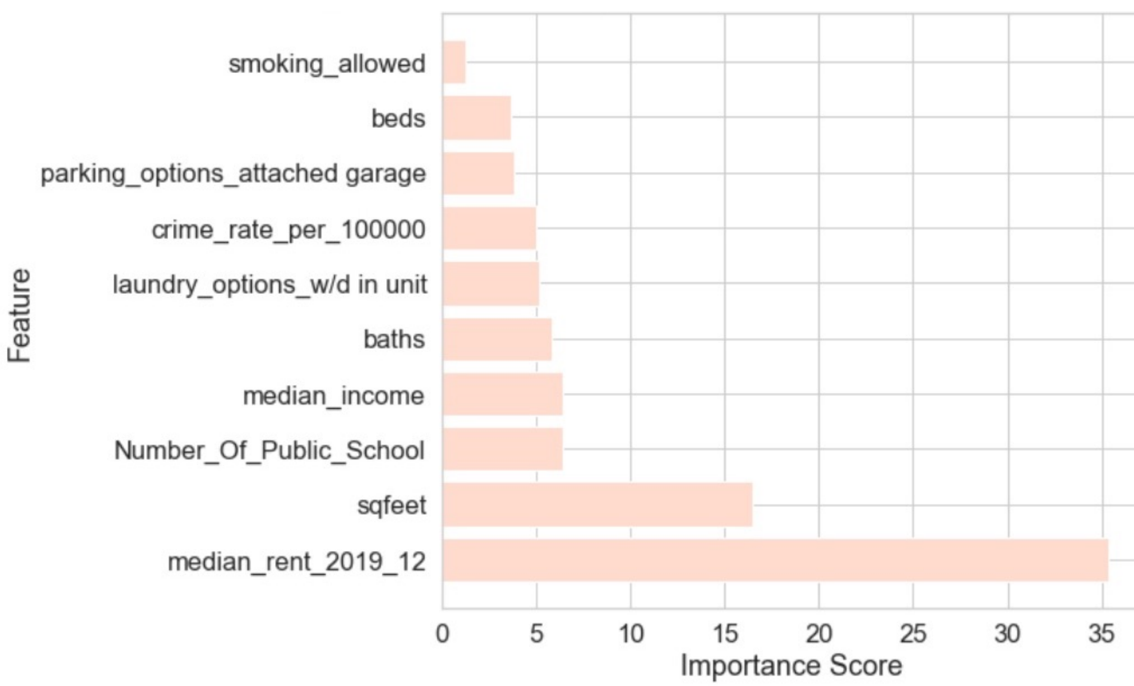


Figure 4: Top 10 features with highest importance scores for random forest regressor

In [162]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set(style = "whitegrid",
        color_codes = True,
        font_scale = 1.5)
sns.set_palette('Reds')

from sklearn.preprocessing import normalize
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.feature_extraction import DictVectorizer

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV, ParameterGrid
from sklearn.model_selection import RandomizedSearchCV

import warnings
warnings.filterwarnings("ignore")
```

```
         cwd: C:\Users\L\AppData\Local\Temp\pip-install-8wik5v1i\fiona_fbee78378f864b
0bb0111101ae501489\
    Complete output (1 lines):
    A GDAL API version must be specified. Provide a path to gdal-config using a GDAL_
CONFIG environment variable or use a GDAL_VERSION environment variable.
    ----------------------------------------
WARNING: Discarding https://files.pythonhosted.org/packages/41/9d/63696e7b1de42aad294
d4781199a408bec593d8fdb80a2b4a788c911a33b/Fiona-1.8.6.tar.gz#sha256=fa31dfe8855b9cd0b
128b47a4df558f1b8eda90d2181bff1dd9854e5556efb3e (https://files.pythonhosted.org/packa
ges/41/9d/63696e7b1de42aad294d4781199a408bec593d8fdb80a2b4a788c911a33b/Fiona-1.8.6.ta
r.gz#sha256=fa31dfe8855b9cd0b128b47a4df558f1b8eda90d2181bff1dd9854e5556efb3e) (from h
ttps://pypi.org/simple/fiona/). (https://pypi.org/simple/fiona/).) Command errored ou
t with exit status 1: python setup.py egg_info Check the logs for full command outpu
t.
    ERROR: Command errored out with exit status 1:
     command: 'C:\Users\L\AppData\Local\Programs\Python\Python39\python.exe' -c 'impo
rt io, os, sys, setuptools, tokenize; sys.argv[0] = '"'"'C:\\Users\\L\\AppData\\Local
\\Temp\\pip-install-8wik5v1i\\fiona_3b7dee46b2eb4fc99d76bb402a7e9e43\\setup.py'"'"';
```

In [163]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [164]:
```
df = pd.read_csv("housing_merged_crime_2.csv")
df
```

Out[164]:

	Unnamed: 0	id	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smol
0	0	7049404148	1400	house	1010	2	1.0	1	1	
1	1	7046093394	1795	apartment	869	1	1.0	1	1	
2	2	7047484892	2595	townhouse	1317	2	2.5	1	1	
3	3	7049005381	1695	house	1100	3	2.0	0	0	
4	4	7049394070	1699	apartment	860	2	1.0	1	1	
...	...	...	...	...	...	...	...	...	...	
264810	265050	7026220138	1800	house	2225	3	2.0	0	0	
264811	265052	7025150381	800	apartment	700	1	1.0	1	0	
264812	265053	7024580936	1450	townhouse	1600	2	2.5	0	0	
264813	265054	7010591533	1350	apartment	1000	3	1.0	0	0	

In [165]:
```
df['state'] = df['county_state'].str[-2:]
df['state']
```

Out[165]:
```
0         CA
1         CA
2         CA
3         CA
4         CA
          ..
264810    CA
264811    CA
264812    CA
264813    CA
264814    CA
Name: state, Length: 264815, dtype: object
```

In [166]:
```
df['county'] = df['county_state'].str[:-11]
```

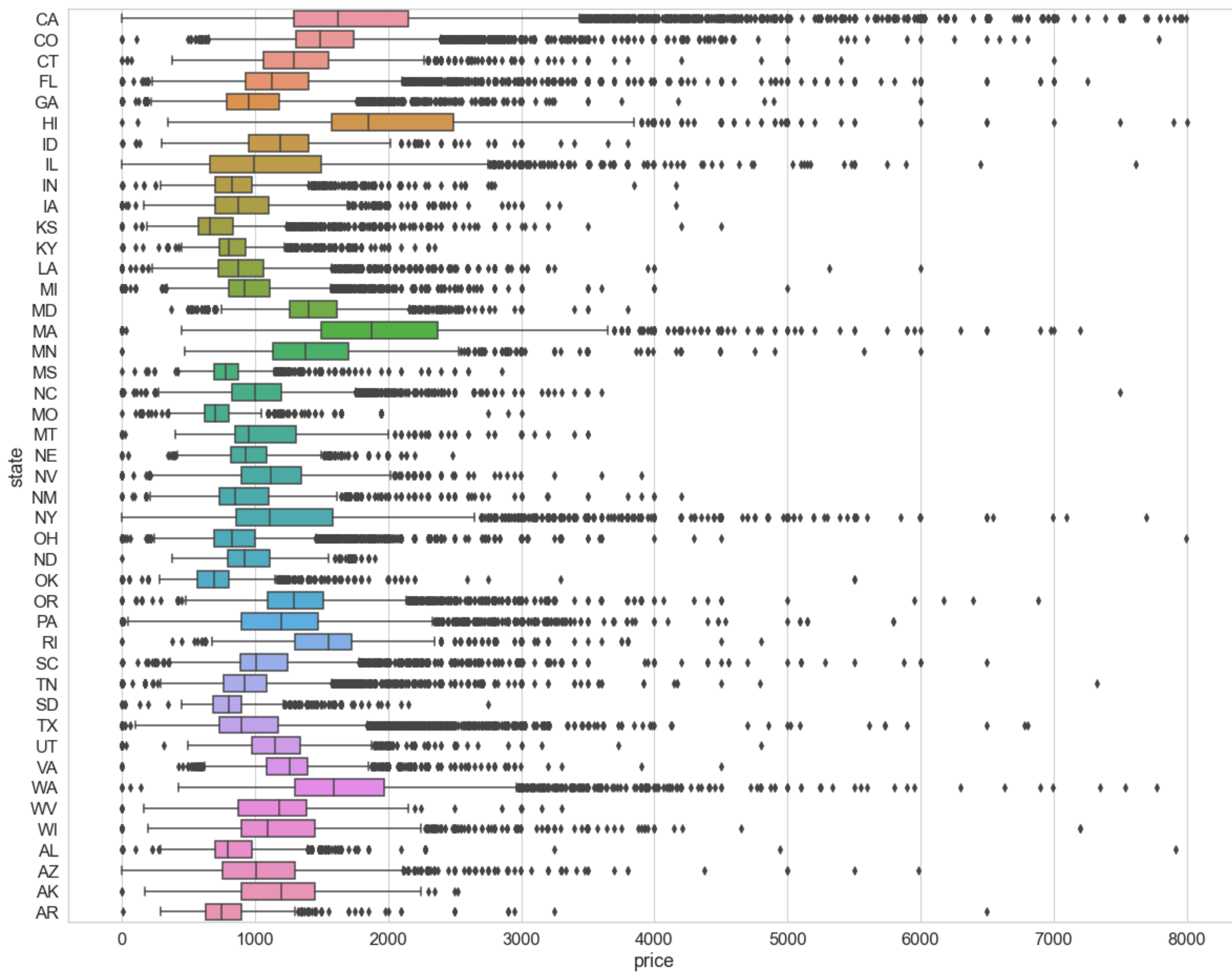
In [167]:
```
df['county']
```

Out[167]:
```
0         Sacramento
1         Sacramento
2         Sacramento
3         Sacramento
4         Sacramento
             ...
264810        Shasta
264811        Shasta
264812        Shasta
264813        Shasta
264814        Shasta
Name: county, Length: 264815, dtype: object
```

In [168]:
```
#housing['region_from_url'] = housing['region_url'].str.replace("https://","").str.replace(
```

In [169]:
```
g = sns.boxplot(y = "state",
                x = 'price',
                data = df)
g.figure.set_size_inches(20,16)
plt.show()

fig = g.get_figure()
fig.savefig("price_by_state.png")
```



In [ ]:

In [170]:
```
df2 = pd.read_csv("Public_Schools.csv")
df2
```

Out[170]:

	X	Y	OBJECTID	NCESID	NAME	ADDRESS	CITY	STATE
0	-81.050895	29.022271	2002	120192008041	SAMSULA ACADEMY	248 N SAMSULA DR	NEW SMYRNA	FL
1	-92.507288	31.180659	2003	220129002344	CAROLINE DORMON JUNIOR HIGH SCHOOL	8906 HWY 165 SOUTH	WOODWORTH	LA
2	-69.971880	43.908147	2004	230378023129	HARRIET BEECHER STOWE ELEMENTARY	44 MCKEEN STREET	BRUNSWICK	ME
3	-89.542799	32.728496	2005	280252001118	LEAKE CENTRAL ELEMENTARY SCHOOL	603 HWY. 16 WEST	CARTHAGE	MS
4	-94.361775	39.364359	2006	291645000891	KEARNEY ELEM.	902 S JEFFERSON	KEARNEY	MO
...	...	...	...	...	...	...	...	...
102365	-83.085229	42.320632	102216	260032201947	COVENANT HOUSE ACADEMY DETROIT - SOUTHWEST SITE	1450 25TH ST	DETROIT	MI
102366	-83.272599	42.062038	102217	260198003940	FRED W. RITTER ELEMENTARY SCHOOL	5650 CARLETON ROCKWOOD RD	SOUTH ROCKWOOD	MI
102367	-88.914089	30.436478	102218	280177000284	DIBERVILLE ELEM	4540 BRODIE ROAD	DIBERVILLE	MS
102368	-94.558365	39.187941	102219	292280001267	DAVIDSON ELEM.	5100 N HIGHLAND	KANSAS CITY	MO
102369	-93.291370	37.220353	102220	292886001480	JUVENILE JUSTICE CTR.	1111 N ROBBERSON	SPRINGFIELD	MO

102370 rows × 33 columns

In [171]:
```
df
```

Out[171]:

	Unnamed: 0	id	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smol
0	0	7049404148	1400	house	1010	2	1.0	1	1	
1	1	7046093394	1795	apartment	869	1	1.0	1	1	
2	2	7047484892	2595	townhouse	1317	2	2.5	1	1	
3	3	7049005381	1695	house	1100	3	2.0	0	0	
4	4	7049394070	1699	apartment	860	2	1.0	1	1	
...	...	...	...	...	...	...	...	...	...	
264810	265050	7026220138	1800	house	2225	3	2.0	0	0	
264811	265052	7025150381	800	apartment	700	1	1.0	1	0	
264812	265053	7024580936	1450	townhouse	1600	2	2.5	0	0	
264813	265054	7010591533	1350	apartment	1000	3	1.0	0	0	

In [172]:
```
df2.groupby('COUNTY').size()
```

Out[172]:
```
COUNTY
ABBEVILLE          9
ACADIA            27
ACCOMACK          13
ADA              131
ADAIR             37
                ...
YUKON-KOYUKUK     31
YUMA              83
ZAPATA             6
ZAVALA             7
ZIEBACH            4
Length: 1908, dtype: int64
```

In [173]:
```
df2.groupby('CITY').size()
```

Out[173]:
```
CITY
ABBEVILLE        17
ABBOTSFORD        2
ABBOTT            1
ABERCROMBIE       1
ABERDEEN         48
                 ..
ZOLFO SPRINGS     2
ZUMBROTA          4
ZUNI              5
ZURICH            1
ZWOLLE            2
Length: 12805, dtype: int64
```

In [174]:
```
number_of_school = df2.groupby('COUNTY').size().reset_index()
```

In [175]:
```
number_of_school['county'] = number_of_school['COUNTY'].str.title()
```

In [176]:
```
number_of_school
```

Out[176]:

	COUNTY	0	county
0	ABBEVILLE	9	Abbeville
1	ACADIA	27	Acadia
2	ACCOMACK	13	Accomack
3	ADA	131	Ada
4	ADAIR	37	Adair
...	...	...	...
1903	YUKON-KOYUKUK	31	Yukon-Koyukuk
1904	YUMA	83	Yuma
1905	ZAPATA	6	Zapata
1906	ZAVALA	7	Zavala
1907	ZIEBACH	4	Ziebach

1908 rows × 3 columns

In [177]:
```

number_of_school.columns = ['COUNTY', 'Number_Of_Public_School', 'county']
number_of_school
```

Out[177]:

	COUNTY	Number_Of_Public_School	county
0	ABBEVILLE	9	Abbeville
1	ACADIA	27	Acadia
2	ACCOMACK	13	Accomack
3	ADA	131	Ada
4	ADAIR	37	Adair
...	...	...	...
1903	YUKON-KOYUKUK	31	Yukon-Koyukuk
1904	YUMA	83	Yuma
1905	ZAPATA	6	Zapata
1906	ZAVALA	7	Zavala
1907	ZIEBACH	4	Ziebach

1908 rows × 3 columns

In [178]:
```python
df
```

Out[178]:

	Unnamed: 0	id	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_a
0	0	7049404148	1400	house	1010	2	1.0	1	1	
1	1	7046093394	1795	apartment	869	1	1.0	1	1	
2	2	7047484892	2595	townhouse	1317	2	2.5	1	1	
3	3	7049005381	1695	house	1100	3	2.0	0	0	
4	4	7049394070	1699	apartment	860	2	1.0	1	1	
...	...	...	...	...	...	...	...	...	...	
264810	265050	7026220138	1800	house	2225	3	2.0	0	0	
264811	265052	7025150381	800	apartment	700	1	1.0	1	0	
264812	265053	7024580936	1450	townhouse	1600	2	2.5	0	0	
264813	265054	7010591533	1350	apartment	1000	3	1.0	0	0	
264814	265055	7023917423	1200	duplex	1144	2	2.0	0	0	

264815 rows × 44 columns

In [179]:
```python
school1 = df.merge(number_of_school, how='inner', left_on = 'county', right_on = 'county')
school1
```

Out[179]:

	Unnamed: 0	id	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_a
0	0	7049404148	1400	house	1010	2	1.0	1	1	
1	1	7046093394	1795	apartment	869	1	1.0	1	1	
2	2	7047484892	2595	townhouse	1317	2	2.5	1	1	
3	3	7049005381	1695	house	1100	3	2.0	0	0	
4	4	7049394070	1699	apartment	860	2	1.0	1	1	
...	...	...	...	...	...	...	...	...	...	
263729	265050	7026220138	1800	house	2225	3	2.0	0	0	
263730	265052	7025150381	800	apartment	700	1	1.0	1	0	
263731	265053	7024580936	1450	townhouse	1600	2	2.5	0	0	
263732	265054	7010591533	1350	apartment	1000	3	1.0	0	0	
263733	265055	7023917423	1200	duplex	1144	2	2.0	0	0	

263734 rows × 46 columns

In [180]:
```python
school1.to_csv('Housing_with_school.csv')
```

In [ ]:

In [181]:
```
df2
```

102365	-83.085229	42.320632	102216	260032201947	ACADEMY DETROIT - SOUTHWEST SITE	1450 25TH ST	DETROIT
102366	-83.272599	42.062038	102217	260198003940	FRED W. RITTER ELEMENTARY SCHOOL	5650 CARLETON ROCKWOOD RD	SOUTH ROCKWOOD
102367	-88.914089	30.436478	102218	280177000284	DIBERVILLE ELEM	4540 BRODIE ROAD	DIBERVILLE
102368	-94.558365	39.187941	102219	292280001267	DAVIDSON ELEM.	5100 N HIGHLAND	KANSAS CITY
102369	-93.291370	37.220353	102220	292886001480	JUVENILE JUSTICE CTR.	1111 N ROBBERSON	SPRINGFIELD

102370 rows × 33 columns

In [182]:
```
df3 = pd.read_csv('income.csv')
```

In [183]:
```
df3 = df3[['State_Name','State_ab','County','City','Median']]
df3
```

Out[183]:

	State_Name	State_ab	County	City	Median
0	Alabama	AL	Mobile County	Chickasaw	30506
1	Alabama	AL	Barbour County	Louisville	19528
2	Alabama	AL	Shelby County	Columbiana	31930
3	Alabama	AL	Mobile County	Satsuma	52814
4	Alabama	AL	Mobile County	Dauphin Island	67225
...	...	...	...	...	...
32521	Puerto Rico	PR	Adjuntas Municipio	Guaynabo	13729
32522	Puerto Rico	PR	Adjuntas Municipio	Aguada	9923
32523	Puerto Rico	PR	Adjuntas Municipio	Aguada	34054
32524	Puerto Rico	PR	Adjuntas Municipio	Aguada	0
32525	Puerto Rico	PR	Adjuntas Municipio	Aguadilla	20229

32526 rows × 5 columns

In [184]:
```
income_median = df3.groupby('County').mean().reset_index()
income_median
```

Out[184]:

	County	Median
0	Abbeville County	69420.066986
1	Acadia Parish	60547.148387
2	Accomack County	104656.813699
3	Ada County	81019.023622
4	Adair County	68828.325098
...	...	...
1128	Young County	36772.000000
1129	Yuba County	46888.000000
1130	Yukon-Koyukuk Census Area	30981.250000
1131	Yuma County	86485.800000
1132	Zapata County	25479.000000

1133 rows × 2 columns

In [185]:
```
income_median['county'] = income_median['County'].str[:-7]
income_median
```

Out[185]:

	County	Median	county
0	Abbeville County	69420.066986	Abbeville
1	Acadia Parish	60547.148387	Acadia
2	Accomack County	104656.813699	Accomack
3	Ada County	81019.023622	Ada
4	Adair County	68828.325098	Adair
...	...	...	...
1128	Young County	36772.000000	Young
1129	Yuba County	46888.000000	Yuba
1130	Yukon-Koyukuk Census Area	30981.250000	Yukon-Koyukuk Cens
1131	Yuma County	86485.800000	Yuma
1132	Zapata County	25479.000000	Zapata

1133 rows × 3 columns

In [186]:
```
income1 = school1.merge(income_median, how='left', left_on = 'county', right_on = 'county')
income1
```

Out[186]:

	Unnamed: 0	id	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_a
0	0	7049404148	1400	house	1010	2	1.0	1	1	
1	1	7046093394	1795	apartment	869	1	1.0	1	1	
2	2	7047484892	2595	townhouse	1317	2	2.5	1	1	
3	3	7049005381	1695	house	1100	3	2.0	0	0	
4	4	7049394070	1699	apartment	860	2	1.0	1	1	
...	...	...	...	...	...	...	...	...	...	
267680	265050	7026220138	1800	house	2225	3	2.0	0	0	
267681	265052	7025150381	800	apartment	700	1	1.0	1	0	
267682	265053	7024580936	1450	townhouse	1600	2	2.5	0	0	
267683	265054	7010591533	1350	apartment	1000	3	1.0	0	0	
267684	265055	7023917423	1200	duplex	1144	2	2.0	0	0	

267685 rows × 48 columns

In [187]:
```
income1.to_csv('Housing_with_school_income_crime.csv')
```

In [188]:
```
income1
```

Out[188]:

	Unnamed: 0	id	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_a
0	0	7049404148	1400	house	1010	2	1.0	1	1	
1	1	7046093394	1795	apartment	869	1	1.0	1	1	
2	2	7047484892	2595	townhouse	1317	2	2.5	1	1	
3	3	7049005381	1695	house	1100	3	2.0	0	0	
4	4	7049394070	1699	apartment	860	2	1.0	1	1	
...	...	...	...	...	...	...	...	...	...	
267680	265050	7026220138	1800	house	2225	3	2.0	0	0	
267681	265052	7025150381	800	apartment	700	1	1.0	1	0	
267682	265053	7024580936	1450	townhouse	1600	2	2.5	0	0	
267683	265054	7010591533	1350	apartment	1000	3	1.0	0	0	
267684	265055	7023917423	1200	duplex	1144	2	2.0	0	0	

267685 rows × 48 columns

In [189]:
```
income1.columns
```

Out[189]:
```
Index(['Unnamed: 0', 'id', 'price', 'type', 'sqfeet', 'beds', 'baths',
       'cats_allowed', 'dogs_allowed', 'smoking_allowed', 'wheelchair_access',
       'electric_vehicle_charge', 'comes_furnished', 'laundry_options',
       'parking_options', 'description', 'lat', 'long', 'region_state',
       'Metro', 'SizeRank', 'median_rent_2019_12', 'county_state',
       'crime_rate_per_100000', 'CPOPARST', 'CPOPCRIM', 'AG_ARRST', 'AG_OFF',
       'COVIND', 'INDEX', 'MODINDX', 'MURDER', 'RAPE', 'ROBBERY', 'AGASSLT',
       'BURGLRY', 'LARCENY', 'MVTHEFT', 'ARSON', 'population', 'FIPS_ST',
       'FIPS_CTY', 'state', 'county', 'COUNTY', 'Number_Of_Public_School',
       'County', 'Median'],
      dtype='object')
```

In [190]:
```
finaldata=income1[['price','type','sqfeet','beds','baths','cats_allowed','dogs_allowed','sı
       'electric_vehicle_charge', 'comes_furnished', 'laundry_options',
       'parking_options','region_state','median_rent_2019_12','crime_rate_per_100000','popu
```

In [191]:
```
finaldata
```

Out[191]:

	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_allowed	wheelchair_
0	1400	house	1010	2	1.0	1	1	1	
1	1795	apartment	869	1	1.0	1	1	0	
2	2595	townhouse	1317	2	2.5	1	1	0	
3	1695	house	1100	3	2.0	0	0	0	
4	1699	apartment	860	2	1.0	1	1	1	
...	...	...	...	...	...	...	...	...	
267680	1800	house	2225	3	2.0	0	0	0	
267681	800	apartment	700	1	1.0	1	0	0	
267682	1450	townhouse	1600	2	2.5	0	0	1	
267683	1350	apartment	1000	3	1.0	0	0	0	

In [192]:
```
finaldata.rename(columns={'Median':'median_income'}, inplace=True)
```

In [193]:
```
finaldata
```

Out[193]:

	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_allowed	wheelchair_acces
0	1400	house	1010	2	1.0	1	1	1	
1	1795	apartment	869	1	1.0	1	1	0	
2	2595	townhouse	1317	2	2.5	1	1	0	
3	1695	house	1100	3	2.0	0	0	0	
4	1699	apartment	860	2	1.0	1	1	1	
...	...	...	...	...	...	...	...	...	
267680	1800	house	2225	3	2.0	0	0	0	
267681	800	apartment	700	1	1.0	1	0	0	
267682	1450	townhouse	1600	2	2.5	0	0	1	
267683	1350	apartment	1000	3	1.0	0	0	0	
267684	1200	duplex	1144	2	2.0	0	0	1	

267685 rows × 21 columns

In [194]:
```
finaldata.to_csv('Housing_rent_.csv')
```

In [195]:
```
finaldata
```

Out[195]:

	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_allowed	wheelchair_acces
0	1400	house	1010	2	1.0	1	1	1	
1	1795	apartment	869	1	1.0	1	1	0	
2	2595	townhouse	1317	2	2.5	1	1	0	
3	1695	house	1100	3	2.0	0	0	0	
4	1699	apartment	860	2	1.0	1	1	1	
...	...	...	...	...	...	...	...	...	
267680	1800	house	2225	3	2.0	0	0	0	
267681	800	apartment	700	1	1.0	1	0	0	
267682	1450	townhouse	1600	2	2.5	0	0	1	
267683	1350	apartment	1000	3	1.0	0	0	0	
267684	1200	duplex	1144	2	2.0	0	0	1	

267685 rows × 21 columns

In [364]:
```
df = finaldata.copy()
```

In [365]:
```
df
```

Out[365]:

	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_allowed	wheelchair_
0	1400	house	1010	2	1.0	1	1	1	
1	1795	apartment	869	1	1.0	1	1	0	
2	2595	townhouse	1317	2	2.5	1	1	0	
3	1695	house	1100	3	2.0	0	0	0	
4	1699	apartment	860	2	1.0	1	1	1	
...	...	...	...	...	...	...	...	...	
267680	1800	house	2225	3	2.0	0	0	0	
267681	800	apartment	700	1	1.0	1	0	0	
267682	1450	townhouse	1600	2	2.5	0	0	1	
267683	1350	apartment	1000	3	1.0	0	0	0	

In [366]:
```
df["median_income"] = df["median_income"].fillna(df.groupby("state")["median_income"].trans
```

In [367]:
```
df
```

Out[367]:

	price	type	sqfeet	beds	baths	cats_allowed	dogs_allowed	smoking_allowed	wheelchair_acces
0	1400	house	1010	2	1.0	1	1	1	
1	1795	apartment	869	1	1.0	1	1	0	
2	2595	townhouse	1317	2	2.5	1	1	0	
3	1695	house	1100	3	2.0	0	0	0	
4	1699	apartment	860	2	1.0	1	1	1	
...	...	...	...	...	...	...	...	...	
267680	1800	house	2225	3	2.0	0	0	0	
267681	800	apartment	700	1	1.0	1	0	0	
267682	1450	townhouse	1600	2	2.5	0	0	1	
267683	1350	apartment	1000	3	1.0	0	0	0	
267684	1200	duplex	1144	2	2.0	0	0	1	

267685 rows × 21 columns

In [368]:
```
df.columns
```

Out[368]:
```
Index(['price', 'type', 'sqfeet', 'beds', 'baths', 'cats_allowed',
       'dogs_allowed', 'smoking_allowed', 'wheelchair_access',
       'electric_vehicle_charge', 'comes_furnished', 'laundry_options',
       'parking_options', 'region_state', 'median_rent_2019_12',
       'crime_rate_per_100000', 'population', 'state', 'county',
       'Number_Of_Public_School', 'median_income'],
      dtype='object')
```

In [369]:
```
df = df[df['median_income'].notna()]
```

In [370]:
```
X = pd.get_dummies(df.drop(['price','state','county','region_state'], axis=1))
```

In [371]:
```
X_lr = pd.get_dummies(df.drop(['price','state','county','region_state'], axis=1), drop_firs
```

In [372]:
```
y = df['price']
```

In [373]:
```
X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(X_lr, y, test_size=0.3, ra
```

In [374]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=88)
```

In [375]:
```
X_train.shape, X_test.shape
```

Out[375]:
```
((186924, 38), (80111, 38))
```

In [ ]:

```

```

In [376]:

```
X.isnull().sum()
```

Out[376]:

```
sqfeet                                0
beds                                  0
baths                                 0
cats_allowed                          0
dogs_allowed                          0
smoking_allowed                       0
wheelchair_access                     0
electric_vehicle_charge               0
comes_furnished                       0
median_rent_2019_12                   0
crime_rate_per_100000                 0
population                            0
Number_Of_Public_School               0
median_income                         0
type_apartment                        0
type_assisted living                  0
type_condo                            0
type_cottage/cabin                    0
type_duplex                           0
type_flat                             0
type_house                            0
type_in-law                           0
type_land                             0
type_loft                             0
type_manufactured                     0
type_townhouse                        0
laundry_options_laundry in bldg       0
laundry_options_laundry on site       0
laundry_options_no laundry on site    0
laundry_options_w/d hookups           0
laundry_options_w/d in unit           0
parking_options_attached garage       0
parking_options_carport               0
parking_options_detached garage       0
parking_options_no parking            0
parking_options_off-street parking    0
parking_options_street parking        0
parking_options_valet parking         0
dtype: int64
```

In [207]:

```
df_log = df.copy()
```

In [208]:

```
df_log['crime_rate_per_100000'] = np.log(df['crime_rate_per_100000'])
df_log['crime_rate_per_100000'] = np.log(df['crime_rate_per_100000'])
```

In [209]:
```
X.columns
```

Out[209]:
```
Index(['sqfeet', 'beds', 'baths', 'cats_allowed', 'dogs_allowed',
       'smoking_allowed', 'wheelchair_access', 'electric_vehicle_charge',
       'comes_furnished', 'median_rent_2019_12', 'crime_rate_per_100000',
       'population', 'Number_Of_Public_School', 'median_income',
       'type_apartment', 'type_assisted living', 'type_condo',
       'type_cottage/cabin', 'type_duplex', 'type_flat', 'type_house',
       'type_in-law', 'type_land', 'type_loft', 'type_manufactured',
       'type_townhouse', 'laundry_options_laundry in bldg',
       'laundry_options_laundry on site', 'laundry_options_no laundry on site',
       'laundry_options_w/d hookups', 'laundry_options_w/d in unit',
       'parking_options_attached garage', 'parking_options_carport',
       'parking_options_detached garage', 'parking_options_no parking',
       'parking_options_off-street parking', 'parking_options_street parking',
       'parking_options_valet parking'],
      dtype='object')
```

1)

In [210]:
```
from sklearn.linear_model import LinearRegression

lr = LinearRegression().fit(X_train, y_train)
```

In [290]:
```
def OSR2(model, X_test, y_test, y_train):

    y_pred = model.predict(X_test)
    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(y_train))**2)

    return (1 - SSE/SST)
```

In [212]:
```
from sklearn.ensemble import GradientBoostingRegressor
from scipy.stats import pearsonr
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from math import sqrt
```

In [213]:
```
comparison_data = {'Linear Regression': ['{:.3f}'.format(OSR2(lr, X_test, y_test, y_train))
                                         '{:.4f}'.format(sqrt(mean_squared_error(y_test, lı
                                         '{:.3f}'.format(mean_absolute_error(y_test, lr.pre
comparison_data
```

Out[213]:
```
{'Linear Regression': ['0.523', '397.8172', '250.768']}
```

In [ ]:

In [214]:
```
comparison_table = pd.DataFrame(data=comparison_data, index=['OSR2', 'Out-of-sample RMSE',
comparison_table.style.set_properties(**{'font-size': '12pt',}).set_table_styles([{'selecto
```

Out[214]:

	Linear Regression
OSR2	0.523
Out-of-sample RMSE	397.8172
Out-of-sample MAE	250.768

2)

In [215]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
# The dataframe passed to VIF must include the intercept term. We add it the same way we d
def VIF(df, columns):
    values = sm.add_constant(df[columns]).values
    num_columns = len(columns)+1
    vif = [variance_inflation_factor(values, i) for i in range(num_columns)]
    return pd.Series(vif[1:], index=columns)

cols = ['sqfeet', 'beds', 'baths', 'cats_allowed', 'dogs_allowed',
       'smoking_allowed', 'wheelchair_access', 'electric_vehicle_charge',
       'comes_furnished', 'median_rent_2019_12', 'crime_rate_per_100000',
       'population', 'Number_Of_Public_School', 'median_income',
       'type_apartment', 'type_assisted living', 'type_condo',
       'type_cottage/cabin', 'type_duplex', 'type_flat', 'type_house',
       'type_in-law', 'type_land', 'type_loft', 'type_manufactured',
       'type_townhouse', 'laundry_options_laundry in bldg',
       'laundry_options_laundry on site', 'laundry_options_no laundry on site',
       'laundry_options_w/d hookups', 'laundry_options_w/d in unit',
       'parking_options_attached garage', 'parking_options_carport',
       'parking_options_detached garage', 'parking_options_no parking',
       'parking_options_off-street parking', 'parking_options_street parking',
       'parking_options_valet parking']
VIF(X_train, cols)
```

Out[215]:

```
sqfeet                                1.006193e+00
beds                                  1.293562e+00
baths                                 1.474819e+00
cats_allowed                          5.275387e+00
dogs_allowed                          5.110245e+00
smoking_allowed                       1.206495e+00
wheelchair_access                     1.171399e+00
electric_vehicle_charge               1.070778e+00
comes_furnished                       1.093723e+00
median_rent_2019_12                   1.648133e+00
crime_rate_per_100000                 1.131938e+00
population                            3.388114e+00
Number_Of_Public_School               2.792757e+00
median_income                         1.131634e+00
type_apartment                                 inf
type_assisted living                  1.801440e+15
type_condo                                     inf
type_cottage/cabin                             inf
type_duplex                                    inf
type_flat                                      inf
type_house                                     inf
type_in-law                                    inf
type_land                                      inf
type_loft                                      inf
type_manufactured                              inf
type_townhouse                                 inf
laundry_options_laundry in bldg       1.791703e+00
laundry_options_laundry on site       2.151620e+00
laundry_options_no laundry on site    1.092024e+00
laundry_options_w/d hookups           2.280311e+00
laundry_options_w/d in unit           2.905800e+00
parking_options_attached garage       1.577847e+00
parking_options_carport               1.414114e+00
parking_options_detached garage       1.191164e+00
parking_options_no parking            1.023455e+00
parking_options_off-street parking    1.793580e+00
```

```
parking_options_street parking        1.170028e+00
parking_options_valet parking         1.004358e+00
dtype: float64
```

In [ ]:

In [377]:
```
cols = ['sqfeet', 'beds', 'baths', 'dogs_allowed',
        'smoking_allowed', 'wheelchair_access', 'electric_vehicle_charge',
        'comes_furnished', 'median_rent_2019_12', 'crime_rate_per_100000',
        'population', 'Number_Of_Public_School', 'median_income',
        'type_apartment', 'type_assisted living', 'type_condo',
        'type_cottage/cabin', 'type_duplex', 'type_flat', 'type_house',
        'type_in-law', 'type_land', 'type_loft', 'type_manufactured',
        'type_townhouse', 'laundry_options_laundry in bldg',
        'laundry_options_laundry on site', 'laundry_options_no laundry on site',
        'laundry_options_w/d hookups', 'laundry_options_w/d in unit',
        'parking_options_attached garage', 'parking_options_carport',
        'parking_options_detached garage', 'parking_options_no parking',
        'parking_options_off-street parking', 'parking_options_street parking',
        'parking_options_valet parking']
VIF(X_train, cols)
```

Out[377]:
```
sqfeet                                1.006189
beds                                  1.293501
baths                                 1.473503
dogs_allowed                          1.239945
smoking_allowed                       1.206051
wheelchair_access                     1.170946
electric_vehicle_charge               1.070564
comes_furnished                       1.090830
median_rent_2019_12                   1.648085
crime_rate_per_100000                 1.131927
population                            3.387434
Number_Of_Public_School               2.792751
median_income                         1.131356
type_apartment                             inf
type_assisted living                       inf
type_condo                                 inf
type_cottage/cabin                         inf
type_duplex                                inf
type_flat                                  inf
type_house                                 inf
type_in-law                                inf
type_land                                  inf
type_loft                                  inf
type_manufactured                          inf
type_townhouse                             inf
laundry_options_laundry in bldg       1.746309
laundry_options_laundry on site       2.127122
laundry_options_no laundry on site    1.091338
laundry_options_w/d hookups           2.254442
laundry_options_w/d in unit           2.864770
parking_options_attached garage       1.577553
parking_options_carport               1.413569
parking_options_detached garage       1.191164
parking_options_no parking            1.023253
parking_options_off-street parking    1.793367
parking_options_street parking        1.169766
parking_options_valet parking         1.004198
dtype: float64
```

In [ ]:

In [378]:
```
cols2 = ['sqfeet', 'beds', 'baths', 'dogs_allowed',
       'smoking_allowed', 'wheelchair_access', 'electric_vehicle_charge',
       'comes_furnished', 'median_rent_2019_12', 'crime_rate_per_100000',
       'population', 'Number_Of_Public_School', 'median_income',
        'type_assisted living', 'type_condo',
       'type_cottage/cabin', 'type_duplex', 'type_flat', 'type_house',
       'type_in-law', 'type_land', 'type_loft', 'type_manufactured',
       'type_townhouse',
       'laundry_options_laundry on site', 'laundry_options_no laundry on site',
       'laundry_options_w/d hookups', 'laundry_options_w/d in unit',
       'parking_options_carport',
       'parking_options_detached garage', 'parking_options_no parking',
       'parking_options_off-street parking', 'parking_options_street parking',
       'parking_options_valet parking']
VIF(X_train, cols2)
```

Out[378]:
```
sqfeet                                1.006066
beds                                  1.291800
baths                                 1.453894
dogs_allowed                          1.206069
smoking_allowed                       1.173535
wheelchair_access                     1.165095
electric_vehicle_charge               1.066233
comes_furnished                       1.089507
median_rent_2019_12                   1.642655
crime_rate_per_100000                 1.131319
population                            3.369284
Number_Of_Public_School               2.785421
median_income                         1.130833
type_assisted living                  1.000087
type_condo                            1.040490
type_cottage/cabin                    1.009216
type_duplex                           1.029965
type_flat                             1.010210
type_house                            1.126434
type_in-law                           1.004197
type_land                             1.000810
type_loft                             1.006366
type_manufactured                     1.037905
type_townhouse                        1.048001
laundry_options_laundry on site       1.514409
laundry_options_no laundry on site    1.052729
laundry_options_w/d hookups           1.630551
laundry_options_w/d in unit           1.854551
parking_options_carport               1.204420
parking_options_detached garage       1.086431
parking_options_no parking            1.019101
parking_options_off-street parking    1.322713
parking_options_street parking        1.082487
parking_options_valet parking         1.002780
dtype: float64
```

In [379]:
```
X_train_lr = X_train_lr.drop(['cats_allowed'], axis=1)
X_test_lr = X_test_lr.drop(['cats_allowed'], axis=1)
```

In [380]:
```
X_train = X_train.drop(['cats_allowed','population'], axis=1)
X_test = X_test.drop(['cats_allowed','population'], axis=1)
```

In [381]:
```
import statsmodels.formula.api as smf
train = X_train_lr.copy()
train['price'] = y_train.copy()
```

In [382]:
```python
lr3 = smf.ols(formula="price ~ " + " + ".join(["Q('" + i +"')" for i in cols2]),
                 data= train).fit()
print(lr3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.521
Model:                            OLS   Adj. R-squared:                  0.521
Method:                 Least Squares   F-statistic:                     5988.
Date:                Fri, 17 Dec 2021   Prob (F-statistic):               0.00
Time:                        20:32:12   Log-Likelihood:            -1.3840e+06
No. Observations:              186924   AIC:                         2.768e+06
Df Residuals:                  186889   BIC:                         2.768e+06
Df Model:                          34
Covariance Type:            nonrobust
==========================================================================================
==================
                                              coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------------------
------------------
Intercept                                 -86.9385      5.623    -15.462      0.000      -
97.959     -75.918
Q('sqfeet')                                 0.0042      0.000     17.385      0.000
0.004       0.005
Q('beds')                                  -7.1392      0.389    -18.335      0.000
-7.902      -6.376
Q('baths')                                219.7905      1.821    120.728      0.000      2
16.222     223.359
Q('dogs_allowed')                         -14.2547      2.229     -6.396      0.000      -
18.623      -9.886
Q('smoking_allowed')                      -86.7169      2.293    -37.826      0.000      -
91.210     -82.224
Q('wheelchair_access')                    -34.9831      3.596     -9.729      0.000      -
42.031     -27.936
Q('electric_vehicle_charge')              188.4801      8.584     21.957      0.000      1
71.655     205.305
Q('comes_furnished')                      -92.4974      4.701    -19.675      0.000     -1
01.712     -83.283
Q('median_rent_2019_12')                    0.6151      0.002    276.125      0.000
0.611       0.619
Q('crime_rate_per_100000')                  0.1224      0.004     28.953      0.000
0.114       0.131
Q('population')                          2.906e-05   1.47e-06     19.751      0.000     2.
62e-05    3.19e-05
Q('Number_Of_Public_School')               -0.0184      0.005     -3.997      0.000
-0.027      -0.009
Q('median_income')                       8.561e-05   1.91e-05      4.471      0.000     4.
81e-05       0.000
Q('type_assisted living')                1953.0117    397.519      4.913      0.000     11
73.884    2732.139
Q('type_condo')                           151.2010      7.478     20.220      0.000      1
36.545     165.857
Q('type_cottage/cabin')                    76.0246     23.631      3.217      0.001
29.709     122.340
Q('type_duplex')                           63.2099      8.305      7.611      0.000
46.932      79.488
Q('type_flat')                            223.7320     24.332      9.195      0.000      1
76.042     271.422
Q('type_house')                           147.7324      3.561     41.484      0.000      1
40.753     154.712
```

```
Q('type_in-law')                                -59.5787     46.315     -1.286      0.198     -1
50.354      31.197
Q('type_land')                                   10.4099    397.663      0.026      0.979     -7
69.000     789.819
Q('type_loft')                                  139.2582     20.370      6.836      0.000
99.333     179.183
Q('type_manufactured')                         -107.4166      9.321    -11.524      0.000     -1
25.686     -89.147
Q('type_townhouse')                              71.2607      5.066     14.068      0.000
61.332      81.189
Q('laundry_options_laundry on site')            -73.1620      3.062    -23.892      0.000      -
79.164     -67.160
Q('laundry_options_no laundry on site')        -140.4749     10.518    -13.356      0.000     -1
61.089    -119.861
Q('laundry_options_w/d hookups')                  8.4785      2.912      2.912      0.004
2.771       14.186
Q('laundry_options_w/d in unit')                175.2748      2.668     65.686      0.000      1
70.045     180.505
Q('parking_options_carport')                    -80.2680      3.206    -25.039      0.000      -
86.551     -73.985
Q('parking_options_detached garage')             38.7407      4.781      8.104      0.000
29.371      48.111
Q('parking_options_no parking')                  93.1629      9.872      9.437      0.000
73.813     112.513
Q('parking_options_off-street parking')         -56.7496      2.270    -24.999      0.000      -
61.199     -52.300
Q('parking_options_street parking')             -60.3856      4.806    -12.565      0.000      -
69.805     -50.966
Q('parking_options_valet parking')              624.7074     45.080     13.858      0.000      5
36.352     713.063
==============================================================================
Omnibus:                   106288.559   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          3562886.464
Skew:                           2.162   Prob(JB):                         0.00
Kurtosis:                      23.947   Cond. No.                     6.04e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
d.
[2] The condition number is large, 6.04e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [218]:

```python
lr2 = LinearRegression().fit(X_train, y_train)
```

In [219]:

```python
comparison_data = {'Linear Regression': ['{:.3f}'.format(OSR2(lr2, X_test, y_test, y_train
                                         '{:.4f}'.format(sqrt(mean_squared_error(y_test, l
                                         '{:.3f}'.format(mean_absolute_error(y_test, lr2.p
comparison_data
```

Out[219]:

```
{'Linear Regression': ['0.523', '397.8712', '250.999']}
```

3) Decision Tree Regressor

In [221]:

```python
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import KFold

grid_values = {'ccp_alpha': np.linspace(0, 0.001, 51)}

dtr = DecisionTreeRegressor(min_samples_leaf=5, min_samples_split=20, random_state=88)

### Note that the line below is important. It ensures that the training data is split into
### five folds randomly. Recall what we've seen in the discussion slides that by default,
### GridSearchCV will split the training data without shuffling.
cv = KFold(n_splits=5,random_state=1,shuffle=True)
### by setting random_state as a fixed number, we ensure that each time the GridSearchCV sp
### we get the same split.
dtr_cv = GridSearchCV(dtr, param_grid=grid_values, scoring='r2', cv=cv, verbose=0)
dtr_cv.fit(X_train, y_train)
```

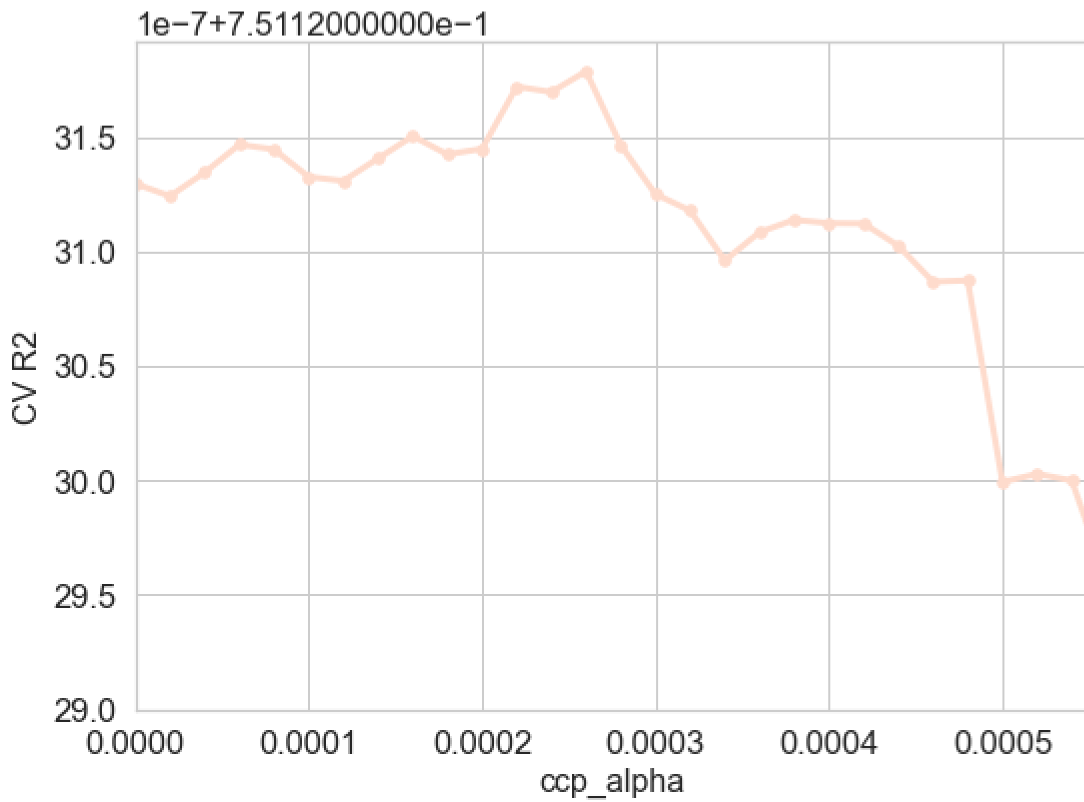
Out[221]:

```
GridSearchCV(cv=KFold(n_splits=5, random_state=1, shuffle=True),
             estimator=DecisionTreeRegressor(min_samples_leaf=5,
                                             min_samples_split=20,
                                             random_state=88),
             param_grid={'ccp_alpha': array([0.0e+00, 2.0e-05, 4.0e-05, 6.0e-05, 8.0e-05,
1.0e-04, 1.2e-04,
       1.4e-04, 1.6e-04, 1.8e-04, 2.0e-04, 2.2e-04, 2.4e-04, 2.6e-04,
       2.8e-04, 3.0e-04, 3.2e-04, 3.4e-04, 3.6e-04, 3.8e-04, 4.0e-04,
       4.2e-04, 4.4e-04, 4.6e-04, 4.8e-04, 5.0e-04, 5.2e-04, 5.4e-04,
       5.6e-04, 5.8e-04, 6.0e-04, 6.2e-04, 6.4e-04, 6.6e-04, 6.8e-04,
       7.0e-04, 7.2e-04, 7.4e-04, 7.6e-04, 7.8e-04, 8.0e-04, 8.2e-04,
       8.4e-04, 8.6e-04, 8.8e-04, 9.0e-04, 9.2e-04, 9.4e-04, 9.6e-04,
       9.8e-04, 1.0e-03])},
             scoring='r2')
```

In [223]:
```
ccp_alpha = dtr_cv.cv_results_['param_ccp_alpha'].data
R2_scores = dtr_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('ccp_alpha', fontsize=16)
plt.ylabel('CV R2', fontsize=16)
plt.scatter(ccp_alpha, R2_scores, s=30)
plt.plot(ccp_alpha, R2_scores, linewidth=3)
plt.grid(True, which='both')
plt.xlim([0, 0.00055])

plt.tight_layout()
plt.show()
```



In [420]:
```
print('Best ccp_alpha', dtr_cv.best_params_)
```

```
Best ccp_alpha {'ccp_alpha': 0.00026000000000000003}
```

In [224]:
```
print('Cross-validated R2:', round(dtr_cv.best_score_, 5))
print('OSR2:', round(OSR2(dtr_cv, X_test, y_test, y_train), 5))
```

```
Cross-validated R2: 0.75112
OSR2: 0.75385
```

In [ ]:

In [227]:
```
comparison_data = {'Decision Tree Regression': ['{:.3f}'.format(OSR2(dtr_cv, X_test, y_tes
                                                '{:.4f}'.format(sqrt(mean_squared_error(y_test, d
                                                '{:.3f}'.format(mean_absolute_error(y_test, dtr_c
comparison_data
```

Out[227]:
```
{'Decision Tree Regression': ['0.754', '285.8934', '129.494']}
```

4) Random Forest Regressor

In [228]:
```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(max_features=5, min_samples_leaf=5,
                           n_estimators = 500, random_state=88, verbose=2)
# Note: you can change the verbose parameter to control how much training progress is prin
rf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
```

In [229]:
```
rf.verbose = False

print('OSR2:', round(OSR2(rf, X_test, y_test, y_train), 5))
```

```
OSR2: 0.77922
```

In [232]:
```
comparison_data = {'RF Regression': ['{:.3f}'.format(OSR2(rf, X_test, y_test, y_train)),
                                     '{:.4f}'.format(sqrt(mean_squared_error(y_test, r
                                     '{:.3f}'.format(mean_absolute_error(y_test, rf.pre
comparison_data
```

Out[232]:
```
{'Decision Tree Regression': ['0.779', '270.7611', '142.560']}
```

5) Random Forest with CV

In [288]:
```
grid_values = {'max_features': ["sqrt", "log2"], 'min_samples_split': [10],
'n_estimators':  [200],
'random_state':  [88]}
rf = RandomForestRegressor()
rf_cv  =  GridSearchCV(rf,  param_grid=grid_values,  cv=5,n_jobs=-1)
rf_cv.fit(X_train, y_train)
```

Out[288]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'max_features': ['sqrt', 'log2'],
                         'min_samples_split': [10], 'n_estimators': [200],
                         'random_state': [88]})
```

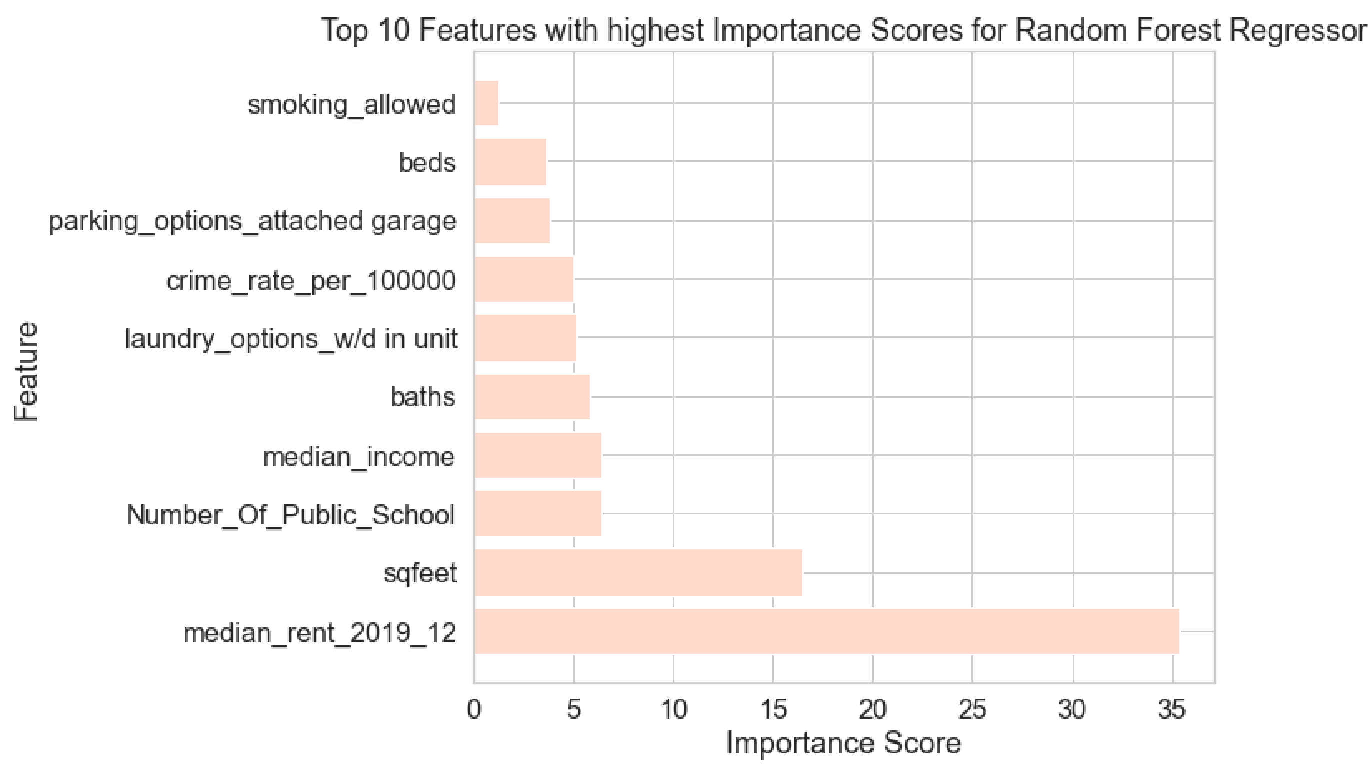
In [291]:
```
comparison_data = {'Random Forest Regression': ['{:.3f}'.format(OSR2(rf_cv, X_test, y_test
                                                '{:.4f}'.format(sqrt(mean_squared_error(y_test, r
                                                '{:.3f}'.format(mean_absolute_error(y_test, rf_cv
comparison_data
```

Out[291]:
```
{'Random Forest Regression': ['0.830', '237.3954', '105.052']}
```

In [347]:
```
sorted_idx = rf_cv.best_estimator_.feature_importances_.argsort()

feature_importances = rf_cv.best_estimator_.feature_importances_[sorted_idx[::-1]]
feature_names = X_train.columns[sorted_idx[::-1]]

plt.figure(figsize=(8,7))
plt.barh(feature_names[:10], 100*feature_importances[:10])
plt.ylabel('Feature')
plt.xlabel('Importance Score')
plt.title('Top 10 Features with highest Importance Scores for Random Forest Regressor')
plt.show()
```



In [330]:
```
feature_names
```

Out[330]:
```
Index(['median_rent_2019_12', 'sqfeet', 'Number_Of_Public_School',
       'median_income', 'baths', 'laundry_options_w/d in unit',
       'crime_rate_per_100000', 'parking_options_attached garage', 'beds',
       'smoking_allowed', 'comes_furnished', 'dogs_allowed',
       'laundry_options_laundry on site', 'parking_options_off-street parking',
       'type_house', 'type_apartment', 'wheelchair_access',
       'laundry_options_w/d hookups', 'electric_vehicle_charge',
       'parking_options_detached garage', 'parking_options_carport',
       'laundry_options_laundry in bldg', 'type_condo',
       'parking_options_street parking', 'type_townhouse',
       'parking_options_no parking', 'type_manufactured', 'type_duplex',
       'parking_options_valet parking', 'laundry_options_no laundry on site',
       'type_loft', 'type_flat', 'type_cottage/cabin', 'type_in-law',
       'type_assisted living', 'type_land'],
      dtype='object')
```

In [309]:
```
rf_importance = pd.DataFrame({'Feature' : X_train.columns,
              'Importance score': 100*rf_cv.best_estimator_.feature_importances_}).round(1
```

In [332]:
```
rf_importance.sort_values('Importance score',ascending=False, inplace=True)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_86304/4085820028.py in <module>
----> 1 rf_importance.sort_values('Importance score',ascending=False, inplace=True).plot(
kind = 'barh')

AttributeError: 'NoneType' object has no attribute 'plot'
```

In [338]:
```
rf_importance
```

Out[338]:

	Feature	Importance score
8	median_rent_2019_12	35.4
0	sqfeet	16.5
10	Number_Of_Public_School	6.4
11	median_income	6.4
2	baths	5.8
28	laundry_options_w/d in unit	5.1
9	crime_rate_per_100000	5.0
29	parking_options_attached garage	3.8
1	beds	3.7
4	smoking_allowed	1.3
7	comes_furnished	1.1
3	dogs_allowed	1.0
25	laundry_options_laundry on site	0.9
18	type_house	0.9
33	parking_options_off-street parking	0.9
5	wheelchair_access	0.8
12	type_apartment	0.8
27	laundry_options_w/d hookups	0.7
6	electric_vehicle_charge	0.6
31	parking_options_detached garage	0.5
30	parking_options_carport	0.5
24	laundry_options_laundry in bldg	0.4
34	parking_options_street parking	0.3
14	type_condo	0.3
23	type_townhouse	0.2
35	parking_options_valet parking	0.1
16	type_duplex	0.1
21	type_loft	0.1
26	laundry_options_no laundry on site	0.1
22	type_manufactured	0.1
32	parking_options_no parking	0.1
15	type_cottage/cabin	0.0
19	type_in-law	0.0
20	type_land	0.0
13	type_assisted living	0.0
17	type_flat	0.0

In [ ]:
```
grid_values = {'ccp_alpha': np.linspace(1,5,20)}

rf2 = RandomForestRegressor(min_samples_leaf=5, min_samples_split=20, random_state=88)
# Note: here we set verbose=2 to keep track of the progress (the running time) of the cross
cv = KFold(n_splits=3,random_state=1,shuffle=True)
rf_cv = GridSearchCV(rf2, param_grid=grid_values, scoring='r2', cv=cv,verbose=0)
rf_cv.fit(X_train, y_train)
```

In [230]:
```
## using GridSearchCV to find best max_features:

import time

grid_values = {'max_features': np.linspace(1,18,18, dtype='int32'),
               'min_samples_leaf': [5],
               'n_estimators': [500],
               'random_state': [88]}

tic = time.time()

rf2 = RandomForestRegressor()
# Note: here we set verbose=2 to keep track of the progress (the running time) of the cross
cv = KFold(n_splits=5,random_state=333,shuffle=True)
rf_cv = GridSearchCV(rf2, param_grid=grid_values, scoring='r2', cv=cv,verbose=2)
rf_cv.fit(X_train, y_train)

toc = time.time()

print('time:', round(toc-tic, 2),'s')
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[CV] END max_features=1, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  37.9s
[CV] END max_features=1, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  38.0s
[CV] END max_features=1, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  39.7s
[CV] END max_features=1, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  37.5s
[CV] END max_features=1, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  35.0s
[CV] END max_features=2, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  44.5s
[CV] END max_features=2, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  42.6s
[CV] END max_features=2, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  43.6s
[CV] END max_features=2, min_samples_leaf=5, n_estimators=500, random_state=88; total
time=  46.4s
```

In [231]:
```
max_features = rf_cv.cv_results_['param_max_features'].data
R2_scores = rf_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('max features', fontsize=16)
plt.ylabel('CV R2', fontsize=16)
plt.scatter(max_features, R2_scores, s=30)
plt.plot(max_features, R2_scores, linewidth=3)
plt.grid(True, which='both')
plt.xlim([1, 19])
plt.ylim([0.3, 0.6])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_86304/4082490374.py in <module>
----> 1 max_features = rf_cv.cv_results_['param_max_features'].data
      2 R2_scores = rf_cv.cv_results_['mean_test_score']
      3 
      4 plt.figure(figsize=(8, 6))
      5 plt.xlabel('max features', fontsize=16)

AttributeError: 'GridSearchCV' object has no attribute 'cv_results_'
```

In [ ]:
```
print(rf_cv.best_params_)
```

In [ ]:
```
print('Cross-validated R2:', round(rf_cv.best_score_, 5))
print('OSR2:', round(OSR2(rf_cv, X_test, y_test, y_train), 5))
```

In [ ]:
```
pd.DataFrame({'Feature' : X_train.columns,
              'Importance score': 100*rf_cv.best_estimator_.feature_importances_}).round(1
```

In [ ]:
```
plt.figure(figsize=(8,7))
plt.barh(X_train.columns, 100*rf_cv.best_estimator_.feature_importances_)
plt.show()
```

In [ ]:
```
comparison_data = {'Decision Tree Regression': ['{:.3f}'.format(OSR2(rf_cv, X_test, y_test
                                                '{:.4f}'.format(sqrt(mean_squared_error(y_test, r
                                                '{:.3f}'.format(mean_absolute_error(y_test, rf_cv
comparison_data
```

In [ ]:
```
grid_values = {'max_features': ["sqrt", "log2"], 'min_samples_split': [10],
'n_estimators':  [200],
'random_state':  [88]}
rf = RandomForestRegressor()
rf_cv  =  GridSearchCV(rf,  param_grid=grid_values,  cv=5,n_jobs=-1)
rf_cv.fit(X_train, y_train)
```

In [ ]:
```
comparison_data = {'Random Forest Regression': ['{:.3f}'.format(OSR2(rf_cv, X_test, y_test
                                                '{:.4f}'.format(sqrt(mean_squared_error(y_test, r
                                                '{:.3f}'.format(mean_absolute_error(y_test, rf_cv
comparison_data
```

In [ ]:
```
sorted_idx = rf_cv.best_estimator_.feature_importances_.argsort()

feature_importances = rf_cv.best_estimator_.feature_importances_[sorted_idx[::-1]]
feature_names = X_train.columns[sorted_idx[::-1]]

plt.figure(figsize=(8,7))
plt.barh(feature_names[:10], 100*feature_importances[:10])
plt.show()
```

4)

In [255]:
```
gbr = GradientBoostingRegressor(max_features=5, min_samples_leaf=5,
                                n_estimators = 500, random_state=88, verbose=2)
gbr.fit(X_train, y_train)
```

```
      Iter       Train Loss   Remaining Time 
         1      318249.4960           25.46s
         2      309892.3465           21.92s
         3      284424.9800           26.35s
         4      264455.3034           25.67s
         5      257600.3145           25.75s
         6      249816.5249           25.78s
         7      244799.4491           24.37s
         8      239331.7135           24.11s
         9      235418.1761           24.23s
        10      232262.2220           23.87s
        11      227529.8978           23.52s
        12      213064.6290           23.76s
        13      207567.8279           23.46s
        14      204071.6426           23.48s
        15      195626.1276           23.65s
        16      186788.9217           24.03s
        17      182483.3867           23.79s
        18      176332.5782           23.84s
```

In [254]:
```
comparison_data = {'Boosting Regression': ['{:.3f}'.format(OSR2(gbr, X_test, y_test, y_tra
                                           '{:.4f}'.format(sqrt(mean_squared_error(y_test, gl
                                           '{:.3f}'.format(mean_absolute_error(y_test, gbr.pı
comparison_data
```

Out[254]:
```
{'RF Regression': ['0.704', '313.4054', '186.340']}
```

In [354]:
```python
import tensorflow
tensorflow.random.set_seed(1)
from tensorflow.python.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.wrappers.scikit_learn import KerasRegressor

model = Sequential()
model.add(Dense(123, input_dim=36, kernel_initializer='normal', activation='relu'))
model.add(Dense(2670, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 123)               4551
_________________________________________________________________
dense_25 (Dense)             (None, 2670)              331080
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 2671
=================================================================
Total params: 338,302
Trainable params: 338,302
Non-trainable params: 0
_________________________________________________________________
```

In [355]:
```python
import sklearn.metrics as metrics
```

In [356]:
```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
model.fit(X_train, y_train, epochs=100, batch_size=150, verbose=1, validation_split=0.2)
predictions = model.predict(X_train)
print('R2 on train---')
print(metrics.r2_score(y_train,predictions))
```

```
Epoch 1/100
997/997 [==============================] - 8s 7ms/step - loss: 2178569.5000 - mse: 217856
9.5000 - mae: 354.5947 - val_loss: 229667.2656 - val_mse: 229667.2656 - val_mae: 263.5162
Epoch 2/100
997/997 [==============================] - 7s 7ms/step - loss: 537332.0000 - mse: 537332.
0000 - mae: 271.1404 - val_loss: 344246.4375 - val_mse: 344246.4375 - val_mae: 259.9053
Epoch 3/100
997/997 [==============================] - 7s 7ms/step - loss: 640761.0000 - mse: 640761.
0000 - mae: 282.6946 - val_loss: 307782.7500 - val_mse: 307782.7188 - val_mae: 293.9029
Epoch 4/100
997/997 [==============================] - 7s 7ms/step - loss: 573024.0000 - mse: 573024.
0000 - mae: 292.7954 - val_loss: 302304.9375 - val_mse: 302304.9375 - val_mae: 252.8325
Epoch 5/100
997/997 [==============================] - 7s 7ms/step - loss: 558458.5000 - mse: 558458.
5000 - mae: 281.6955 - val_loss: 3319483.5000 - val_mse: 3319483.2500 - val_mae: 538.5547
Epoch 6/100
997/997 [==============================] - 7s 7ms/step - loss: 1775622.5000 - mse: 177562
2.5000 - mae: 339.1584 - val_loss: 283360.6562 - val_mse: 283360.6562 - val_mae: 348.7722
Epoch 7/100
997/997 [==============================] - 7s 7ms/step - loss: 210920.6094 - mse: 210920.
6094 - mae: 305.3047 - val_loss: 208179.7344 - val_mse: 208179.7344 - val_mae: 295.4032
Epoch 8/100
997/997 [==============================] - 7s 7ms/step - loss: 203088.7500 - mse: 203088.
7500 - mae: 299.0921 - val_loss: 212223.0312 - val_mse: 212223.0312 - val_mae: 298.1946
Epoch 9/100
997/997 [==============================] - 7s 7ms/step - loss: 202227.7656 - mse: 202227.
7656 - mae: 297.5894 - val_loss: 221459.5938 - val_mse: 221459.6094 - val_mae: 293.5531
Epoch 10/100
997/997 [==============================] - 7s 7ms/step - loss: 198412.2656 - mse: 198412.
2656 - mae: 294.3273 - val_loss: 208627.8594 - val_mse: 208627.8594 - val_mae: 285.8324
Epoch 11/100
997/997 [==============================] - 7s 7ms/step - loss: 215395.5156 - mse: 215395.
5156 - mae: 294.1674 - val_loss: 424617.2500 - val_mse: 424617.2500 - val_mae: 279.3413
Epoch 12/100
997/997 [==============================] - 7s 7ms/step - loss: 312863.5938 - mse: 312863.
5938 - mae: 293.6694 - val_loss: 193498.6562 - val_mse: 193498.6562 - val_mae: 290.1169
Epoch 13/100
997/997 [==============================] - 7s 7ms/step - loss: 189348.7344 - mse: 189348.
7344 - mae: 286.8516 - val_loss: 189911.2344 - val_mse: 189911.2344 - val_mae: 288.9438
Epoch 14/100
997/997 [==============================] - 7s 7ms/step - loss: 186013.1719 - mse: 186013.
1719 - mae: 283.1584 - val_loss: 197230.1094 - val_mse: 197230.1094 - val_mae: 271.8239
Epoch 15/100
997/997 [==============================] - 7s 7ms/step - loss: 196889.2188 - mse: 196889.
2188 - mae: 284.5005 - val_loss: 249418.9531 - val_mse: 249418.9531 - val_mae: 308.9533
Epoch 16/100
997/997 [==============================] - 7s 7ms/step - loss: 189807.2188 - mse: 189807.
2188 - mae: 281.3250 - val_loss: 183400.8750 - val_mse: 183400.8750 - val_mae: 267.5586
Epoch 17/100
997/997 [==============================] - 7s 7ms/step - loss: 188053.8281 - mse: 188053.
8281 - mae: 278.3419 - val_loss: 181965.2031 - val_mse: 181965.1875 - val_mae: 278.5824
Epoch 18/100
997/997 [==============================] - 7s 7ms/step - loss: 166472.2812 - mse: 166472.
2812 - mae: 262.7895 - val_loss: 154322.8438 - val_mse: 154322.8438 - val_mae: 239.8181
Epoch 19/100
```

```
997/997 [==============================] - 7s 7ms/step - loss: 154245.1719 - mse: 154245.
1562 - mae: 250.8082 - val_loss: 158143.5312 - val_mse: 158143.5312 - val_mae: 263.7492
Epoch 20/100
997/997 [==============================] - 7s 7ms/step - loss: 152124.6094 - mse: 152124.
6406 - mae: 249.3533 - val_loss: 147414.5938 - val_mse: 147414.5938 - val_mae: 243.4297
Epoch 21/100
997/997 [==============================] - 7s 7ms/step - loss: 149244.5312 - mse: 149244.
5312 - mae: 245.3097 - val_loss: 176109.1406 - val_mse: 176109.1406 - val_mae: 258.3642
Epoch 22/100
997/997 [==============================] - 7s 7ms/step - loss: 157562.6562 - mse: 157562.
6562 - mae: 244.4856 - val_loss: 160852.9844 - val_mse: 160852.9844 - val_mae: 274.4952
Epoch 23/100
997/997 [==============================] - 7s 7ms/step - loss: 148627.6250 - mse: 148627.
6250 - mae: 245.8055 - val_loss: 153339.2812 - val_mse: 153339.2812 - val_mae: 238.0928
Epoch 24/100
997/997 [==============================] - 7s 7ms/step - loss: 146266.1875 - mse: 146266.
1875 - mae: 242.9894 - val_loss: 144851.4844 - val_mse: 144851.5000 - val_mae: 237.7784
Epoch 25/100
997/997 [==============================] - 7s 7ms/step - loss: 146189.0625 - mse: 146189.
0625 - mae: 241.8049 - val_loss: 146550.9375 - val_mse: 146550.9375 - val_mae: 246.2441
Epoch 26/100
997/997 [==============================] - 7s 7ms/step - loss: 144877.5469 - mse: 144877.
5469 - mae: 241.5396 - val_loss: 151231.9219 - val_mse: 151231.9375 - val_mae: 239.6295
Epoch 27/100
997/997 [==============================] - 7s 7ms/step - loss: 145196.0469 - mse: 145196.
0469 - mae: 241.8298 - val_loss: 143789.1562 - val_mse: 143789.1562 - val_mae: 241.4591
Epoch 28/100
997/997 [==============================] - 7s 7ms/step - loss: 147538.3438 - mse: 147538.
3438 - mae: 240.2508 - val_loss: 142825.7344 - val_mse: 142825.7188 - val_mae: 232.2558
Epoch 29/100
997/997 [==============================] - 6s 6ms/step - loss: 143203.3125 - mse: 143203.
3125 - mae: 239.8859 - val_loss: 146435.2500 - val_mse: 146435.2500 - val_mae: 234.5027
Epoch 30/100
997/997 [==============================] - 6s 6ms/step - loss: 143572.5000 - mse: 143572.
5000 - mae: 240.5518 - val_loss: 195250.6719 - val_mse: 195250.6719 - val_mae: 259.0006
Epoch 31/100
997/997 [==============================] - 6s 6ms/step - loss: 202824.1719 - mse: 202824.
1719 - mae: 242.3080 - val_loss: 168241.5938 - val_mse: 168241.5938 - val_mae: 235.7322
Epoch 32/100
997/997 [==============================] - 6s 6ms/step - loss: 150024.5000 - mse: 150024.
5000 - mae: 239.5619 - val_loss: 159463.5938 - val_mse: 159463.5938 - val_mae: 236.4779
Epoch 33/100
997/997 [==============================] - 6s 6ms/step - loss: 145919.2656 - mse: 145919.
2656 - mae: 238.5564 - val_loss: 158583.6719 - val_mse: 158583.6719 - val_mae: 248.3937
Epoch 34/100
997/997 [==============================] - 6s 6ms/step - loss: 140972.0469 - mse: 140972.
0469 - mae: 237.5797 - val_loss: 140824.4375 - val_mse: 140824.4375 - val_mae: 231.9164
Epoch 35/100
997/997 [==============================] - 6s 6ms/step - loss: 140542.7188 - mse: 140542.
7188 - mae: 237.4907 - val_loss: 153144.2656 - val_mse: 153144.2656 - val_mae: 243.9295
Epoch 36/100
997/997 [==============================] - 6s 6ms/step - loss: 141597.5000 - mse: 141597.
5000 - mae: 237.6303 - val_loss: 148295.7344 - val_mse: 148295.7344 - val_mae: 234.0125
Epoch 37/100
997/997 [==============================] - 6s 6ms/step - loss: 144061.8438 - mse: 144061.
8438 - mae: 237.4074 - val_loss: 140636.9688 - val_mse: 140636.9688 - val_mae: 231.6590
Epoch 38/100
997/997 [==============================] - 6s 6ms/step - loss: 139939.1250 - mse: 139939.
1250 - mae: 237.6498 - val_loss: 137519.1562 - val_mse: 137519.1562 - val_mae: 232.2253
Epoch 39/100
997/997 [==============================] - 6s 6ms/step - loss: 139682.1406 - mse: 139682.
1406 - mae: 236.1107 - val_loss: 137594.1719 - val_mse: 137594.1719 - val_mae: 231.4776
```

```
Epoch 40/100
997/997 [==============================] - 6s 6ms/step - loss: 178297.8281 - mse: 178297.
8281 - mae: 237.6006 - val_loss: 139431.4375 - val_mse: 139431.4375 - val_mae: 227.9279
Epoch 41/100
997/997 [==============================] - 6s 6ms/step - loss: 150346.8438 - mse: 150346.
8438 - mae: 236.9604 - val_loss: 140907.8750 - val_mse: 140907.8750 - val_mae: 232.3125
Epoch 42/100
997/997 [==============================] - 7s 7ms/step - loss: 139282.8281 - mse: 139282.
8281 - mae: 236.6678 - val_loss: 141798.7500 - val_mse: 141798.7500 - val_mae: 229.9337
Epoch 43/100
997/997 [==============================] - 6s 6ms/step - loss: 138065.2500 - mse: 138065.
2344 - mae: 235.3316 - val_loss: 162280.6250 - val_mse: 162280.6250 - val_mae: 255.6343
Epoch 44/100
997/997 [==============================] - 6s 6ms/step - loss: 138370.4688 - mse: 138370.
4688 - mae: 236.1839 - val_loss: 137590.1875 - val_mse: 137590.1875 - val_mae: 232.1606
Epoch 45/100
997/997 [==============================] - 6s 6ms/step - loss: 137186.9375 - mse: 137186.
9375 - mae: 234.3390 - val_loss: 151677.1875 - val_mse: 151677.1875 - val_mae: 261.7673
Epoch 46/100
997/997 [==============================] - 6s 6ms/step - loss: 137171.8594 - mse: 137171.
8594 - mae: 234.3788 - val_loss: 136255.7500 - val_mse: 136255.7500 - val_mae: 225.0896
Epoch 47/100
997/997 [==============================] - 6s 6ms/step - loss: 137214.8594 - mse: 137214.
8594 - mae: 234.4433 - val_loss: 140046.2656 - val_mse: 140046.2656 - val_mae: 228.1329
Epoch 48/100
997/997 [==============================] - 7s 7ms/step - loss: 137443.1719 - mse: 137443.
1719 - mae: 234.7038 - val_loss: 141975.5781 - val_mse: 141975.5781 - val_mae: 226.9374
Epoch 49/100
997/997 [==============================] - 6s 6ms/step - loss: 137587.5938 - mse: 137587.
5938 - mae: 234.2762 - val_loss: 143345.6406 - val_mse: 143345.6406 - val_mae: 249.1173
Epoch 50/100
997/997 [==============================] - 6s 6ms/step - loss: 136733.4219 - mse: 136733.
4219 - mae: 233.7984 - val_loss: 140832.0000 - val_mse: 140832.0000 - val_mae: 230.3743
Epoch 51/100
997/997 [==============================] - 6s 6ms/step - loss: 135216.1875 - mse: 135216.
1875 - mae: 232.8970 - val_loss: 150601.1250 - val_mse: 150601.1094 - val_mae: 244.7561
Epoch 52/100
997/997 [==============================] - 6s 6ms/step - loss: 135455.9531 - mse: 135455.
9531 - mae: 233.4963 - val_loss: 133022.7500 - val_mse: 133022.7500 - val_mae: 227.1853
Epoch 53/100
997/997 [==============================] - 6s 6ms/step - loss: 134660.1562 - mse: 134660.
1562 - mae: 232.6239 - val_loss: 136473.5312 - val_mse: 136473.5312 - val_mae: 227.7013
Epoch 54/100
997/997 [==============================] - 6s 6ms/step - loss: 134411.9375 - mse: 134411.
9375 - mae: 232.0114 - val_loss: 134262.5156 - val_mse: 134262.5156 - val_mae: 225.4827
Epoch 55/100
997/997 [==============================] - 6s 6ms/step - loss: 134462.8750 - mse: 134462.
8750 - mae: 232.3732 - val_loss: 158749.2656 - val_mse: 158749.2656 - val_mae: 255.6693
Epoch 56/100
997/997 [==============================] - 6s 6ms/step - loss: 134512.5156 - mse: 134512.
5156 - mae: 232.8555 - val_loss: 137489.9062 - val_mse: 137489.9062 - val_mae: 231.5816
Epoch 57/100
997/997 [==============================] - 6s 6ms/step - loss: 134530.1250 - mse: 134530.
1250 - mae: 232.4032 - val_loss: 147388.1406 - val_mse: 147388.1406 - val_mae: 258.8509
Epoch 58/100
997/997 [==============================] - 6s 6ms/step - loss: 135816.5312 - mse: 135816.
5312 - mae: 233.1191 - val_loss: 133903.1875 - val_mse: 133903.1875 - val_mae: 229.1787
Epoch 59/100
997/997 [==============================] - 6s 6ms/step - loss: 133560.8281 - mse: 133560.
8281 - mae: 230.4206 - val_loss: 141709.0625 - val_mse: 141709.0625 - val_mae: 230.3804
Epoch 60/100
997/997 [==============================] - 6s 6ms/step - loss: 134334.2031 - mse: 134334.
```

```
2031 - mae: 231.9054 - val_loss: 133806.5938 - val_mse: 133806.5938 - val_mae: 232.8995
Epoch 61/100
997/997 [==============================] - 6s 6ms/step - loss: 132655.3906 - mse: 132655.
3906 - mae: 230.7782 - val_loss: 131806.9844 - val_mse: 131806.9844 - val_mae: 230.1927
Epoch 62/100
997/997 [==============================] - 6s 6ms/step - loss: 132948.5312 - mse: 132948.
5312 - mae: 231.3015 - val_loss: 152064.3594 - val_mse: 152064.3594 - val_mae: 269.6455
Epoch 63/100
997/997 [==============================] - 6s 6ms/step - loss: 132738.2188 - mse: 132738.
2188 - mae: 230.9946 - val_loss: 133140.4219 - val_mse: 133140.4219 - val_mae: 223.4678
Epoch 64/100
997/997 [==============================] - 6s 6ms/step - loss: 135523.6875 - mse: 135523.
6875 - mae: 232.9394 - val_loss: 132390.6875 - val_mse: 132390.6875 - val_mae: 224.4917
Epoch 65/100
997/997 [==============================] - 6s 6ms/step - loss: 135861.8438 - mse: 135861.
8438 - mae: 232.5321 - val_loss: 136733.7812 - val_mse: 136733.7812 - val_mae: 229.7266
Epoch 66/100
997/997 [==============================] - 6s 6ms/step - loss: 131263.3906 - mse: 131263.
3906 - mae: 228.8659 - val_loss: 141008.1094 - val_mse: 141008.1094 - val_mae: 225.3926
Epoch 67/100
997/997 [==============================] - 6s 6ms/step - loss: 133431.8438 - mse: 133431.
8438 - mae: 230.1638 - val_loss: 154275.1875 - val_mse: 154275.1875 - val_mae: 239.1538
Epoch 68/100
997/997 [==============================] - 6s 6ms/step - loss: 135864.4844 - mse: 135864.
4844 - mae: 231.3972 - val_loss: 132181.5000 - val_mse: 132181.5000 - val_mae: 226.4135
Epoch 69/100
997/997 [==============================] - 6s 6ms/step - loss: 130738.4766 - mse: 130738.
4766 - mae: 229.2983 - val_loss: 140112.5156 - val_mse: 140112.5156 - val_mae: 246.9799
Epoch 70/100
997/997 [==============================] - 6s 6ms/step - loss: 131618.2500 - mse: 131618.
2500 - mae: 230.3023 - val_loss: 145452.6562 - val_mse: 145452.6562 - val_mae: 258.5497
Epoch 71/100
997/997 [==============================] - 6s 6ms/step - loss: 130537.2734 - mse: 130537.
2734 - mae: 229.2898 - val_loss: 134677.2188 - val_mse: 134677.2188 - val_mae: 227.3626
Epoch 72/100
997/997 [==============================] - 6s 6ms/step - loss: 130393.5547 - mse: 130393.
5547 - mae: 228.4528 - val_loss: 131647.1094 - val_mse: 131647.1094 - val_mae: 224.1965
Epoch 73/100
997/997 [==============================] - 6s 6ms/step - loss: 130271.0938 - mse: 130271.
0938 - mae: 228.6604 - val_loss: 132871.7969 - val_mse: 132871.7812 - val_mae: 225.6233
Epoch 74/100
997/997 [==============================] - 6s 6ms/step - loss: 132382.3750 - mse: 132382.
3750 - mae: 229.5387 - val_loss: 128724.8906 - val_mse: 128724.8906 - val_mae: 220.2993
Epoch 75/100
997/997 [==============================] - 6s 6ms/step - loss: 130922.2969 - mse: 130922.
2969 - mae: 228.8475 - val_loss: 129740.3750 - val_mse: 129740.3750 - val_mae: 220.8179
Epoch 76/100
997/997 [==============================] - 6s 6ms/step - loss: 129174.0781 - mse: 129174.
0781 - mae: 226.7299 - val_loss: 134263.9219 - val_mse: 134263.9062 - val_mae: 238.1487
Epoch 77/100
997/997 [==============================] - 6s 6ms/step - loss: 133215.9531 - mse: 133215.
9531 - mae: 231.0426 - val_loss: 129335.8828 - val_mse: 129335.8984 - val_mae: 222.3159
Epoch 78/100
997/997 [==============================] - 6s 6ms/step - loss: 128892.8672 - mse: 128892.
8672 - mae: 227.3899 - val_loss: 129791.5391 - val_mse: 129791.5391 - val_mae: 221.1563
Epoch 79/100
997/997 [==============================] - 6s 6ms/step - loss: 129332.0625 - mse: 129332.
0625 - mae: 228.2420 - val_loss: 128334.2109 - val_mse: 128334.2109 - val_mae: 222.3342
Epoch 80/100
997/997 [==============================] - 7s 7ms/step - loss: 129289.8984 - mse: 129289.
8984 - mae: 228.2259 - val_loss: 130148.3750 - val_mse: 130148.3750 - val_mae: 231.8275
Epoch 81/100
```

```
997/997 [==============================] - 6s 6ms/step - loss: 128364.0000 - mse: 128364.
0000 - mae: 226.7438 - val_loss: 129775.0547 - val_mse: 129775.0625 - val_mae: 220.8748
Epoch 82/100
997/997 [==============================] - 6s 6ms/step - loss: 129357.0078 - mse: 129357.
0078 - mae: 227.1084 - val_loss: 130788.7109 - val_mse: 130788.7109 - val_mae: 223.5847
Epoch 83/100
997/997 [==============================] - 6s 6ms/step - loss: 132237.6250 - mse: 132237.
6250 - mae: 228.0929 - val_loss: 129354.5625 - val_mse: 129354.5625 - val_mae: 221.7545
Epoch 84/100
997/997 [==============================] - 6s 6ms/step - loss: 128261.6719 - mse: 128261.
6719 - mae: 226.7873 - val_loss: 126664.9688 - val_mse: 126664.9688 - val_mae: 222.6024
Epoch 85/100
997/997 [==============================] - 6s 6ms/step - loss: 128452.1094 - mse: 128452.
1094 - mae: 227.1759 - val_loss: 140759.6719 - val_mse: 140759.6719 - val_mae: 253.7267
Epoch 86/100
997/997 [==============================] - 7s 7ms/step - loss: 127767.4688 - mse: 127767.
4688 - mae: 226.3548 - val_loss: 134513.7812 - val_mse: 134513.7812 - val_mae: 225.1212
Epoch 87/100

997/997 [==============================] - 7s 7ms/step - loss: 140892.8750 - mse: 140892.
8750 - mae: 234.5445 - val_loss: 135510.4688 - val_mse: 135510.4688 - val_mae: 230.9668
Epoch 88/100
997/997 [==============================] - 6s 6ms/step - loss: 134975.5000 - mse: 134975.
5000 - mae: 232.4763 - val_loss: 133506.1719 - val_mse: 133506.1719 - val_mae: 228.5215
Epoch 89/100
997/997 [==============================] - 7s 7ms/step - loss: 133718.6719 - mse: 133718.
6719 - mae: 231.9824 - val_loss: 133511.7031 - val_mse: 133511.7031 - val_mae: 228.3299
Epoch 90/100
997/997 [==============================] - 6s 6ms/step - loss: 132401.5312 - mse: 132401.
5312 - mae: 230.0097 - val_loss: 132807.0781 - val_mse: 132807.0781 - val_mae: 226.3419
Epoch 91/100
997/997 [==============================] - 6s 6ms/step - loss: 132583.0781 - mse: 132583.
0781 - mae: 230.8392 - val_loss: 139836.2656 - val_mse: 139836.2656 - val_mae: 230.8142
Epoch 92/100
997/997 [==============================] - 6s 6ms/step - loss: 132191.9219 - mse: 132191.
9219 - mae: 229.8558 - val_loss: 134768.7344 - val_mse: 134768.7344 - val_mae: 231.2129
Epoch 93/100
997/997 [==============================] - 6s 6ms/step - loss: 133300.1250 - mse: 133300.
1250 - mae: 230.9251 - val_loss: 133013.4844 - val_mse: 133013.4844 - val_mae: 226.7774
Epoch 94/100
997/997 [==============================] - 6s 6ms/step - loss: 131756.8594 - mse: 131756.
8594 - mae: 229.3573 - val_loss: 145338.5312 - val_mse: 145338.5312 - val_mae: 256.9131
Epoch 95/100
997/997 [==============================] - 6s 6ms/step - loss: 131917.5156 - mse: 131917.
5156 - mae: 230.0249 - val_loss: 150185.9375 - val_mse: 150185.9375 - val_mae: 245.4980
Epoch 96/100
997/997 [==============================] - 6s 6ms/step - loss: 131589.3906 - mse: 131589.
3906 - mae: 229.5597 - val_loss: 131196.6562 - val_mse: 131196.6562 - val_mae: 229.0036
Epoch 97/100
997/997 [==============================] - 6s 6ms/step - loss: 132038.2500 - mse: 132038.
2500 - mae: 230.3468 - val_loss: 132394.4531 - val_mse: 132394.4531 - val_mae: 230.9092
Epoch 98/100
997/997 [==============================] - 6s 6ms/step - loss: 130944.3750 - mse: 130944.
3594 - mae: 228.8186 - val_loss: 131998.9062 - val_mse: 131998.9062 - val_mae: 222.6915
Epoch 99/100
997/997 [==============================] - 6s 6ms/step - loss: 131559.3750 - mse: 131559.
3750 - mae: 229.4675 - val_loss: 138134.0312 - val_mse: 138134.0312 - val_mae: 230.4541
Epoch 100/100
997/997 [==============================] - 6s 6ms/step - loss: 130541.3672 - mse: 130541.
3672 - mae: 228.7942 - val_loss: 141408.8750 - val_mse: 141408.8750 - val_mae: 246.5306
R2 on train---
0.581007686457187
```

In [357]:
```
def OSR2(model, X_test, y_test, y_train):

    y_pred = model.predict(X_test)
    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(y_train))**2)

    return (1 - SSE/SST)
```

In [385]:
```
def OSR22(y_train, y_test, y_pred):

    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(y_train))**2)

    return (1 - SSE/SST)
```

In [358]:
```
print(metrics.r2_score(y_train,predictions))
```

```
0.581007686457187
```

In [386]:
```
model_pred = model.predict(X_test)
print(OSR22(y_train, y_test, model_pred.flatten()))
```

```
0.5719576608494573
```

In [ ]:
```
model = Sequential()
model.add(Dense(123, input_dim=36, kernel_initializer='normal', activation='relu'))
model.add(Dense(2670, activation='relu'))
model.add(Dense(2670, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

In [267]:
```
from tensorflow.keras.layers import Input
```

In [273]:

```python
nn_mod_2 = tensorflow.keras.Sequential()
nn_mod_2.add(Input(shape=(36,)))
nn_mod_2.add(Dense(15, activation='sigmoid'))
nn_mod_2.add(Dense(15, activation='sigmoid'))
nn_mod_2.add(Dense(15, activation='sigmoid'))
nn_mod_2.add(Dense(1))

opt = RMSprop()
nn_mod_2.compile(optimizer=opt,
                 loss='mse',
               metrics=['mean_squared_error'])

tic = time.time()
nn_mod_2.fit(X_train,
           y_train,
           epochs=50,
           validation_split=0.2)
toc = time.time()
print('Neural Net 2 time:', round(toc-tic, 2),'s')
```

```
Epoch 1/50
4674/4674 [==============================] - 5s 987us/step - loss: 1587074.7500 - mea
n_squared_error: 1587074.7500 - val_loss: 1520060.5000 - val_mean_squared_error: 1520
060.5000
Epoch 2/50
4674/4674 [==============================] - 4s 955us/step - loss: 1425526.3750 - mea
n_squared_error: 1425526.3750 - val_loss: 1363698.6250 - val_mean_squared_error: 1363
698.6250
Epoch 3/50
4674/4674 [==============================] - 4s 926us/step - loss: 1275170.7500 - mea
n_squared_error: 1275170.7500 - val_loss: 1218539.5000 - val_mean_squared_error: 1218
539.5000
Epoch 4/50
4674/4674 [==============================] - 5s 1ms/step - loss: 1136008.0000 - mean_
squared_error: 1136008.0000 - val_loss: 1084529.6250 - val_mean_squared_error: 108452
9.6250
Epoch 5/50
4674/4674 [==============================] - 5s 1ms/step - loss: 1008156.7500 - mean_
squared_error: 1008156.7500 - val_loss: 961760.4375 - val_mean_squared_error: 961760.
```

In [277]:

```python
def OSR2(y_train, y_test, y_pred):

    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(y_train))**2)

    return (1 - SSE/SST)
```

In [278]:

```python
nn_pred_2 = nn_mod_2.predict(X_test)
print(OSR2(y_train, y_test, nn_pred_2.flatten()))
```

```
-3.620839420626076e-05
```

In [387]:
```
comparison_data = {'Linear Regression': ['{:.3f}'.format(OSR2(lr3, X_test_lr, y_test_lr, y
                                         '{:.4f}'.format(sqrt(mean_squared_error(y_test_lr
                                         '{:.3f}'.format(mean_absolute_error(y_test_lr, lr
                   'Decision Tree Regressor': ['{:.3f}'.format(OSR2(dtr_cv, X_test, y_test
                                               '{:.4f}'.format(sqrt(mean_squared_error(y_te
                                               '{:.3f}'.format(mean_absolute_error(y_test,
                   'Random Forest Regressor': ['{:.3f}'.format(OSR2(rf_cv, X_test, y_test,
                                     '{:.4f}'.format(sqrt(mean_squared_error(y_test, rf_cv
                                     '{:.3f}'.format(mean_absolute_error(y_test, rf_cv.pre
                   'Gradient Boosted Regressor': ['{:.3f}'.format(OSR2(gbr, X_test, y_test
                                              '{:.4f}'.format(sqrt(mean_squared_error(y_tes
                                              '{:.3f}'.format(mean_absolute_error(y_test,
                  'Neural Networks':['{:.3f}'.format(OSR22(y_train, y_test, model_pred.fla
                                              '{:.4f}'.format(sqrt(130541.3672)),
                                              '{:.3f}'.format(228.7942)]}

comparison_table = pd.DataFrame(data=comparison_data, index=['OSR2', 'Out-of-sample RMSE',
comparison_table.style.set_properties(**{'font-size': '12pt',}).set_table_styles([{'select
```

Out[387]:

	Linear Regression	Decision Tree Regressor	Random Forest Regressor	Gradient Boosted Regressor	Neural Networks
OSR2	0.512	0.754	0.830	0.704	0.572
Out-of-sample RMSE	402.3966	285.8934	237.3954	313.4054	361.3051
Out-of-sample MAE	255.757	129.494	105.052	186.340	228.794

In [388]:
```
import time

def bootstrap_validation(test_data, test_label, train_label, model, metrics_list, sample=5
    tic = time.time()
    n_sample = sample
    n_metrics = len(metrics_list)
    output_array=np.zeros([n_sample, n_metrics])
    output_array[:]=np.nan
    print(output_array.shape)
    for bs_iter in range(n_sample):
        bs_index = np.random.choice(test_data.index, len(test_data.index), replace=True)
        bs_data = test_data.loc[bs_index]
        bs_label = test_label.loc[bs_index]
        bs_predicted = model.predict(bs_data)
        for metrics_iter in range(n_metrics):
            metrics = metrics_list[metrics_iter]
            output_array[bs_iter, metrics_iter]=metrics(bs_predicted,bs_label,train_label)
#         if bs_iter % 100 == 0:
#             print(bs_iter, time.time()-tic)
    output_df = pd.DataFrame(output_array)
    return output_df
```

In [389]:
```python
def OS_R_squared(predictions, y_test,y_train):
    SSE = np.sum((y_test-predictions)**2)
    SST = np.sum((y_test-np.mean(y_train))**2)
    r2 = 1-SSE/SST
    return r2

def mean_squared_error(predictions, y_test,y_train):
    MSE = np.mean((y_test-predictions)**2)
    return MSE

def mean_absolute_error(predictions, y_test,y_train):
    MAE = np.mean(np.abs(y_test-predictions))
    return MAE
```

In [392]:
```python
bs_output = bootstrap_validation(X_test,y_test,y_train,rf_cv,
                                 metrics_list=[OS_R_squared, mean_squared_error,mean_absol
                                 sample = 500)
```

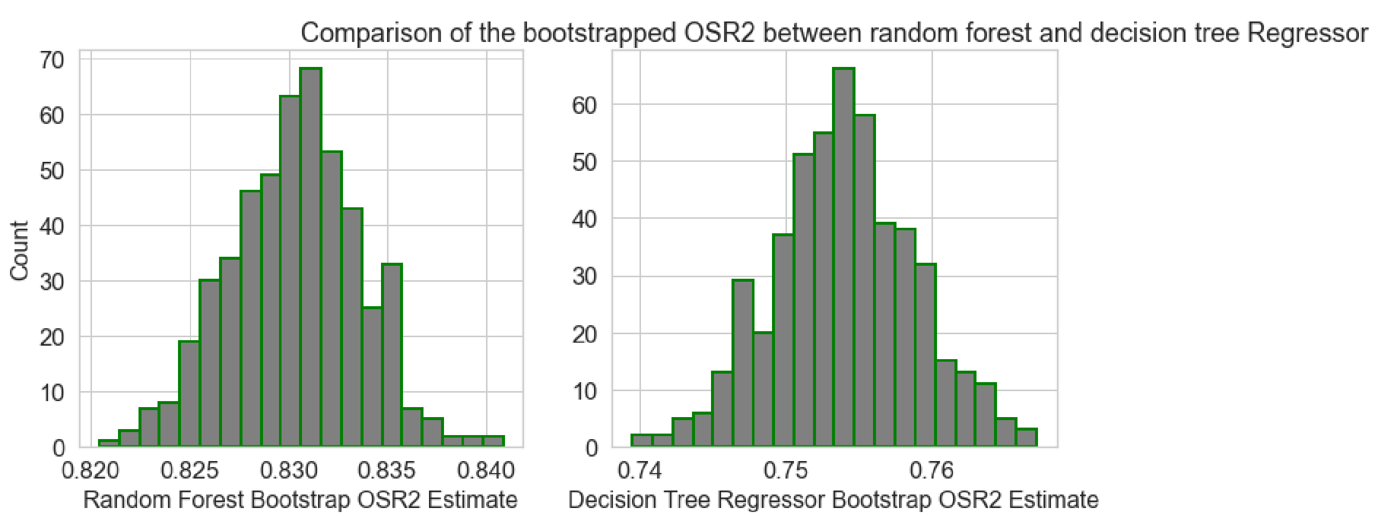
```
(500, 3)
```

In [394]:
```python
y_pred = rf_cv.predict(X_test)
```

In [419]:
```python
test_OSR2 = OS_R_squared(y_pred,y_test,y_train)

fig, axs = plt.subplots(ncols=2, figsize=(12,5))
axs[0].set_xlabel('Random Forest Bootstrap OSR2 Estimate', fontsize=16)
axs[1].set_xlabel('Decision Tree Regressor Bootstrap OSR2 Estimate', fontsize=16)
axs[0].set_ylabel('Count', fontsize=16)
axs[0].hist(bs_output.iloc[:,0], bins=20,edgecolor='green', linewidth=2,color = "grey")
#axs[0].set_xlim([0.7,0.85])
axs[1].hist(bs_output_dtr.iloc[:,0], bins=20,edgecolor='green', linewidth=2,color = "grey"
#axs[1].set_xlim([0.7,0.85])
plt.title('Comparison of the bootstrapped OSR2 between random forest and decision tree Regr
```

Out[419]:
```
Text(0.5, 1.0, 'Comparison of the bootstrapped OSR2 between random forest and decision tr
ee Regressor')
```



In [398]:
```
# The 95% confidence interval
CI=[0,1]
CI_0 = np.quantile(bs_output.iloc[:,0]-test_OSR2,np.array([0.025,0.975]))
CI[0] = test_OSR2 - CI_0[1]
CI[1] = test_OSR2 - CI_0[0]
print("The 95-percent confidence interval of OSR2 is %s" % CI) #0.5,0.64
```

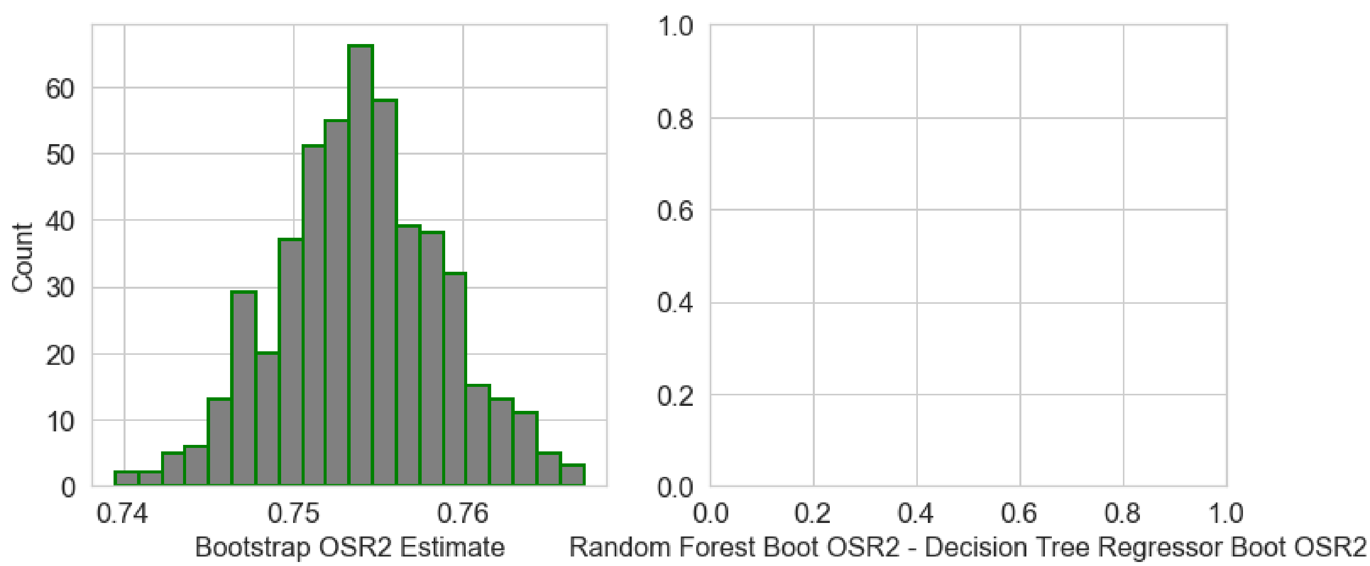
```
The 95-percent confidence interval of OSR2 is [0.8242669151163888, 0.836637059410585]
```

In [408]:
```
bs_output_dtr = bootstrap_validation(X_test,y_test,y_train,dtr_cv,
                                 metrics_list=[OS_R_squared, mean_squared_error,mean_absolu
                                 sample = 500)
```

```
(500, 3)
```

In [413]:
```
fig, axs = plt.subplots(ncols=2, figsize=(12,5))
axs[0].set_xlabel('Bootstrap OSR2 Estimate', fontsize=16)
axs[1].set_xlabel('Random Forest Boot OSR2 - Decision Tree Regressor Boot OSR2', fontsize=1
axs[0].set_ylabel('Count', fontsize=16)
axs[0].hist(bs_output_dtr.iloc[:,0], bins=20,edgecolor='green', linewidth=2,color = "grey")
#axs[0].set_xlim([0.4,0.7])
#axs[1].hist(bs_output.iloc[:,0]-bs_output_dtr.iloc[:0], bins=20,edgecolor='green', linewic
#axs[1].set_xlim([-0.15,0.15])
```

Out[413]:
```
(array([ 2.,  2.,  5.,  6., 13., 29., 20., 37., 51., 55., 66., 58., 39.,
        38., 32., 15., 13., 11.,  5.,  3.]),
 array([0.73951323, 0.7408955 , 0.74227776, 0.74366003, 0.74504229,
        0.74642456, 0.74780683, 0.74918909, 0.75057136, 0.75195362,
        0.75333589, 0.75471816, 0.75610042, 0.75748269, 0.75886495,
        0.76024722, 0.76162949, 0.76301175, 0.76439402, 0.76577628,
        0.76715855]),
 <BarContainer object of 20 artists>)
```



In [ ]: