Regression on Mobile Phone Dataset (Divar)

Mahsa Eskandari Ghadi
Student No. 810196597

In this project we tried Linear Regression, Decision Tree Regression and Random Forest Regression, on a cellphone dataset in order to predict prices. The main steps include: loading data, cleaning data, feature selection, check model performances and tuning the hyperparameters. (repeat last the last two until we get acceptable scores and error values.)

In [49]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

from sklearn.feature_selection import mutual_info_classif #For calculati

from sklearn.preprocessing import LabelEncoder #For numeralizing the cat

from sklearn.preprocessing import StandardScaler #For scaling data

from sklearn.model_selection import train_test_split #For spliting data

from collections import Counter

#Models
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

#Performance Measures
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

#Farsi language processing lib
#!pip install hazm
from hazm import *

import csv
```

The function below is used through out the code to plot and see the R2 scores vs the hyperparameter value in order to determine the best performance with the least overfitting.

In [50]:
```
def plot_param_vs_r2 (param, test_r2, train_r2, param_name) :

    acc_fig = plt.figure()
    ax = acc_fig.add_axes([0,0,1,1])
    ax.plot(param, train_r2, label="train", linewidth=1, color='#2E4053'
    ax.plot(param, test_r2, label='test', linewidth=1, color='#C0392B')
    ax.legend()
    ax.set_title('R2 vs. {}'.format(param_name))
    ax.set_xlabel(param_name)
    ax.set_ylabel('r2')
```

Part 0: Quick Glance at the Dataset

The data of our project is a record of Divar users who are looking to sell cellphones with helpful details of the brand of the cellphone, the city the seller resides in, the name and model of the phone, a short description, the number of images they included in the ad, the day of the week and hour they posted the ad and lastly the price that they put on their phones. Our goal is to construct a model that will predict the price for mobile phones with a reasonable R2score and mean absolute error.

In [51]:
```
df = pd.read_csv("mobile_phone_dataset.csv")
df.head()
```

Out[51]:

	Unnamed: 0	brand	city	title	desc	image_count	created_at	price
0	0	Nokia::نوکیا	Qom	نوکیا6303	سلام.یه گوشیه6303سالم که فقط دوتا خط کوچیک رو ...	2	Wednesday 07AM	60000
1	1	Apple::اپل	Tehran	ایفون ۵اس۳۲گیگ	درحد نو سالم اصلی بدون ضربه مهلت تست میدم	0	Wednesday 11AM	1150000
2	2	Samsung::سامسونگ	Mashhad	سامسونگ j5	گوشی بسیار بسیار تمیز و فقط سه هفته کارکرده و ...	2	Wednesday 02PM	590000
3	3	Apple::اپل	Karaj	ایفون 5 گری 32گیگ	گلس پشت و رو .کارت اپل ای دی. لوازم جانبی اصلی...	3	Wednesday 04PM	1100000
4	4	Samsung::سامسونگ	Tehran	galaxy S5 Gold در حد آک	کاملا تمیز و بدون حتی 1 به\nخط و خش همراه گلاس...	2	Friday 01PM	900000

Part 1: Preprocessing and Cleaning Data

For cleaning the data I took these steps:

- Change the created at format (seperate day from time)
- Change the brand format (delete the Farsi part)
- Delete the "exchange" rows with the -1 price.
- Define features and target (X, y)
- Drop some features (Unnamed: 0, desc)
- Label Encode and OneHot Encode the categorical features
- Plot Information Gain
- Feature select
- Extract features from the title

After each step the head of the dataframe is shown to check the change has really been made.

In [52]:
```
df[['day','time']] = df['created_at'].str.split(' ', 1, expand=True)
df = df.drop(columns=['created_at'])
df.head()
```

Out[52]:

	Unnamed: 0	brand	city	title	desc	image_count	price	day	time
0	0	Nokia::نوکیا	Qom	نوکیا6303	سلام.یه گوشیه6303سالم که فقط دوتا خط کوچیک رو ...	2	60000	Wednesday	07AM
1	1	Apple::اپل	Tehran	ایفون ۵اس۳۲گیگ	درحد نو سالم اصلی بدون ضربه مهلت تست میدم	0	1150000	Wednesday	11AM
2	2	Samsung::سامسونگ	Mashhad	سامسونگ j5	گوشی بسیار بسیار تمیز و فقط سه هفته کارکرده و ...	2	590000	Wednesday	02PM
3	3	Apple::اپل	Karaj	ایفون 5 گری sگیگ32	گلس پشت و رو .کارت اپل ای دی. لوازم جانبی اصلی...	3	1100000	Wednesday	04PM
4	4	Samsung::سامسونگ	Tehran	galaxy S5 Gold در حد آک	کاملا تمیز و بدون حتی 1 خط و به\nخش همراه گلاس...	2	900000	Friday	01PM

In [53]:
```
df['brand'] = df['brand'].str.split('::', 1, expand=True)
df.head()
```

Out[53]:

	Unnamed: 0	brand	city	title	desc	image_count	price	day	time
0	0	Nokia	Qom	نوکیا6303	سلام.یه گوشیه6303سالم که فقط دوتا خط کوچیک رو ...	2	60000	Wednesday	07AM
1	1	Apple	Tehran	ایفون ۵اس۳۲گیگ	درحد نو سالم اصلی بدون ضربه مهلت تست میدم	0	1150000	Wednesday	11AM
2	2	Samsung	Mashhad	سامسونگ j5	گوشی بسیار بسیار تمیز و فقط سه هفته کارکرده و ...	2	590000	Wednesday	02PM
3	3	Apple	Karaj	sگری 5 ایفون 32گیگ	گلس پشت و رو .کارت اپل ای دی. لوازم جانبی اصلی...	3	1100000	Wednesday	04PM
4	4	Samsung	Tehran	galaxy S5 Gold در حد آک	کاملا تمیز و بدون حتی 1 خط و به\nخش همراه گلاس...	2	900000	Friday	01PM

In [54]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59189 entries, 0 to 59188
Data columns (total 9 columns):
Unnamed: 0     59189 non-null int64
brand          59189 non-null object
city           59189 non-null object
title          59189 non-null object
desc           59189 non-null object
image_count    59189 non-null int64
price          59189 non-null int64
day            59189 non-null object
time           59189 non-null object
dtypes: int64(3), object(6)
memory usage: 4.1+ MB
```

In [55]:
```
1 # split dataset in features and target variable
2 X = df.drop(columns=['price'])  # Features
3 y = df['price']  # Target variable
```

In [56]:
```
1 y.head()
```

Out[56]:
```
0      60000
1    1150000
2     590000
3    1100000
4     900000
Name: price, dtype: int64
```

In [57]:
```
1 X = X.drop(columns=['Unnamed: 0', 'desc'])
2 X.head()
```

Out[57]:

	brand	city	title	image_count	day	time
0	Nokia	Qom	نوکیا6303	2	Wednesday	07AM
1	Apple	Tehran	ایفون ۵اس۳۲گیگ	0	Wednesday	11AM
2	Samsung	Mashhad	سامسونگ j5	2	Wednesday	02PM
3	Apple	Karaj	ایفون 32گیگ s گری 5	3	Wednesday	04PM
4	Samsung	Tehran	galaxy S5 Gold در حد آک	2	Friday	01PM

In [58]:
```
X_encoded = pd.get_dummies(X, columns=['brand', 'city'], drop_first=True
X_encoded.head()
```

Out[58]:

	title	image_count	day	time	brand_HTC	brand_Huawei	brand_LG	brand_Lenc
0	نوکیا6303	2	Wednesday	07AM	0	0	0	
1	ایفون ۵اس۳۲گیگ	0	Wednesday	11AM	0	0	0	
2	سامسونگ j5	2	Wednesday	02PM	0	0	0	
3	5 گری sایفون 32گیگ	3	Wednesday	04PM	0	0	0	
4	galaxy S5 در Gold حد آک	2	Friday	01PM	0	0	0	

In [59]:
```
le = LabelEncoder()
X_encoded['day'] = le.fit_transform(X_encoded['day'])
times = X_encoded['time']
hours = []
for i in range(len(times)):
    time_str = str(times[i])
    if (time_str.find('AM') != -1):
        time_str = time_str.replace('AM', '')
        time = int(time_str)
    else:
        time_str = time_str.replace('PM', '')
        time = int(time_str) + 12
    hours.append(time)
X_encoded['time'] = hours
X_encoded.head()
```

Out[59]:

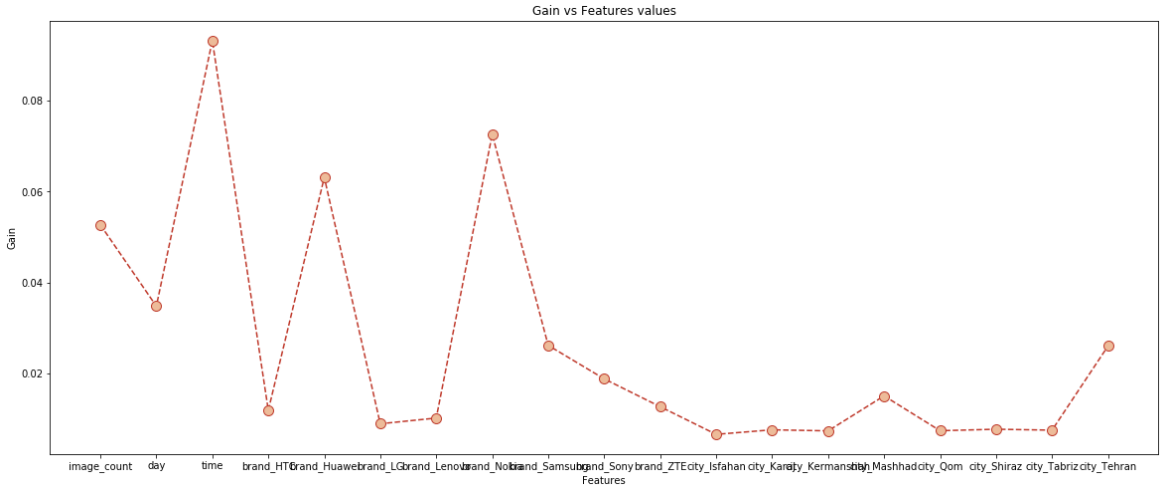
	title	image_count	day	time	brand_HTC	brand_Huawei	brand_LG	brand_Lenovo	bra
0	نوکیا6303	2	6	7	0	0	0	0	
1	ایفون ۵اس۳۲گیگ	0	6	11	0	0	0	0	
2	سامسونگ j5	2	6	14	0	0	0	0	
3	5 گری sایفون 32گیگ	3	6	16	0	0	0	0	
4	galaxy S5 Gold در حد آک	2	0	13	0	0	0	0	

Part 2: Feature Selection

In [60]:
```
X_dropped = X_encoded.drop(columns=['title'])
```

In [61]:
```
mic = mutual_info_classif(X_dropped, y, discrete_features=True)
fig = plt.figure(figsize=(20,8))
plt.plot(X_dropped.columns, mic, color='#C0392B', linestyle='dashed', ma
plt.title('Gain vs Features values')
plt.xlabel('Features')
plt.ylabel('Gain')
```

Out[61]: Text(0, 0.5, 'Gain')



As it can be seen the time column has the most gain which doesn't really make sense because obviously the time of the day a person posts the ad should't really be affecting the price of the product. This can lead the model to make false predictions. It is only logical to drop these columns.

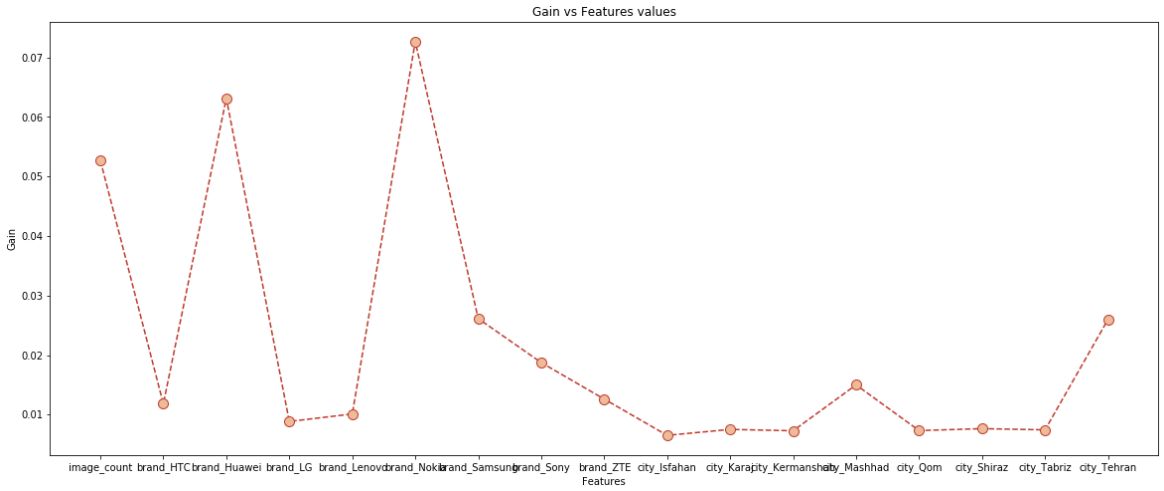
In [62]:
```
X_encoded = X_encoded.drop(columns=['time', 'day'])
X_encoded.head()
```

Out[62]:

	title	image_count	brand_HTC	brand_Huawei	brand_LG	brand_Lenovo	brand_Nokia	b
0	نوکیا6303	2	0	0	0	0	1	
1	ایفون ۵اس۳۲گیگ	0	0	0	0	0	0	
2	سامسونگ j5	2	0	0	0	0	0	
3	5 گری sایفون 32گیگ	3	0	0	0	0	0	
4	galaxy S5 Gold در حد آک	2	0	0	0	0	0	

In [63]:
```
X_dropped = X_encoded.drop(columns=['title'])
mic = mutual_info_classif(X_dropped, y, discrete_features=True)
fig = plt.figure(figsize=(20,8))
plt.plot(X_dropped.columns, mic, color='#C0392B', linestyle='dashed', ma
plt.title('Gain vs Features values')
plt.xlabel('Features')
plt.ylabel('Gain')
```

Out[63]: Text(0, 0.5, 'Gain')



To extract features from the title these steps were taken:

- Normalize each title (remove un necessary spaces and such)
- Tokenize each title
- Get the count of each unique word
- Feature Select (remove some of the words)

In [64]:
```
normalizer = Normalizer()
titles = X_encoded['title']
titles_normalized = []
```

In [65]:
```
for title in titles:
    titles_normalized.append(normalizer.normalize(title))
```

In [66]:
```
X_encoded['title'] = titles_normalized
X_encoded.head()
```

Out[66]:

	title	image_count	brand_HTC	brand_Huawei	brand_LG	brand_Lenovo	brand_Nokia	b
0	نوکیا۶۳۰۳	2	0	0	0	0	1	
1	ایفون ۵اس۳۲گیگ	0	0	0	0	0	0	
2	سامسونگ j۵	2	0	0	0	0	0	
3	sگری ۵ ایفون ۳۲گیگ	3	0	0	0	0	0	
4	galaxy S۵ در Gold حد آک	2	0	0	0	0	0	

In [67]:
```
titles = X_encoded['title']
titles_tokenized = []
```

In [68]:
```
for title in titles:
    titles_tokenized.append(word_tokenize(title))
```

In [69]:
```
for i in range(len(titles_tokenized)):
    titles_tokenized[i] = [x.lower() for x in titles_tokenized[i]]
```

In [70]:
```
titles_flattened = [j for sub in titles_tokenized for j in sub]
```

In [71]:
```
titles_df = pd.DataFrame(titles_flattened)
word_series = titles_df[0].value_counts()
word_dict = word_series.to_dict()

word_dict
```

Out[71]:
```
{'18193 :'گوشی,
 '8644 :'سامسونگ,
 '5059 :'موبایل,
 'g': 3577,
 'iphone': 3308,
 '۱۶': 3085,
 '3002 :'هواوی,
 '2948 :'ایفون,
 '2843 :'گیگ,
 '۶۴': 2837,
 '2716 :'نوکیا,
 '۵s': 2697,
 '2611 :'جی,
 '2484 :'سونی,
 '2441 :'اپل,
 '۶': 2345,
 '۳۲': 2343,
 '2313 :'آیفون,
 '2238 :'گلکسی,
```

WORDS TO REMOVE AS FEATURES

The most word used is **"گوشی"** meaning "phone" which there is no gain in keeping it since we know our whole dataset is phones and phones only. Next we should remove the **brands** since again we already have the brands in a whole column and keeping them is just repeating features. Also **stop-words** like **"در", "با", "و" and "یا"** were removed.

In [72]:
```
words_to_delete = ["آیفون", "گیگ" ,"هواوی" ,"سامسونگ" ,"موبایل" ,"گوشی","samsun
for word in words_to_delete:
    del word_dict[word]
```

In [73]:
```
1 word_dict
```

```
'271 :'خط,
'267 :'فور,
'۷۰۰': 267,
'265 :'تعویض,
'۸۱۶': 265,
'g۳': 262,
'c': 262,
'v': 256,
'۳c': 255,
'۳۰۰': 254,
'253 :'وای,
'248 :'خش,
'248 :'ایکس,
'247 :'سیکس,
'۴c': 245,
'245 :'ساده,
'mini': 244,
'p۸': 243,
'iphon': 243,
'i۱': 240.
```

In [74]:
```
1 len(word_dict)
```

Out[74]: 12338

The less likely a word is used the more information it contains thus the more special it is. but also the looking at the least used words we realize that they are usually spelling errors. So we'll take the words that have been used at least 10 times and throw the rest away, leaving us 1103 words out of 11350. Note that, we also don't want too many features because that will lead the model to overfit. 1103 to 53301 of the whole data is reasonable in this situation.

In [75]:
```
1 word_dict = dict((key, value) for key, value in word_dict.items() if val
```

In [76]:
```
1 len(word_dict)
```

Out[76]: 1178

Now we want to give values to those features by checking a box of yes or no. If the tile contains that word we check yes(1) else no(0). Then we concat these new columns to the dataframe for X.

In [77]:
```
1 new_columns = np.zeros((len(word_dict), len(titles_tokenized)))
```

In [78]:
```
1 for i in range(len(titles_tokenized)):
2     for j, word in enumerate(word_dict):
3         if word in titles_tokenized[i]:
4             new_columns[j][i] = 1
```

In [79]:
```
cols = list(range(len(word_dict)))
cols_str = [str(x) for x in cols]
rows = list(range(len(titles_tokenized)))
rows_str = [str(x) for x in rows]
new_cols = pd.DataFrame(new_columns)
new_cols = new_cols.T
```

In [80]:
```
X_final = pd.concat([X_encoded, new_cols], axis=1)
```

In [81]:
```
X_final = X_final.drop(columns=['title'])
X_final.head()
```

Out[81]:

	image_count	brand_HTC	brand_Huawei	brand_LG	brand_Lenovo	brand_Nokia	brand_Sa
0	2	0	0	0	0	1	
1	0	0	0	0	0	0	
2	2	0	0	0	0	0	
3	3	0	0	0	0	0	
4	2	0	0	0	0	0	

5 rows × 1195 columns

In [82]:
```
indices = df[df['price'] == -1 ].index

X_train = X_final.drop(indices)
X_train = X_train.reset_index(drop=True)

y_train = df.drop(indices)['price']

X_test = X_final.iloc[indices]
X_test = X_test.reset_index(drop=True)

y_test = df[df['price'] == -1 ]['price']
```

In [83]:
```
print(len(X_train),len(X_train),len(X_test),len(y_test))
```

```
53301 53301 5888 5888
```

Part 3: Check Model Performances

In [84]:
```
lr = LinearRegression(normalize=True)
lr.fit(X_train, y_train)

y_pred_test = lr.predict(X_test)
y_pred_train = lr.predict(X_train)

print('Train R2: ', r2_score(y_train, y_pred_train))
print('Train MAE: ', mean_absolute_error(y_train, y_pred_train))
print()

out_csv = [['price']]
for r in enumerate(y_pred_test):
    out_csv.append([r])

with open('output.csv', 'w') as csvFile:
    writer = csv.writer(csvFile)
    writer.writerows(out_csv)
csvFile.close()
```

```
Train R2:  0.7439069383377497
Train MAE:  178400.64198412755
```

R-squared(R2) is a goodness-of-fit measure for linear regression models. This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively.

$$R^2 = \frac{Variance\ explained\ by\ the\ model}{Total\ variance}$$

Mean Absolute Error (MAE): MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

$$MAE = \frac{1}{n}\sum_{i=1}^{i=n}|y_i - \hat{y_i}|$$

In [45]:
```
X_train, X_test, y_train, y_test = train_test_split(X_final, y, test_siz
```

In [81]:
```
dtr = DecisionTreeRegressor(random_state=101)
dtr.fit(X_train, y_train)

y_pred_test = dtr.predict(X_test)
y_pred_train = dtr.predict(X_train)

print('Train R2: ', r2_score(y_train, y_pred_train))
print('Test MAE: ', mean_absolute_error(y_train, y_pred_train))

```

```
Test R2:  0.6244406594974545
Test MSE:  112554357438.30193

Train R2:  0.9407708823689316
Test MSE:  18003696496.75302
```

The dtr model is clearly overfitting by 31%. We will try to fix that by tuning the hyperparameters

Part 4: Hyperparameter Tuning

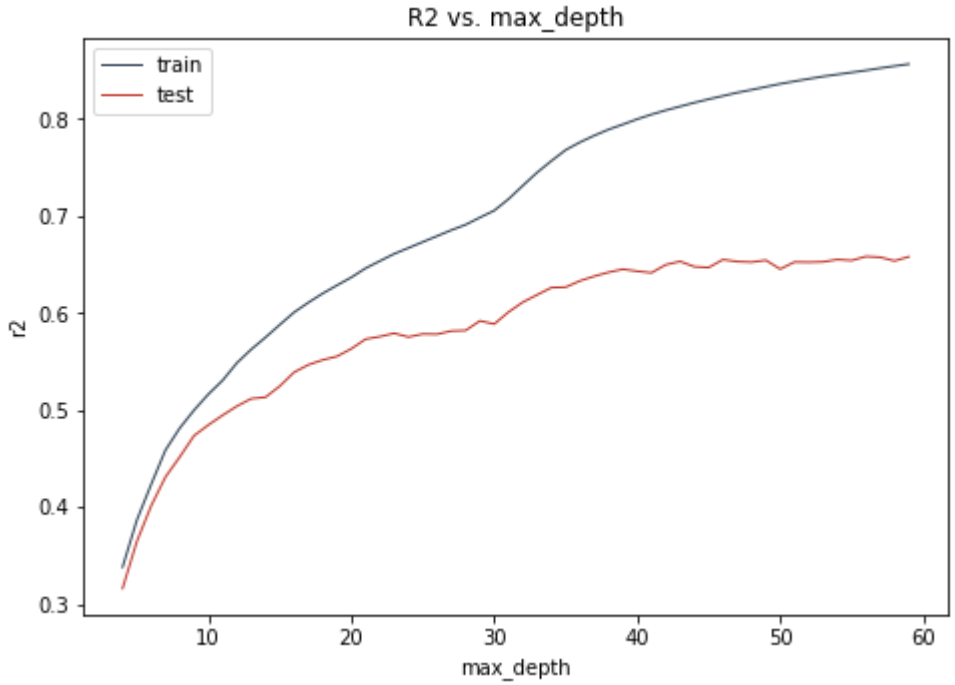
In [99]:
```
depths = list(range(4,60))
test_r2s = []
train_r2s = []
```

In [100]:
```
for depth in depths:
    dtr = DecisionTreeRegressor(random_state=101, max_depth=depth)
    dtr.fit(X_train, y_train)

    y_pred_test = dtr.predict(X_test)
    y_pred_train = dtr.predict(X_train)

    test_r2s.append(r2_score(y_test, y_pred_test))
    train_r2s.append(r2_score(y_train, y_pred_train))
```

In [101]:
```
1 plot_param_vs_r2(depths, test_r2s, train_r2s, 'max_depth')
```



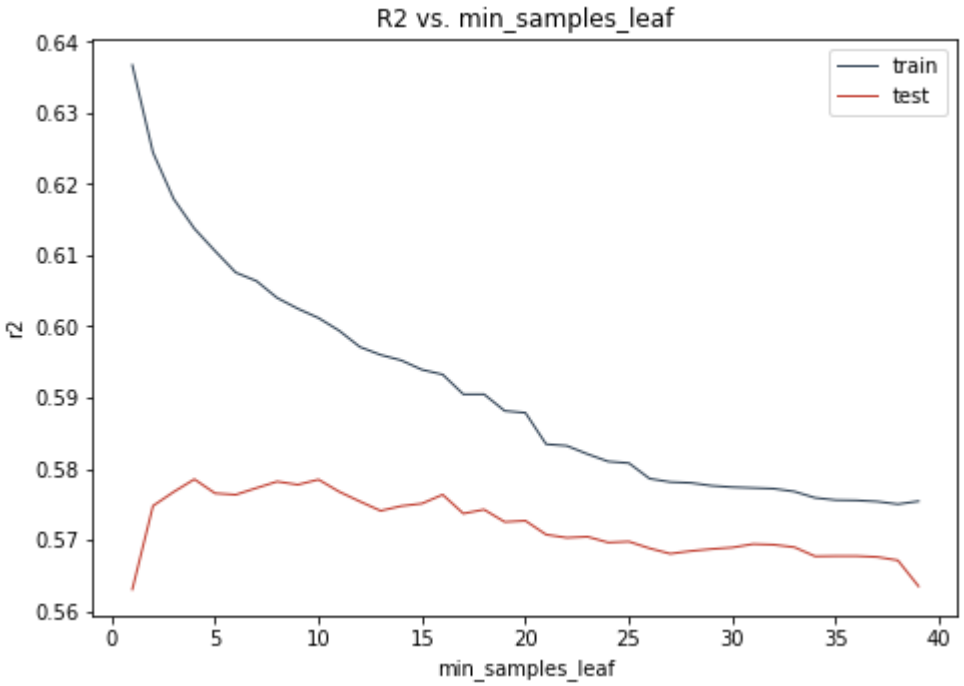
In [102]:
```
1 max_depth = 20
2 min_samples_leaves = list(range(1,40))
3 test_r2s = []
4 train_r2s = []
```

In [103]:
```
for min_samples in min_samples_leaves:
    dtr = DecisionTreeRegressor(random_state=101, max_depth=max_depth, m
    dtr.fit(X_train, y_train)

    y_pred_test = dtr.predict(X_test)
    y_pred_train = dtr.predict(X_train)

    test_r2s.append(r2_score(y_test, y_pred_test))
    train_r2s.append(r2_score(y_train, y_pred_train))
```

In [104]:
```
plot_param_vs_r2(min_samples_leaves, test_r2s, train_r2s, 'min_samples_l
```



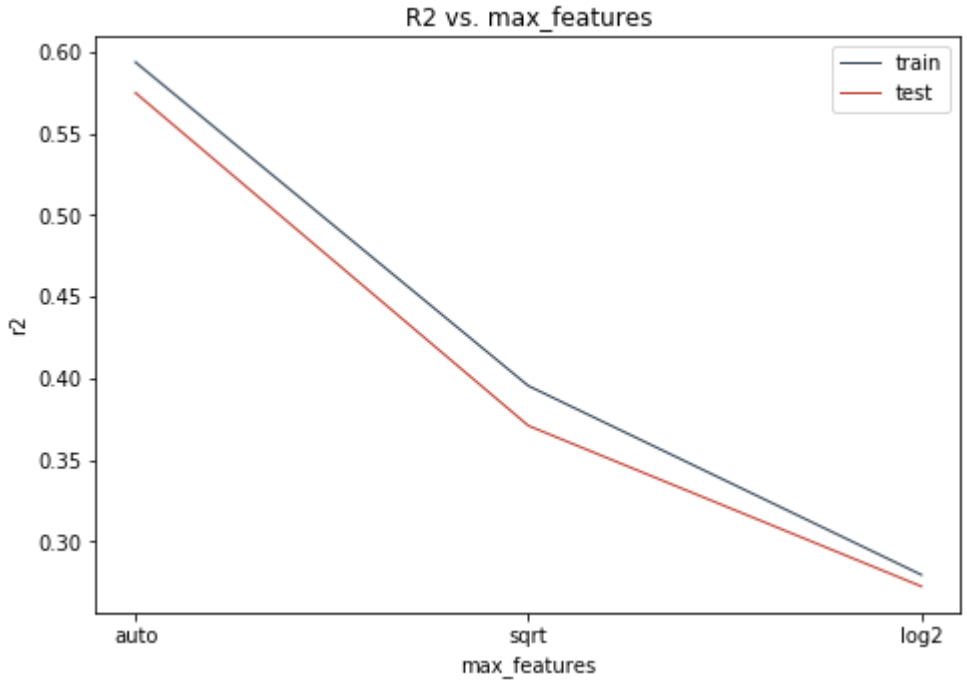
In [105]:
```
min_samples_leaf = 15
max_features = ['auto', 'sqrt', 'log2']
test_r2s = []
train_r2s = []
```

In [106]:
```
for max_feature in max_features:
    dtr = DecisionTreeRegressor(random_state=101, max_depth=max_depth, m
    dtr.fit(X_train, y_train)

    y_pred_test = dtr.predict(X_test)
    y_pred_train = dtr.predict(X_train)

    test_r2s.append(r2_score(y_test, y_pred_test))
    train_r2s.append(r2_score(y_train, y_pred_train))
```

In [107]:
```
1 plot_param_vs_r2(max_features, test_r2s, train_r2s, 'max_features')
```



In [108]:
```
1  dtr = DecisionTreeRegressor(random_state=101, max_depth=20, min_samples_
2  dtr.fit(X_train, y_train)
3
4  y_pred_test = dtr.predict(X_test)
5  y_pred_train = dtr.predict(X_train)
6
7  print('Test R2: ', r2_score(y_test, y_pred_test))
8  print('Test MSE: ', mean_squared_error(y_test, y_pred_test))
9  print()
10 print('Train R2: ', r2_score(y_train, y_pred_train))
11 print('Test MSE: ', mean_squared_error(y_train, y_pred_train))
```

```
Test R2:  0.5751356234409368
Test MSE:  127330974748.33266

Train R2:  0.5939229023521089
Test MSE:  123434032326.35826
```

In [112]:
```
rfr = RandomForestRegressor(n_estimators=10, max_depth=20, min_samples_l
rfr.fit(X_train, y_train)

y_pred_test = rfr.predict(X_test)
y_pred_train = rfr.predict(X_train)

print('Test R2: ', r2_score(y_test, y_pred_test))
print('Test MSE: ', mean_squared_error(y_test, y_pred_test))
print()
print('Train R2: ', r2_score(y_train, y_pred_train))
print('Test MSE: ', mean_squared_error(y_train, y_pred_train))

```

```
Test R2:  0.5847959909565348
Test MSE:  124435782588.0711

Train R2:  0.5970606762639716
Test MSE:  122480252640.89195
```

Part 5: Conclusion

Linear regression works really nicely when the data has a linear shape. But, when the data has a non-linear shape, then a linear model cannot capture the non-linear features. In that case Decision Tree Regressors do a better job at capturing the non-linearity in the data by dividing the space into smaller sub-spaces depending on the questions asked. In the end we have Random Forrest Regressors. A random forest is a meta estimator that fits a bunch of decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. So if a decision tree is not working well a random forrest isn't likely to be of much help since it's averaging the decision trees. But it did reduce overfitting by 1% and also the MSE for both. From the above explanations and all the performances we saw from the each regressor, It is likely ok to assume that our data had more of a linear shape. Also for continiuous data like prices linear regressors usually work better.

References

- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html)
- https://scikit-learn.org/0.15/modules/generated/sklearn.ensemble.RandomForestRegressor.html (https://scikit-learn.org/0.15/modules/generated/sklearn.ensemble.RandomForestRegressor.html)