Supervised Machine Learning: Regression Project

House Sales In King County, USA Dataset

Jean Paul Tuyikunde

May 28, 2021

Objectives

- Cleaning Data
- Performing Exploratory analysis on data
- Develop prediction models
- Evaluation of developed models

I. Data Description

This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015. Get the data here

id : A notation for a house

date: Date house was sold

price: Price is prediction target

bedrooms: Number of bedrooms

bathrooms: Number of bathrooms

sqft_living: Square footage of the home

sqft_lot: Square footage of the lot

floors :Total floors (levels) in house

waterfront :House which has a view to a waterfront

view: Has been viewed

condition :How good the condition is overall

grade: overall grade given to the housing unit, based on King County grading system

sqft_above : Square footage of house apart from basement

sqft_basement: Square footage of the basement

yr_built : Built Year

yr_renovated : Year when house was renovated

zipcode: Zip code

lat: Latitude coordinate

long: Longitude coordinate

sqft_living15 : Living room area in 2015(implies-- some renovations) This might or might not have affected the lotsize area

sqft_lot15 : LotSize area in 2015(implies-- some renovations)

II. Data Cleaning

Importing required libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style = "ticks")
```

read the data set in pandas DataFrame

In [2]:
```
import os

path = ['data']
filename = 'kc_house_data.csv'
filepath = os.sep.join(path + [filename])

data = pd.read_csv(filepath, sep=',')
data.head()
```

Out[2]:

	Unnamed: 0	Unnamed: 0.1	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	...	grade	sqft_abo
0	0	0	7129300520	20141013T000000	221900.0	3.0	1.00	1180	5650	1.0	...	7	1
1	1	1	6414100192	20141209T000000	538000.0	3.0	2.25	2570	7242	2.0	...	7	2
2	2	2	5631500400	20150225T000000	180000.0	2.0	1.00	770	10000	1.0	...	6	7
3	3	3	2487200875	20141209T000000	604000.0	4.0	3.00	1960	5000	1.0	...	7	10
4	4	4	1954400510	20150218T000000	510000.0	3.0	2.00	1680	8080	1.0	...	8	16

5 rows × 23 columns

In [3]:
```
data.shape
```

Out[3]: (21613, 23)

Dropping the "id","Unnamed: cols" and checking types of every columns

In [4]:
```
data.drop(['id','Unnamed: 0','Unnamed: 0.1'], axis=1, inplace = True)
data.dtypes.to_frame()
```

Out[4]:

	0
date	object
price	float64
bedrooms	float64
bathrooms	float64
sqft_living	int64
sqft_lot	int64
floors	float64
waterfront	int64
view	int64
condition	int64
grade	int64
sqft_above	int64
sqft_basement	int64
yr_built	int64
yr_renovated	int64
zipcode	int64
lat	float64
long	float64
sqft_living15	int64
sqft_lot15	int64

In [5]:
```
data.describe()
```

Out[5]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000

Let's Check the Missing values in our data

In [6]:
```
for col in data.columns:
    if data[col].isnull().sum() >0:
        print("The missing value(s) in {} is = {}".format(col, data[col].isnull().sum()))
    else:
        print("No missing values in data")
```

```
No missing values in data
No missing values in data
The missing value(s) in bedrooms is = 13
The missing value(s) in bathrooms is = 10
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
No missing values in data
```

Replacing the missing values of `'bedrooms'&'bathrooms'` columns with the mean values using `replace()` method.

In [7]:
```
mean1 = data['bedrooms'].mean()
data['bedrooms'].replace(np.nan, mean1, inplace = True)
mean2 = data['bathrooms'].mean()
data['bathrooms'].replace(np.nan, mean2, inplace = True)
```

In [8]:
```
for col in data.columns:
    if data[col].isnull().sum() >0:
        print("The missing value(s) in {} is = {}".format(col, data[col].isnull().sum()))
else:
    print("No missing values in data")
```

```
No missing values in data
```

III. Exploration Data Analysis

Find correlation between some features

In [9]:
```
features = ['sqft_living','sqft_basement','sqft_lot','price']
data[features].corr()
```

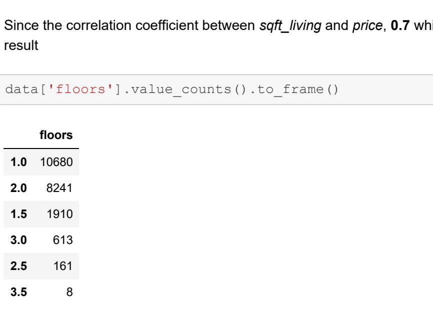
Out[9]:

	sqft_living	sqft_basement	sqft_lot	price
sqft_living	1.000000	0.435043	0.172826	0.702035
sqft_basement	0.435043	1.000000	0.015286	0.323816
sqft_lot	0.172826	0.015286	1.000000	0.089661
price	0.702035	0.323816	0.089661	1.000000

according to the above table we can use the `regplot()` to see the either if there is a positive or negative correlation between features and `*price*`

In [10]:
```
sns.regplot(x = 'sqft_living', y = 'price', data = data)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x24f88cebfc8>



According to the graph there is more house with Square footage of the home `'sqft_living'` that ranges from 0 to 8000, and according to the numbers above there is a quite a significant positive relationship, but we can verify again with `Pearson correlation`

In [11]:
```
from scipy import stats
pear_coef, p_value = stats.pearsonr(data['sqft_living'], data['price'])
print("The correlation coefficient = {}\nP-value = {}".format(pear_coef, p_value))
```

```
The correlation coefficient = 0.7020350546118002
P-value = 0.0
```

Since the correlation coefficient between *sqft_living* and *price*, **0.7** which is close to 1 but the P-value shows there's a weak **certainty** in the result

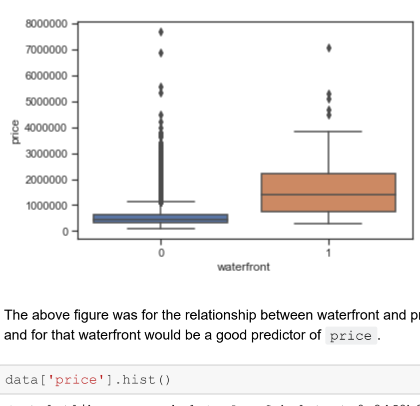
In [12]:
```
data['floors'].value_counts().to_frame()
```

Out[12]:

	floors
1.0	10680
2.0	8241
1.5	1910
3.0	613
2.5	161
3.5	8

In [13]:
```
sns.countplot(x=data['floors'])
```

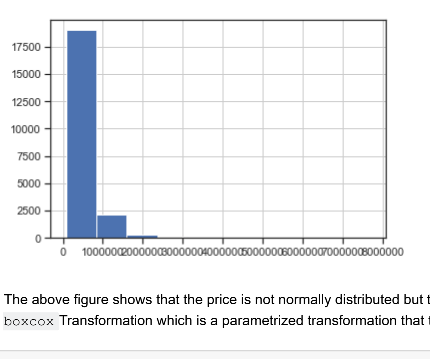
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x24f8a5bb248>



We can see that there is more houses with 1 floor in our data set

In [14]:
```
sns.boxplot(x = 'waterfront', y = 'price', data = data)
```

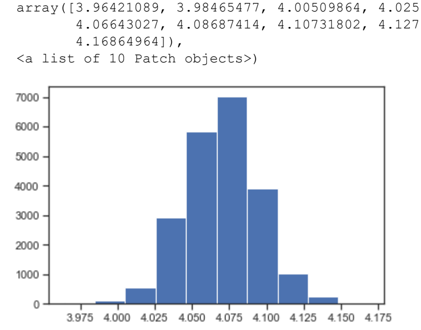
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x24f8b20b648>



The above figure was for the relationship between waterfront and price, and the distributions of price to different waterfront do not *overlap* and for that waterfront would be a good predictor of `price`.

In [15]:
```
data['price'].hist()
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x24f8b25ea88>

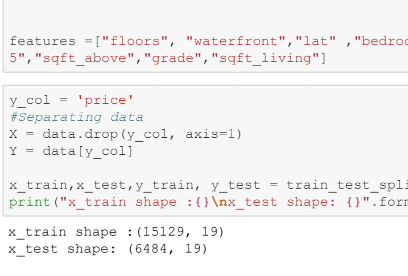


The above figure shows that the price is not normally distributed but the next cell show how you can transform the *Price* with `boxcox` Transformation which is a parametrized transformation that tries to get distributions "as close to a normal distribution as possible".

In [16]:
```
from scipy.stats import boxcox

bc_results = boxcox(data.price)[0]
plt.hist(bc_results)
```

Out[16]:
```
(array([  31.,   95.,  540., 2908., 5831., 7012., 3918., 1017.,  240.,
          21.]),
 array([3.96421089, 3.98465477, 4.00509864, 4.02554252, 4.04598639,
        4.06643027, 4.08687414, 4.10731802, 4.12776189, 4.14820576,
        4.16864964]),
 <a list of 10 Patch objects>)
```



IV. Model Development

For our models we will be using the following list of features and libraries

In [17]:
```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import (StandardScaler,
                                   PolynomialFeatures)


features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","sqft_living1
5","sqft_above","grade","sqft_living"]
```

In [18]:
```
y_col = 'price'
#Separating data
X = data.drop(y_col, axis=1)
Y = data[y_col]

x_train,x_test,y_train, y_test = train_test_split(X, Y, test_size = 0.30, random_state = 42)
print("x_train shape :{}\nx_test shape: {}".format(x_train.shape, x_test.shape))
```

```
x_train shape :(15129, 19)
x_test shape: (6484, 19)
```

All model will be fitted through Pipeline and calculate the **R^2** score and **MSE** (mean squared error).

Linear Regression model

In [19]:
```
lr = LinearRegression()
st = StandardScaler()
pr = PolynomialFeatures(degree= 2, include_bias=True)

Input = [('scale',st),('polynomial',pr), ('model',lr)]
pipe = Pipeline(Input)
pipe.fit(x_train[features], y_train)
lr_prediction = pipe.predict(x_test[features])
```

In [20]:
```
#Taking the R^2 score
print("The Linear Regression R^2 score=",pipe.score(x_test[features], y_test))
print("The MSE for Linear Regression =",mean_squared_error(y_test, lr_prediction))
```

```
The Linear Regression R^2 score= 0.7114426981295834
The MSE for Linear Regression = 41658016837.75108
```

Ridge Regression model

In [21]:
```
#with Ridge regression and rest will be the same
RR = Ridge(alpha = 0.1)
Input1 = [('scale',st),('polynomial',pr), ('model',RR)]

pipe1 = Pipeline(Input1)
pipe1.fit(x_train[features], y_train)
Rg_prediction = pipe1.predict(x_test[features])
```

In [22]:
```
#R^2 score
print("Ridge R^2 score=", pipe1.score(x_test[features], y_test))
print("The MSE for Ridge Regression =",mean_squared_error(y_test, Rg_prediction))
```

```
Ridge R^2 score= 0.7117768928182427
The MSE for Ridge Regression = 41609770309.67151
```

Lasso Regression model

In [23]:
```
#with Ridge regression and rest will be the same
Lass = Lasso(alpha = 0.1)
Input2 = [('scale',st),('polynomial',pr), ('model',Lass)]

pipe2 = Pipeline(Input2)
pipe2.fit(x_train[features], y_train)
Lass_prediction = pipe2.predict(x_test[features])
#This takes a little while
```

```
C:\Users\User\Anaconda3\envs\tensoflow\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:
476: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterati
ons. Duality gap: 74139943876732.03, tolerance: 197654197140.59027
  positive)
```

In [24]:
```
#R^2 score
print("Lasso R^2 score=", pipe2.score(x_test[features], y_test))
print("The MSE for Lasso Regression =",mean_squared_error(y_test, Lass_prediction))
```

```
Lasso R^2 score= 0.7117767757461126
The MSE for Lasso Regression = 41609787210.96887
```

Insights

What would be the best model? The best model should have the low `MSE` and High `R^2 score`

Let's take a look at the values for the different models.

Linear Regression: Using features highlighted as a Predictor Variable of Price.

- R-squared: 0.7114426981295834
- MSE: 41658016837.75108

Ridge Regression: Using features as Predictor Variables of Price.

- R-squared: 0.7117768928182427
- MSE: 41609770309.67151

Lasso Regression: Using features as Predictor Variable of Price.

- R-squared: 0.7117767757461126
- MSE: 41609787210.96887

- R^2 of Ridge is greater than the R^2 of Linear model
- R^2 of Ridge is greater than the R^2 of Lasso
- R^2 of lasso is greater than the R^2 of Linear
- Mse_ridge is less to linear
- Mse_ridge is less to lasso

We can keep refining our models by using *GridSearch* to find the **best hyperparameters**

In [25]:
```
X_train1 = x_train[features]
X_test1 = x_test[features]

RR2 = Ridge() # constructor of ridge
parameters= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000], 'normalize':[True, False]}]
Grid = GridSearchCV(RR2, parameters, cv = 4) #using 4 folds
Grid.fit(X_train1, y_train)
Grid.best_estimator_
```

Out[25]:
```
Ridge(alpha=0.001, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=True, random_state=None, solver='auto', tol=0.001)
```

Let's use the results of the above model, and change the alpha from 0.1 to 0.001

In [26]:
```
# Ridge regression with best estimator
RR = Ridge(alpha = 0.001, normalize = True)
Input3 = [('scale',st),('polynomial',pr), ('model',RR)]

pipe = Pipeline(Input3)
pipe.fit(X_train1, y_train)
ridge_pred = pipe.predict(X_test1)
#R^2 score and MSE
print("Ridge R^2 score=", pipe.score(X_test1, y_test))
print("The MSE for Ridge Regression =",mean_squared_error(y_test, ridge_pred))
```

```
Ridge R^2 score= 0.7117982405688242
The MSE for Ridge Regression = 41606688408.962685
```

By using the Grid search results which is `GridSearch.best_estimator_` to our second Ridge model the `R^2` score was *increased* and The **MSE** was **decreased**.

V. Conclusions

- The data quality is good since there was less amount of missing data
- The best model to be employed could be Ridge and Lasso
- Further addition of features can be made
- Data analysis can be improved by exploring different `approaches`, `models` and `hyperparameters`