1 Project 2 : Regression Analysis for House Price

1.1 Overview

The Project is to provide business insigts about housing price in the King County to the business stakeholder

The project has utilized public data to analyze the housing price market. Based upon the findings about Housing price and the factors of house price, recommendatios are given to the stakerholder

1.2 Business Understanding

A real estate consulting and services company helps homeowners and other customers to buy and sell properties. They provide services to customers to evaluate house prices and check what factors are affecting house values.

We work on this project to provide insights to this real estate company

Business insights to investigate:

1. Estimate house values
2. What factors are affecting house price? By how much ?

1.3 Data Understanding

Source of the data is the King County Housing Price dataset. The Dataset contains the house price, with other aspects of the house such as living room area, lot size , number of bedrooms, number of bathrooms etc. we will use the dataset to develop models for regression analysis

1.4 Data Preparation

1.4.1 Loading the Data

We load in the dataset. We use some data as the training data for our model, and some other data as the testing data to check model performance

In [56]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

from sklearn.linear_model import LinearRegression

pd.set_option('max_columns', None)

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

In [57]:
```
df = pd.read_csv('data\\kc_house_data.csv')
```

In [58]:
```
df
```

Out[58]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	NONE	Average	7 Average	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO	NONE	Average	7 Average	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO	NONE	Average	6 Low Average	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO	NONE	Very Good	7 Average	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO	NONE	Average	8 Good	
...	...	...	...	...	...	...	...	...	...	...	...	...	
21592	263000018	5/21/2014	360000.0	3	2.50	1530	1131	3.0	NO	NONE	Average	8 Good	
21593	6600060120	2/23/2015	400000.0	4	2.50	2310	5813	2.0	NO	NONE	Average	8 Good	
21594	1523300141	6/23/2014	402101.0	2	0.75	1020	1350	2.0	NO	NONE	Average	7 Average	
21595	291310100	1/16/2015	400000.0	3	2.50	1600	2388	2.0	NaN	NONE	Average	8 Good	
21596	1523300157	10/15/2014	325000.0	2	0.75	1020	1076	2.0	NO	NONE	Average	7 Average	

21597 rows × 21 columns

Some more information about the features of this dataset:

1.4.2 Display and Explore the Datasets

In [59]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  object 
 9   view           21534 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(6), int64(9), object(6)
memory usage: 3.5+ MB
```

In [60]:
```
df.describe()
```

Out[60]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	yr_built	y
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.000000	21597.000000	1
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	1788.596842	1970.999676	
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	827.759761	29.375234	
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	370.000000	1900.000000	
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	1190.000000	1951.000000	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	1560.000000	1975.000000	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	2210.000000	1997.000000	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	9410.000000	2015.000000	

Average house Price $540,000

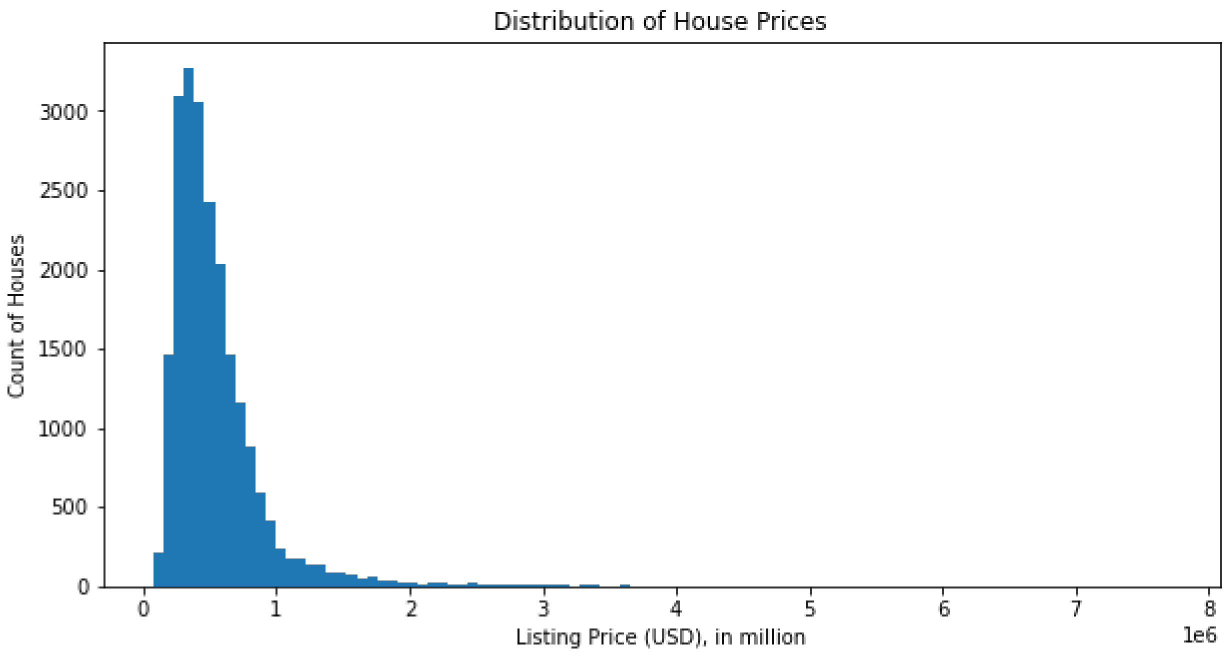
1.4.3 Check Missing Data

In [61]:
```
df.isnull().sum()
```

Out[61]:
```
id                  0
date                0
price               0
bedrooms            0
bathrooms           0
sqft_living         0
sqft_lot            0
floors              0
waterfront       2376
view               63
condition           0
grade               0
sqft_above          0
sqft_basement       0
yr_built            0
yr_renovated     3842
zipcode             0
lat                 0
long                0
sqft_living15       0
sqft_lot15          0
dtype: int64
```

1.4.4 Check House Price Distribution

In [62]:
```python
fig, ax = plt.subplots(figsize=(10, 5))

ax.hist(df['price'], bins=100)

ax.set_xlabel("Listing Price (USD), in million")
ax.set_ylabel("Count of Houses")
ax.set_title("Distribution of House Prices");
```

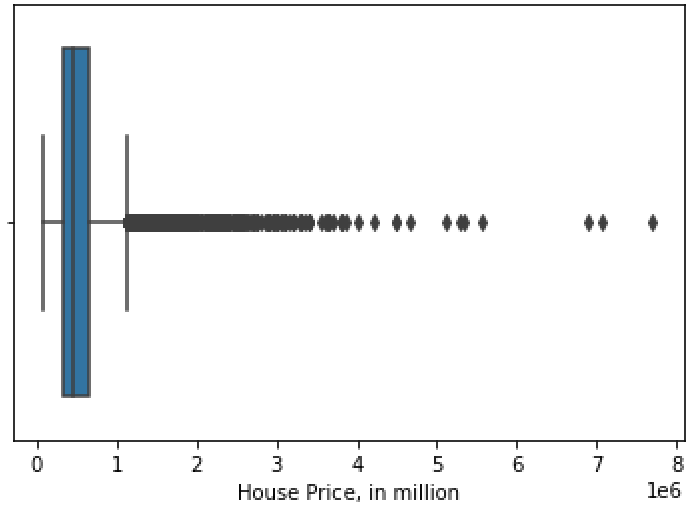


House Price distribution shows some outlier values

House Price Distribution, Box Plot

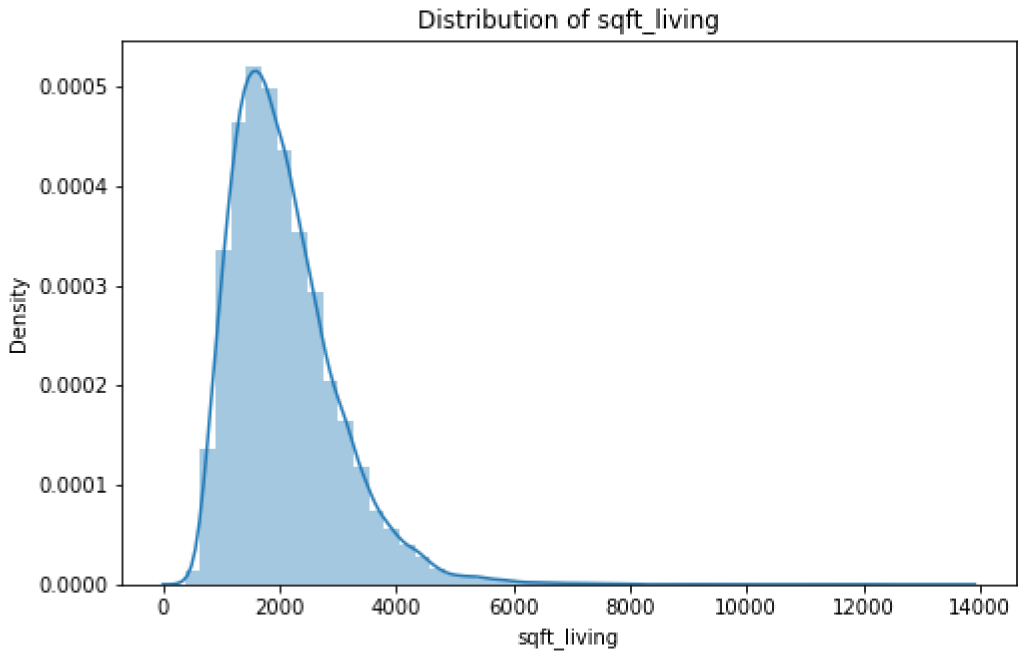
In [63]:
```
sns.boxplot(x='price', data=df ).set( xlabel='House Price, in million')
```

Out[63]:
```
[Text(0.5, 0, 'House Price, in million')]
```



Check sqft_living variable distribution

In [64]:
```
fig, ax = plt.subplots(figsize=(8, 5))
ax = sns.distplot(df['sqft_living'])
ax.set_title("Distribution of sqft_living");
```

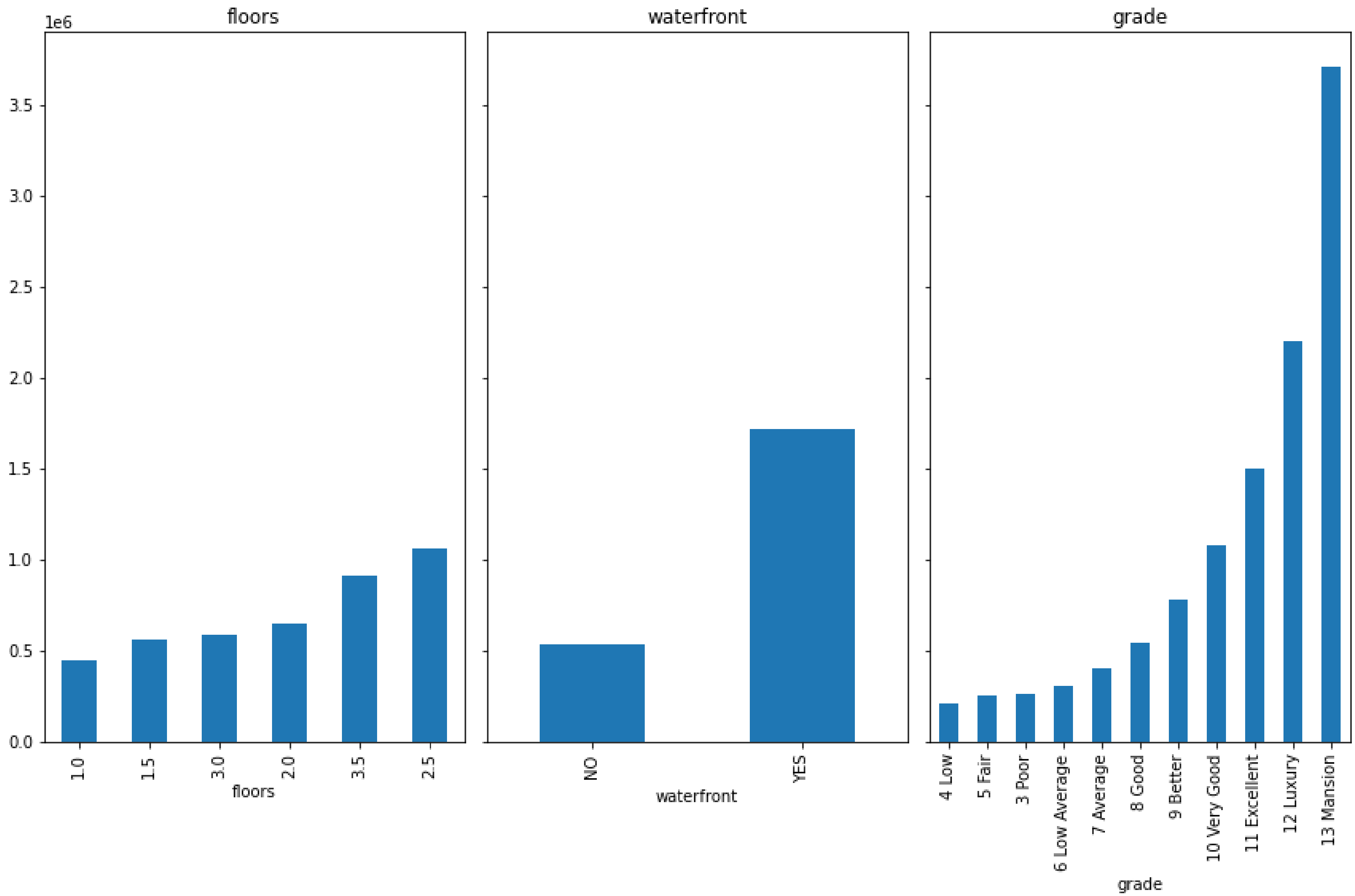


Scatter Plot of condition

In [65]:
```
fig, ax = plt.subplots(figsize=(10, 5))
sns.scatterplot(data=df, x='condition', y='price')
ax.set_title("Scatter Plot of condition");
```

1.4.5 Other Categorical Variables

In [66]:
```python
import matplotlib.pyplot as plt
%matplotlib inline

# Create bar plots
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(12,8), sharey=True)

categoricals = ['floors', 'waterfront', 'grade']

for col, ax in zip(categoricals, axes.flatten()):
    (df.groupby(col)                # group values together by column of interest
         .mean()['price']        # take the mean of the saleprice for each group
         .sort_values()              # sort the groups in ascending order
         .plot
         .bar(ax=ax))                # create a bar graph on the ax

    ax.set_title(col)                # Make the title the name of the column

fig.tight_layout()
```



1.4.6 Relevant Features

Our Team has identified the most relevant features for house price

In [67]:
```
features = df.columns
features
```

Out[67]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [68]:
```
columns_drop = ['id', 'date', 'view', 'sqft_above', 'sqft_basement', 'yr_renovated',
                'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15']
```

In [69]:
```
df.drop(columns=columns_drop, inplace=True)
df.head()
```

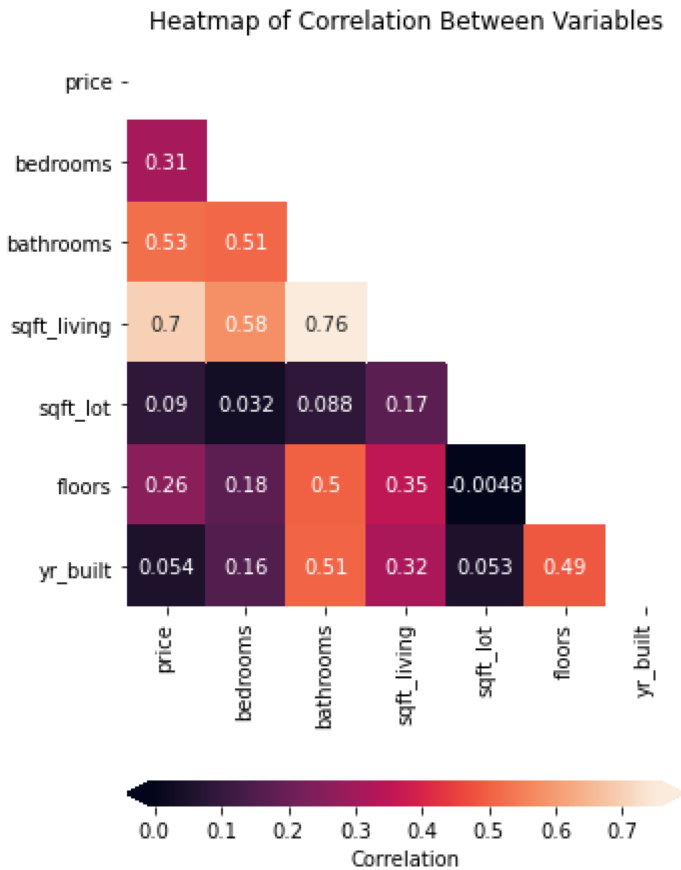
Out[69]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	yr_built
0	221900.0	3	1.00	1180	5650	1.0	NaN	Average	7 Average	1955
1	538000.0	3	2.25	2570	7242	2.0	NO	Average	7 Average	1951
2	180000.0	2	1.00	770	10000	1.0	NO	Average	6 Low Average	1933
3	604000.0	4	3.00	1960	5000	1.0	NO	Very Good	7 Average	1965
4	510000.0	3	2.00	1680	8080	1.0	NO	Average	8 Good	1987

1.4.7 Correlation Heatmap

In [70]:
```python
# compute the correlation matrix
heatmap_data = df
corr = heatmap_data.corr()

fig, ax = plt.subplots(figsize=(5, 8))

# Plot a heatmap of the correlation matrix, with both
# numbers and colors indicating the correlations
sns.heatmap(
    data=corr,
    # The mask means we only show half the values,
    mask=np.triu(np.ones_like(corr, dtype=bool)),
    # Specifies that we should use the existing axes
    ax=ax,
    # Specifies that we want labels, not just colors
    annot=True,
    # Customizes colorbar appearance
    cbar_kws={"label": "Correlation", "orientation": "horizontal", "pad": .2, "extend": "both"}
)

ax.set_title("Heatmap of Correlation Between Variables")
```

Out[70]:
```
Text(0.5, 1.0, 'Heatmap of Correlation Between Variables')
```



From the Heatmap , the most correlated featrue with price is sqft_living

In [71]:
```
most_correlated_feature = "sqft_living"
```

1.4.8 Data Preprocessing

Missing Values - Drop

In [72]:
```
df.shape
```

Out[72]: (21597, 10)

In [73]:
```
df.dropna(inplace=True)
```

In [74]:
```
df.shape
```

Out[74]:
```
(19221, 10)
```

Remove Outliers

In [75]:
```
df=df[df['price']<1750000]
df.head()
```

Out[75]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	yr_built
1	538000.0	3	2.25	2570	7242	2.0	NO	Average	7 Average	1951
2	180000.0	2	1.00	770	10000	1.0	NO	Average	6 Low Average	1933
3	604000.0	4	3.00	1960	5000	1.0	NO	Very Good	7 Average	1965
4	510000.0	3	2.00	1680	8080	1.0	NO	Average	8 Good	1987
5	1230000.0	4	4.50	5420	101930	1.0	NO	Average	11 Excellent	2001

Categorical Variables Encoding

Encode 'condition' Variable

In [76]:
```
LE_condition = LabelEncoder()
```

In [77]:
```
condition_encoded = LE_condition.fit_transform(df['condition'])
```

In [78]:
```
df['condition_encoded'] = condition_encoded
```

Encode 'grade' Variable

Extract the numerical values

In [79]:
```
def grade_encode(string):
    return int(string[0])
```

In [80]:
```
df['grade_encoded']= df['grade'].apply(grade_encode)
```

Encode 'waterfront' Variable

In [81]:
```
LE_waterfront = LabelEncoder()
df['waterfront_encoded'] = LE_waterfront.fit_transform(df['waterfront'])
```

Encode Year_built Variable

Assign each decade to a label. i.e 1950-1959 : label 5 1980-1989 : label 8

In [82]:
```
def year_encode(year):
    if year<1930:
        return 0
    elif year>=2000 and year<2011:
        return 10
    elif year>=2011:
        return 11
    else:
        return int(str(year)[2])
```

In [83]:
```
df['yr_built_encoded']=df['yr_built'].apply(year_encode)
```

In [84]:
```python
df.head(5)
```

Out[84]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade	yr_built	condition_encoded	grade_enco
1	538000.0	3	2.25	2570	7242	2.0	NO	Average	7 Average	1951	0	
2	180000.0	2	1.00	770	10000	1.0	NO	Average	6 Low Average	1933	0	
3	604000.0	4	3.00	1960	5000	1.0	NO	Very Good	7 Average	1965	4	
4	510000.0	3	2.00	1680	8080	1.0	NO	Average	8 Good	1987	0	
5	1230000.0	4	4.50	5420	101930	1.0	NO	Average	11 Excellent	2001	0	

In [85]:
```python
columns_drop = ['waterfront', 'grade', 'condition', 'yr_built']
df.drop(columns= columns_drop, inplace=True)
```

In [86]:
```python
df.head()
```

Out[86]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition_encoded	grade_encoded	waterfront_encoded	yr_built_encode
1	538000.0	3	2.25	2570	7242	2.0	0	7	0	
2	180000.0	2	1.00	770	10000	1.0	0	6	0	
3	604000.0	4	3.00	1960	5000	1.0	4	7	0	
4	510000.0	3	2.00	1680	8080	1.0	0	8	0	
5	1230000.0	4	4.50	5420	101930	1.0	0	1	0	1

In [87]:
```python
df.shape
```

Out[87]: (18926, 10)

1.4.9 Training Data and Testing Data

In [88]:
```
df.columns
```

Out[88]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'condition_encoded', 'grade_encoded', 'waterfront_encoded',
       'yr_built_encoded'],
      dtype='object')
```

In [89]:
```
Y=df['price']
X=df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'condition_encoded', 'grade_encoded', 'waterfront_encoded',
       'yr_built_encoded']]
```

In [90]:
```
X.describe()
```

Out[90]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition_encoded	grade_encoded	waterfront_encoded	yr
count	18926.000000	18926.000000	18926.000000	1.892600e+04	18926.000000	18926.000000	18926.000000	18926.000000	
mean	3.360404	2.092730	2040.603825	1.490190e+04	1.489142	0.850893	7.015059	0.004703	
std	0.920992	0.739256	846.167014	4.037491e+04	0.539118	1.263683	1.804013	0.068415	
min	1.000000	0.500000	370.000000	5.200000e+02	1.000000	0.000000	1.000000	0.000000	
25%	3.000000	1.500000	1420.000000	5.014250e+03	1.000000	0.000000	7.000000	0.000000	
50%	3.000000	2.250000	1900.000000	7.560000e+03	1.500000	0.000000	7.000000	0.000000	
75%	4.000000	2.500000	2510.000000	1.050400e+04	2.000000	2.000000	8.000000	0.000000	
max	33.000000	7.500000	7620.000000	1.651359e+06	3.500000	4.000000	9.000000	1.000000	

Type *Markdown* and LaTeX: α^2

In [91]:
```
### std of the encoded Variables

lst_std=[]
for column in X.describe().columns:
    std=X.describe()[column]['std']
    lst_std.append(std)
```

1.4.10 Dataset Train Test Split

In [92]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=40)
```

1.4.11 Feature Scaling

In [93]:
```
scaler= StandardScaler()
```

In [94]:
```
X_train_scaled = scaler.fit_transform(X_train)
```

In [95]:
```
X_train_scaled
```

Out[95]:
```
array([[-1.45864228, -1.47266711, -1.27280381, ..., -0.56548928,
        -0.06569733,  0.48244049],
       [-0.38650886,  0.55685248,  1.05628415, ...,  1.10069287,
        -0.06569733,  1.10780672],
       [ 0.68562457,  2.2481188 , -0.06688009, ...,  1.10069287,
        -0.06569733,  1.10780672],
       ...,
       [ 0.68562457,  0.21859921, -0.19693069, ..., -0.01009523,
        -0.06569733,  1.10780672],
       [-0.38650886, -1.47266711, -0.68166473, ..., -0.01009523,
        -0.06569733, -0.45560887],
       [-1.45864228, -0.79616058, -1.22551268, ..., -0.01009523,
        -0.06569733, -0.76829198]])
```

In [96]:
```
X_test_scaled = scaler.transform(X_test)
```

1.5 Modeling

1.5.1 Building a Baseline Model

Now, we'll build a linear regression model using just the most correlated feature, which will serve as our baseline model:

In [97]:
```
from sklearn.linear_model import LinearRegression

baseline_model = LinearRegression()
```

Then we evaluate the model using `cross_validate` , we perform 3 separate train-test splits within our `X_train` and `y_train` , then we find both the train and the test scores for each.

In [98]:
```
from sklearn.model_selection import cross_validate, ShuffleSplit

splitter = ShuffleSplit(n_splits=3, test_size=0.25, random_state=0)

baseline_scores = cross_validate(
    estimator=baseline_model,
    X=X_train[[most_correlated_feature]],
    y=y_train,
    return_train_score=True,
    cv=splitter
)

print("Train score:     ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

```
Train score:      0.44901908790995093
Validation score: 0.4485073282167599
```

The coefficient of determination scores on both the training set and validation sets for the baseline model are about 0.44

1.5.2 2. Build a Model with Relevant Features

    Build and Evaluate second Model , with the trasfromed featrues

In [99]:
```
second_model = LinearRegression()

second_model_scores = cross_validate(
    estimator=second_model,
    X=X_train_scaled,
    y=y_train,
    return_train_score=True,
    cv=splitter
)
```

In [100]:
```
print("Current second Model")
print("Train score:      ", second_model_scores["train_score"].mean())
print("Validation score:", second_model_scores["test_score"].mean())
print()

print("Baseline Model")
print("Train score:      ", baseline_scores["train_score"].mean())
print("Validation score:", baseline_scores["test_score"].mean())
```

```
Current second Model
Train score:      0.5322276193151804
Validation score: 0.5287335014527756

Baseline Model
Train score:      0.44901908790995093
Validation score: 0.4485073282167599
```

Our second model got slightly better scores on both the training data and the validation data. It seems that adding additional features will help the model to capture the relationships between the independent variables and the target

1.5.3 Build and Evaluate a Final Predictive Model

We use the second model features to train our final predictive Model. We will then evaluate the model performance on the test set

In [101]:
```
final_model = LinearRegression()

final_model.fit(X_train_scaled, y_train)

# Score the model on X_test_final and y_test
score=final_model.score(X_test_scaled, y_test)
print('Coefficient of Determination score', score)
```

```
Coefficient of Determination score 0.5285701284093244
```

1.6 Model Prediction

In [102]:
```
predictions=final_model.predict(X_test_scaled)
```

In [103]:
```
d={'Price':y_test, 'Predicted Price':predictions}
data_frame=pd.DataFrame(data=d)
```

In [104]:
```
data_frame
```

Out[104]:

	Price	Predicted Price
20422	530000.0	706963.963583
5784	415000.0	990927.482799
11293	1050000.0	739515.499954
9982	655000.0	650575.759705
14776	670000.0	411108.828087
...	...	...
12342	450000.0	402462.437803
15264	765000.0	691685.389436
18820	355000.0	308993.187359
17043	530000.0	462702.700391
3508	749950.0	737145.600616

4732 rows × 2 columns

1.7 Model Evaluation and Interpretation

1.7.1 Metrics RMSE

The previous score above is an r-squared score. Let's compute the RMSE as well, since this would be more applicable to the business audience.

In [105]:
```
from sklearn.metrics import mean_squared_error

print('Estimation Error')
mean_squared_error(y_test,predictions, squared=False)
```

```
Estimation Error
```

Out[105]:
```
183580.36088270726
```

This means that for an average House Price, this algorithm prediction will be off by about 183580. Given that the averge house price is $540296, this prediction is not the best

1.7.2 Interpret the Final Model

Below, we display the coefficients and intercept for the final model with sacled features:

In [106]:
```
print('Variable Coeffeicinets, scaling features\n')
print(pd.Series(final_model.coef_, index=X_train.columns, name="Coefficients"))
print()
print("Intercept:", final_model.intercept_)
```

```
Variable Coeffeicinets, scaling features

bedrooms              -34418.593619
bathrooms              35303.901476
sqft_living           182907.321046
sqft_lot               -5892.706855
floors                 35577.751546
condition_encoded      12649.665516
grade_encoded         -19775.275562
waterfront_encoded     26478.812761
yr_built_encoded      -70036.884498
Name: Coefficients, dtype: float64

Intercept: 511735.0945469917
```

Any features and variables will have the effects of chaning the housing price. Since the features are scaled, it shows that if the sqft_living, the living room areas, increases by one stardard deviation, 915 sqft, the house price will go up by 182907.

Since the Features are scaled in the data preparation process. The coefficients are also scaled. we could perform the convertion to check unit change

Conversion and Check the effect of unit change of feature on the target variable

In [107]:

```
coef_scaled=final_model.coef_
#lst_std
coef_lst=[]
for coef, std in zip(coef_scaled,lst_std ):
    coef_lst.append(coef/std)

print('Variable Coeffeicinets \n')
print(pd.Series(coef_lst, index=X_train.columns, name="Coefficients"))
print()
```

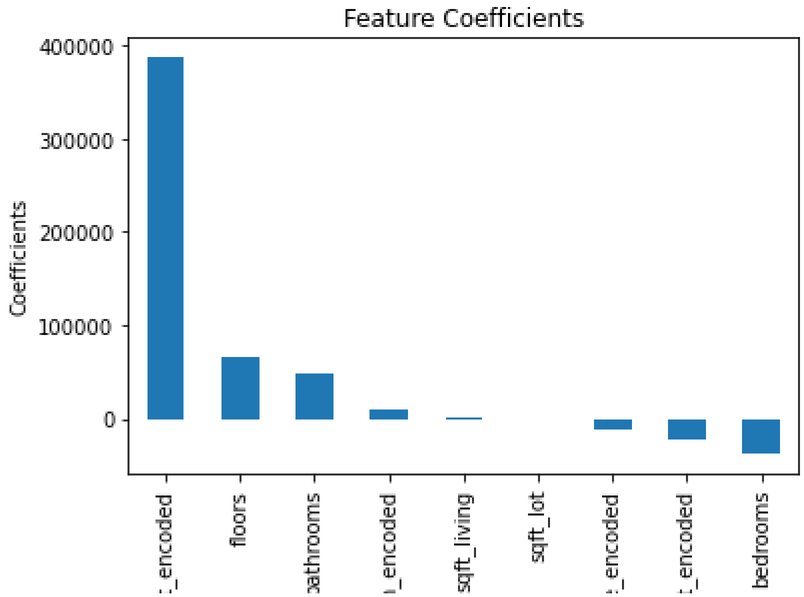
```
Variable Coeffeicinets

bedrooms              -37371.213679
bathrooms              47755.971065
sqft_living              216.159834
sqft_lot                  -0.145950
floors                 65992.506014
condition_encoded      10010.159340
grade_encoded         -10961.825614
waterfront_encoded    387030.328313
yr_built_encoded      -21906.701728
Name: Coefficients, dtype: float64
```

In [108]:

```
s=(pd.Series(coef_lst, index=X_train.columns, name="Coefficients"))
```

In [109]:
```python
fig, ax = plt.subplots()
ax=s.sort_values(ascending=False).plot.bar()
ax.set_title("Feature Coefficients");
ax.set_xlabel('Features')
ax.set_ylabel('Coefficients')
```

Out[109]: `Text(0, 0.5, 'Coefficients')`



Model with No Scaling Features

In [110]:
```
final_model_no_scale = LinearRegression()
final_model_no_scale.fit(X_train, y_train)

print('final model no scaling feature\n')
print(pd.Series(final_model_no_scale.coef_, index=X_train.columns, name="Coefficients"))
print()
print("Intercept:", final_model_no_scale.intercept_)
```

```
final model no scaling feature

bedrooms              -36901.324650
bathrooms              47766.639735
sqft_living              216.247329
sqft_lot                  -0.140204
floors                 66225.946635
condition_encoded      10004.068172
grade_encoded         -10983.070375
waterfront_encoded    404782.019225
yr_built_encoded      -21899.351372
Name: Coefficients, dtype: float64

Intercept: 207385.94403028378
```

We can see from the coefficients that the features are related to the house price, such as sqft_living, floors, waterfront, condition etc. These features has the positive effects on house price, and The waterfront variable has the largest positive effect on price

If sqrt_living the area of the living room increases, the house price would increases as well.

If there is waterfront, the house price will increase

if the condition of house improves, its price will increase as well

Therefore, for modeling with no scaled features, base price 207385. For a unit chage of sqft in living room, the house price will change by $216

1.7.3 Recommendations

The Business Stakeholders can focus on one or more areas to improve the housing price. They could focus on condition. House condition is positively related to the house price. With other factors the same, improving the house condition will increase the house value. Increasing the house condition by one level may increase the house price by 10010

The business stakeholders can also focus on living room area as well. It shows a positive relationship between living room area and the house price

Limitations

The final model shows a coefficient of determination socre of 0.5 on the unseen dataset. It is not the greatest score. And therefore, the model still needs a lot of improvement to capture all the relationshps between variables.

The model also shows a large RMSE error of 183580. Given that the average house price is at 540296, the prediction of the house price is not the best with this large error. Therefore, it also shows that the model needs improvement.

The coefficients may show the effects. But the effect quantity needs to be investigated more based upon the model performance

1.7.4 Next Steps for Modeling

1. Use advanced non-linear models
2. Improve model performance or develop a better model. With a good model, Check if bringing in more data would help

In [ ]: