King County House Price Prediction

Regression Analysis Project -1 and 2 ¶

To Predict the sales price of Houses in King's County, USA

Importing Relevant Modules/Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
from sklearn.tree import export_graphviz
from io import StringIO
from IPython.display import Image
import pydotplus
import pydot
```

In [2]:
```
df = pd.read_csv('kc_house_data.csv')
```

Variable Description

id - Unique ID for each home sold

date - Date of the home sale

price - Price of each home sold

bedrooms - Number of bedrooms

bathrooms - Number of bathrooms, where .5 accounts for a room with a toilet but no shower

sqft_living - Square footage of the apartments interior living space

sqft_lot - Square footage of the land space

floors - Number of floors

waterfront - A dummy variable for whether the apartment was overlooking the waterfront or not

view - An index from 0 to 4 of how good the view of the property was

condition - An index from 1 to 5 on the condition of the apartment,

grade - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design.

sqft_above - The square footage of the interior housing space that is above ground level

sqft_basement - The square footage of the interior housing space that is below ground level

yr_built - The year the house was initially built

yr_renovated - The year of the house's last renovation

zipcode - What zipcode area the house is in

lat - Lattitude

long - Longitude

sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors

sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

The Initial Dataset contains 20 explanatory variables and 1 response/dependent variable(price)

The dataset contains 21,613 rows of data

In [3]:
```
df.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0

5 rows × 21 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcod
e',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

Summary Statistics Of The Data

In [6]:
```
df.describe()
```

Out[6]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06

In [7]:
```
df['view'].unique()
```

Out[7]:
```
array([0, 3, 4, 2, 1], dtype=int64)
```

Data Cleaning

As the columns id and date would not provide any value in predicting the price of the houses its better to drop these irrelevant variables from the dataset.

In [8]:
```
df.drop(['id','date'],axis=1,inplace=True)
```

In [9]:
```
df.loc[(df['yr_renovated'] == 0),'yr_renovated']= 0
df.loc[(df['yr_renovated'] != 0),'yr_renovated']= 1
df.head()
```

Out[9]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0	0	3	
1	538000.0	3	2.25	2570	7242	2.0	0	0	3	
2	180000.0	2	1.00	770	10000	1.0	0	0	3	
3	604000.0	4	3.00	1960	5000	1.0	0	0	5	
4	510000.0	3	2.00	1680	8080	1.0	0	0	3	

As we see that there were a lot many houses that were not renovted so to make this column of information useful we converted this column to a binary column taking the value of 1 if the house was ever renovated and 0 if it was never renovated.

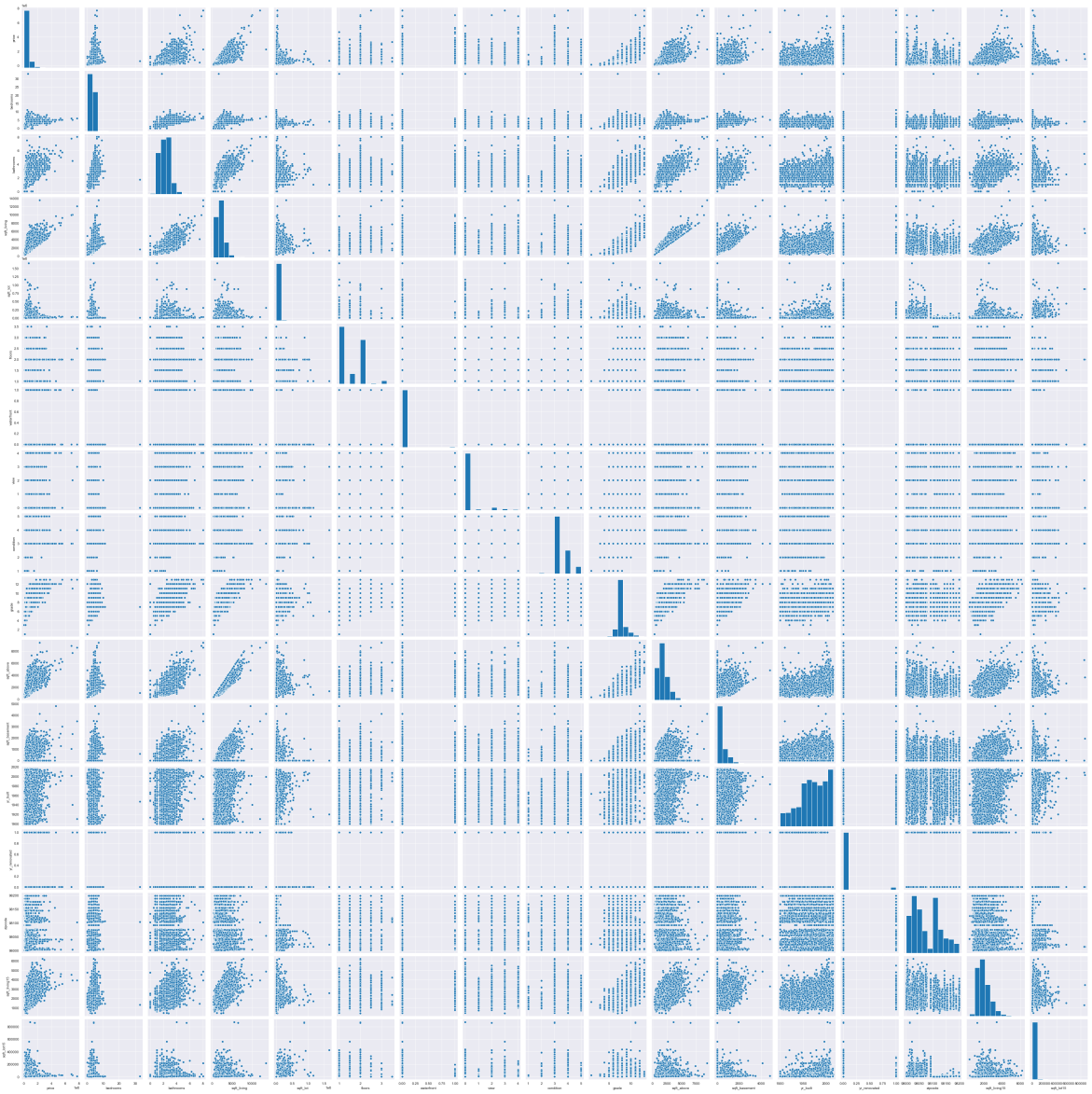
In [10]:
```
df.drop(['lat','long'],axis=1,inplace=True)
```

EDA and Data Visualization

In [11]:
```
sns.set_style('darkgrid')
```

In [12]:
```
sns.pairplot(df)
```

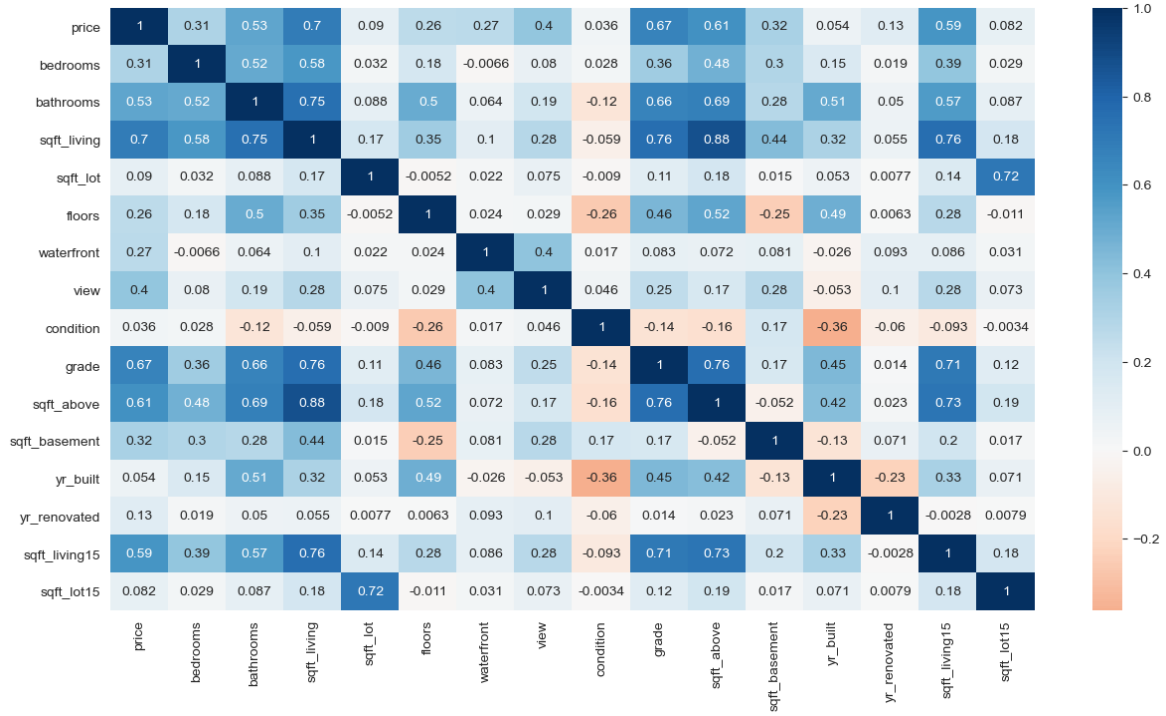
Out[12]: <seaborn.axisgrid.PairGrid at 0x17fbfbbac88>



Correlation Matrix

In [13]:
```
fig = plt.figure(figsize=(15,8),dpi=100)
sns.heatmap(df.drop('zipcode',axis=1).corr(),cmap="RdBu",annot=True,center=0)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x17fc88a3ac8>



We see that most of the variables have positive relationship with the dependent variable price. we see that the explanatory variable price has the highest correlation with the dependent variable price and that too a positive one which indicates that as the living area of the house increases so does the price of the house

Variables like grade(the type of construction of the house) and the average area of 15 houses near the house of concern also have a very significant correlation with the price of the house of concern.

Variables such as condition of the house and whether the house was ever renovated or not did not have significant correlations with the dependent variables price.

In [14]:
```
df.head()
```

Out[14]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0	0	3	
1	538000.0	3	2.25	2570	7242	2.0	0	0	3	
2	180000.0	2	1.00	770	10000	1.0	0	0	3	
3	604000.0	4	3.00	1960	5000	1.0	0	0	5	
4	510000.0	3	2.00	1680	8080	1.0	0	0	3	

In [15]:
```
# Distribution Of Price
```

In [16]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.distplot(df['price'],bins=50,kde=False)
plt.ylabel('Frequency')
plt.xlabel('Price')
plt.title('Distribution Of Price')
```

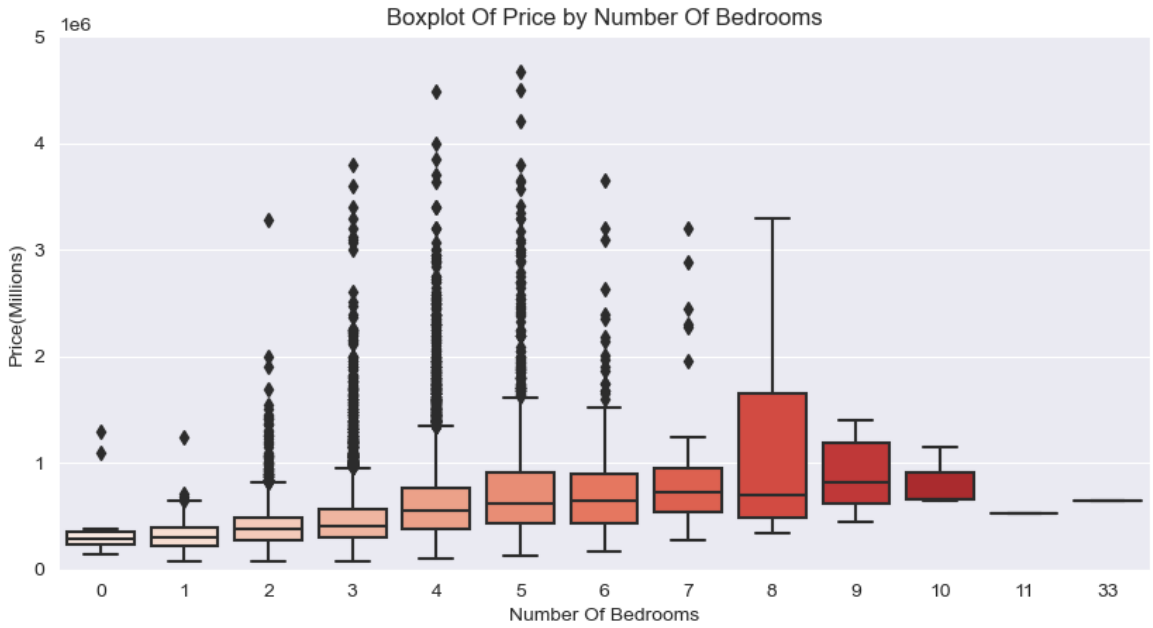
Out[16]:
```
Text(0.5, 1.0, 'Distribution Of Price')
```

The dependent variable price seems to be very positively skewed which makes sense as there are very less houses that are extremely expensive and more houses that are affordable by the common folks in our datasets.

Price Vs Bedrooms

In [17]:
```
fig = plt.figure(figsize=(10,5),dpi=100)
sns.boxplot(x='bedrooms',y='price',data=df,palette='Reds')
plt.ylabel('Price(Millions)')
plt.xlabel('Number Of Bedrooms')
plt.title('Boxplot Of Price by Number Of Bedrooms')
plt.ylim([0,5000000])
```

Out[17]: (0.0, 5000000.0)

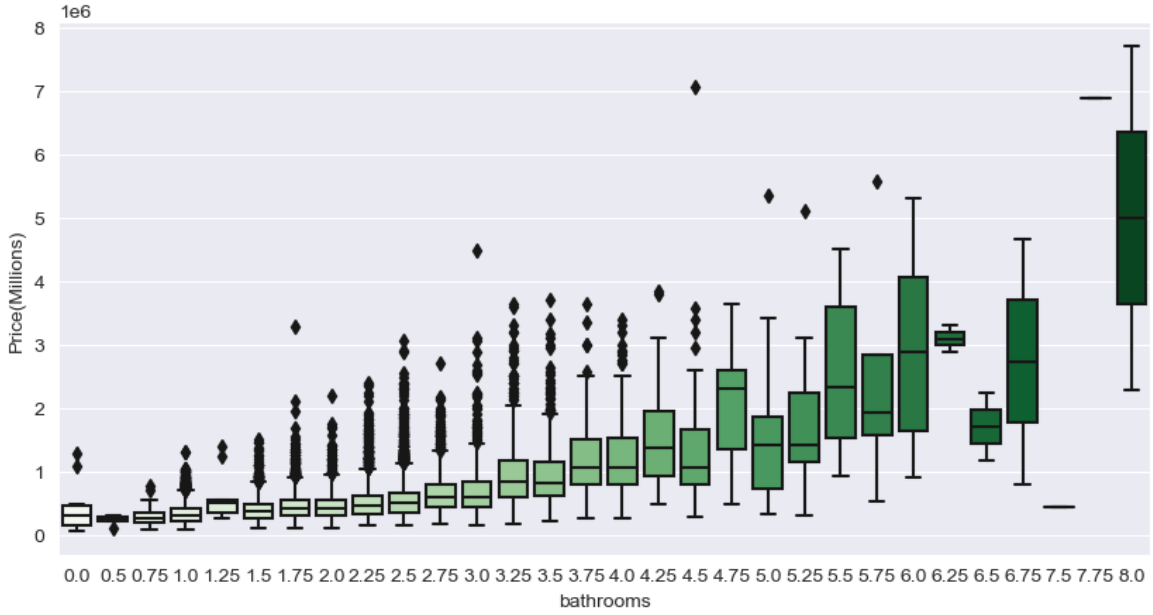


Price Of The House Increases Significantly With Increase in Number Of bedrooms that are there in the house

Price Vs Bathrooms

In [18]:
```
fig = plt.figure(figsize=(10,5),dpi=100)
sns.boxplot(x='bathrooms',y='price',data=df,palette='Greens')
plt.ylabel('Price(Millions)')
```

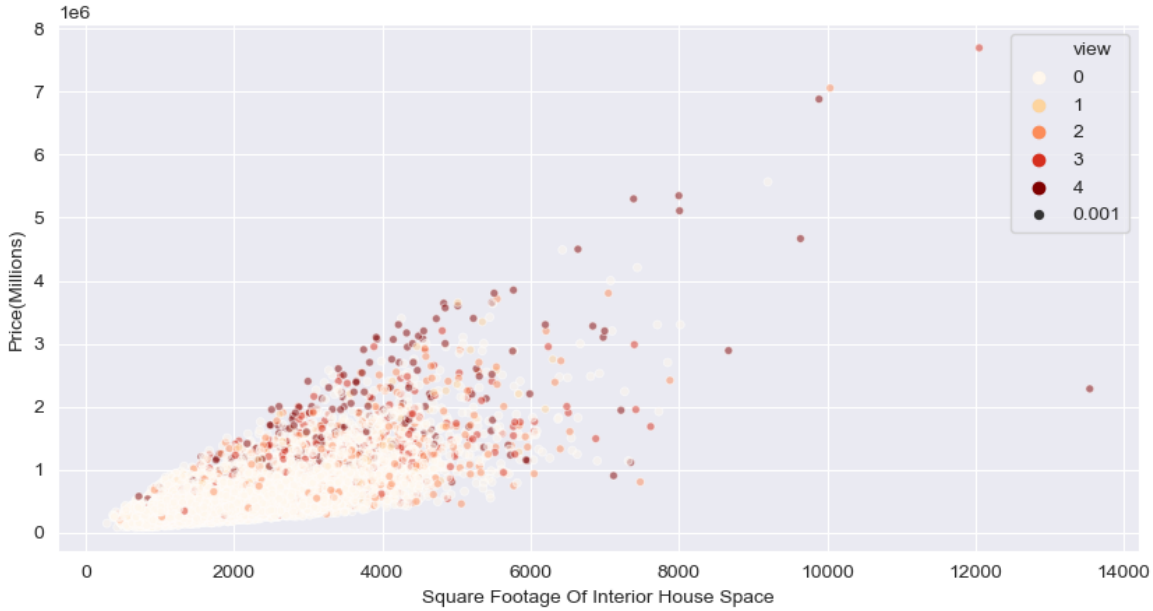
Out[18]: Text(0, 0.5, 'Price(Millions)')



Price Of The House Increases Significantly With Increase in Number Of bathrooms that are there in the house

In [19]:
```
# Square Foot Of the interior Space
fig = plt.figure(figsize=(10,5),dpi=100,edgecolor='black')
sns.scatterplot(x=df['sqft_living'],y='price',data=df,
                size=0.001,alpha=0.5,hue=df['view'],legend='full',palette='Or
plt.xlabel('Square Footage Of Interior House Space')
plt.ylabel('Price(Millions)')
```

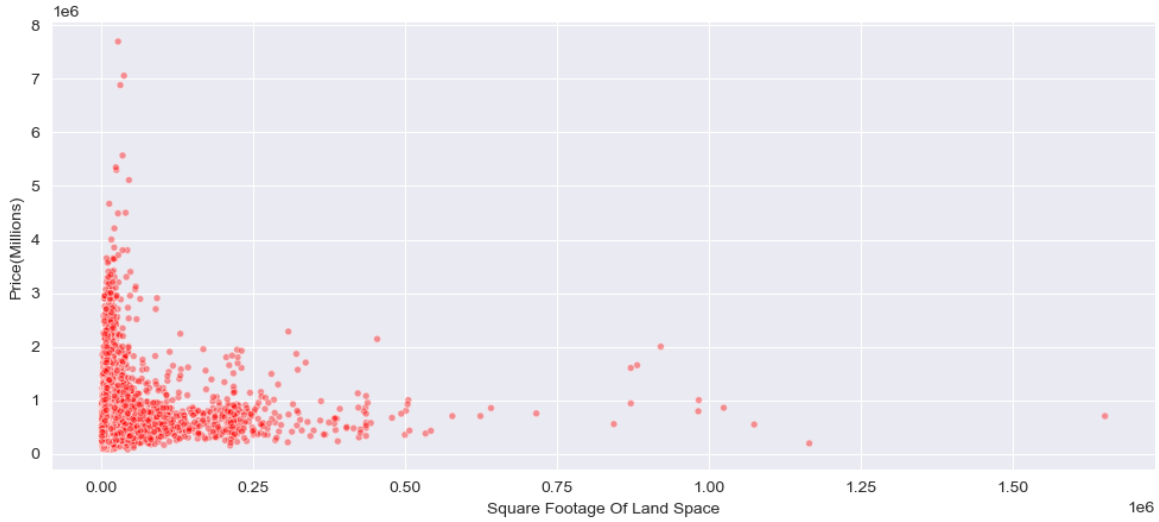
Out[19]: Text(0, 0.5, 'Price(Millions)')



A positive correlation between the price of the house and the area of living of the house is very evident and the dark spots rising towards the upper right hand side of the graph indicates that view also has a positive correlation with the house as the view increases(or getts better) so does the price of the house.

In [20]:
```
# Square Foot Of The Plot
fig = plt.figure(figsize=(12,5),dpi=100)
sns.scatterplot(x=df['sqft_lot'],y='price',data=df,
                size=0.001,alpha=0.4,color='red',legend=False)
plt.xlabel('Square Footage Of Land Space')
plt.ylabel('Price(Millions)')
```

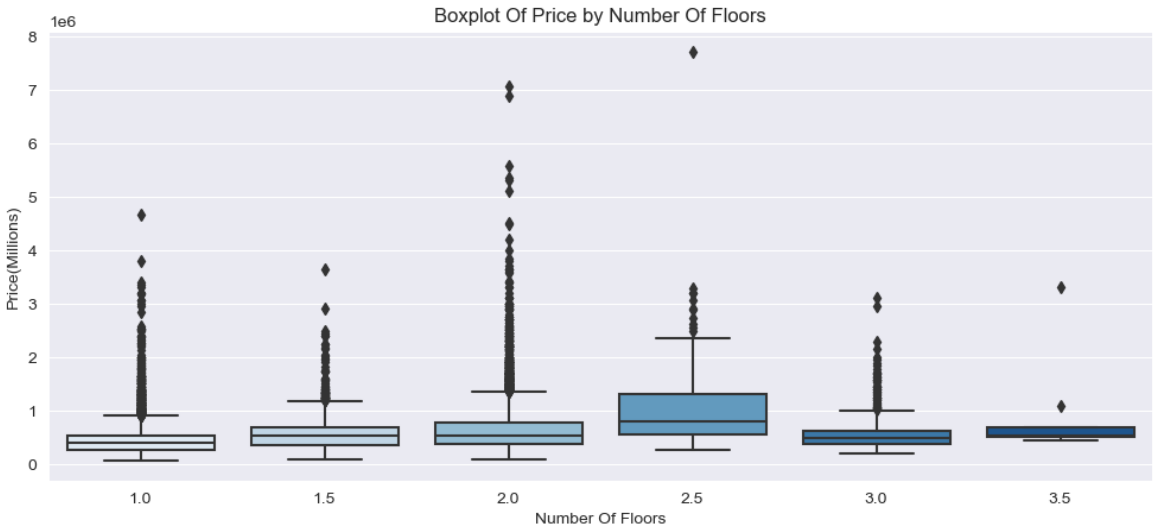
Out[20]: Text(0, 0.5, 'Price(Millions)')



It can be seen that the area of the entire plot dosen't seem to affect the overall price of the house as the lack of correlatio is very evident in the plot above.

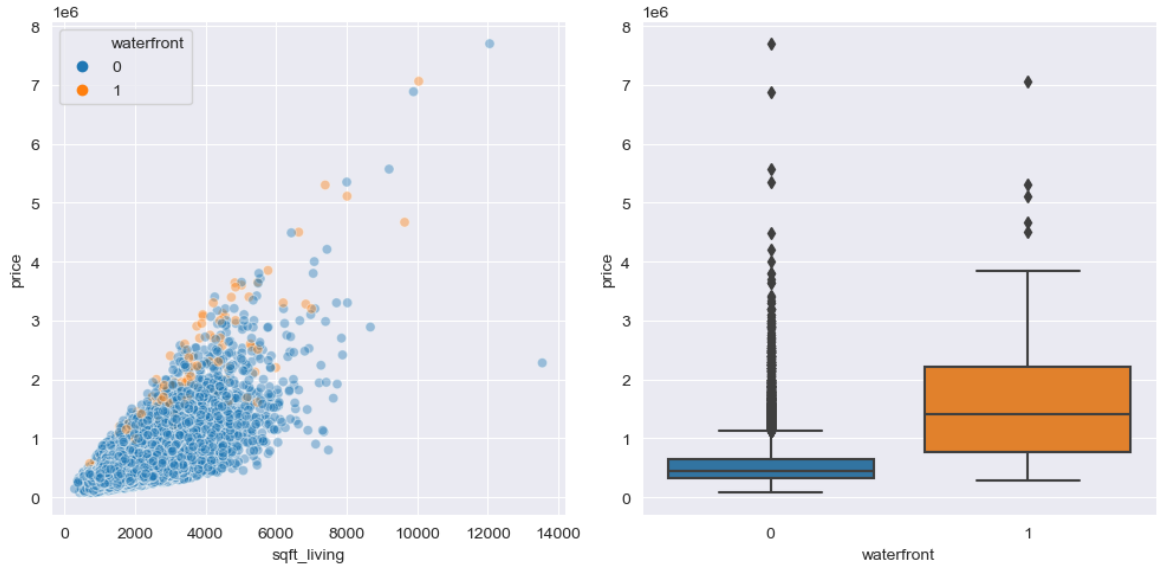
In [21]:
```
# Floors
fig = plt.figure(figsize=(12,5),dpi=100)
sns.boxplot(x='floors',y='price',data=df,palette='Blues')
plt.ylabel('Price(Millions)')
plt.xlabel('Number Of Floors')
plt.title('Boxplot Of Price by Number Of Floors')
```

Out[21]: Text(0.5, 1.0, 'Boxplot Of Price by Number Of Floors')



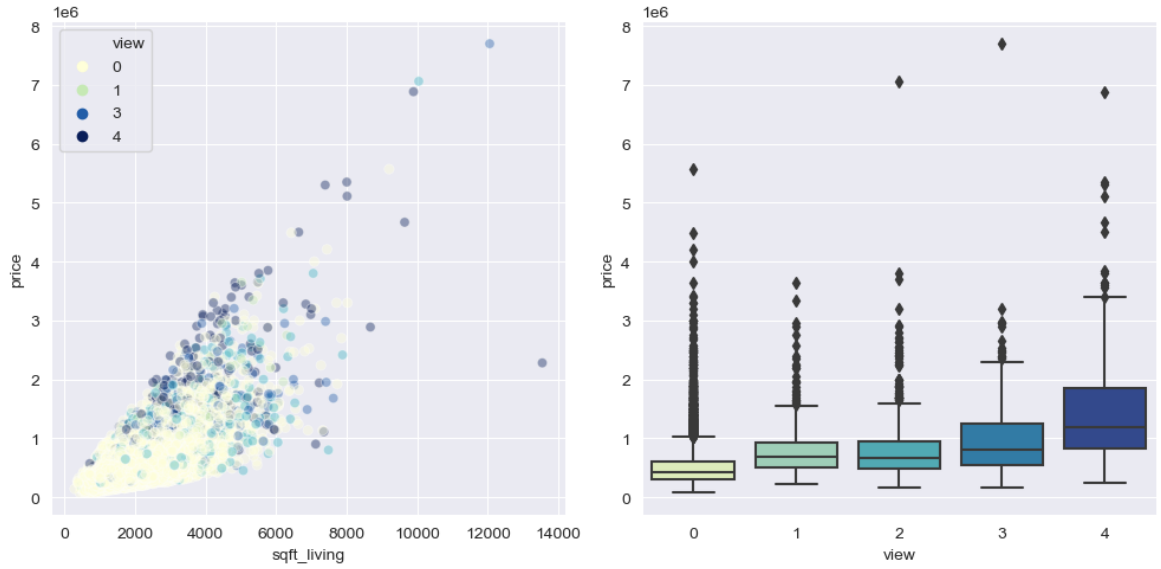
We can see that the price of the houses as the floors increase form 1 to 2.5 but we can see a sudden decline in prices for houses that have more than 2.5 floors

In [22]:
```
# Waterview
fig,axs = plt.subplots(ncols=2,figsize=(10,5),dpi=100)
sns.scatterplot(x='sqft_living', y='price', data=df,hue=df['waterfront'],alph
sns.boxplot(x='waterfront', y='price', data=df, ax=axs[1])
plt.tight_layout()
```



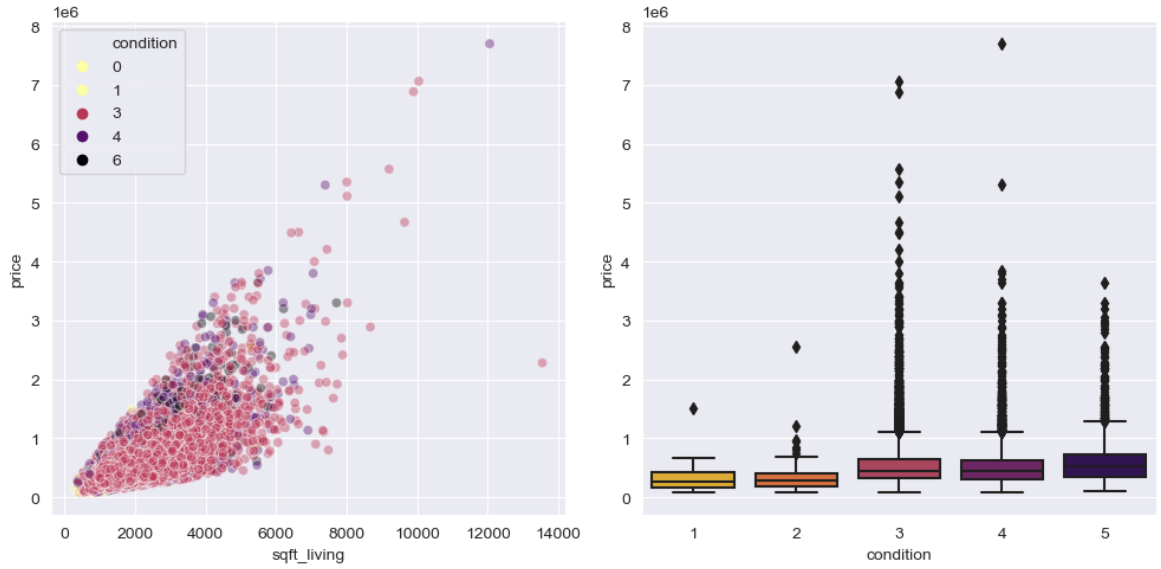
From the above scatterplot and the boxplot we can clearly see that houses that have waterfront are significantly more expensive than the houses that do not have the waterfront.

In [23]:
```
# View
fig,axs = plt.subplots(ncols=2,figsize=(10,5),dpi=100)
sns.scatterplot(x='sqft_living', y='price', data=df,hue=df['view'],alpha=0.4,
sns.boxplot(x='view', y='price', data=df, ax=axs[1],palette='YlGnBu')
plt.tight_layout()
```



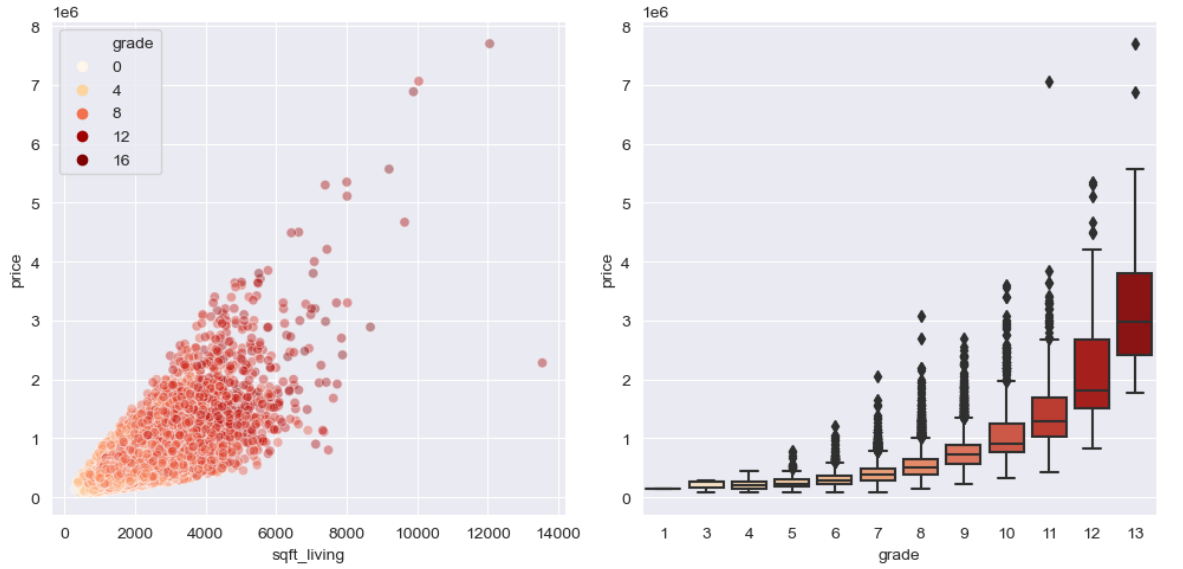
From the above scatterplot and the boxplot we can clearly see that houses that have better views(More the value of View)are significantly more expensive than the houses that have poorer views(less value of View).

In [24]:
```
# Condition
fig,axs = plt.subplots(ncols=2,figsize=(10,5),dpi=100)
sns.scatterplot(x='sqft_living', y='price', data=df,hue=df['condition'],alpha
sns.boxplot(x='condition', y='price', data=df, ax=axs[1],palette='inferno_r')
plt.tight_layout()
```



From the above scatterplot and the boxplot we can clearly see that houses that are in better condition(More the value of condition)are significantly more expensive than the houses that are in poorer condition(less value of condition).

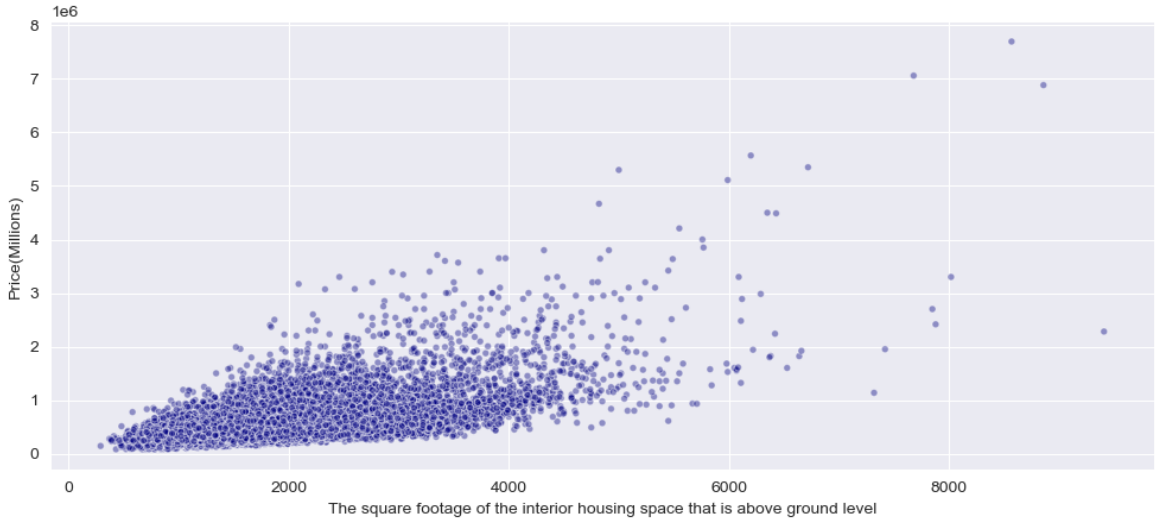
In [25]:
```
# grade
fig,axs = plt.subplots(ncols=2,figsize=(10,5),dpi=100)
sns.scatterplot(x='sqft_living', y='price', data=df,hue=df['grade'],alpha=0.4
sns.boxplot(x='grade', y='price', data=df, ax=axs[1],palette='OrRd')
plt.tight_layout()
```



From the above scatterplot and the boxplot we can clearly see that houses that have better Construction or Grade (More the value of grade)are significantly more expensive than the houses that have poorer construction(less value of grade).

In [26]:
```
# Condition
fig = plt.figure(figsize=(12,5),dpi=100)
sns.scatterplot(x=df['sqft_above'],y='price',data=df,
                size=0.001,alpha=0.4,color='navy',legend=False)
plt.xlabel('The square footage of the interior housing space that is above gr
plt.ylabel('Price(Millions)')
```

Out[26]: Text(0, 0.5, 'Price(Millions)')



We see that the area of the houses without the basement has a fair amount of correlation with the dependent variable price of the house

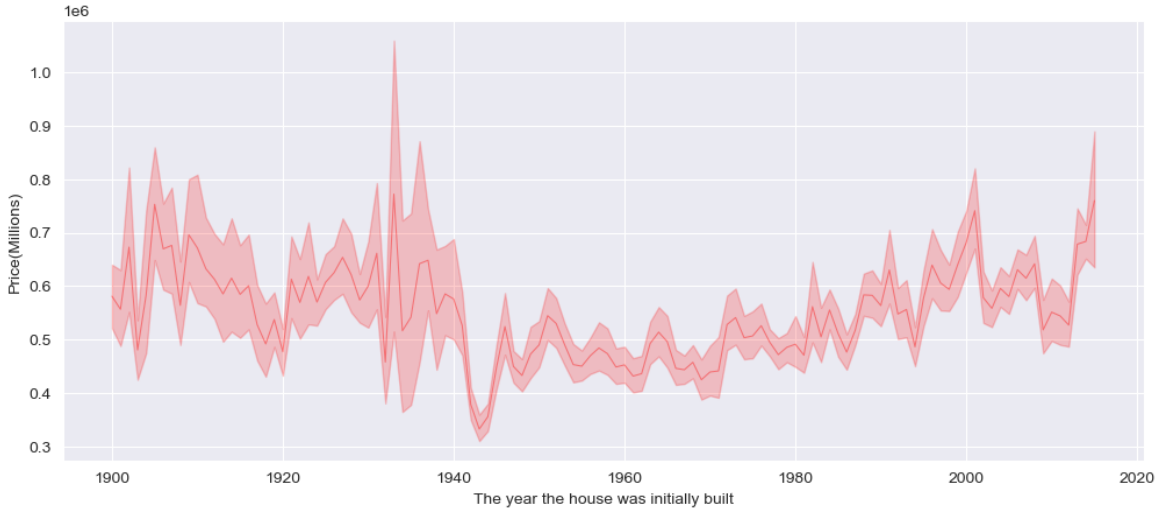
In [27]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.scatterplot(x=df['sqft_basement'],y='price',data=df,
                size=0.001,alpha=0.4,color='black',legend=False)
plt.xlabel('The square footage of the interior housing space that is below gr
plt.ylabel('Price(Millions)')
```

Out[27]: Text(0, 0.5, 'Price(Millions)')

We see that the area of the basemenent of the house has absolutely no correlation with the dependent variable price of the house.

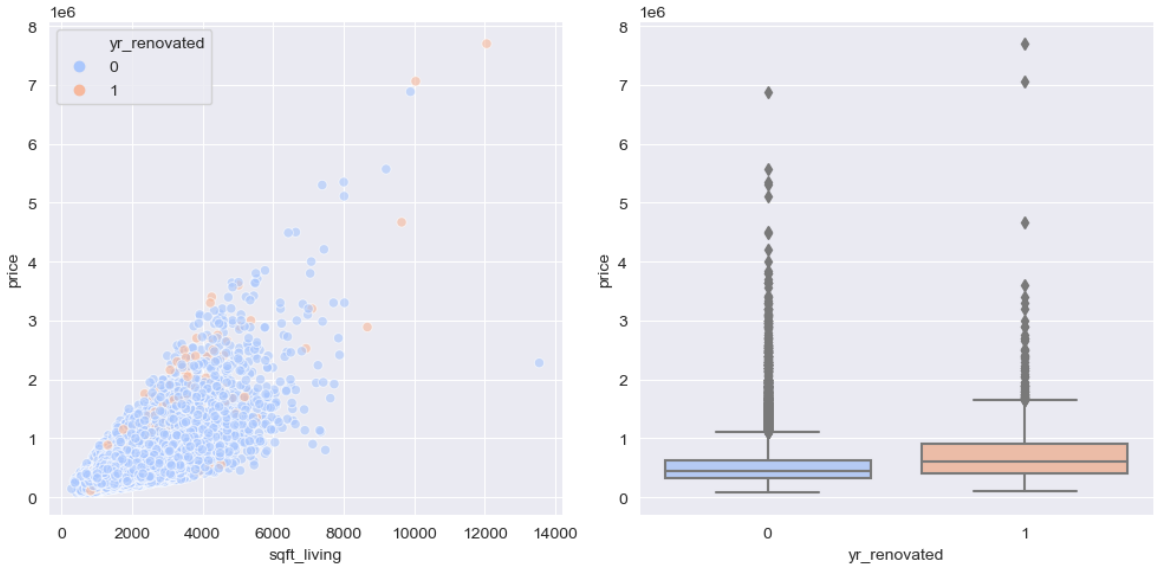
In [28]:
```
# year
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(x=df['yr_built'],y='price',data=df,
                size=0.001,alpha=0.4,color='red',legend=False)
plt.xlabel('The year the house was initially built')
plt.ylabel('Price(Millions)')
```

Out[28]:
```
Text(0, 0.5, 'Price(Millions)')
```



We can clearly see that the house prices in the King's County has remained pretty stable but there was a sudden decrease in the prices of the house just after 1940 somewhere around 1942. It remained stable for a while and is on the rise in 1980.

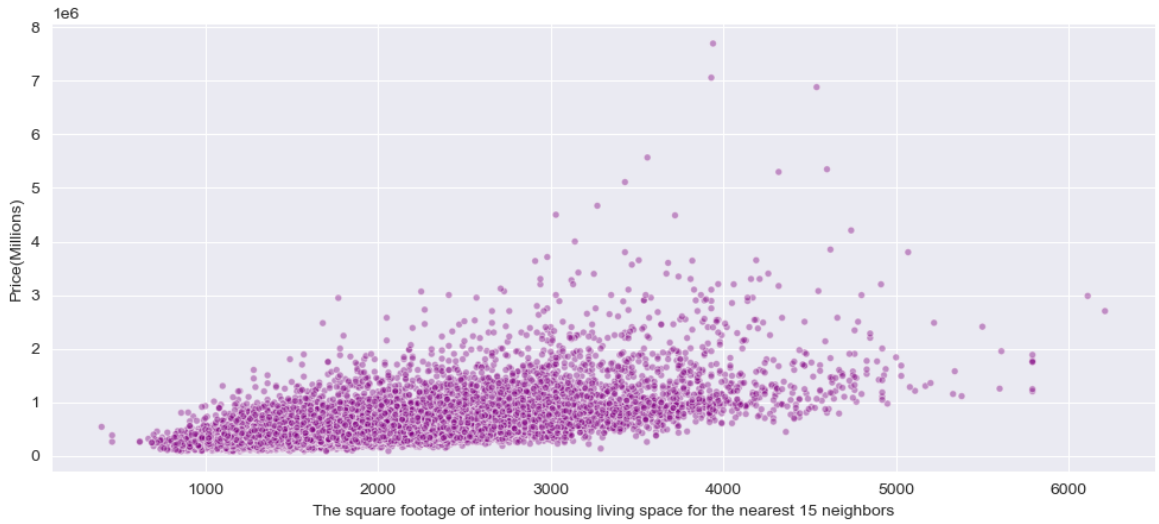
In [29]:
```
fig,axs = plt.subplots(ncols=2,figsize=(10,5),dpi=100)
sns.scatterplot(x='sqft_living', y='price', data=df,hue=df['yr_renovated'],al
sns.boxplot(x='yr_renovated', y='price', data=df, ax=axs[1],palette='coolwarm
plt.tight_layout()
```



From the scatterplot and the boxplot we can see that the houses that have been renovated have higer price than the houses that were never renovated.

In [30]:
```
# sqft_living15
fig = plt.figure(figsize=(12,5),dpi=100)
sns.scatterplot(x=df['sqft_living15'],y='price',data=df,
                size=0.001,alpha=0.4,color='Purple',legend=False)
plt.xlabel('The square footage of interior housing living space for the neare
plt.ylabel('Price(Millions)')
```

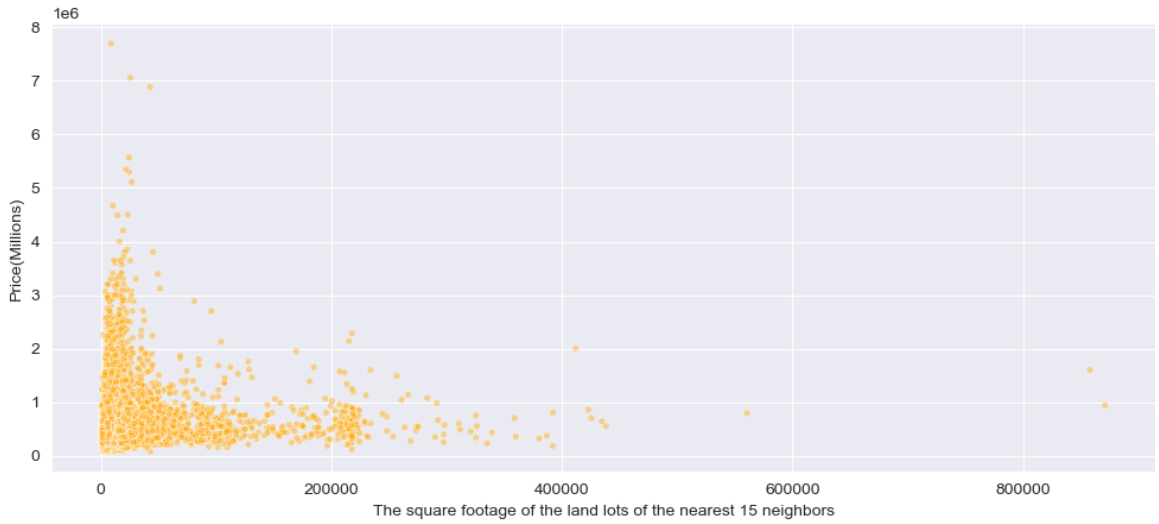
Out[30]: Text(0, 0.5, 'Price(Millions)')



The living area of the 15 neighbourhood houses has a less correlation with the dependent variable price of the house of concern.

In [31]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.scatterplot(x=df['sqft_lot15'],y='price',data=df,
                size=0.001,alpha=0.4,color='Orange',legend=False)
plt.xlabel('The square footage of the land lots of the nearest 15 neighbors')
plt.ylabel('Price(Millions)')
```

Out[31]: Text(0, 0.5, 'Price(Millions)')



The area of the plot of the 15 houses near the chosen house has no correlation with the price of chosen house.

Machine Learning Implementations

In [32]:
```
df.head()
```

Out[32]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0	0	3	
1	538000.0	3	2.25	2570	7242	2.0	0	0	3	
2	180000.0	2	1.00	770	10000	1.0	0	0	3	
3	604000.0	4	3.00	1960	5000	1.0	0	0	5	
4	510000.0	3	2.00	1680	8080	1.0	0	0	3	

In [33]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21613 non-null  float64
 1   bedrooms       21613 non-null  int64  
 2   bathrooms      21613 non-null  float64
 3   sqft_living    21613 non-null  int64  
 4   sqft_lot       21613 non-null  int64  
 5   floors         21613 non-null  float64
 6   waterfront     21613 non-null  int64  
 7   view           21613 non-null  int64  
 8   condition      21613 non-null  int64  
 9   grade          21613 non-null  int64  
 10  sqft_above     21613 non-null  int64  
 11  sqft_basement  21613 non-null  int64  
 12  yr_built       21613 non-null  int64  
 13  yr_renovated   21613 non-null  int64  
 14  zipcode        21613 non-null  int64  
 15  sqft_living15  21613 non-null  int64  
 16  sqft_lot15     21613 non-null  int64  
dtypes: float64(3), int64(14)
memory usage: 2.8 MB
```

In [34]:
```
df.head()
```

Out[34]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0	0	3	
1	538000.0	3	2.25	2570	7242	2.0	0	0	3	
2	180000.0	2	1.00	770	10000	1.0	0	0	3	
3	604000.0	4	3.00	1960	5000	1.0	0	0	5	
4	510000.0	3	2.00	1680	8080	1.0	0	0	3	

In [35]:
```
df.drop('zipcode',axis=1,inplace=True)
```

There are many zipcodes and hence deriving relevant information from it form the regression models is very less hence we decided to drop the zipcodes of the houses.

In [36]:
```
df.head()
```

Out[36]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
0	221900.0	3	1.00	1180	5650	1.0	0	0	3	
1	538000.0	3	2.25	2570	7242	2.0	0	0	3	
2	180000.0	2	1.00	770	10000	1.0	0	0	3	
3	604000.0	4	3.00	1960	5000	1.0	0	0	5	
4	510000.0	3	2.00	1680	8080	1.0	0	0	3	

In [37]:
```
df.columns
```

Out[37]:
```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floor
s',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15',
       'sqft_lot15'],
      dtype='object')
```

In [38]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21613 non-null  float64
 1   bedrooms       21613 non-null  int64  
 2   bathrooms      21613 non-null  float64
 3   sqft_living    21613 non-null  int64  
 4   sqft_lot       21613 non-null  int64  
 5   floors         21613 non-null  float64
 6   waterfront     21613 non-null  int64  
 7   view           21613 non-null  int64  
 8   condition      21613 non-null  int64  
 9   grade          21613 non-null  int64  
 10  sqft_above     21613 non-null  int64  
 11  sqft_basement  21613 non-null  int64  
 12  yr_built       21613 non-null  int64  
 13  yr_renovated   21613 non-null  int64  
 14  sqft_living15  21613 non-null  int64  
 15  sqft_lot15     21613 non-null  int64  
dtypes: float64(3), int64(13)
memory usage: 2.6 MB
```

Setting Up Predictor and Response Variables

Setting the predictor and response variables

In [39]:
```
X = df[['bedrooms',     'bathrooms',   'sqft_living',
            'sqft_lot',        'floors',    'waterfront',          'view',
          'condition',         'grade',    'sqft_above', 'sqft_basement',
           'yr_built',  'yr_renovated', 'sqft_living15',
         'sqft_lot15']]
```

In [40]:
```
X.head()
```

Out[40]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft
0	3	1.00	1180	5650	1.0	0	0	3	7	
1	3	2.25	2570	7242	2.0	0	0	3	7	
2	2	1.00	770	10000	1.0	0	0	3	6	
3	4	3.00	1960	5000	1.0	0	0	5	7	
4	3	2.00	1680	8080	1.0	0	0	3	8	

In [41]:
```
X.shape
```

Out[41]:
```
(21613, 15)
```

In [42]:
```
y = df['price']
```

In [43]:
```
y.head()
```

Out[43]:
```
0    221900.0
1    538000.0
2    180000.0
3    604000.0
4    510000.0
Name: price, dtype: float64
```

Imputing the 10% Null values at random in the dataset

In [44]:
```
# Imputing 10% of the columns with Null Values
for col in X.columns: X.loc[X.sample(frac=0.1).index, col] = pd.np.nan
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureW
arning: The pandas.np module is deprecated and will be removed from pandas
in a future version. Import numpy directly instead

C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:966: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self.obj[item] = s
```

In [45]:
```
X.head(30)
```

Out[45]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	s
0	3.0	1.00	1180.0	5650.0	NaN	0.0	0.0	3.0	7.0	
1	3.0	2.25	2570.0	7242.0	2.0	0.0	0.0	3.0	7.0	
2	2.0	1.00	770.0	10000.0	1.0	0.0	0.0	3.0	6.0	
3	4.0	3.00	1960.0	5000.0	1.0	0.0	0.0	5.0	7.0	
4	3.0	2.00	NaN	8080.0	1.0	0.0	NaN	3.0	8.0	
5	4.0	4.50	NaN	101930.0	1.0	NaN	NaN	3.0	11.0	
6	3.0	2.25	1715.0	6819.0	2.0	0.0	0.0	3.0	7.0	
7	3.0	1.50	1060.0	9711.0	1.0	0.0	0.0	3.0	7.0	
8	3.0	1.00	1780.0	7470.0	1.0	0.0	0.0	3.0	7.0	
9	3.0	2.50	1890.0	6560.0	2.0	0.0	0.0	3.0	7.0	
10	3.0	2.50	3560.0	9796.0	1.0	0.0	0.0	3.0	NaN	
11	2.0	1.00	1160.0	6000.0	1.0	0.0	NaN	4.0	7.0	
12	3.0	1.00	1430.0	19901.0	1.5	0.0	0.0	4.0	7.0	
13	3.0	1.75	1370.0	9680.0	1.0	0.0	0.0	4.0	7.0	
14	NaN	2.00	1810.0	4850.0	1.5	0.0	0.0	3.0	7.0	
15	4.0	3.00	2950.0	5000.0	NaN	0.0	3.0	3.0	9.0	
16	3.0	2.00	1890.0	14040.0	2.0	0.0	0.0	3.0	7.0	
17	NaN	1.00	1600.0	4300.0	NaN	0.0	0.0	4.0	7.0	
18	NaN	1.00	1200.0	9850.0	1.0	NaN	0.0	4.0	7.0	
19	3.0	1.00	1250.0	9774.0	1.0	0.0	0.0	4.0	7.0	
20	4.0	1.75	1620.0	4980.0	1.0	0.0	0.0	4.0	7.0	
21	3.0	2.75	3050.0	44867.0	1.0	NaN	4.0	3.0	9.0	
22	5.0	NaN	NaN	6300.0	2.0	0.0	NaN	3.0	8.0	
23	2.0	1.50	1070.0	9643.0	1.0	NaN	0.0	3.0	7.0	
24	3.0	2.25	2450.0	6500.0	2.0	0.0	0.0	4.0	8.0	
25	3.0	NaN	1710.0	4697.0	1.5	0.0	0.0	5.0	6.0	
26	3.0	1.75	2450.0	2691.0	2.0	0.0	0.0	3.0	8.0	
27	3.0	1.00	1400.0	1581.0	1.5	0.0	0.0	5.0	8.0	
28	3.0	1.75	1520.0	6380.0	1.0	0.0	0.0	3.0	7.0	
29	4.0	2.50	2570.0	7173.0	2.0	NaN	0.0	3.0	8.0	

In [46]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       19452 non-null  float64
 1   bathrooms      19452 non-null  float64
 2   sqft_living    19452 non-null  float64
 3   sqft_lot       19452 non-null  float64
 4   floors         19452 non-null  float64
 5   waterfront     19452 non-null  float64
 6   view           19452 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     19452 non-null  float64
 10  sqft_basement  19452 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  19452 non-null  float64
 14  sqft_lot15     19452 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

In [47]:
```
X.isna().sum()
```

Out[47]:
```
bedrooms         2161
bathrooms        2161
sqft_living      2161
sqft_lot         2161
floors           2161
waterfront       2161
view             2161
condition        2161
grade            2161
sqft_above       2161
sqft_basement    2161
yr_built         2161
yr_renovated     2161
sqft_living15    2161
sqft_lot15       2161
dtype: int64
```

We have Successfully Made 10% of all columns Randomly as NaN

Imputing The NaN Values

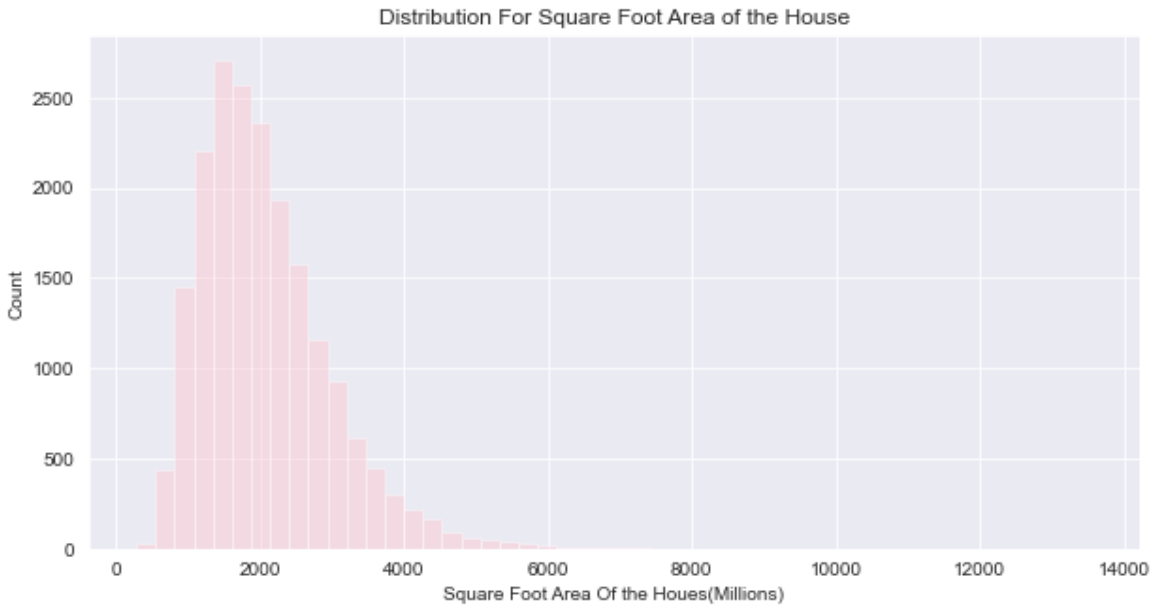
Imputation Of Quantitative Variables

sqft_living

sqft_living

In [48]:
```
fig=plt.figure(figsize=(10,5))
sns.distplot(X['sqft_living'],color='pink',kde=False)
plt.title('Distribution For Square Foot Area of the House')
plt.ylabel('Count')
plt.xlabel('Square Foot Area Of the Houes(Millions)')
```

Out[48]:
```
Text(0.5, 0, 'Square Foot Area Of the Houes(Millions)')
```



The Distribution Of the Variable suggests that the variable is positively skewed

In [49]:
```
X.groupby('grade')['sqft_living'].mean()
```

Out[49]:
```
grade
1.0      290.000000
3.0      596.666667
4.0      661.478261
5.0      971.735751
6.0     1190.512867
7.0     1685.845294
8.0     2181.810443
9.0     2865.151544
10.0    3531.699566
11.0    4392.553191
12.0    5491.527027
13.0    7483.076923
Name: sqft_living, dtype: float64
```

In [50]:
```
X['sqft_living'] = X['sqft_living'].fillna(X.groupby('grade')['sqft_living'].
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [51]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   bedrooms       19452 non-null  float64
 1   bathrooms      19452 non-null  float64
 2   sqft_living    21402 non-null  float64
 3   sqft_lot       19452 non-null  float64
 4   floors         19452 non-null  float64
 5   waterfront     19452 non-null  float64
 6   view           19452 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     19452 non-null  float64
 10  sqft_basement  19452 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  19452 non-null  float64
 14  sqft_lot15     19452 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

In [52]:
```
X.head(30)
```

Out[52]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
0	3.0	1.00	1180.000000	5650.0	NaN	0.0	0.0	3.0	7.0
1	3.0	2.25	2570.000000	7242.0	2.0	0.0	0.0	3.0	7.0
2	2.0	1.00	770.000000	10000.0	1.0	0.0	0.0	3.0	6.0
3	4.0	3.00	1960.000000	5000.0	1.0	0.0	0.0	5.0	7.0
4	3.0	2.00	2181.810443	8080.0	1.0	0.0	NaN	3.0	8.0
5	4.0	4.50	4392.553191	101930.0	1.0	NaN	NaN	3.0	11.0
6	3.0	2.25	1715.000000	6819.0	2.0	0.0	0.0	3.0	7.0
7	3.0	1.50	1060.000000	9711.0	1.0	0.0	0.0	3.0	7.0
8	3.0	1.00	1780.000000	7470.0	1.0	0.0	0.0	3.0	7.0
9	3.0	2.50	1890.000000	6560.0	2.0	0.0	0.0	3.0	7.0
10	3.0	2.50	3560.000000	9796.0	1.0	0.0	0.0	3.0	NaN
11	2.0	1.00	1160.000000	6000.0	1.0	0.0	NaN	4.0	7.0
12	3.0	1.00	1430.000000	19901.0	1.5	0.0	0.0	4.0	7.0
13	3.0	1.75	1370.000000	9680.0	1.0	0.0	0.0	4.0	7.0
14	NaN	2.00	1810.000000	4850.0	1.5	0.0	0.0	3.0	7.0
15	4.0	3.00	2950.000000	5000.0	NaN	0.0	3.0	3.0	9.0
16	3.0	2.00	1890.000000	14040.0	2.0	0.0	0.0	3.0	7.0
17	NaN	1.00	1600.000000	4300.0	NaN	0.0	0.0	4.0	7.0
18	NaN	1.00	1200.000000	9850.0	1.0	NaN	0.0	4.0	7.0
19	3.0	1.00	1250.000000	9774.0	1.0	0.0	0.0	4.0	7.0
20	4.0	1.75	1620.000000	4980.0	1.0	0.0	0.0	4.0	7.0
21	3.0	2.75	3050.000000	44867.0	1.0	NaN	4.0	3.0	9.0
22	5.0	NaN	2181.810443	6300.0	2.0	0.0	NaN	3.0	8.0
23	2.0	1.50	1070.000000	9643.0	1.0	NaN	0.0	3.0	7.0
24	3.0	2.25	2450.000000	6500.0	2.0	0.0	0.0	4.0	8.0
25	3.0	NaN	1710.000000	4697.0	1.5	0.0	0.0	5.0	6.0
26	3.0	1.75	2450.000000	2691.0	2.0	0.0	0.0	3.0	8.0
27	3.0	1.00	1400.000000	1581.0	1.5	0.0	0.0	5.0	8.0
28	3.0	1.75	1520.000000	6380.0	1.0	0.0	0.0	3.0	7.0
29	4.0	2.50	2570.000000	7173.0	2.0	NaN	0.0	3.0	8.0

In [53]:
```
X['sqft_living'].isna().sum()
```

Out[53]:
```
211
```

202 Have still not been imputed as the values for grades in the corresponding columns were also NaN

In [54]:
```
X.groupby('bathrooms')['sqft_living'].mean()
```

Out[54]:
```
bathrooms
0.00     1636.800000
0.50     1312.628217
0.75      947.121324
1.00     1207.162588
1.25     1864.093570
1.50     1558.016005
1.75     1795.356620
2.00     1792.896258
2.25     2093.810654
2.50     2373.770465
2.75     2627.233947
3.00     2709.916817
3.25     3175.179011
3.50     3361.181251
3.75     3779.179064
4.00     4100.745063
4.25     4388.413215
4.50     4245.818483
4.75     5228.333333
5.00     4843.277778
5.25     5006.879433
5.50     6506.000000
5.75     6876.666667
6.00     6443.333333
6.25     8345.000000
6.50     6765.000000
6.75     8560.000000
7.50     4050.000000
7.75     9890.000000
8.00    12795.000000
Name: sqft_living, dtype: float64
```

In [55]:
```
X['sqft_living'] = X['sqft_living'].fillna(X.groupby('bathrooms')['sqft_livin
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [56]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       19452 non-null  float64
 1   bathrooms      19452 non-null  float64
 2   sqft_living    21594 non-null  float64
 3   sqft_lot       19452 non-null  float64
 4   floors         19452 non-null  float64
 5   waterfront     19452 non-null  float64
 6   view           19452 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     19452 non-null  float64
 10  sqft_basement  19452 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  19452 non-null  float64
 14  sqft_lot15     19452 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

In [57]:
```
X['sqft_living'].isna().sum()
```

Out[57]:
```
19
```

26 values are still missing

In [58]:
```
X.groupby('bedrooms')['sqft_living'].mean()
```

Out[58]:
```
bedrooms
0.0     1977.547826
1.0      918.674468
2.0     1276.184642
3.0     1818.391647
4.0     2532.397204
5.0     3019.008373
6.0     3230.147062
7.0     3590.447076
8.0     3699.622299
9.0     3966.000000
10.0    3706.666667
11.0    3000.000000
33.0    1620.000000
Name: sqft_living, dtype: float64
```

In [59]:
```
X['sqft_living'] = X['sqft_living'].fillna(X.groupby('bedrooms')['sqft_living
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [60]:
```
X['sqft_living'].isna().sum()
```

Out[60]: 1

In [61]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   bedrooms       19452 non-null  float64
 1   bathrooms      19452 non-null  float64
 2   sqft_living    21612 non-null  float64
 3   sqft_lot       19452 non-null  float64
 4   floors         19452 non-null  float64
 5   waterfront     19452 non-null  float64
 6   view           19452 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     19452 non-null  float64
 10  sqft_basement  19452 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  19452 non-null  float64
 14  sqft_lot15     19452 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

1 values are still missing

In [62]:
```
X.groupby('floors')['sqft_living'].mean()
```

Out[62]:
```
floors
1.0    1736.972444
1.5    1896.007811
2.0    2565.394594
2.5    3036.384817
3.0    1826.261339
3.5    2518.000000
Name: sqft_living, dtype: float64
```

In [63]:
```
X['sqft_living'] = X['sqft_living'].fillna(X.groupby('floors')['sqft_living']
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [64]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   bedrooms       19452 non-null  float64
 1   bathrooms      19452 non-null  float64
 2   sqft_living    21612 non-null  float64
 3   sqft_lot       19452 non-null  float64
 4   floors         19452 non-null  float64
 5   waterfront     19452 non-null  float64
 6   view           19452 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     19452 non-null  float64
 10  sqft_basement  19452 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  19452 non-null  float64
 14  sqft_lot15     19452 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

In [65]:
```
X['sqft_living'].isna().sum()
```

Out[65]:
```
1
```

As the distribution of the sqft_living is positively skewed and not normally distributed hence it would

be appropriate to impute the 1 nan that is remaining with the median of the column

In [66]:
```
X['sqft_living'].median()
```

Out[66]:
```
1909.5
```

In [67]:
```
X.loc[(X['sqft_living'].isna()),'sqft_living']=  X['sqft_living'].median()
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [68]:
```
X.loc[(X['sqft_living'].isna()),'sqft_living']= X['sqft_living'].median()
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [69]:
```
X['sqft_living'].isna().sum()
```

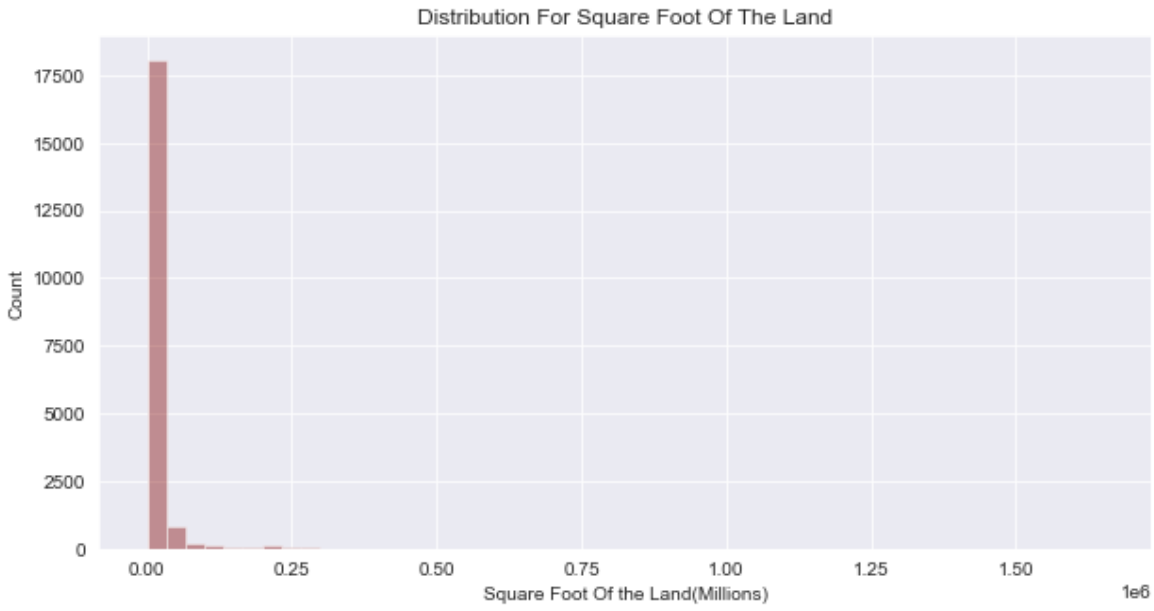
Out[69]:
```
0
```

The column was imputed by the grouped mean of the variables to with which it had very high correlation and once it was done by that and still the null values prevailed then it was done by the median of the column as the distribution of the column was positively skewed and not normally distributed

sqft_lot

In [70]:
```
fig=plt.figure(figsize=(10,5))
sns.distplot(X['sqft_lot'],color='maroon',bins=50,kde=False)
plt.title('Distribution For Square Foot Of The Land')
plt.ylabel('Count')
plt.xlabel('Square Foot Of the Land(Millions)')
```

Out[70]:
```
Text(0.5, 0, 'Square Foot Of the Land(Millions)')
```



The Distribution is positively skewed and has correlations with none of the other variables hence we are imputing it with the median of the column

In [71]:
```
X['sqft_lot'].median()
```

Out[71]:
```
7605.0
```

In [72]:
```
X.loc[(X['sqft_lot'].isna()),'sqft_lot']= X['sqft_lot'].median()
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [73]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       19452 non-null  float64
 1   bathrooms      19452 non-null  float64
 2   sqft_living    21613 non-null  float64
 3   sqft_lot       21613 non-null  float64
 4   floors         19452 non-null  float64
 5   waterfront     19452 non-null  float64
 6   view           19452 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     19452 non-null  float64
 10  sqft_basement  19452 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  19452 non-null  float64
 14  sqft_lot15     19452 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

In [74]:
```
X['sqft_lot'].isna().sum()
```

Out[74]: 0

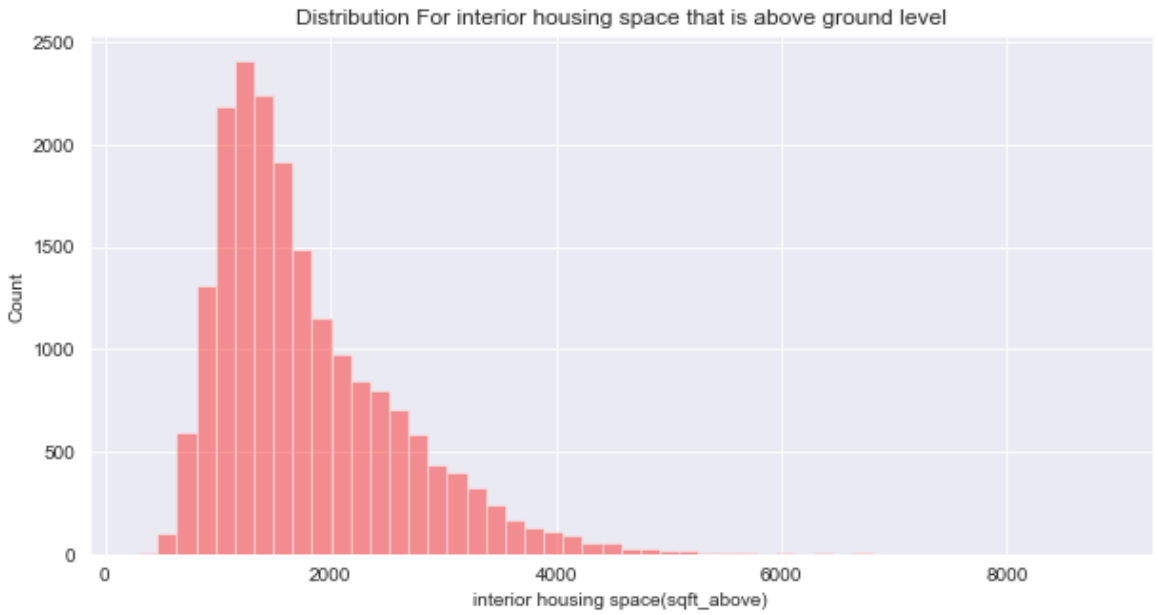
The Null Values were imputing was done by the median of the column as the distribution of the column was positively skewed and not normally distributed

The Variable sqft_lot has been successfully imputed

sqft_above

In [75]:
```
fig=plt.figure(figsize=(10,5))
sns.distplot(X['sqft_above'],color='red',kde=False)
plt.title('Distribution For interior housing space that is above ground level
plt.ylabel('Count')
plt.xlabel('interior housing space(sqft_above)')
```

Out[75]: Text(0.5, 0, 'interior housing space(sqft_above)')



In [76]:
```
X.groupby('grade')['sqft_above'].mean()
```

Out[76]:
```
grade
1.0      290.000000
3.0      596.666667
4.0      677.000000
5.0      927.596774
6.0     1066.188494
7.0     1409.519598
8.0     1869.188397
9.0     2553.294592
10.0    3107.479826
11.0    3855.443038
12.0    4518.381579
13.0    6060.769231
Name: sqft_above, dtype: float64
```

In [77]:
```
X.head()
```

Out[77]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sq
0	3.0	1.00	1180.000000	5650.0	NaN	0.0	0.0	3.0	7.0	
1	3.0	2.25	2570.000000	7242.0	2.0	0.0	0.0	3.0	7.0	
2	2.0	1.00	770.000000	10000.0	1.0	0.0	0.0	3.0	6.0	
3	4.0	3.00	1960.000000	5000.0	1.0	0.0	0.0	5.0	7.0	
4	3.0	2.00	2181.810443	8080.0	1.0	0.0	NaN	3.0	8.0	

In [78]:
```
X['sqft_above'] = X['sqft_above'].fillna(X.groupby('grade')['sqft_above'].tra
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [79]:
```
X['sqft_above'].isna().sum()
```

Out[79]:
```
203
```

212 More Null values remain that have to be taken care of.

In [80]:
```
X.groupby('bathrooms')['sqft_above'].mean()
```

Out[80]:
```
bathrooms
0.00    1636.800000
0.50     992.500000
0.75     876.676499
1.00    1111.666314
1.25    1248.888889
1.50    1358.092395
1.75    1425.274999
2.00    1491.422181
2.25    1752.698749
2.50    2166.307228
2.75    2149.058227
3.00    2227.018154
3.25    2661.374071
3.50    2793.843421
3.75    3193.777762
4.00    3413.659729
4.25    3607.078488
4.50    3652.186371
4.75    4265.259192
5.00    3825.524581
5.25    4120.833333
5.50    5146.544304
5.75    5520.000000
6.00    4988.333333
6.25    7070.000000
6.50    5630.000000
6.75    4950.000000
7.50    4050.000000
7.75    8860.000000
8.00    6544.190789
Name: sqft_above, dtype: float64
```

In [81]:
```
X['sqft_above'] = X['sqft_above'].fillna(X.groupby('bathrooms')['sqft_above']
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [82]:
```
X['sqft_above'].isna().sum()
```

Out[82]:
```
18
```

17 More Null values remain that have to be taken care of.

In [83]:
```
X.groupby('floors')['sqft_above'].mean()
```

Out[83]:
```
floors
1.0    1349.682730
1.5    1633.261261
2.0    2378.952368
2.5    2692.549815
3.0    1747.873513
3.5    2315.500000
Name: sqft_above, dtype: float64
```

In [84]:
```
X['sqft_above'] = X['sqft_above'].fillna(X.groupby('floors')['sqft_above'].tr
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [85]:
```
X['sqft_above'].isna().sum()
```

Out[85]:
```
5
```

In [86]:
```
X.groupby('bedrooms')['sqft_above'].mean()
```

Out[86]:
```
bedrooms
0.0     1950.400000
1.0      883.008995
2.0     1134.150416
3.0     1594.950442
4.0     2178.384519
5.0     2386.179664
6.0     2457.189196
7.0     2989.557725
8.0     2835.454545
9.0     3146.000000
10.0    2456.666667
11.0    2400.000000
33.0    1040.000000
Name: sqft_above, dtype: float64
```

In [87]:
```
X['sqft_above'] = X['sqft_above'].fillna(X.groupby('bedrooms')['sqft_above'].
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [88]:
```
X.loc[(X['sqft_above'].isna()),'sqft_above']= X['sqft_above'].median()
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [89]:
```
X['sqft_above'].isna().sum()
```

Out[89]: 0

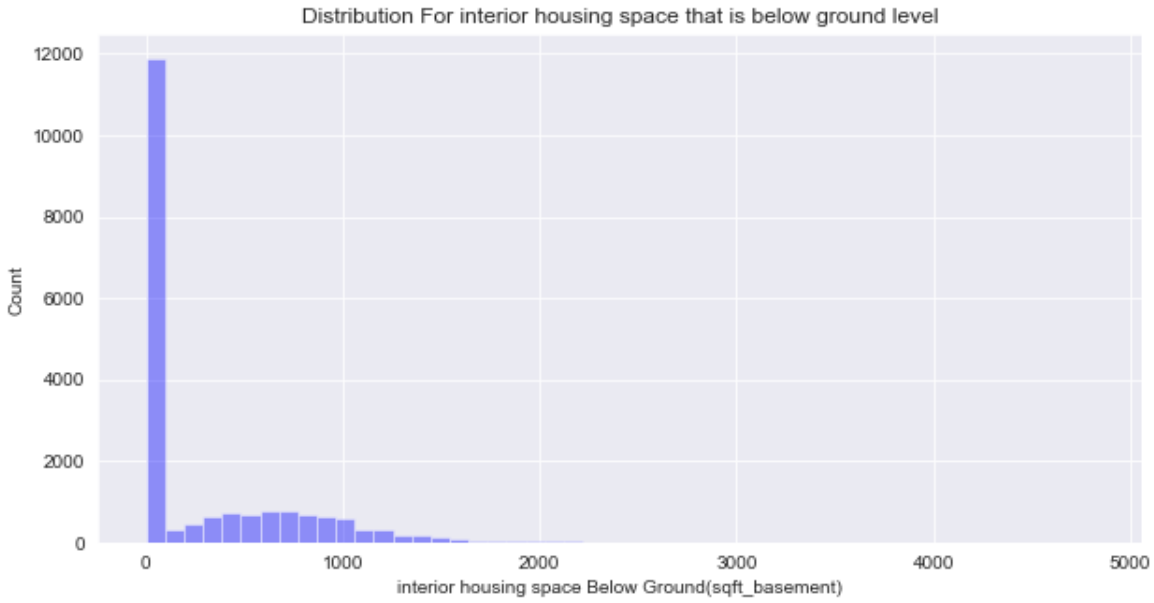
The column was imputed by the grouped mean of the variables to with which it had very high correlation and once it was done by that and still the null values prevailed then it was done by the median of the column as the distribution of the column was positively skewed and not normally distributed

The Variable sqft_above has been successfully imputed

sqft_basement

In [90]:
```
fig=plt.figure(figsize=(10,5))
sns.distplot(X['sqft_basement'],color='blue',kde=False)
plt.title('Distribution For interior housing space that is below ground level
plt.ylabel('Count')
plt.xlabel('interior housing space Below Ground(sqft_basement)')
```

Out[90]: Text(0.5, 0, 'interior housing space Below Ground(sqft_basement)')



In [91]:
```
X.groupby('bedrooms')['sqft_basement'].mean()
```

Out[91]:
```
bedrooms
0.0        0.000000
1.0       65.568182
2.0      133.283662
3.0      221.607515
4.0      358.453990
5.0      637.324615
6.0      786.728111
7.0      674.000000
8.0     1106.000000
9.0      820.000000
10.0    1250.000000
11.0     600.000000
33.0     580.000000
Name: sqft_basement, dtype: float64
```

In [92]:
```
X['sqft_basement'] = X['sqft_basement'].fillna(X.groupby('bedrooms')['sqft_ba
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [93]:
```
X['sqft_basement'].isna().sum()
```

Out[93]:
```
210
```

221 More Null values remain that have to be taken care of.

In [94]:
```
X.groupby('view')['sqft_basement'].mean()
```

Out[94]:
```
view
0.0    253.418062
1.0    579.053658
2.0    536.668960
3.0    689.788565
4.0    814.442885
Name: sqft_basement, dtype: float64
```

In [95]:
```
X['sqft_basement'] = X['sqft_basement'].fillna(X.groupby('view')['sqft_baseme
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [96]:
```
X['sqft_basement'].isna().sum()
```

Out[96]:
```
19
```

19 More Null values remain that have to be taken care of.

In [97]:
```
X.groupby('bathrooms')['sqft_basement'].mean()
```

Out[97]:
```
bathrooms
0.00       0.000000
0.50     242.500000
0.75      75.015336
1.00      97.006545
1.25     186.845279
1.50     203.150225
1.75     370.612068
2.00     303.576028
2.25     340.159274
2.50     207.675463
2.75     493.253010
3.00     506.706368
3.25     506.122038
3.50     578.478376
3.75     536.174448
4.00     629.954998
4.25     804.259536
4.50     601.348204
4.75     937.389624
5.00     853.397553
5.25     741.937449
5.50    1092.000000
5.75    1356.666667
6.00    1455.000000
6.25    1275.000000
6.50    1135.000000
6.75    2747.000000
7.50       0.000000
7.75    1030.000000
8.00    3805.000000
Name: sqft_basement, dtype: float64
```

In [98]:
```
X['sqft_basement'] = X['sqft_basement'].fillna(X.groupby('bathrooms')['sqft_b
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [99]:
```
X['sqft_basement'].isna().sum()
```

Out[99]:
```
1
```

In [100]:
```
X['sqft_basement'] = X['sqft_basement'].fillna(X.groupby('grade')['sqft_basem
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [101]:
```
X.loc[(X['sqft_basement'].isna()),'sqft_basement']= X['sqft_basement'].median
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [102]:
```
X['sqft_basement'].isna().sum()
```

Out[102]: 0

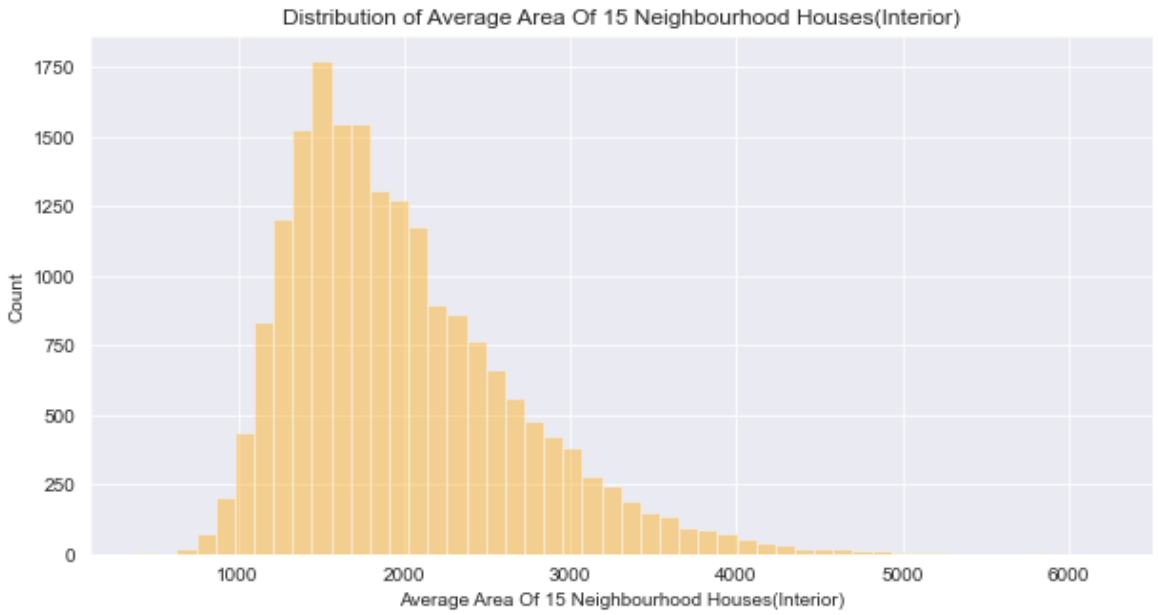
The column was imputed by the grouped mean of the variables to with which it had very high correlation and once it was done by that and still the null values prevailed then it was done by the median of the column as the distribution of the column was positively skewed and not normally distributed

The Variable sqft_basement has been successfully imputed

sqft_living15

In [103]:
```
fig = plt.figure(figsize=(10,5))
sns.distplot(X['sqft_living15'],color='orange',kde=False)
plt.title('Distribution of Average Area Of 15 Neighbourhood Houses(Interior)'
plt.xlabel('Average Area Of 15 Neighbourhood Houses(Interior)')
plt.ylabel('Count')
```

Out[103]:
```
Text(0, 0.5, 'Count')
```



In [104]:
```
X.groupby('grade')['sqft_living15'].mean()
```

Out[104]:
```
grade
1.0             NaN
3.0     1250.000000
4.0     1501.153846
5.0     1375.712042
6.0     1386.870948
7.0     1677.116420
8.0     2065.355237
9.0     2618.858016
10.0    3034.796926
11.0    3521.765432
12.0    4026.833333
13.0    4110.833333
Name: sqft_living15, dtype: float64
```

In [105]:
```
X['sqft_living15'] = X['sqft_living15'].fillna(X.groupby('grade')['sqft_livin
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [106]:
```
X['sqft_living15'].isna().sum()
```

Out[106]:
```
234
```

229 More Null values remain that have to be taken care of.

In [107]:
```
X.groupby('bathrooms')['sqft_living15'].mean()
```

Out[107]:
```
bathrooms
0.00    1976.941295
0.50    1606.338809
0.75    1480.400682
1.00    1460.153772
1.25    1659.359205
1.50    1624.962599
1.75    1797.055005
2.00    1732.345388
2.25    2023.189024
2.50    2256.418952
2.75    2339.514296
3.00    2245.302805
3.25    2650.143730
3.50    2676.147765
3.75    2835.443384
4.00    3163.178129
4.25    3171.956987
4.50    3006.473333
4.75    3411.468254
5.00    3283.726779
5.25    3073.750000
5.50    3662.000000
5.75    2910.000000
6.00    3691.666667
6.25    4150.000000
6.50    2255.000000
6.75    3040.000000
7.50    1448.000000
7.75    4540.000000
8.00    4395.000000
Name: sqft_living15, dtype: float64
```

In [108]:
```
X['sqft_living15'] = X['sqft_living15'].fillna(X.groupby('bathrooms')['sqft_l
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [109]:
```
X['sqft_living15'].isna().sum()
```

Out[109]:
```
24
```

18 More Null values remain that have to be taken care of.

In [110]:
```
X.groupby('bedrooms')['sqft_living15'].mean()
```

Out[110]:
```
bedrooms
0.0     2222.941295
1.0     1462.413856
2.0     1523.364574
3.0     1841.953194
4.0     2276.888666
5.0     2433.183281
6.0     2286.702955
7.0     2271.667779
8.0     2291.832015
9.0     2017.600000
10.0    2193.333333
11.0    1420.000000
33.0    1330.000000
Name: sqft_living15, dtype: float64
```

In [111]:
```
X['sqft_living15'] = X['sqft_living15'].fillna(X.groupby('bedrooms')['sqft_li
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [112]:
```
X['sqft_living15'].isna().sum()
```

Out[112]:
```
5
```

In [113]:
```
X.loc[(X['sqft_living15'].isna()),'sqft_living15']= X['sqft_living15'].median
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [114]:
```
X['sqft_living15'].isna().sum()
```

Out[114]:
```
0
```

The column was imputed by the grouped mean of the variables to with which it had very high correlation and once it was done by that and still the null values prevailed then it was done by the

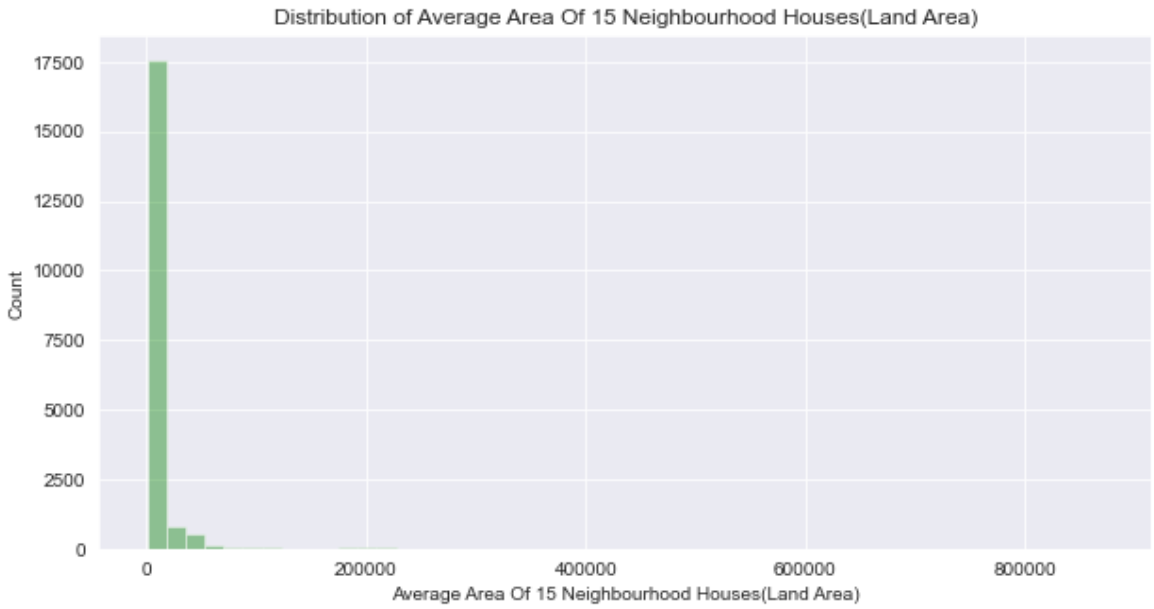
median of the column as the distribution of the column was positively skewed and not normally distributed

The Variable sqft_living15 has been successfully imputed

sqft_lot15

In [115]:
```
fig = plt.figure(figsize=(10,5))
sns.distplot(X['sqft_lot15'],color='green',kde=False)
plt.title('Distribution of Average Area Of 15 Neighbourhood Houses(Land Area)')
plt.xlabel('Average Area Of 15 Neighbourhood Houses(Land Area)')
plt.ylabel('Count')
```

Out[115]:
```
Text(0, 0.5, 'Count')
```



The Distribution is positively skewed and has correlations with none of the other variables hence we are imputing it with the median of the column

In [116]:
```
X['sqft_lot15'].median()
```

Out[116]:
```
7615.0
```

In [117]:
```
X.loc[(X['sqft_lot15'].isna()),'sqft_lot15']= X['sqft_lot15'].median()
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [118]:
```
X['sqft_lot15'].isna().sum()
```

Out[118]:
```
0
```

The Variable sqft_lot15 has been successfully imputed

In [119]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       19452 non-null  float64
 1   bathrooms      19452 non-null  float64
 2   sqft_living    21613 non-null  float64
 3   sqft_lot       21613 non-null  float64
 4   floors         19452 non-null  float64
 5   waterfront     19452 non-null  float64
 6   view           19452 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     21613 non-null  float64
 10  sqft_basement  21613 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  21613 non-null  float64
 14  sqft_lot15     21613 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

The Null Values were imputing was done by the median of the column as the distribution of the column was positively skewed and not normally distributed

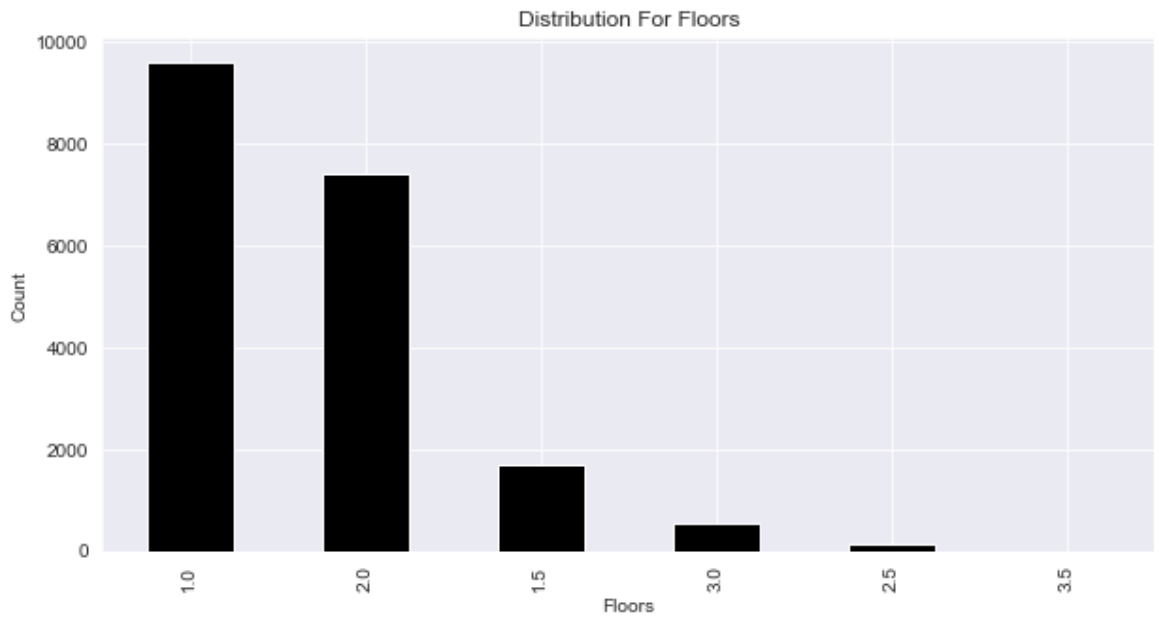
Imputing Categorical Variables

Floors

The number of floors a house will have is very highly correlated with the mean Area that the house occupies above the land hence we will use the sqft_above variable to impute the floors variable

In [120]:
```
fig=plt.figure(figsize=(10,5))
X['floors'].value_counts().plot.bar(color='black')
plt.title('Distribution For Floors')
plt.ylabel('Count')
plt.xlabel('Floors')
```

Out[120]: Text(0.5, 0, 'Floors')



In [121]:
```
X.groupby('floors')['sqft_above'].mean()
```

Out[121]:
```
floors
1.0    1349.682730
1.5    1633.261261
2.0    2378.952368
2.5    2692.549815
3.0    1747.873513
3.5    2315.500000
Name: sqft_above, dtype: float64
```

In [122]:
```
df['floors'].value_counts(normalize=True)
```

Out[122]:
```
1.0    0.494147
2.0    0.381298
1.5    0.088373
3.0    0.028363
2.5    0.007449
3.5    0.000370
Name: floors, dtype: float64
```

Almost 90% of the houses have floors 1 or 2 hence it makes sense to impute the null values in the floor column with either 1 or 2 based on a cutoff

In [123]:
```
# Cutoff
```

In [124]:
```
((2368.027330 - 1349.777303)/2) + (1349.777303)
```

Out[124]:
```
1858.9023164999999
```

In [125]:
```
X.loc[((X['floors'].isna()) & ((X['sqft_above'])<= 1858.9023164999999)),'floo
X.loc[((X['floors'].isna()) & ((X['sqft_above']) > 1858.9023164999999)),'floo
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
```

Floors 3 and 3.5 breaks the general trend and we see that there are way many houses with 3 floors than 3.5 floors hence we want to impute the rest of the values with 3 floors as there are only 8 houses out of 21,613 houses that have 3.5 floors

In [126]:
```
X['floors'].isna().sum()
```

Out[126]:
```
0
```

The Variable floors has been successfully imputed

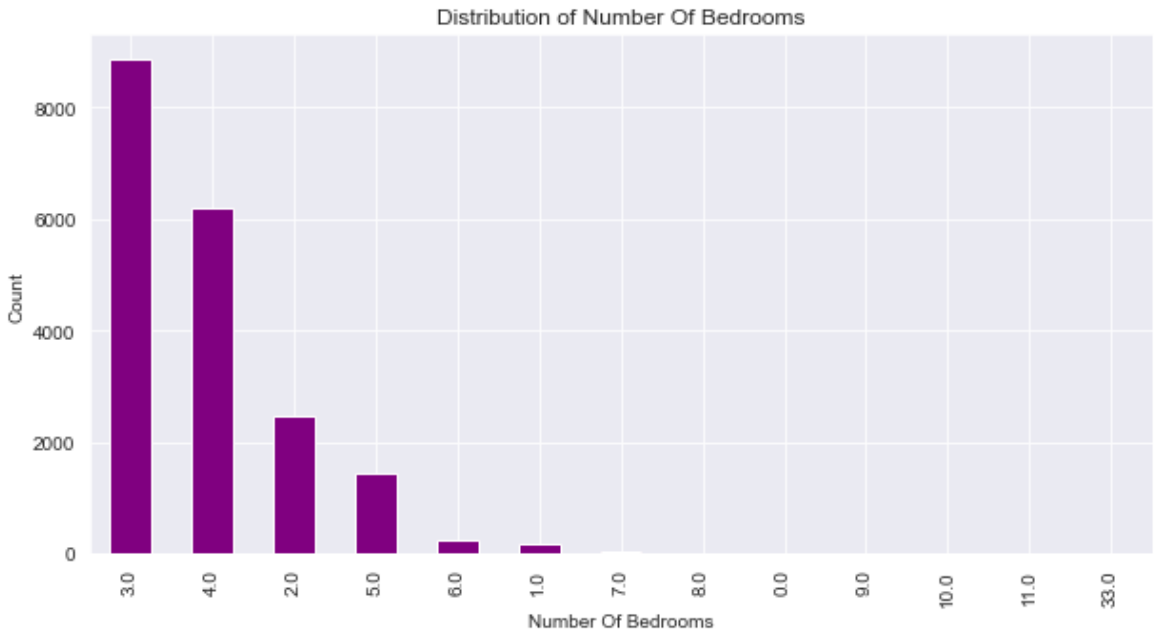
Bedrooms

There is a mistake while recording the data as we can see that the number of bedrooms in 1 of the

record is 33 and by the area and other variables it looks like its 3

In [127]:
```
fig=plt.figure(figsize=(10,5))
X['bedrooms'].value_counts().plot.bar(color='purple')
plt.title('Distribution of Number Of Bedrooms')
plt.ylabel('Count')
plt.xlabel('Number Of Bedrooms')
```

Out[127]:
```
Text(0.5, 0, 'Number Of Bedrooms')
```



In [128]:
```
X.loc[(X['bedrooms'] == 33),'bedrooms']= 3.0
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [129]:
```
X.groupby('bedrooms')['sqft_living'].mean()
```

Out[129]:
```
bedrooms
0.0     1977.547826
1.0      918.674468
2.0     1276.184642
3.0     1818.369242
4.0     2532.397204
5.0     3019.008373
6.0     3230.147062
7.0     3590.447076
8.0     3699.622299
9.0     3966.000000
10.0    3706.666667
11.0    3000.000000
Name: sqft_living, dtype: float64
```

In [130]:
```
X['bedrooms'].value_counts(normalize=True)
```

Out[130]:
```
3.0     0.455223
4.0     0.318065
2.0     0.127133
5.0     0.073874
6.0     0.012801
1.0     0.009613
7.0     0.001748
8.0     0.000565
0.0     0.000514
9.0     0.000257
10.0    0.000154
11.0    0.000051
Name: bedrooms, dtype: float64
```

It can be seen that correlation between the square foot area of the house and that of the number of bedrooms are very highly correlated hence we can make use of the square foot column to impute the bedrooms column

We can see that almost 95% of the houses have bedrooms such as 2, 3 , 4, 5, 6, 7. Hence we will use the ranges of mean square foot values of all these bedrooms to impute the bedrooms column

In [131]:
```
X.loc[((X['bedrooms'].isna()) & ((X['sqft_living'])<= 1278.056771)),'bedrooms
X.loc[((X['bedrooms'].isna()) & (((((X['sqft_living'])<= 1816.872496)) & (X['
X.loc[((X['bedrooms'].isna()) & (((((X['sqft_living'])<= 2533.530047)) & (X['
X.loc[((X['bedrooms'].isna()) & (((((X['sqft_living'])<= 3000.203482)) & (X['
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)

C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  This is separate from the ipykernel package so we can avoid doing imports
until
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  after removing the cwd from sys.path.
```

In [132]:
```
X['bedrooms'].isna().sum()
```

Out[132]:
```
273
```

There are still 304 nulls and as the bedrooms 3 and 4 are in almost 80% of the houses hence the house is either going have 3 or 4 bedrooms according to the following cutoff

In [133]:
```
# Cuttoff
```

In [134]:
```
((2533.530047-1816.872496)/2) + (1816.872496)
```

Out[134]: 2175.2012715

In [135]:
```
X.loc[((X['bedrooms'].isna()) & ((X['sqft_living'])<= 2175.2012715)),'bedroom
X.loc[((X['bedrooms'].isna()) & ((X['sqft_living']) > 2175.2012715)),'bedroom
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
```

In [136]:
```
X['bedrooms'].isna().sum()
```

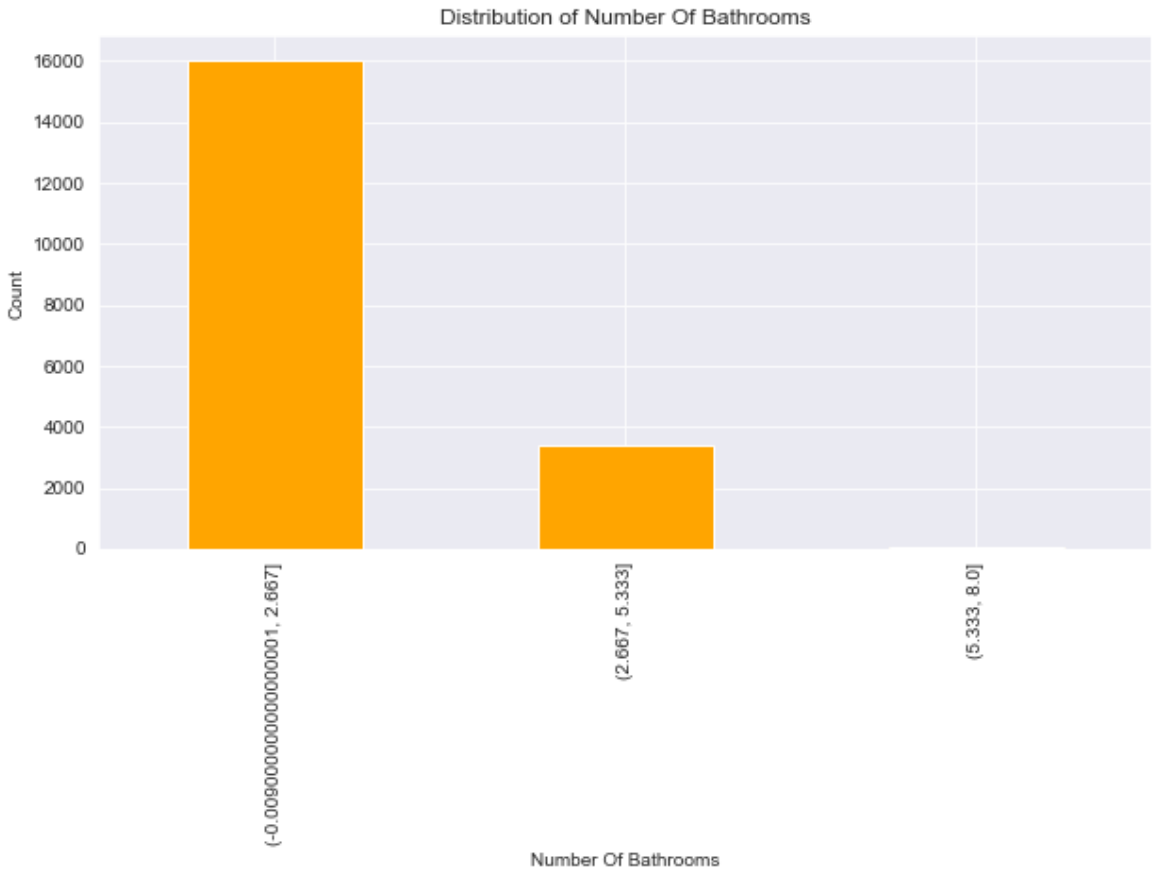
Out[136]: 0

The Variable bedrooms has been successfully imputed

Bathrooms

In [137]:
```
fig=plt.figure(figsize=(10,5))
X['bathrooms'].value_counts(bins=3).plot.bar(color='orange')
plt.title('Distribution of Number Of Bathrooms')
plt.ylabel('Count')
plt.xlabel('Number Of Bathrooms')
```

Out[137]: Text(0.5, 0, 'Number Of Bathrooms')



In [138]:
```
X.groupby('bathrooms')['sqft_living'].mean()
```

Out[138]:
```
bathrooms
0.00     1636.800000
0.50     1312.628217
0.75      947.121324
1.00     1207.162588
1.25     1864.093570
1.50     1558.016005
1.75     1795.356620
2.00     1792.896258
2.25     2093.810654
2.50     2373.770465
2.75     2627.233947
3.00     2709.916817
3.25     3175.179011
3.50     3361.181251
3.75     3779.179064
4.00     4100.745063
4.25     4388.413215
4.50     4245.818483
4.75     5228.333333
5.00     4843.277778
5.25     5006.879433
5.50     6506.000000
5.75     6876.666667
6.00     6443.333333
6.25     8345.000000
6.50     6765.000000
6.75     8560.000000
7.50     4050.000000
7.75     9890.000000
8.00    12795.000000
Name: sqft_living, dtype: float64
```

In [139]:
```
X['bathrooms'].value_counts(normalize=True)
```

Out[139]:
```
2.50    0.251131
1.00    0.177103
1.75    0.140243
2.25    0.094746
2.00    0.088885
1.50    0.066780
2.75    0.054802
3.00    0.034649
3.50    0.034084
3.25    0.027452
3.75    0.006889
4.00    0.006118
4.50    0.004678
4.25    0.003701
0.75    0.003444
4.75    0.001080
5.00    0.000925
5.25    0.000617
0.00    0.000514
5.50    0.000514
1.25    0.000463
6.00    0.000308
0.50    0.000206
5.75    0.000154
8.00    0.000103
6.25    0.000103
6.50    0.000103
6.75    0.000103
7.50    0.000051
7.75    0.000051
Name: bathrooms, dtype: float64
```

Maximum houses have the bathrooms either 1 or 2.5 hence choosing appropriate values of mean living area to impute the number of bathrooms as the two variables are very highly correlated. We will be selecting an appropriate cuttoff value for the imputation

In [140]:
```
((2376.896569 - 1209.031391)/2) + 1209.031391
```

Out[140]:
```
1792.96398
```

In [141]:
```
X.loc[((X['bathrooms'].isna()) & ((X['sqft_living'])<= 1792.96398)),'bathroom
X.loc[((X['bathrooms'].isna()) & ((X['sqft_living']) > 1792.96398)),'bathroom

#X.loc[((X['bathrooms'].isna()) & (((((X['sqft_living'])<= 1822.186282)) & (X
#X.loc[((X['bathrooms'].isna()) & (((((X['sqft_living'])<= 2530.850591)) & (X
#X.loc[((X['bathrooms'].isna()) & (((((X['sqft_living'])<= 3003.129655)) & (X
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
```

In [142]:
```
X['bathrooms'].isna().sum()
```

Out[142]: 0

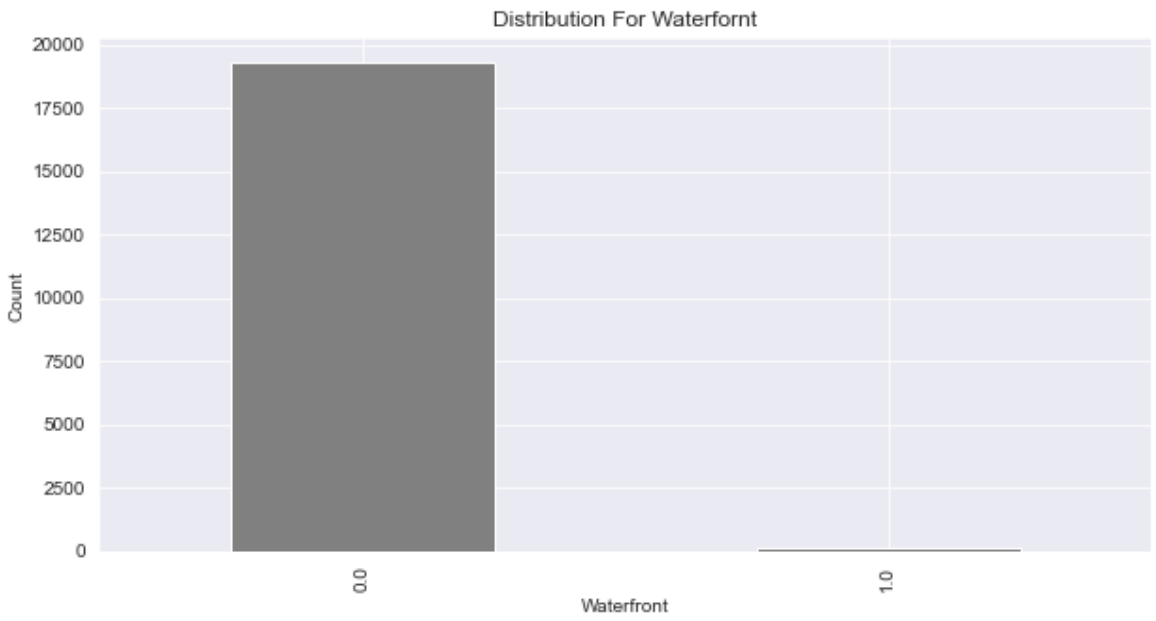
In [143]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       21613 non-null  float64
 1   bathrooms      21613 non-null  float64
 2   sqft_living    21613 non-null  float64
 3   sqft_lot       21613 non-null  float64
 4   floors         21613 non-null  float64
 5   waterfront     19452 non-null  float64
 6   view           19452 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     21613 non-null  float64
 10  sqft_basement  21613 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  21613 non-null  float64
 14  sqft_lot15     21613 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

Waterfront

In [144]:
```
fig=plt.figure(figsize=(10,5))
X['waterfront'].value_counts().plot.bar(color='gray')
plt.title('Distribution For Waterfornt')
plt.ylabel('Count')
plt.xlabel('Waterfront')
```

Out[144]: Text(0.5, 0, 'Waterfront')



In [145]:
```
X['waterfront'].value_counts(normalize=True)
```

Out[145]:
```
0.0    0.99234
1.0    0.00766
Name: waterfront, dtype: float64
```

99% of the houses do not have waterfront hence we would impute it with 0 (The Mode)

In [146]:
```
X['waterfront'].mode()
```

Out[146]:
```
0    0.0
dtype: float64
```

In [147]:
```
X.loc[(X['waterfront'].isna()),'waterfront']= 0
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

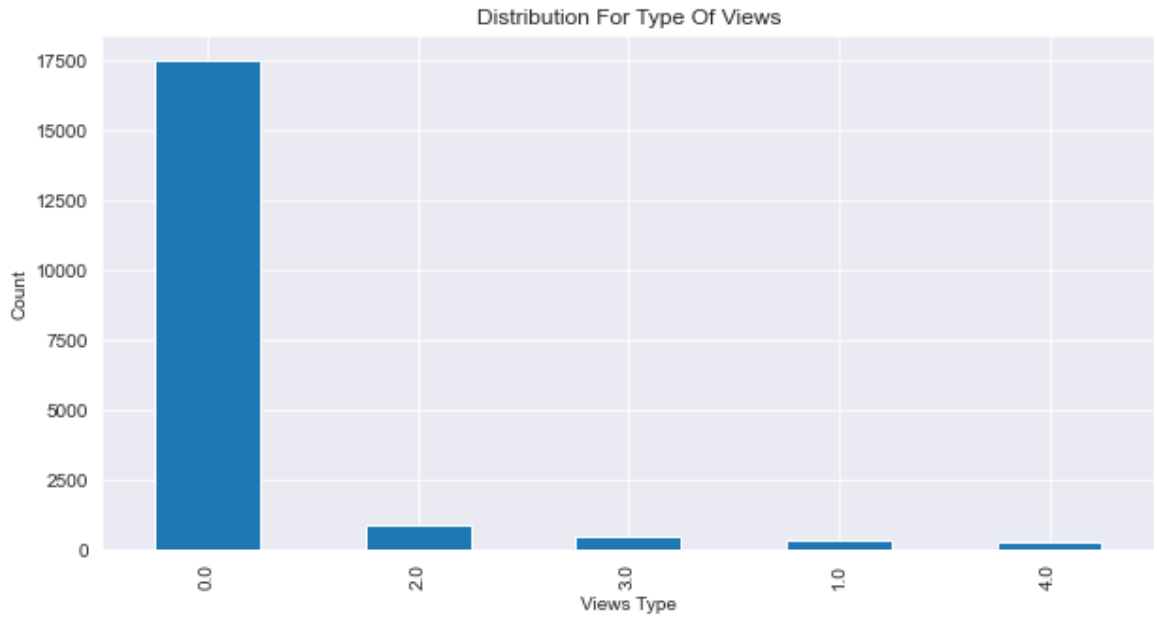
In [148]:
```
X['waterfront'].isna().sum()
```

Out[148]:
```
0
```

View

In [149]:
```
fig=plt.figure(figsize=(10,5))
X['view'].value_counts().plot.bar()
plt.title('Distribution For Type Of Views')
plt.ylabel('Count')
plt.xlabel('Views Type')
```

Out[149]: Text(0.5, 0, 'Views Type')



In [150]:
```
X.groupby('view')['sqft_living'].mean()
```

Out[150]:
```
view
0.0    1995.300476
1.0    2555.469071
2.0    2652.534937
3.0    2990.136946
4.0    3346.935324
Name: sqft_living, dtype: float64
```

In [151]:
```
X['view'].value_counts(normalize=True)
```

Out[151]:
```
0.0    0.901295
2.0    0.044725
3.0    0.023597
1.0    0.015525
4.0    0.014857
Name: view, dtype: float64
```

In [152]:
```
X.loc[(X['view'].isna()),'view']= 0
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [153]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   bedrooms       21613 non-null  float64
 1   bathrooms      21613 non-null  float64
 2   sqft_living    21613 non-null  float64
 3   sqft_lot       21613 non-null  float64
 4   floors         21613 non-null  float64
 5   waterfront     21613 non-null  float64
 6   view           21613 non-null  float64
 7   condition      19452 non-null  float64
 8   grade          19452 non-null  float64
 9   sqft_above     21613 non-null  float64
 10  sqft_basement  21613 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  21613 non-null  float64
 14  sqft_lot15     21613 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

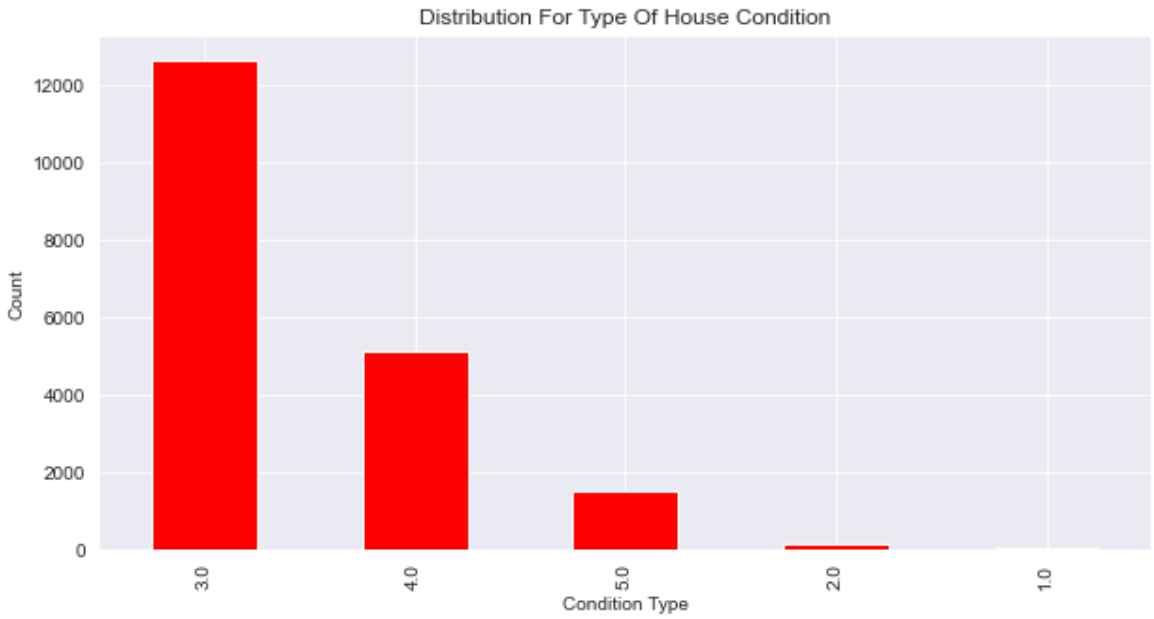
In [154]:
```
X['view'].isnull().sum()
```

Out[154]:
```
0
```

Condition

In [155]:
```
fig=plt.figure(figsize=(10,5))
X['condition'].value_counts().plot.bar(color='red')
plt.title('Distribution For Type Of House Condition')
plt.ylabel('Count')
plt.xlabel('Condition Type')
```

Out[155]:
```
Text(0.5, 0, 'Condition Type')
```



In [156]:
```
X['condition'].value_counts(normalize=True)
```

Out[156]:
```
3.0    0.649702
4.0    0.262955
5.0    0.078090
2.0    0.007814
1.0    0.001439
Name: condition, dtype: float64
```

In [157]:
```
X.groupby('condition')['yr_built'].max()
```

Out[157]:
```
condition
1.0    1966.0
2.0    1995.0
3.0    2015.0
4.0    2009.0
5.0    2005.0
Name: yr_built, dtype: float64
```

In [158]:
```
X.loc[((X['condition'].isna()) & ((X['yr_built']) >= 2000 )),'bathrooms']= 3
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [159]:
```
X['condition']
```

Out[159]:
```
0        3.0
1        3.0
2        3.0
3        5.0
4        3.0
        ...
21608    3.0
21609    3.0
21610    3.0
21611    NaN
21612    3.0
Name: condition, Length: 21613, dtype: float64
```

In [160]:
```
X.loc[(X['condition'].isna()),'condition']= 3
```

```
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

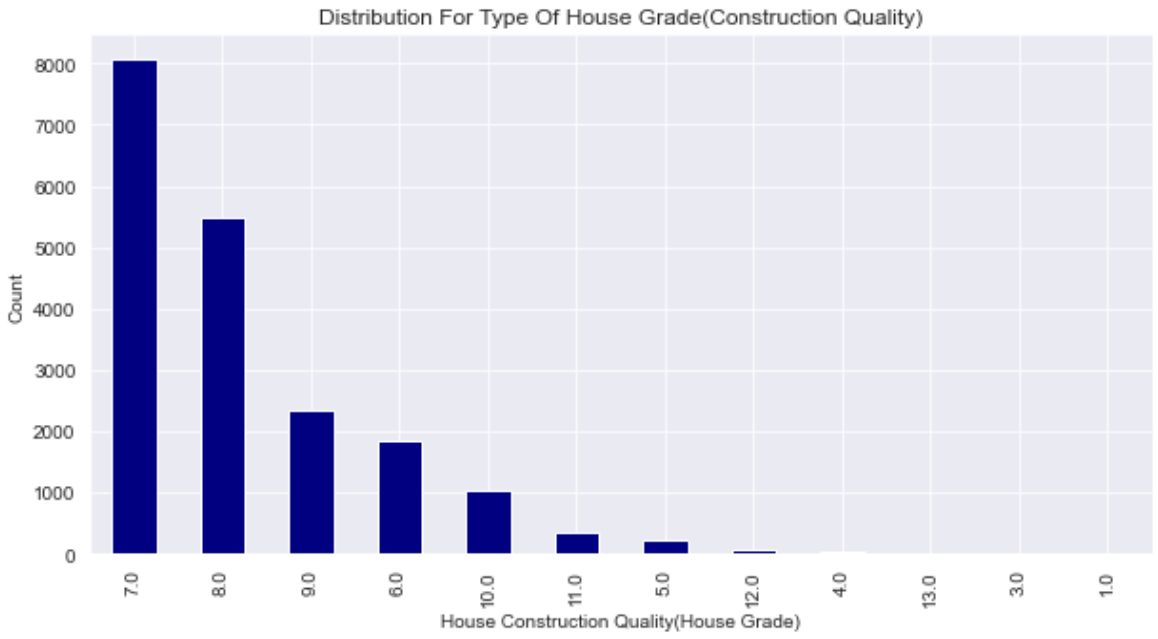
In [161]:
```
X['condition'].isna().sum()
```

Out[161]:
```
0
```

Grade

In [162]:
```
fig=plt.figure(figsize=(10,5))
X['grade'].value_counts().plot.bar(color='navy')
plt.title('Distribution For Type Of House Grade(Construction Quality)')
plt.ylabel('Count')
plt.xlabel('House Construction Quality(House Grade)')
```

Out[162]: Text(0.5, 0, 'House Construction Quality(House Grade)')



In [163]:
```
X['grade'].value_counts(normalize=True)
```

Out[163]:
```
7.0     0.414559
8.0     0.281976
9.0     0.120193
6.0     0.094797
10.0    0.052488
11.0    0.018301
5.0     0.011104
12.0    0.004267
4.0     0.001439
13.0    0.000668
3.0     0.000154
1.0     0.000051
Name: grade, dtype: float64
```

In [164]:
```
X.groupby('grade')['sqft_living'].mean()
```

Out[164]:
```
grade
1.0      290.000000
3.0      596.666667
4.0      661.478261
5.0      971.735751
6.0     1190.512867
7.0     1685.845294
8.0     2181.810443
9.0     2865.151544
10.0    3531.699566
11.0    4392.553191
12.0    5491.527027
13.0    7483.076923
Name: sqft_living, dtype: float64
```

In [165]:
```
X.loc[(X['grade'].isna()),'grade']= 7
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [166]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       21613 non-null  float64
 1   bathrooms      21613 non-null  float64
 2   sqft_living    21613 non-null  float64
 3   sqft_lot       21613 non-null  float64
 4   floors         21613 non-null  float64
 5   waterfront     21613 non-null  float64
 6   view           21613 non-null  float64
 7   condition      21613 non-null  float64
 8   grade          21613 non-null  float64
 9   sqft_above     21613 non-null  float64
 10  sqft_basement  21613 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   19452 non-null  float64
 13  sqft_living15  21613 non-null  float64
 14  sqft_lot15     21613 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

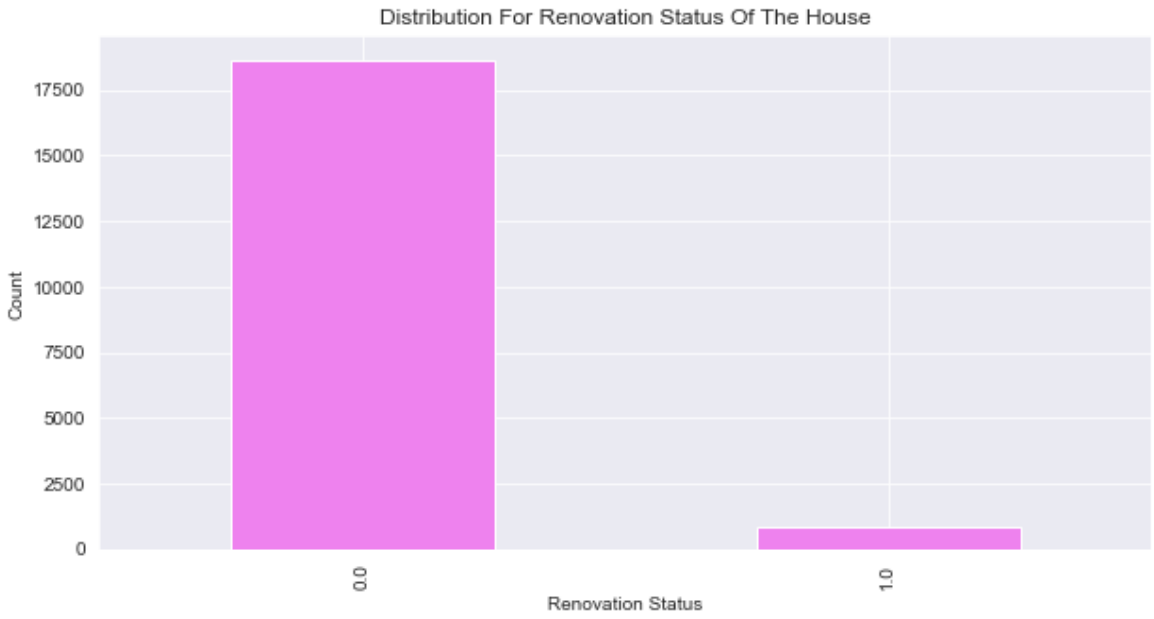
In [167]:
```
X['grade'].isna().sum()
```

Out[167]: 0

yr_renovated

In [168]:
```
fig=plt.figure(figsize=(10,5))
X['yr_renovated'].value_counts().plot.bar(color='violet')
plt.title('Distribution For Renovation Status Of The House')
plt.ylabel('Count')
plt.xlabel('Renovation Status')
```

Out[168]:
```
Text(0.5, 0, 'Renovation Status')
```



In [169]:
```
X['yr_renovated'].value_counts()
```

Out[169]:
```
0.0    18629
1.0      823
Name: yr_renovated, dtype: int64
```

Almost all the houses in our dataset have never been renovated hence we will impute this column by the mode that is 0

In [170]:
```
X.loc[(X['yr_renovated'].isna()),'yr_renovated']= 0
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [171]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   bedrooms       21613 non-null  float64
 1   bathrooms      21613 non-null  float64
 2   sqft_living    21613 non-null  float64
 3   sqft_lot       21613 non-null  float64
 4   floors         21613 non-null  float64
 5   waterfront     21613 non-null  float64
 6   view           21613 non-null  float64
 7   condition      21613 non-null  float64
 8   grade          21613 non-null  float64
 9   sqft_above     21613 non-null  float64
 10  sqft_basement  21613 non-null  float64
 11  yr_built       19452 non-null  float64
 12  yr_renovated   21613 non-null  float64
 13  sqft_living15  21613 non-null  float64
 14  sqft_lot15     21613 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

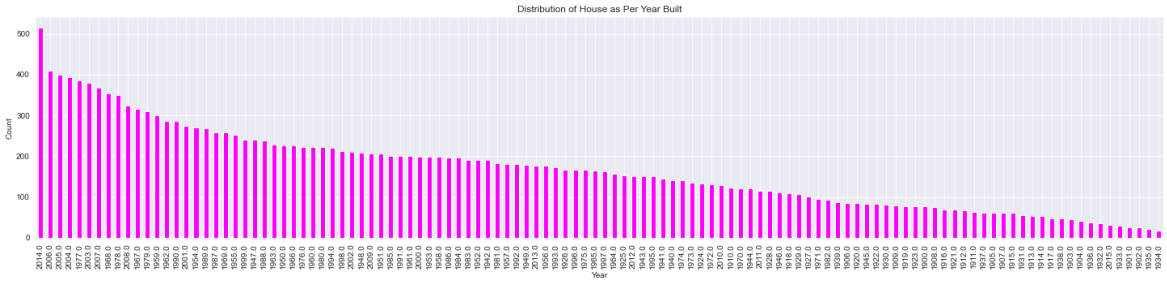
In [172]:
```
X['yr_renovated'].isna().sum()
```

Out[172]: 0

yr_built

In [173]:
```
fig=plt.figure(figsize=(25,5))
X['yr_built'].value_counts().plot.bar(color='magenta')
plt.title('Distribution of House as Per Year Built')
plt.xlabel('Year')
plt.ylabel('Count')
```

Out[173]: Text(0, 0.5, 'Count')



In [174]:
```
X.groupby('condition')['yr_built'].mean()
```

Out[174]:
```
condition
1.0    1930.739130
2.0    1947.121429
3.0    1978.225130
4.0    1958.145411
5.0    1946.803412
Name: yr_built, dtype: float64
```

In [175]:
```
X['yr_built'].value_counts(bins=11)
```

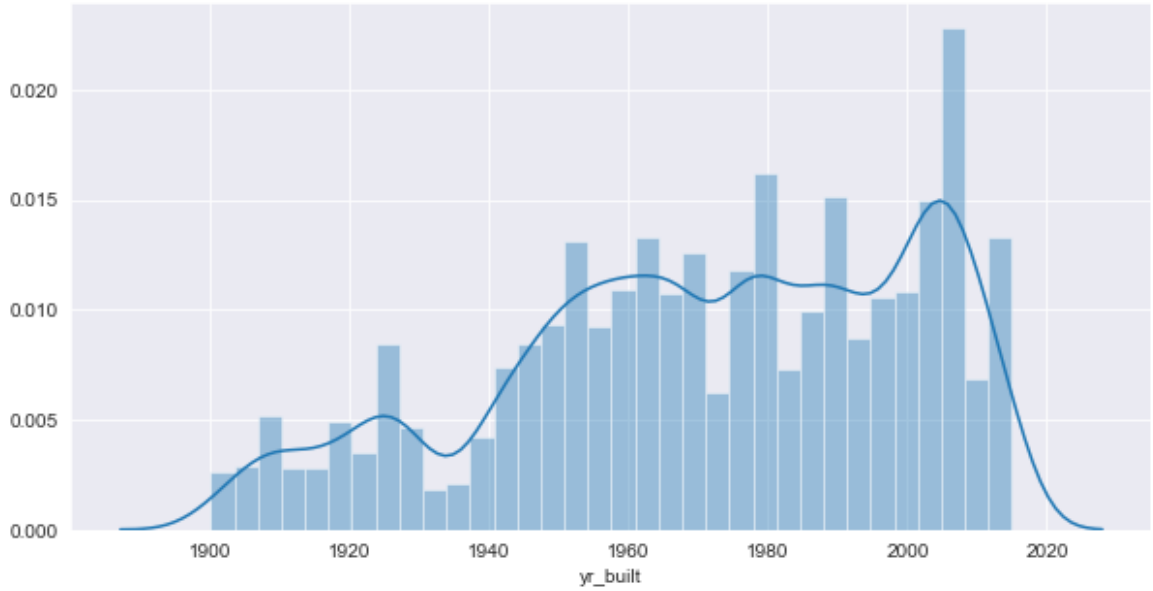
Out[175]:
```
(2004.545, 2015.0]      2829
(1983.636, 1994.091]    2420
(1994.091, 2004.545]    2393
(1952.273, 1962.727]    2285
(1973.182, 1983.636]    2268
(1962.727, 1973.182]    2189
(1941.818, 1952.273]    1911
(1920.909, 1931.364]    1146
(1899.884, 1910.455]     700
(1910.455, 1920.909]     688
(1931.364, 1941.818]     623
Name: yr_built, dtype: int64
```

In [176]:
```
X['yr_built'].median()
```

Out[176]: 1975.0

In [177]:
```
fig = plt.figure(figsize=(10,5))
sns.distplot(X['yr_built'])
```

Out[177]: <matplotlib.axes._subplots.AxesSubplot at 0x17fd6532dd8>



In [178]:
```
X.loc[(X['yr_built'].isna()),'yr_built']= 1975
```

```
C:\Users\behab\Anaconda3\lib\site-packages\pandas\core\indexing.py:671: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\behab\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://p
andas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [179]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       21613 non-null  float64
 1   bathrooms      21613 non-null  float64
 2   sqft_living    21613 non-null  float64
 3   sqft_lot       21613 non-null  float64
 4   floors         21613 non-null  float64
 5   waterfront     21613 non-null  float64
 6   view           21613 non-null  float64
 7   condition      21613 non-null  float64
 8   grade          21613 non-null  float64
 9   sqft_above     21613 non-null  float64
 10  sqft_basement  21613 non-null  float64
 11  yr_built       21613 non-null  float64
 12  yr_renovated   21613 non-null  float64
 13  sqft_living15  21613 non-null  float64
 14  sqft_lot15     21613 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

In [180]:
```
X.isna().sum().sum()
```

Out[180]:
```
0
```

The Dataset has finally been imputed

Checking Whether The Predictors Or The Response variable Has any suspecious or unexplained values

In [181]:
```
X.columns
```

Out[181]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15',
       'sqft_lot15'],
      dtype='object')
```

In [182]:
```
X['bedrooms'].unique()
```

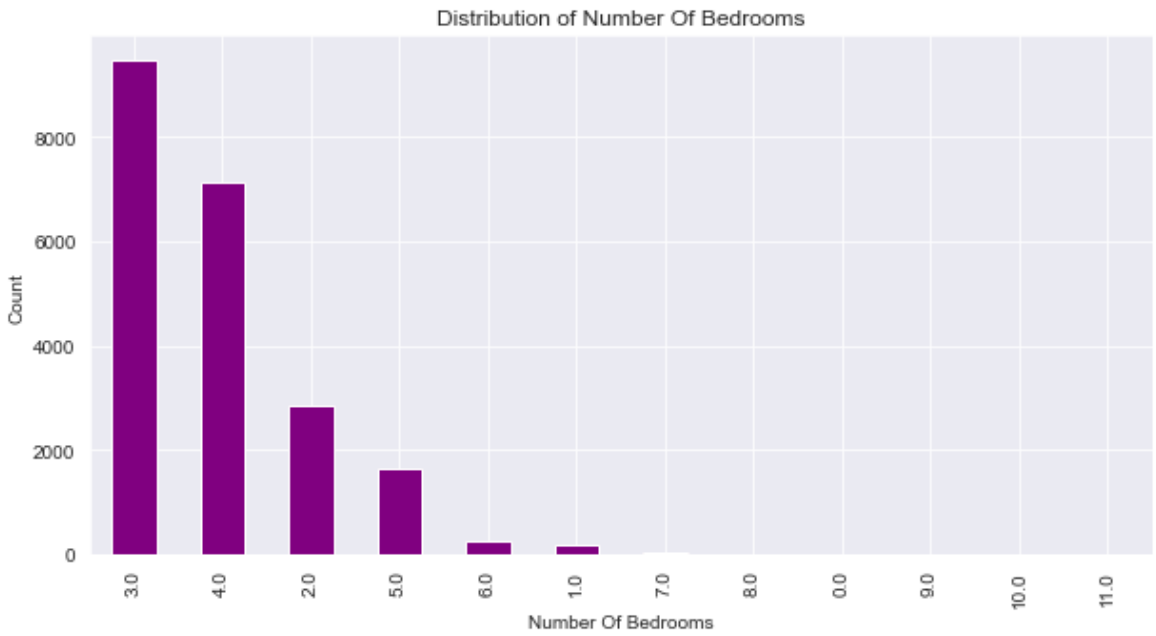
Out[182]:
```
array([ 3.,  2.,  4.,  5.,  1.,  6.,  0.,  7.,  8.,  9., 11., 10.])
```

In [183]:
```
X['bedrooms'].value_counts()
```

Out[183]:
```
3.0     9459
4.0     7123
2.0     2863
5.0     1668
6.0      249
1.0      187
7.0       34
8.0       11
0.0       10
9.0        5
10.0       3
11.0       1
Name: bedrooms, dtype: int64
```

In [184]:
```
fig=plt.figure(figsize=(10,5))
X['bedrooms'].value_counts().plot.bar(color='purple')
plt.title('Distribution of Number Of Bedrooms')
plt.ylabel('Count')
plt.xlabel('Number Of Bedrooms')
```

Out[184]:
```
Text(0.5, 0, 'Number Of Bedrooms')
```



The Bedrooms columns looks perfectly fine with all the understandable values

In [185]:
```
X['bathrooms'].unique()
```

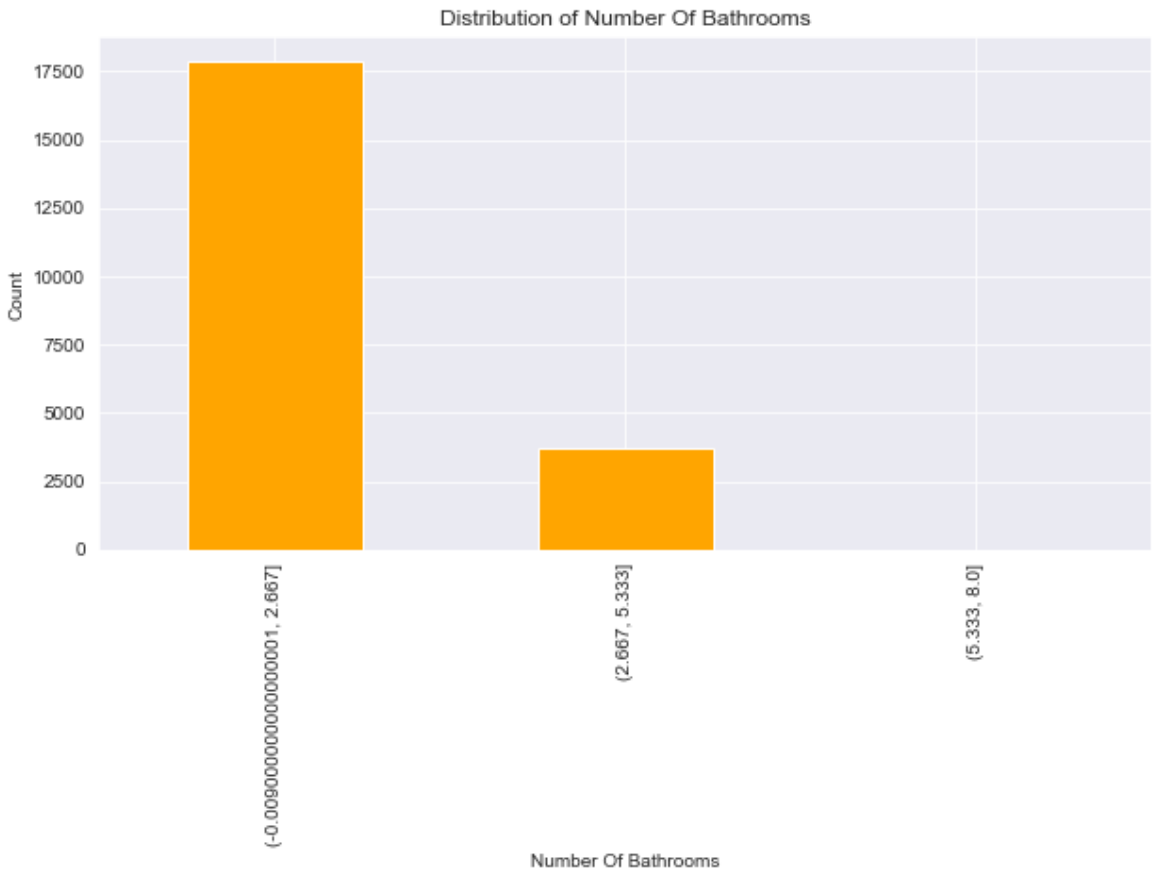
Out[185]:
```
array([1.  , 2.25, 3.  , 2.  , 4.5 , 1.5 , 2.5 , 1.75, 2.75, 3.25, 4.  ,
       3.5 , 0.75, 4.75, 5.  , 4.25, 3.75, 0.  , 1.25, 5.25, 6.  , 0.5 ,
       5.5 , 6.75, 8.  , 7.5 , 7.75, 6.25, 5.75, 6.5 ])
```

In [186]:
```
X['bathrooms'].value_counts(bins=3)
```

Out[186]:
```
(-0.009000000000000001, 2.667]    17865
(2.667, 5.333]                     3720
(5.333, 8.0]                         28
Name: bathrooms, dtype: int64
```

In [187]:
```
fig=plt.figure(figsize=(10,5))
X['bathrooms'].value_counts(bins=3).plot.bar(color='orange')
plt.title('Distribution of Number Of Bathrooms')
plt.ylabel('Count')
plt.xlabel('Number Of Bathrooms')
```

Out[187]:
```
Text(0.5, 0, 'Number Of Bathrooms')
```



The Bathrooms column looks perfectly fine with all the understandable or comprehendible values

In [188]:
```
X['sqft_living'].unique()
```

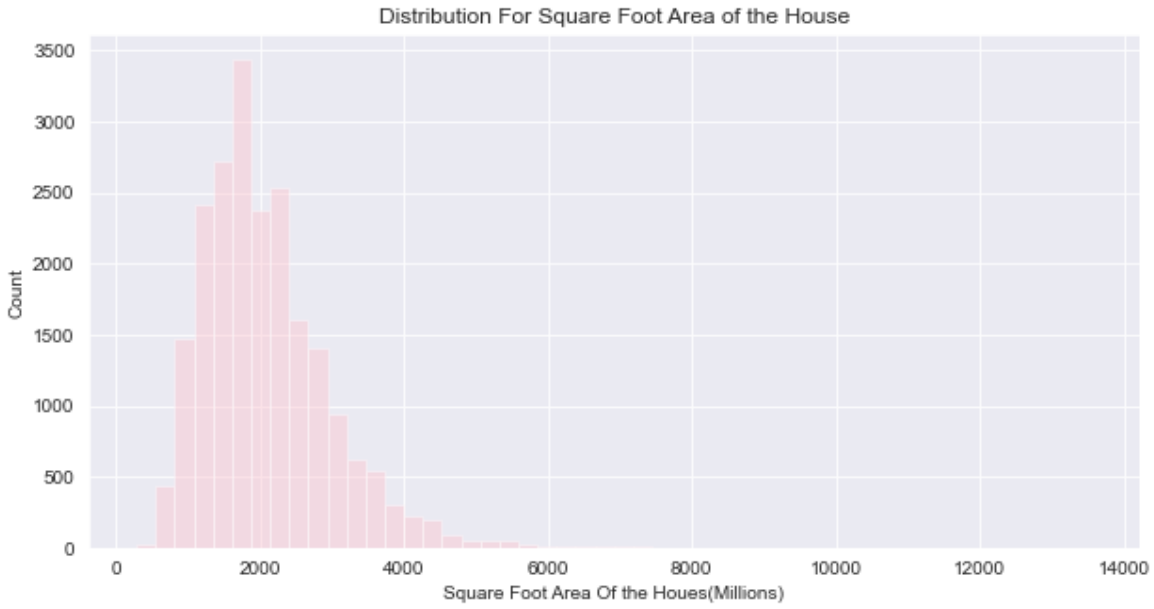
Out[188]:
```
array([1180., 2570.,  770., ..., 3087., 3118., 1425.])
```

In [189]:
```
X['sqft_living'].value_counts(bins=3)
```

Out[189]:
```
(276.749, 4706.667]     21344
(4706.667, 9123.333]      263
(9123.333, 13540.0]         6
Name: sqft_living, dtype: int64
```

In [190]:
```
fig=plt.figure(figsize=(10,5))
sns.distplot(X['sqft_living'],color='pink',kde=False)
plt.title('Distribution For Square Foot Area of the House')
plt.ylabel('Count')
plt.xlabel('Square Foot Area Of the Houes(Millions)')
```

Out[190]:
```
Text(0.5, 0, 'Square Foot Area Of the Houes(Millions)')
```



The Sqft_living column looks perfectly fine with all the understandable or comprehendible values

In [191]:
```
X['sqft_lot'].unique()
```

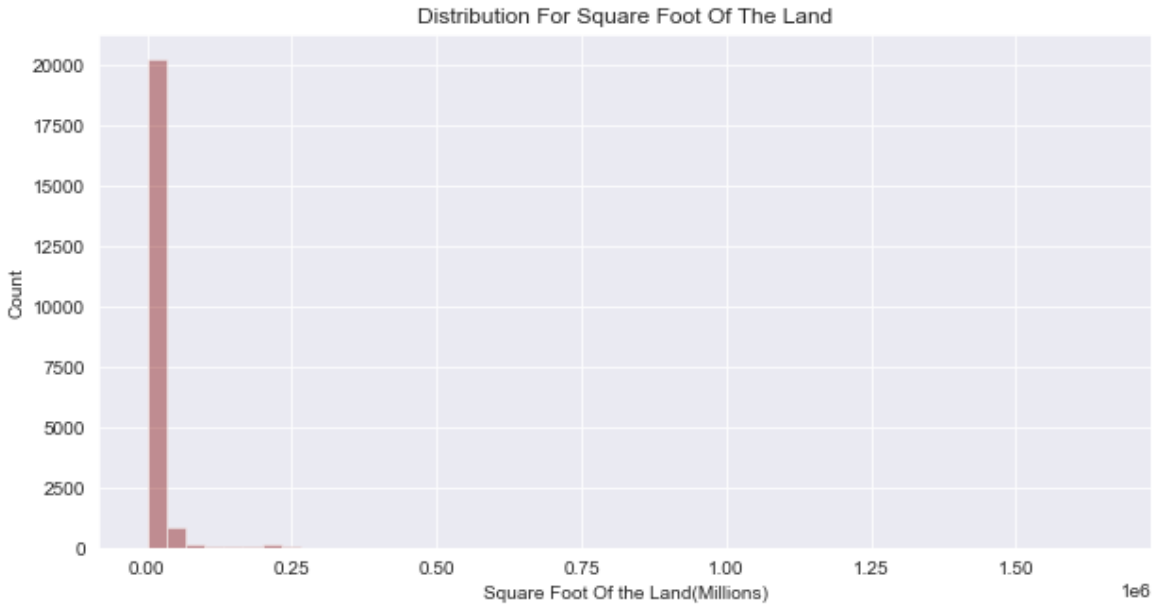
Out[191]:
```
array([ 5650.,  7242., 10000., ...,  5813.,  2388.,  1076.])
```

In [192]:
```
X['sqft_lot'].value_counts(bins=20)
```

Out[192]:
```
(-1130.84, 83061.95]       21138
(83061.95, 165603.9]         206
(165603.9, 248145.85]        174
(248145.85, 330687.8]         46
(413229.75, 495771.7]         17
(330687.8, 413229.75]         13
(495771.7, 578313.65]          7
(908481.45, 991023.4]          3
(825939.5, 908481.45]          3
(578313.65, 660855.6]          2
(660855.6, 743397.55]          1
(1568817.05, 1651359.0]        1
(991023.4, 1073565.35]         1
(1073565.35, 1156107.3]        1
(1486275.1, 1568817.05]        0
(1156107.3, 1238649.25]        0
(1238649.25, 1321191.2]        0
(1321191.2, 1403733.15]        0
(1403733.15, 1486275.1]        0
(743397.55, 825939.5]          0
Name: sqft_lot, dtype: int64
```

In [193]:
```
fig=plt.figure(figsize=(10,5))
sns.distplot(X['sqft_lot'],color='maroon',bins=50,kde=False)
plt.title('Distribution For Square Foot Of The Land')
plt.ylabel('Count')
plt.xlabel('Square Foot Of the Land(Millions)')
```

Out[193]:
```
Text(0.5, 0, 'Square Foot Of the Land(Millions)')
```



The sqft_lot column looks perfectly fine with all the understandable or comprehendible values

In [194]:
```
X['floors'].unique()
```

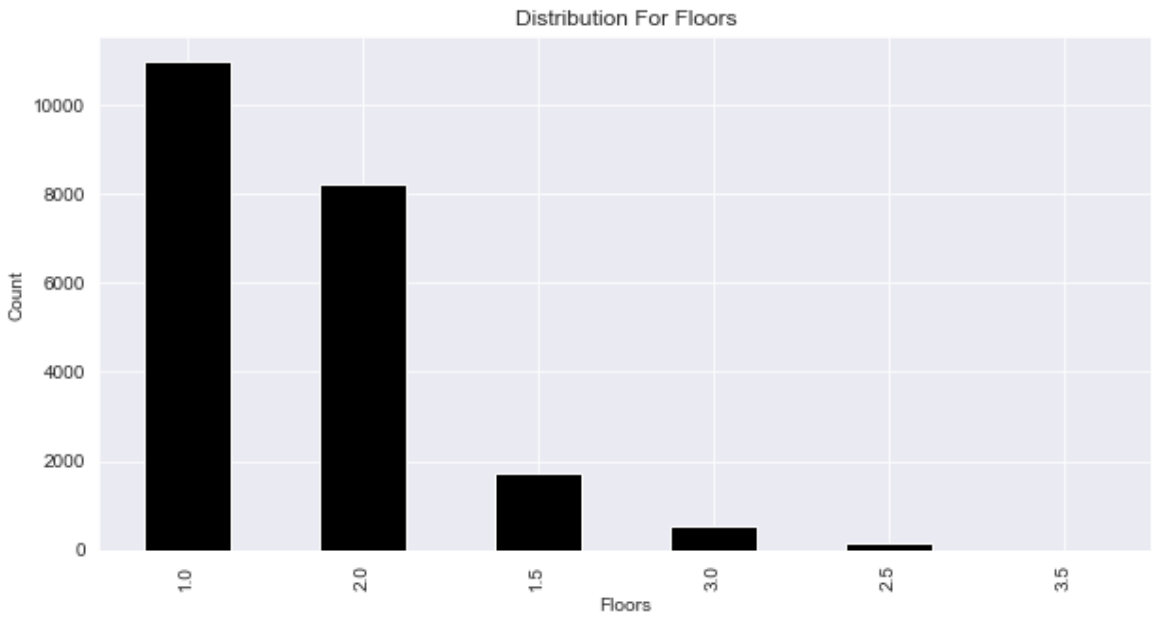
Out[194]:
```
array([1. , 2. , 1.5, 3. , 2.5, 3.5])
```

In [195]:
```
X['floors'].value_counts()
```

Out[195]:
```
1.0    10974
2.0     8226
1.5     1718
3.0      544
2.5      143
3.5        8
Name: floors, dtype: int64
```

In [196]:
```
fig=plt.figure(figsize=(10,5))
X['floors'].value_counts().plot.bar(color='black')
plt.title('Distribution For Floors')
plt.ylabel('Count')
plt.xlabel('Floors')
```

Out[196]:
```
Text(0.5, 0, 'Floors')
```



The floors column looks perfectly fine with all the understandable or comprehendible values

In [197]:
```
X['waterfront'].unique()
```

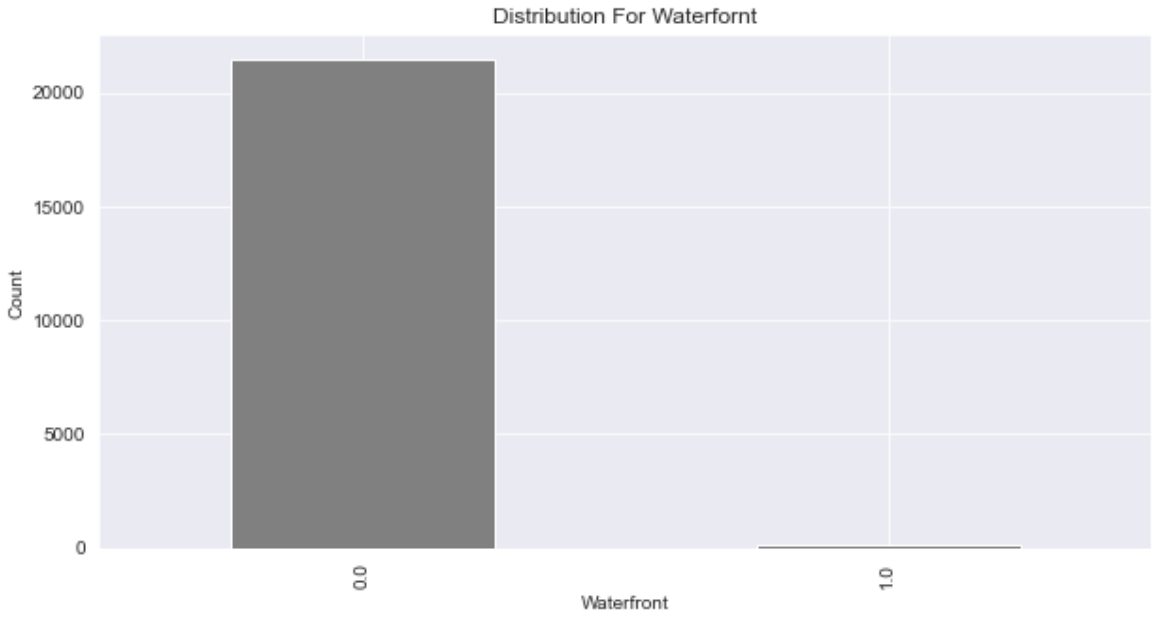
Out[197]:
```
array([0., 1.])
```

In [198]:
```
X['waterfront'].value_counts()
```

Out[198]:
```
0.0    21464
1.0      149
Name: waterfront, dtype: int64
```

In [199]:
```
fig=plt.figure(figsize=(10,5))
X['waterfront'].value_counts().plot.bar(color='gray')
plt.title('Distribution For Waterfornt')
plt.ylabel('Count')
plt.xlabel('Waterfront')
```

Out[199]:
```
Text(0.5, 0, 'Waterfront')
```



The Waterfront column looks perfectly fine with all the understandable or comprehendible values

In [200]:
```
X.columns
```

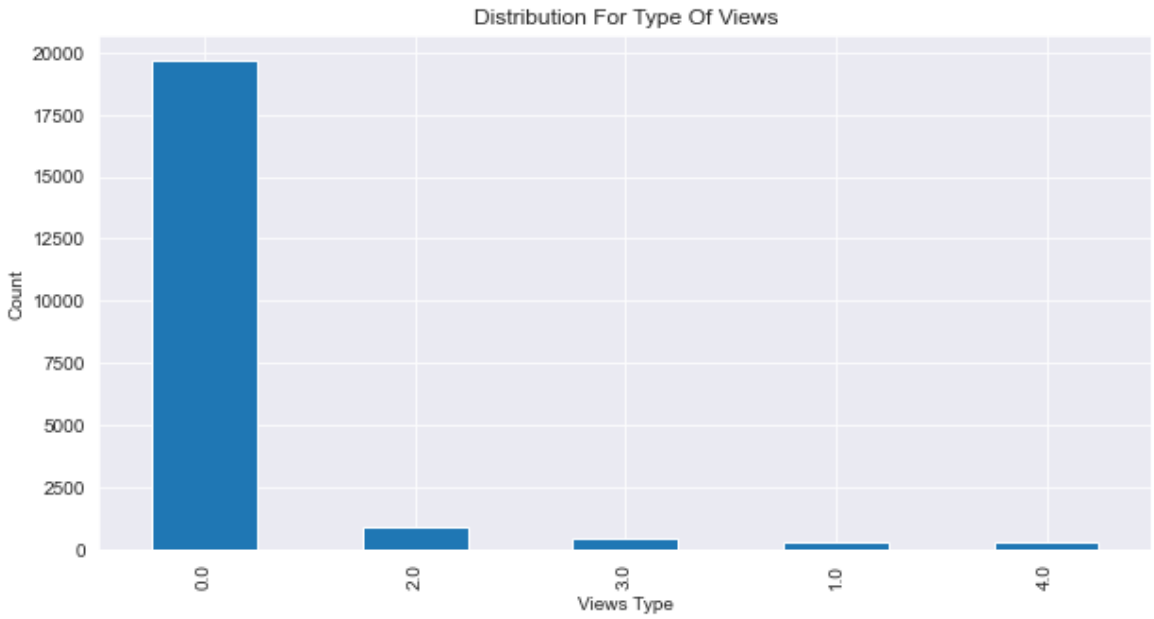
Out[200]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15',
       'sqft_lot15'],
      dtype='object')
```

In [201]:
```
X['view'].unique()
```

Out[201]:
```
array([0., 3., 4., 2., 1.])
```

In [202]:
```
fig=plt.figure(figsize=(10,5))
X['view'].value_counts().plot.bar()
plt.title('Distribution For Type Of Views')
plt.ylabel('Count')
plt.xlabel('Views Type')
```

Out[202]:
```
Text(0.5, 0, 'Views Type')
```



The Views column looks perfectly fine with all the understandable or comprehendible values

In [203]:
```
X['condition'].unique()
```

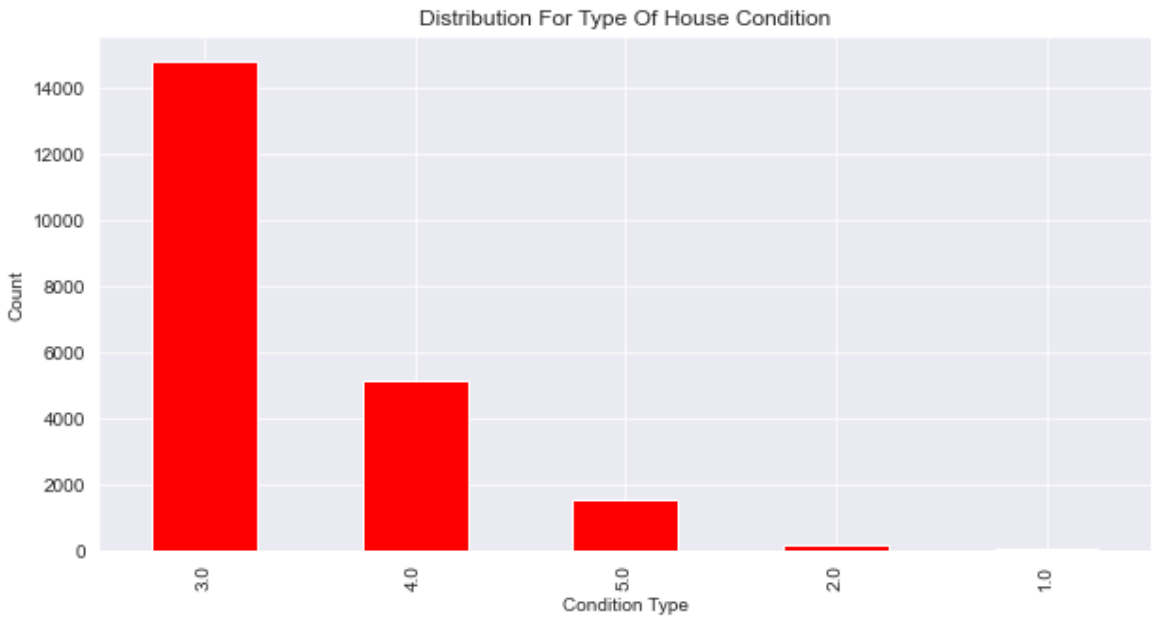
Out[203]:
```
array([3., 5., 4., 1., 2.])
```

In [204]:
```
X['condition'].value_counts()
```

Out[204]:
```
3.0    14799
4.0     5115
5.0     1519
2.0      152
1.0       28
Name: condition, dtype: int64
```

In [205]:
```
fig=plt.figure(figsize=(10,5))
X['condition'].value_counts().plot.bar(color='red')
plt.title('Distribution For Type Of House Condition')
plt.ylabel('Count')
plt.xlabel('Condition Type')
```

Out[205]:
```
Text(0.5, 0, 'Condition Type')
```



The Condition column looks perfectly fine with all the understandable or comprehendible values

In [206]:
```
X['grade'].unique()
```

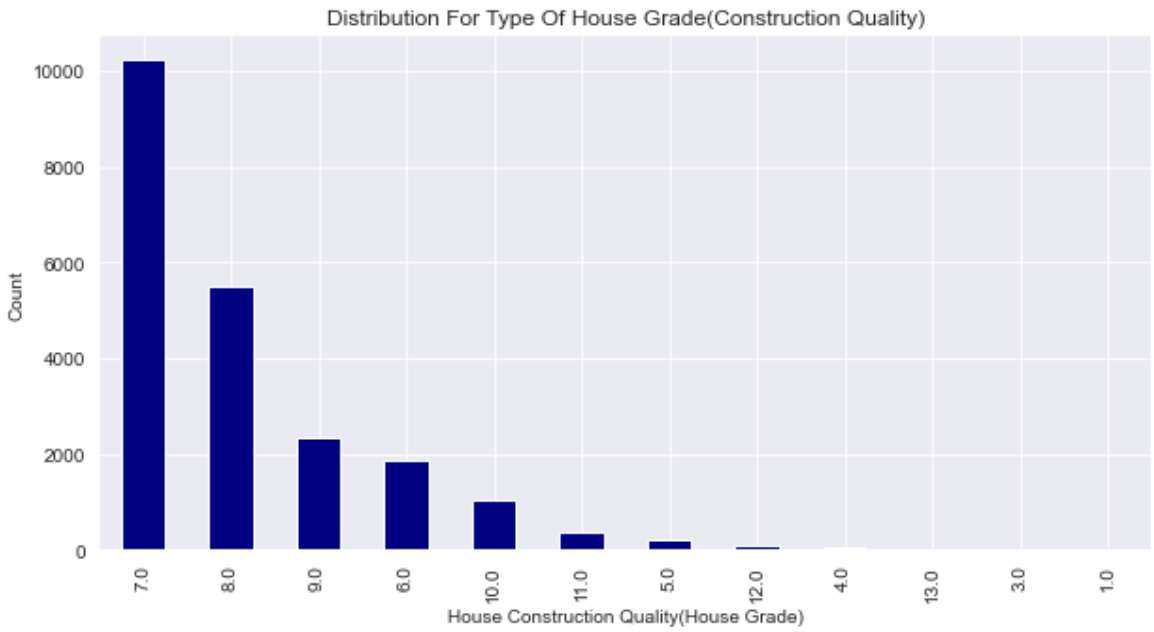
Out[206]:
```
array([ 7.,  6.,  8., 11.,  9.,  5., 10., 12.,  4.,  3., 13.,  1.])
```

In [207]:
```
X['grade'].value_counts()
```

Out[207]:
```
7.0     10225
8.0      5485
9.0      2338
6.0      1844
10.0     1021
11.0      356
5.0       216
12.0       83
4.0        28
13.0       13
3.0         3
1.0         1
Name: grade, dtype: int64
```

In [208]:
```
fig=plt.figure(figsize=(10,5))
X['grade'].value_counts().plot.bar(color='navy')
plt.title('Distribution For Type Of House Grade(Construction Quality)')
plt.ylabel('Count')
plt.xlabel('House Construction Quality(House Grade)')
```

Out[208]:
```
Text(0.5, 0, 'House Construction Quality(House Grade)')
```



The Grade column looks perfectly fine with all the understandable or comprehendible values

In [209]:
```
X['sqft_above'].unique()
```

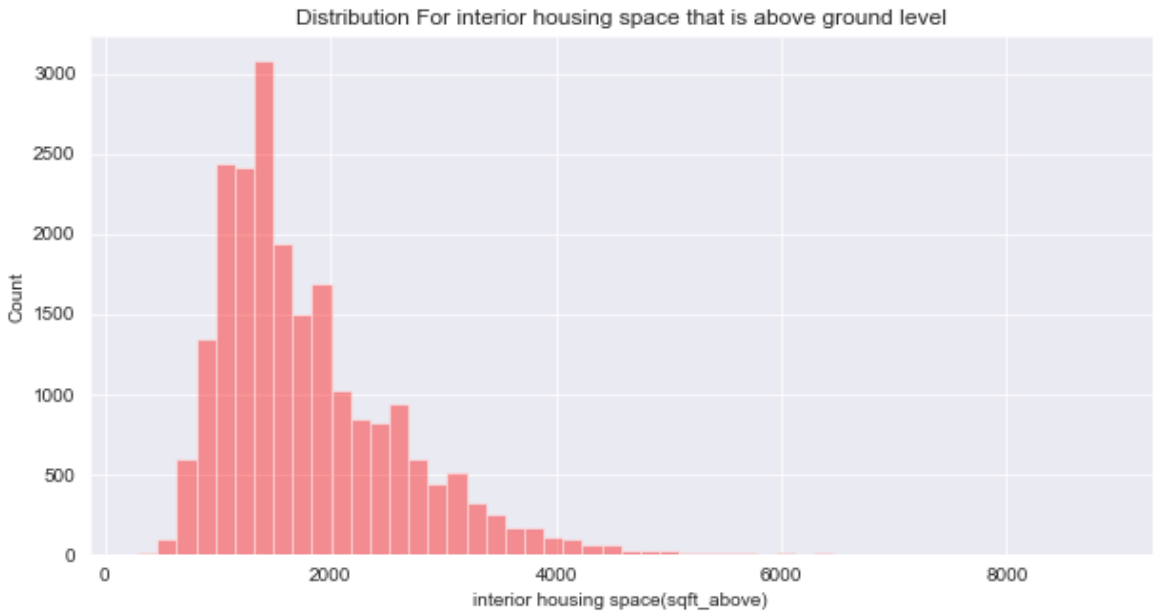
Out[209]:
```
array([1180.        , 2170.        ,  770.        , 1050.        ,
       1680.        , 3890.        , 1715.        , 1060.        ,
       1409.5195984 , 1890.        , 1860.        ,  860.        ,
       1430.        , 1370.        , 1810.        , 1980.        ,
       1600.        , 1200.        , 1250.        , 2330.        ,
       2270.        , 1070.        , 2450.        , 1710.        ,
       1750.        , 1400.        ,  790.        , 2570.        ,
       2320.        , 1190.        , 1090.        , 1510.        ,
        927.59677419, 2360.        ,  890.        , 2620.        ,
       2600.        , 3595.        , 1570.        ,  920.        ,
       3160.        ,  990.        , 2290.        , 2165.        ,
       1640.        , 2130.        , 2830.        , 2250.        ,
       2420.        , 3250.        , 1850.        , 1590.        ,
       1260.        , 2519.        , 1540.        , 1110.        ,
       1770.        , 2720.        , 2240.        , 1000.        ,
       3070.        , 2380.        , 2390.        ,  880.        ,
       1040.        ,  910.        , 3450.        , 2350.        ,
       1900.        ,  960.        , 2660.        , 1610.        ,
        765.        , 3520.        , 1290.        , 1960.        ,
```

In [210]:
```
X['sqft_above'].value_counts(bins=5)
```

Out[210]:
```
(281.42900000000003, 2004.0]    15102
(2004.0, 3718.0]                 5888
(3718.0, 5432.0]                  579
(5432.0, 7146.0]                   37
(7146.0, 8860.0]                    7
Name: sqft_above, dtype: int64
```

In [211]:
```
fig=plt.figure(figsize=(10,5))
sns.distplot(X['sqft_above'],color='red',kde=False)
plt.title('Distribution For interior housing space that is above ground level
plt.ylabel('Count')
plt.xlabel('interior housing space(sqft_above)')
```

Out[211]:
```
Text(0.5, 0, 'interior housing space(sqft_above)')
```



The sqft_above column looks perfectly fine with all the understandable or comprehendible values

In [212]:
```
X['sqft_basement'].unique()
```

Out[212]:
```
array([   0.        ,  400.        ,  910.        , 1530.        ,
        730.        , 1700.        ,  300.        ,  221.6075154 ,
        970.        ,  358.45398994,  720.        ,  700.        ,
        133.28366248,  780.        ,  790.        ,  330.        ,
       1620.        ,  360.        ,  588.        , 1510.        ,
        410.        ,  990.        ,  600.        ,  560.        ,
        550.        , 1000.        , 1600.        ,  500.        ,
       1040.        ,  880.        , 1010.        ,  240.        ,
        265.        ,  637.32461538,  290.        ,  800.        ,
        540.        ,  380.        ,  710.        ,  840.        ,
        770.        ,  480.        ,  570.        , 1490.        ,
        620.        , 1250.        , 1270.        ,  650.        ,
        180.        , 1130.        ,  450.        , 1640.        ,
       1460.        , 1020.        , 1030.        ,  750.        ,
        640.        , 1070.        ,  490.        , 1310.        ,
        630.        , 2000.        ,  390.        ,  760.        ,
        850.        ,  210.        , 1430.        , 1950.        ,
        440.        ,  220.        , 1160.        ,  860.        ,
        580.        , 2060.        , 1820.        , 1180.        ,
        200.        , 1150.        , 1200.        ,  680.        ,
        530.        , 1450.        , 1170.        , 1080.        ,
       1100.        ,  280.        ,  870.        ,  460.        ,
       1400.        ,  253.41806244, 1320.        ,  660.        ,
       1220.        , 1580.        , 1380.        ,  475.        ,
        690.        ,  270.        ,  350.        ,  935.        ,
       1370.        , 1470.        ,  160.        ,  950.        ,
         50.        ,  740.        , 1780.        ,  820.        ,
        340.        ,  470.        ,  370.        ,  120.        ,
       1760.        ,  130.        ,  520.        ,  890.        ,
       1110.        ,  150.        ,  140.        , 1720.        ,
        960.        ,  810.        ,  190.        ,   97.0065449 ,
       1290.        ,  207.67546348, 1800.        , 1120.        ,
       1810.        ,   60.        ,  900.        , 1050.        ,
        940.        ,  310.        ,  930.        ,  610.        ,
       1830.        , 1300.        ,  510.        , 1330.        ,
       1590.        ,  430.        ,  920.        ,  420.        ,
       1240.        ,  670.        , 1960.        , 1560.        ,
       2020.        , 1190.        , 2110.        , 1280.        ,
        250.        , 2390.        , 1230.        ,  830.        ,
       1260.        , 1410.        ,  590.        , 1140.        ,
        786.7281106 ,  260.        ,  100.        , 1480.        ,
       1060.        , 1284.        ,   65.56818182, 1670.        ,
       1350.        , 2570.        ,  320.        , 2590.        ,
       1090.        ,  203.1502247 ,  110.        ,  980.        ,
         90.        , 1550.        , 2350.        , 2490.        ,
       1340.        , 1481.        , 1360.        , 1135.        ,
       1520.        , 1660.        , 1390.        , 2130.        ,
       2600.        , 1850.        , 1690.        ,  243.        ,
       1210.        , 2620.        ,  170.        , 1024.        ,
       1610.        , 1440.        , 1500.        , 1570.        ,
       1650.        , 1910.        , 1630.        , 1852.        ,
       2090.        ,  674.        , 1790.        , 2150.        ,
        230.        ,   70.        , 1680.        , 2100.        ,
       3000.        , 1870.        , 1420.        ,  689.78856461,
```

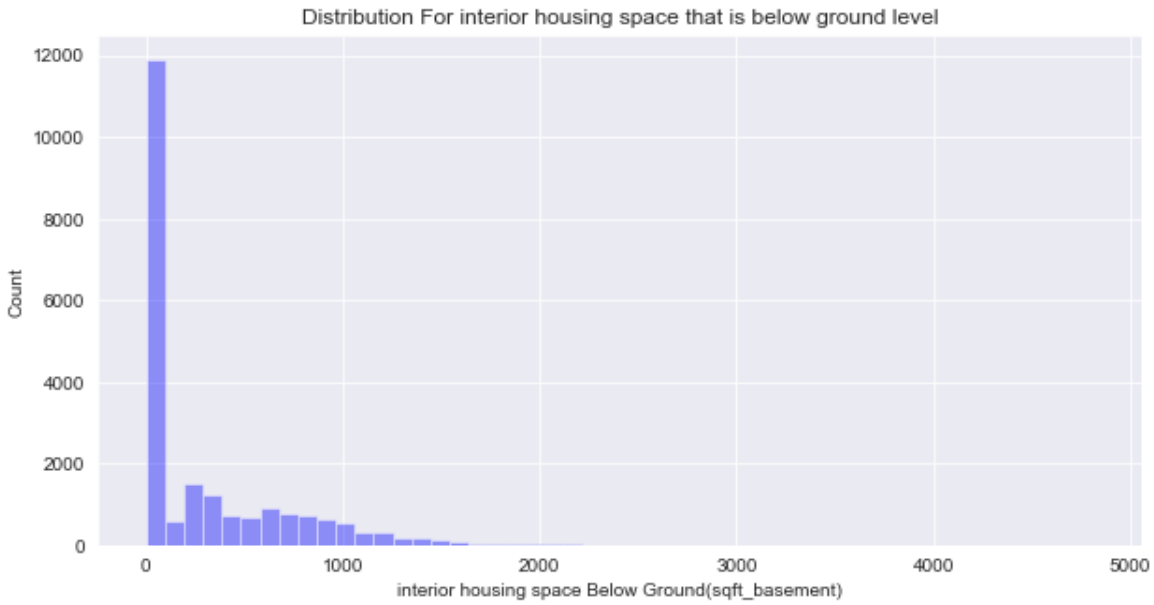
```
1710.        ,  340.15927442, 2030.        ,  875.        ,
1540.        , 2850.        , 2170.        ,  506.        ,
 906.        ,  145.        , 1940.        , 2040.        ,
1750.        ,  493.25300997,  374.        ,  536.66895992,
 518.        , 2720.        ,  370.61206755, 2730.        ,
3480.        , 2160.        , 1920.        , 2330.        ,
1860.        , 2050.        , 4820.        , 1913.        ,
  80.        , 2010.        , 3260.        , 2200.        ,
 303.57602776, 1730.        ,  579.05365769,  652.        ,
2196.        , 1930.        ,  515.        ,   40.        ,
2080.        , 1900.        , 2580.        , 1548.        ,
1740.        ,  235.        ,  861.        , 1890.        ,
2220.        ,  792.        , 2070.        , 4130.        ,
1106.        , 2240.        ,  894.        , 1990.        ,
 278.29466104, 2550.        ,  435.        , 2300.        ,
 666.        , 3500.        ,  172.        , 1816.        ,
2190.        , 1245.        , 1525.        , 1880.        ,
1840.        ,  862.        ,  946.        , 1281.        ,
 414.        , 2180.        ,  276.        , 1248.        ,
 602.        ,  516.        ,  506.12203752,  176.        ,
 225.        , 1275.        ,  266.        ,  283.        ,
  65.        , 2310.        ,   10.        , 1770.        ,
2120.        ,  295.        ,  915.        ,  556.        ,
 417.        ,  143.        , 2810.        ,  274.        ,
 248.        ])
```

In [213]:
```
X['sqft_basement'].value_counts(bins=5)
```

Out[213]:
```
(-4.821000000000001, 964.0]    19665
(964.0, 1928.0]                 1867
(1928.0, 2892.0]                  75
(2892.0, 3856.0]                   4
(3856.0, 4820.0]                   2
Name: sqft_basement, dtype: int64
```

In [214]:
```
fig=plt.figure(figsize=(10,5))
sns.distplot(X['sqft_basement'],color='blue',kde=False)
plt.title('Distribution For interior housing space that is below ground level
plt.ylabel('Count')
plt.xlabel('interior housing space Below Ground(sqft_basement)')
```

Out[214]:
```
Text(0.5, 0, 'interior housing space Below Ground(sqft_basement)')
```



The sqft_basement column looks perfectly fine with all the understandable or comprehendible values

In [215]:
```
X['yr_built'].unique()
```

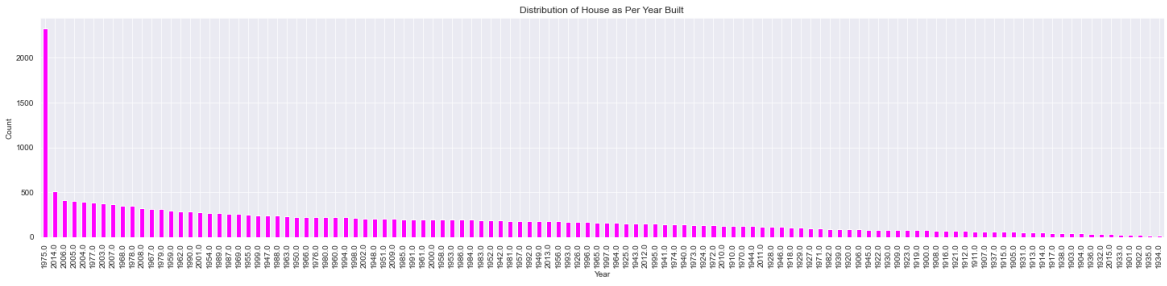
Out[215]:
```
array([1955., 1951., 1933., 1965., 1987., 2001., 1995., 1963., 1960.,
       2003., 1942., 1927., 1975., 1900., 1979., 1994., 1921., 1969.,
       1947., 1968., 1985., 1941., 1915., 1909., 2005., 1929., 1981.,
       1930., 1904., 1996., 2000., 1984., 2014., 1922., 1959., 1966.,
       1953., 1950., 2008., 1991., 1954., 1973., 1925., 1989., 1972.,
       1986., 1956., 2002., 1992., 1948., 1964., 1952., 1961., 2006.,
       1988., 1962., 1939., 1946., 1967., 1980., 1910., 1978., 1905.,
       1971., 2010., 1945., 1924., 1990., 1914., 1926., 2004., 1923.,
       2007., 1916., 1976., 1949., 1999., 1993., 1901., 1977., 1920.,
       1997., 1943., 1983., 1957., 1940., 1918., 1928., 1974., 1911.,
       1937., 1982., 1908., 1931., 1998., 1913., 2013., 1907., 1958.,
       2012., 1912., 2011., 1917., 1932., 1944., 1902., 2009., 1903.,
       1970., 2015., 1938., 1919., 1906., 1936., 1935., 1934.])
```

In [216]:
```
X['yr_built'].value_counts()
```

Out[216]:
```
1975.0    2327
2014.0     515
2006.0     410
2005.0     400
2004.0     394
          ...
1933.0      29
1901.0      25
1902.0      25
1935.0      21
1934.0      17
Name: yr_built, Length: 116, dtype: int64
```

In [217]:
```
fig=plt.figure(figsize=(25,5))
X['yr_built'].value_counts().plot.bar(color='magenta')
plt.title('Distribution of House as Per Year Built')
plt.xlabel('Year')
plt.ylabel('Count')
```

Out[217]:
```
Text(0, 0.5, 'Count')
```



The Year Built column looks perfectly fine with all the understandable or comprehendible values

In [218]:
```
X['yr_renovated'].unique()
```

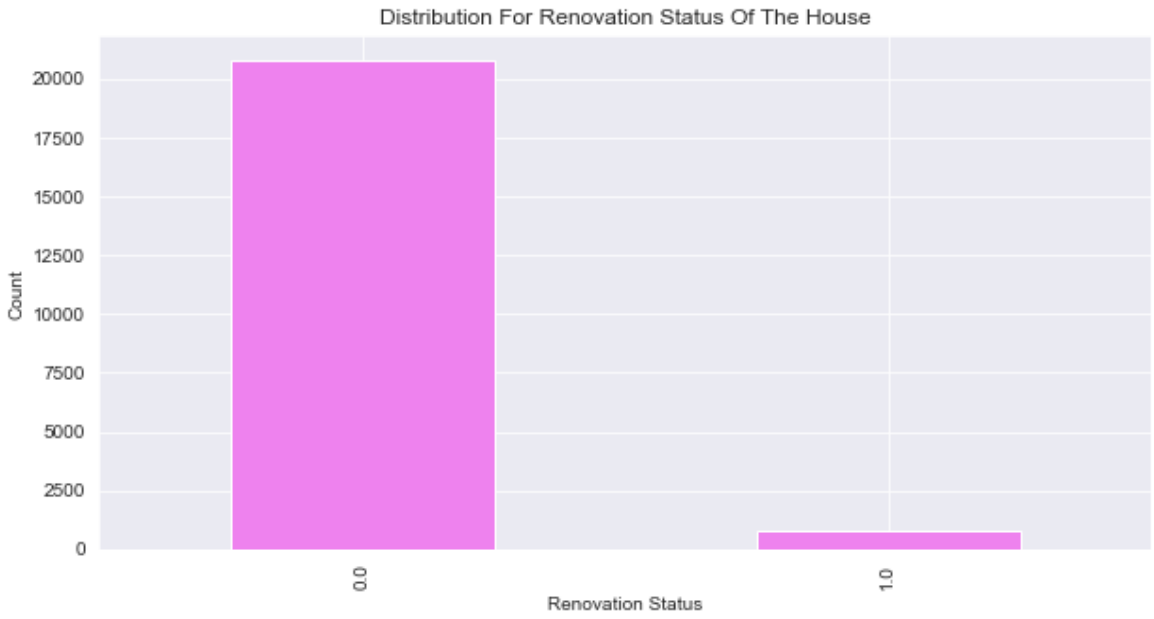
Out[218]:
```
array([0., 1.])
```

In [219]:
```
X['yr_renovated'].value_counts()
```

Out[219]:
```
0.0    20790
1.0      823
Name: yr_renovated, dtype: int64
```

In [220]:
```
fig=plt.figure(figsize=(10,5))
X['yr_renovated'].value_counts().plot.bar(color='violet')
plt.title('Distribution For Renovation Status Of The House')
plt.ylabel('Count')
plt.xlabel('Renovation Status')
```

Out[220]: Text(0.5, 0, 'Renovation Status')



The yr_renovated column looks perfectly fine with all the understandable or comprehendible values

In [221]:
```
X['sqft_living15'].unique()
```

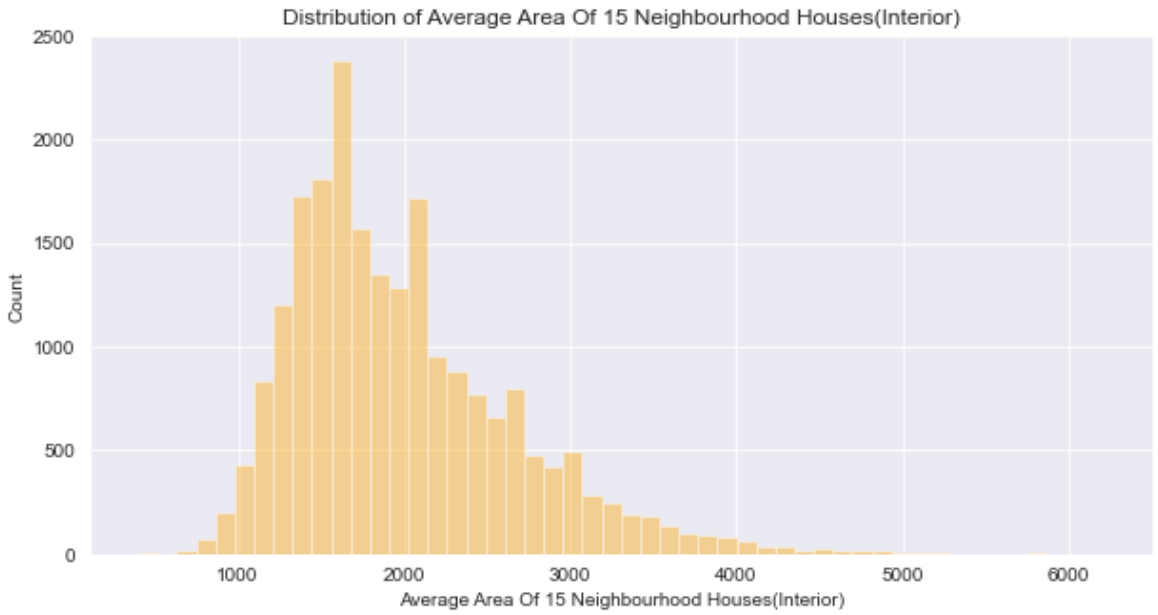
Out[221]:
```
array([1677.11642038, 1690.        , 2720.        , 1360.        ,
       1800.        , 4760.        , 2238.        , 1650.        ,
       2390.        , 2210.        , 1330.        , 1370.        ,
       2140.        , 1890.        , 1610.        , 1060.        ,
       1280.        , 1400.        , 4110.        , 2240.        ,
       1220.        , 2200.        , 1030.        , 1760.        ,
       1860.        , 1520.        , 2630.        , 2580.        ,
       1390.        , 1460.        , 1570.        , 2020.        ,
       1590.        , 2160.        , 1730.        , 1290.        ,
       2620.        , 2470.        , 2410.        , 3625.        ,
       1580.        , 1340.        , 3050.        , 1228.        ,
       2680.        ,  970.        , 1190.        , 1990.        ,
       1410.        , 1480.        , 2065.35523655, 1950.        ,
       2250.        , 2690.        , 2960.        , 2270.        ,
       2570.        , 1440.        , 2750.        , 2221.        ,
       1010.        , 3390.        , 3530.        , 1640.        ,
       1510.        , 2420.        , 3240.        , 1680.        ,
       1130.        , 3350.        , 1720.        , 1850.        ,
       1900.        , 1980.        , 2520.        , 1350.        ,
```

In [222]:
```
X['sqft_living15'].value_counts(bins=5)
```

Out[222]:
```
(1561.2, 2723.4]                12370
(393.18800000000005, 1561.2]     6310
(2723.4, 3885.6]                 2614
(3885.6, 5047.8]                  300
(5047.8, 6210.0]                   19
Name: sqft_living15, dtype: int64
```

In [223]:
```
fig = plt.figure(figsize=(10,5))
sns.distplot(X['sqft_living15'],color='orange',kde=False)
plt.title('Distribution of Average Area Of 15 Neighbourhood Houses(Interior)'
plt.xlabel('Average Area Of 15 Neighbourhood Houses(Interior)')
plt.ylabel('Count')
```

Out[223]:
```
Text(0, 0.5, 'Count')
```



The sqft_living15 column looks perfectly fine with all the understandable or comprehendible values

In [224]:
```
X['sqft_lot15'].unique()
```

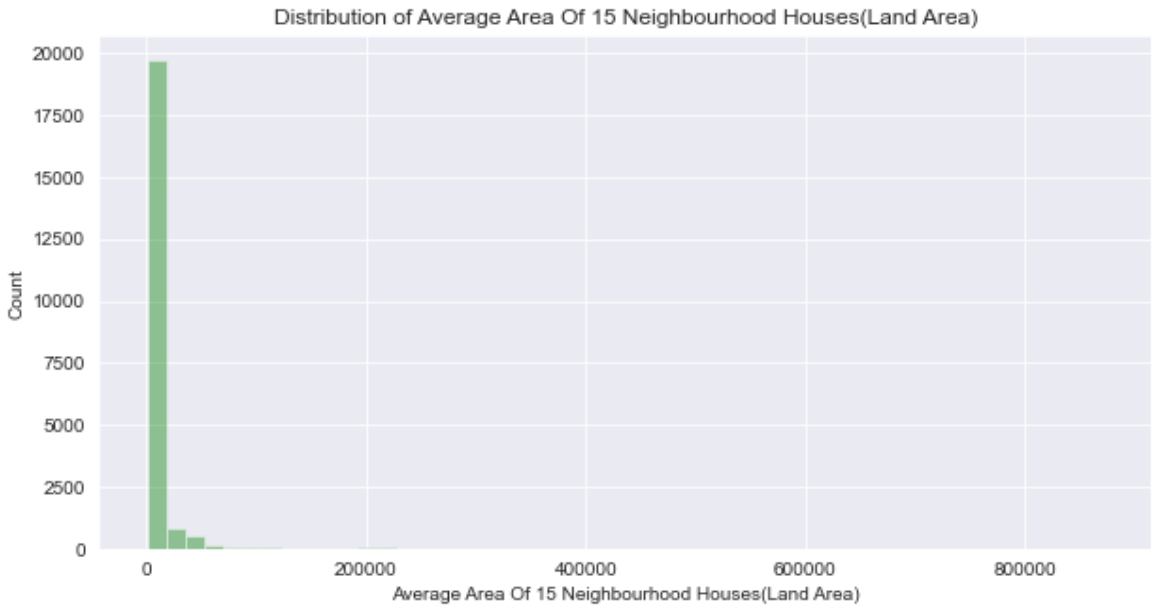
Out[224]:
```
array([5650., 7639., 8062., ..., 5731., 1509., 2007.])
```

In [225]:
```
X['sqft_lot15'].value_counts(bins=5)
```

Out[225]:
```
(-219.55, 174760.8]     21429
(174760.8, 348870.6]      170
(348870.6, 522980.4]       11
(697090.2, 871200.0]        2
(522980.4, 697090.2]        1
Name: sqft_lot15, dtype: int64
```

In [226]:
```
fig = plt.figure(figsize=(10,5))
sns.distplot(X['sqft_lot15'],color='green',kde=False)
plt.title('Distribution of Average Area Of 15 Neighbourhood Houses(Land Area)')
plt.xlabel('Average Area Of 15 Neighbourhood Houses(Land Area)')
plt.ylabel('Count')
```

Out[226]:
```
Text(0, 0.5, 'Count')
```



The sqft_lot15 column looks perfectly fine with all the understandable or comprehendible values

In [227]:
```
y.value_counts(bins=5)
```

Out[227]:
```
(67374.999, 1600000.0]    21187
(1600000.0, 3125000.0]      388
(3125000.0, 4650000.0]       30
(4650000.0, 6175000.0]        5
(6175000.0, 7700000.0]        3
Name: price, dtype: int64
```

In [228]:
```
fig = plt.figure(figsize=(10,5))
sns.distplot(y,kde=False)
plt.title('Distribution of Price Of The Houses')
plt.xlabel('Price in Millions')
plt.ylabel('Count')
```

Out[228]: Text(0, 0.5, 'Count')

In [229]:
```
X.head()
```

Out[229]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sq
0	3.0	1.00	1180.000000	5650.0	1.0	0.0	0.0	3.0	7.0	
1	3.0	2.25	2570.000000	7242.0	2.0	0.0	0.0	3.0	7.0	
2	2.0	1.00	770.000000	10000.0	1.0	0.0	0.0	3.0	6.0	
3	4.0	3.00	1960.000000	5000.0	1.0	0.0	0.0	5.0	7.0	
4	3.0	2.00	2181.810443	8080.0	1.0	0.0	0.0	3.0	8.0	

In [230]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       21613 non-null  float64
 1   bathrooms      21613 non-null  float64
 2   sqft_living    21613 non-null  float64
 3   sqft_lot       21613 non-null  float64
 4   floors         21613 non-null  float64
 5   waterfront     21613 non-null  float64
 6   view           21613 non-null  float64
 7   condition      21613 non-null  float64
 8   grade          21613 non-null  float64
 9   sqft_above     21613 non-null  float64
 10  sqft_basement  21613 non-null  float64
 11  yr_built       21613 non-null  float64
 12  yr_renovated   21613 non-null  float64
 13  sqft_living15  21613 non-null  float64
 14  sqft_lot15     21613 non-null  float64
dtypes: float64(15)
memory usage: 2.5 MB
```

In [231]:
```
X.describe()
```

Out[231]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	2
count	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	
mean	3.378892	2.095093	2077.623664	1.417626e+04	1.481539	0.006894	
std	0.908661	0.776276	900.450808	3.847108e+04	0.535248	0.082745	
min	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	
25%	3.000000	1.500000	1450.000000	5.314000e+03	1.000000	0.000000	
50%	3.000000	2.250000	1909.500000	7.605000e+03	1.000000	0.000000	
75%	4.000000	2.500000	2520.000000	1.012500e+04	2.000000	0.000000	
max	11.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	

In [232]:
```
y.head()
```

Out[232]:
```
0    221900.0
1    538000.0
2    180000.0
3    604000.0
4    510000.0
Name: price, dtype: float64
```

Train-Test Split

Splitting The Data Into Train and Test For Model Evaluation Purposes

For this Case we take 75% of the data as training data and 25% of the data as testing data

In [233]:
```
X_train  , X_test , y_train , y_test = train_test_split(X,y,random_state = 0)
```

In [234]:
```
X_train.shape
```

Out[234]: (16209, 15)

In [235]:
```
X_test.shape
```

Out[235]: (5404, 15)

In [236]:
```
y_test.shape
```

Out[236]: (5404,)

In [237]:
```
y_train.shape
```

Out[237]: (16209,)

Data Scaling

Scaling for better Predictive Accuracy as the variables have different ranges

What Type of Scaling to use?

MinMax Scaling is the scaling type that has been chosen for this project because as we have witnessed before the distribution of the variables in our dataset is highly skewed and applying Standard Scaler would only have made sense if the variables were normally distributed to start with as it normalizes the variable with mean 0 and SD 1. As the distribution of variable is not normal hence applying Min Max scaling makes more sense here as it dosen't assume any distribution of variables and dose not distort it either as it only just scales the variables between 0 and 1.

In [238]:
```
scaler = MinMaxScaler()
```

In [239]:
```
X_trn = scaler.fit_transform(X_train)
X_tst = scaler.transform(X_test)
```

In [240]:
```
X_train = pd.DataFrame(X_trn,columns=X_train.columns)
X_test = pd.DataFrame(X_tst,columns=X_test.columns)
```

In [241]:
```
X_train.head()
```

Out[241]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
0	0.090909	0.12500	0.050566	0.024488	0.0	0.0	0.0	1.00	0.500000
1	0.272727	0.21875	0.092075	0.005573	0.0	0.0	0.0	0.50	0.500000
2	0.272727	0.28125	0.104151	0.003570	0.0	0.0	0.0	0.75	0.583333
3	0.272727	0.25000	0.194351	0.116049	0.0	0.0	0.0	0.75	0.666667
4	0.272727	0.12500	0.105660	0.007863	0.0	0.0	0.0	0.75	0.500000

Predictive Models

The metric that we are going to use to access the model is going to be the R-Squared it basically tells us the amount of variation in the dependent variable(price) that is explaned by the explanatory variables or the predictor variables for a model. It ranges from 0 to 1. One being the best value or the ideal value and 0 being the worst possible value

A model will have certainly higher predictive power if it explains higher amount of variation in the dependent variable by the predictors. A model with the higher value of R-Squared will have better predictive power and will make better predictions so the best model would be the one that will have the highest R-Squared Value.

Model - 1 KNN Regressor

In [242]:
```
knn = KNeighborsRegressor()
```

Default Value Of K is taken which is 5

In [243]:
```
knn.fit(X_train,y_train)
```

Out[243]:
```
KNeighborsRegressor()
```

In [244]:
```
prediction_knn = knn.predict(X_test)
```

In [245]:
```
# Training Score
```

In [246]:
```
knn.score(X_train,y_train)
```

Out[246]:
```
0.7565876015012198
```

In [247]:
```
# Test Score
```

In [248]:
```
knn.score(X_test,y_test)
```

Out[248]: 0.629922795672869

In [249]:
```
from sklearn import metrics
```

In [250]:
```
rmse_knn = metrics.mean_squared_error(y_test,prediction_knn)**(0.5)
print('The RMSE of Knn with 5 neighbours is {}'.format(round(rmse_knn,2)))
```

```
The RMSE of Knn with 5 neighbours is 221723.42
```

Hyperparameter Tuning To Find The Best Value Of K

In [251]:
```
n = list(range(1,51))
#lf_size = list(range(1,51))
```

In [252]:
```
knn = KNeighborsRegressor()
```

In [253]:
```
# grid_params  = {'n_neighbors':n,'leaf_size':lf_size}
```

In [254]:
```
grid_params  = {'n_neighbors':n}
```

In [255]:
```
kfold = KFold(n_splits=5,random_state=0)
```

```
C:\Users\behab\Anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

In [256]:
```
grid = GridSearchCV(knn,grid_params,cv=kfold,verbose=3)
```

In [257]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
[CV] n_neighbors=1 ...................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.

[CV] ....................... n_neighbors=1, score=0.463, total=   0.6s
[CV] n_neighbors=1 ...................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.5s remaining:
0.0s

[CV] ....................... n_neighbors=1, score=0.317, total=   0.6s
[CV] n_neighbors=1 ...................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.1s remaining:
0.0s
```

In [258]:
```
grid.best_params_
```

Out[258]:
```
{'n_neighbors': 11}
```

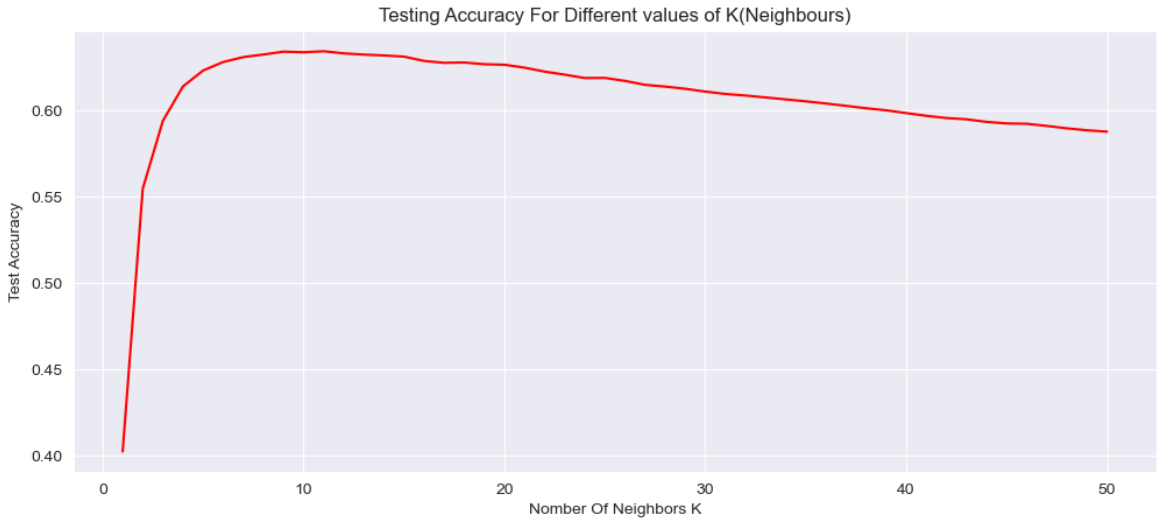
In [259]:
```
grid.best_score_
```

Out[259]:
```
0.6345814346206223
```

In [260]:
```
results_knn = pd.DataFrame(grid.cv_results_)
```

In [261]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(data=results_knn,x=results_knn['param_n_neighbors'],y=results_kn
plt.xlabel('Nomber Of Neighbors K')
plt.ylabel('Test Accuracy')
plt.title('Testing Accuracy For Different values of K(Neighbours)')
```

Out[261]: Text(0.5, 1.0, 'Testing Accuracy For Different values of K(Neighbours)')



In [262]:
```
results_knn.head()
```

Out[262]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	pa
0	0.136422	0.011161	0.497081	0.061471	1	{'n_neighl
1	0.163769	0.057086	0.592615	0.134181	2	{'n_neighl
2	0.124676	0.022462	0.523000	0.060143	3	{'n_neighl
3	0.109531	0.008449	0.479303	0.045087	4	{'n_neighl
4	0.121114	0.022475	0.519196	0.070092	5	{'n_neighl

In [263]:
```
knn = KNeighborsRegressor(n_neighbors=8)
```

In [264]:
```
cross_val_knn = cross_val_score(knn,X_train,y_train,cv=kfold,verbose=3)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.

[CV]  ................................................................
[CV] ................................... , score=0.595, total=   0.7s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remaining:
0.0s

[CV] ................................... , score=0.599, total=   0.7s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.3s remaining:
0.0s

[CV] ................................... , score=0.631, total=   0.6s
[CV]  ................................................................
[CV] ................................... , score=0.689, total=   0.7s
[CV]  ................................................................
[CV] ................................... , score=0.650, total=   0.7s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    3.3s finished
```

In [265]:
```
knn_mean_score_cv = cross_val_knn.mean()
print('The cross validated R-Squared score of the best knn model with 10 neig
```

```
The cross validated R-Squared score of the best knn model with 10 neighbour
s is: 0.6327
```

This Means that the in this model the explanatory variables can successfully explain 62.99% of the variance in the price.

Model -2 Linear Regression

In [266]:
```
lm = LinearRegression()
```

In [267]:
```
lm.fit(X_train,y_train)
```

Out[267]:
```
LinearRegression()
```

In [268]:
```
lm.score(X_train,y_train)
```

Out[268]:
```
0.6337299251163646
```

In [269]:
```
lm.score(X_test,y_test)
```

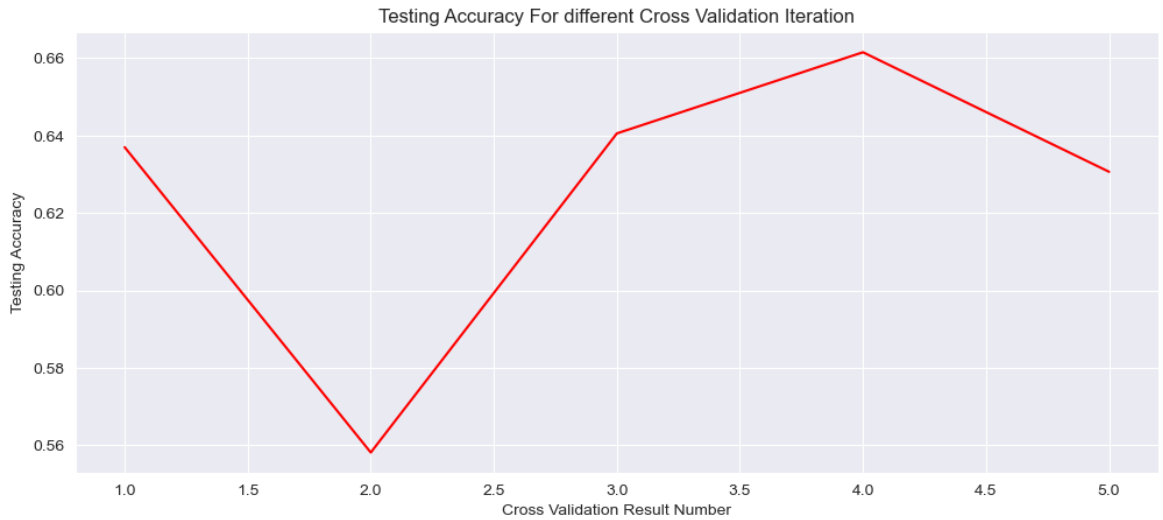
Out[269]:
```
0.6249205036278962
```

In [270]:
```
cross_val_lm = cross_val_score(lm,X_train,y_train,cv=kfold,verbose=3)
lm_mean_score_cv= cross_val_lm.mean()
```

```
[CV]  ................................................................
[CV] ................................... , score=0.637, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.558, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.641, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.662, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.631, total=   0.0s


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

In [271]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
plt.plot(np.arange(1,6),cross_val_lm,c='red')
plt.xlabel('Cross Validation Result Number')
plt.ylabel('Testing Accuracy')
plt.title('Testing Accuracy For different Cross Validation Iteration')
```

Out[271]: Text(0.5, 1.0, 'Testing Accuracy For different Cross Validation Iteration')



In [272]:
```
print('The cross validated Test score(R-Squared) of the best Linear model is:
```

```
The cross validated Test score(R-Squared) of the best Linear model is: 0.62
56
```

In [273]:
```
X_train.head()
```

Out[273]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
0	0.090909	0.12500	0.050566	0.024488	0.0	0.0	0.0	1.00	0.500000
1	0.272727	0.21875	0.092075	0.005573	0.0	0.0	0.0	0.50	0.500000
2	0.272727	0.28125	0.104151	0.003570	0.0	0.0	0.0	0.75	0.583333
3	0.272727	0.25000	0.194351	0.116049	0.0	0.0	0.0	0.75	0.666667
4	0.272727	0.12500	0.105660	0.007863	0.0	0.0	0.0	0.75	0.500000

This Means that the in this model the explanatory variables can successfully explain 62.03% of the variance in the price.

Model 3 - Ridge Regression

In [274]:
```
# With Default Value of Alpha = 1
```

In [275]:
```
ridge = Ridge(random_state=0)
```

In [276]:
```
ridge.fit(X_train,y_train)
```

Out[276]:
```
Ridge(random_state=0)
```

In [277]:
```
ridge.score(X_train,y_train)
```

Out[277]:
```
0.633646611285265
```

In [278]:
```
ridge.score(X_test,y_test)
```

Out[278]:
```
0.6248217938179781
```

In [279]:
```
ridge = Ridge(random_state=0)
```

In [280]:
```
alpha_range = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
```

In [281]:
```
grid_params = {'alpha':[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]}
```

In [282]:
```
grid = GridSearchCV(ridge,grid_params,cv=kfold,verbose=3)
```

In [283]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.637, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.558, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.641, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.662, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.631, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.637, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.558, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.641, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.662, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.631, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.637, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.558, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.641, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s


[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.662, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.631, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.637, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.558, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.641, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.662, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.631, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ............................. alpha=1, score=0.636, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ............................. alpha=1, score=0.559, total=   0.0s
[CV] alpha=1 .........................................................
```

```
[CV] ............................ alpha=1, score=0.640, total=   0.0s
[CV] alpha=1 .....................................................
[CV] ............................ alpha=1, score=0.662, total=   0.0s
[CV] alpha=1 .....................................................
[CV] ............................ alpha=1, score=0.631, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.627, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.565, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.635, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.662, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.629, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.568, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.553, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.581, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.627, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.590, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.314, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.352, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.322, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.373, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.347, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ......................... alpha=10000, score=0.058, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ......................... alpha=10000, score=0.069, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ......................... alpha=10000, score=0.059, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ......................... alpha=10000, score=0.071, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ......................... alpha=10000, score=0.067, total=   0.0s

[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    0.5s finished
```

Out[283]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=0, shuffle=False),
             estimator=Ridge(random_state=0),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 10
00,
                                   10000]},
             verbose=3)
```

In [284]:
```
grid.best_params_
```

Out[284]:
```
{'alpha': 1}
```

In [285]:
```
grid.best_score_
```

Out[285]:
```
0.6256162008940382
```

In [286]:
```
results_ridge = pd.DataFrame(grid.cv_results_)
```

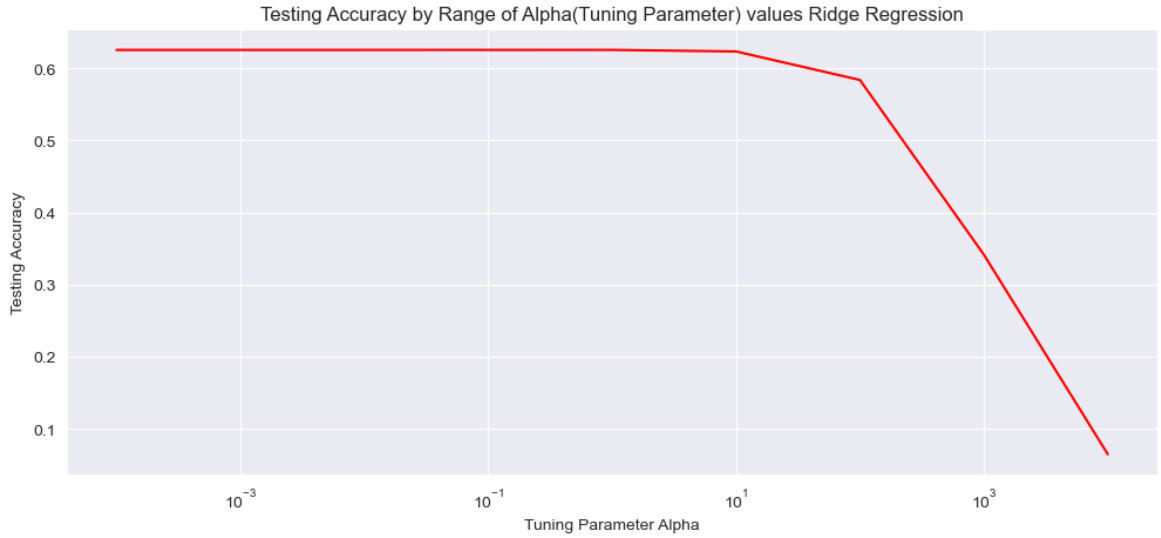
In [287]:
```
results_ridge.head()
```

Out[287]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_
0	0.010173	0.000746	0.003191	0.000399	0.0001	{'alpha': 0.0001}	
1	0.008577	0.001196	0.002992	0.000631	0.001	{'alpha': 0.001}	
2	0.009375	0.001352	0.003391	0.000488	0.01	{'alpha': 0.01}	
3	0.010571	0.000798	0.002792	0.000746	0.1	{'alpha': 0.1}	
4	0.008777	0.001163	0.003191	0.000746	1	{'alpha': 1}	

In [288]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(data=results_ridge,x='param_alpha',y='mean_test_score',color = '
plt.xlabel('Tuning Parameter Alpha')
plt.ylabel('Testing Accuracy')
plt.xscale('log')
plt.title('Testing Accuracy by Range of Alpha(Tuning Parameter) values Ridge
```

Out[288]:
```
Text(0.5, 1.0, 'Testing Accuracy by Range of Alpha(Tuning Parameter) values
Ridge Regression')
```



In [289]:
```
cross_val_ridge = cross_val_score(ridge,X_train,y_train,cv=kfold,verbose=3)
ridge_mean_score_cv= cross_val_ridge.mean()
```

```
[CV]  ................................................................
[CV] ................................. , score=0.636, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.559, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.640, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.662, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.631, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

In [290]:
```
print('The cross validated Testing score (R-Squared) of the best Ridge model
```

```
The cross validated Testing score (R-Squared) of the best Ridge model which
is a linear model initself is: 0.6256
```

This Means that the in this model the explanatory variables can successfully explain 62.03% of the variance in the price.

In [291]:
```
X_train.head()
```

Out[291]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
0	0.090909	0.12500	0.050566	0.024488	0.0	0.0	0.0	1.00	0.500000
1	0.272727	0.21875	0.092075	0.005573	0.0	0.0	0.0	0.50	0.500000
2	0.272727	0.28125	0.104151	0.003570	0.0	0.0	0.0	0.75	0.583333
3	0.272727	0.25000	0.194351	0.116049	0.0	0.0	0.0	0.75	0.666667
4	0.272727	0.12500	0.105660	0.007863	0.0	0.0	0.0	0.75	0.500000

Model 4 - Lasso Regression

In [292]:
```
lasso = Lasso(random_state=0)
```

In [293]:
```
lasso.fit(X_train,y_train)
```

Out[293]:
```
Lasso(random_state=0)
```

In [294]:
```
lasso.score(X_train,y_train)
```

Out[294]:
```
0.6337299002646608
```

In [295]:
```
lasso.score(X_test,y_test)
```

Out[295]:
```
0.6249145999819712
```

In [296]:
```
alpha_range = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
grid_params = {'alpha':[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]}
```

In [297]:
```
grid = GridSearchCV(lasso,grid_params,cv=kfold,verbose=3)
```

In [298]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] alpha=0.0001 ....................................................
[CV] ........................ alpha=0.0001, score=0.637, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ........................ alpha=0.0001, score=0.558, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ........................ alpha=0.0001, score=0.641, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ........................ alpha=0.0001, score=0.662, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s


[CV] alpha=0.0001 ....................................................
[CV] ........................ alpha=0.0001, score=0.631, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ......................... alpha=0.001, score=0.637, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ......................... alpha=0.001, score=0.558, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ......................... alpha=0.001, score=0.641, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ......................... alpha=0.001, score=0.662, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ......................... alpha=0.001, score=0.631, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] .......................... alpha=0.01, score=0.637, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] .......................... alpha=0.01, score=0.558, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] .......................... alpha=0.01, score=0.641, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] .......................... alpha=0.01, score=0.662, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] .......................... alpha=0.01, score=0.631, total=   0.1s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.637, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.558, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.641, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.662, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.631, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ............................ alpha=1, score=0.637, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ............................ alpha=1, score=0.558, total=   0.0s
[CV] alpha=1 .........................................................
```

```
[CV] ............................ alpha=1, score=0.641, total=   0.0s
[CV] alpha=1 .....................................................
[CV] ............................ alpha=1, score=0.662, total=   0.0s
[CV] alpha=1 .....................................................
[CV] ............................ alpha=1, score=0.631, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.637, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.558, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.640, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.662, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.631, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.635, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.560, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.639, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.662, total=   0.0s
[CV] alpha=100 ...................................................
[CV] .......................... alpha=100, score=0.632, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.614, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.565, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.624, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.654, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ......................... alpha=1000, score=0.624, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................ alpha=10000, score=0.377, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................ alpha=10000, score=0.428, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................ alpha=10000, score=0.399, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................ alpha=10000, score=0.456, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................ alpha=10000, score=0.421, total=   0.0s

[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    1.4s finished
```

Out[298]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=0, shuffle=False),
             estimator=Lasso(random_state=0),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 10
00,
                                   10000]},
             verbose=3)
```

In [299]:
```
grid.best_params_
```

Out[299]:
```
{'alpha': 100}
```

In [300]:
```
grid.best_score_
```

Out[300]:
```
0.6256330292321816
```

In [301]:
```
results_lasso = pd.DataFrame(grid.cv_results_)
```

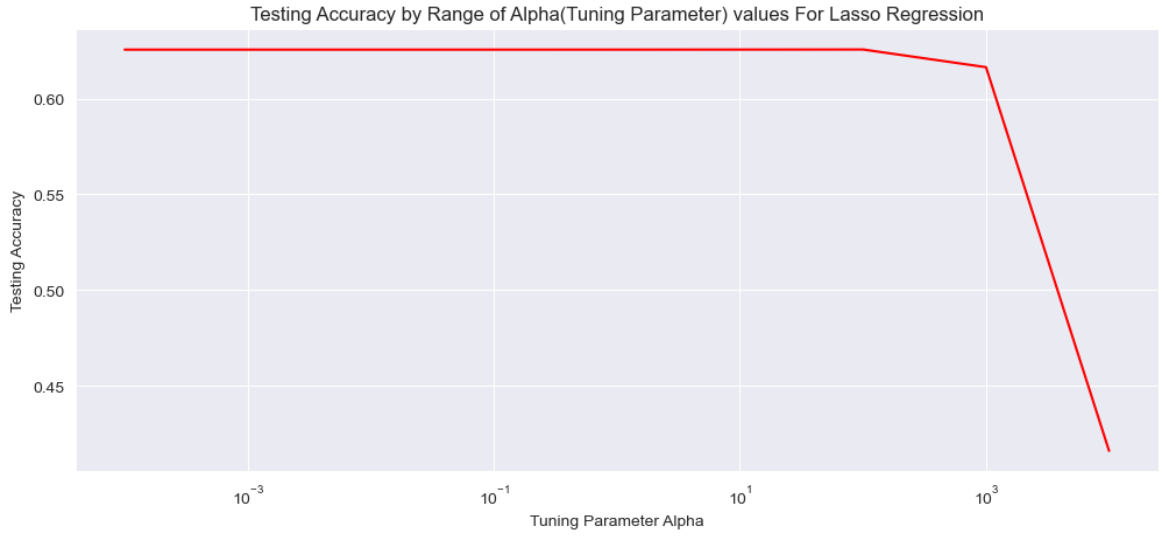
In [302]:
```
results_lasso.head()
```

Out[302]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_
0	0.041689	0.002918	0.003391	0.001018	0.0001	{'alpha': 0.0001}	
1	0.039095	0.001163	0.003790	0.000399	0.001	{'alpha': 0.001}	
2	0.036702	0.012669	0.003591	0.001017	0.01	{'alpha': 0.01}	
3	0.031915	0.000892	0.003391	0.000488	0.1	{'alpha': 0.1}	
4	0.028922	0.001261	0.003790	0.000399	1	{'alpha': 1}	

In [303]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(data=results_lasso,x='param_alpha',y='mean_test_score',color = '
plt.xlabel('Tuning Parameter Alpha')
plt.ylabel('Testing Accuracy')
plt.xscale('log')
plt.title('Testing Accuracy by Range of Alpha(Tuning Parameter) values For La
```

Out[303]:
```
Text(0.5, 1.0, 'Testing Accuracy by Range of Alpha(Tuning Parameter) values
For Lasso Regression')
```



In [304]:
```
lasso = Lasso(alpha=100)
```

In [305]:
```
cross_val_lasso = cross_val_score(lasso,X_train,y_train,cv=kfold,verbose=3)
lasso_mean_score_cv = cross_val_lasso.mean()
```

```
[CV]  ................................................................
[CV] ................................... , score=0.635, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.560, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.639, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.662, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.632, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

In [306]:
```
print('The cross validated Testing score(R-Squared) of the best Lasso model i
```

```
The cross validated Testing score(R-Squared) of the best Lasso model is: 0.
6256
```

This Means that the in this model the explanatory variables can successfully explain 62.03% of the variance in the price.

In [307]:
```
X_train.head()
```

Out[307]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
0	0.090909	0.12500	0.050566	0.024488	0.0	0.0	0.0	1.00	0.500000
1	0.272727	0.21875	0.092075	0.005573	0.0	0.0	0.0	0.50	0.500000
2	0.272727	0.28125	0.104151	0.003570	0.0	0.0	0.0	0.75	0.583333
3	0.272727	0.25000	0.194351	0.116049	0.0	0.0	0.0	0.75	0.666667
4	0.272727	0.12500	0.105660	0.007863	0.0	0.0	0.0	0.75	0.500000

Model 5 -Decision Tree Regressor

In [308]:
```
tree = DecisionTreeRegressor(random_state=0)
```

In [309]:
```
tree.fit(X_train,y_train)
```

Out[309]:
```
DecisionTreeRegressor(random_state=0)
```

In [310]:
```
tree.score(X_train,y_train)
```

Out[310]:
```
0.9999287418430224
```

In [311]:
```
tree.score(X_test,y_test)
```

Out[311]:
```
0.46837168901280835
```

In [312]:
```
depth = list(range(1,101))
```

In [313]:
```
grid_params = {'max_depth':depth}
```

In [314]:
```
grid = GridSearchCV(tree,grid_params,cv=kfold,verbose=3)
```

In [315]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
[CV] max_depth=1 .....................................................
[CV] ........................ max_depth=1, score=0.309, total=   0.0s
[CV] max_depth=1 .....................................................
[CV] ........................ max_depth=1, score=0.285, total=   0.0s
[CV] max_depth=1 .....................................................
[CV] ........................ max_depth=1, score=0.300, total=   0.0s
[CV] max_depth=1 .....................................................
[CV] ........................ max_depth=1, score=0.332, total=   0.0s
[CV] max_depth=1 .....................................................
[CV] ........................ max_depth=1, score=0.299, total=   0.0s
[CV] max_depth=2 .....................................................
[CV] ........................ max_depth=2, score=0.456, total=   0.0s
[CV] max_depth=2 .....................................................
[CV] ........................ max_depth=2, score=0.394, total=   0.0s
[CV] max_depth=2 .....................................................
[CV] ........................ max_depth=2, score=0.469, total=   0.0s
[CV] max_depth=2 .....................................................
[CV] ........................ max_depth=2, score=0.463, total=   0.0s
```

In [316]:
```
grid.best_params_
```

Out[316]:
```
{'max_depth': 6}
```

In [317]:
```
grid.best_score_
```

Out[317]:
```
0.590926130248538
```

In [318]:
```
results_tree = pd.DataFrame(grid.cv_results_)
```

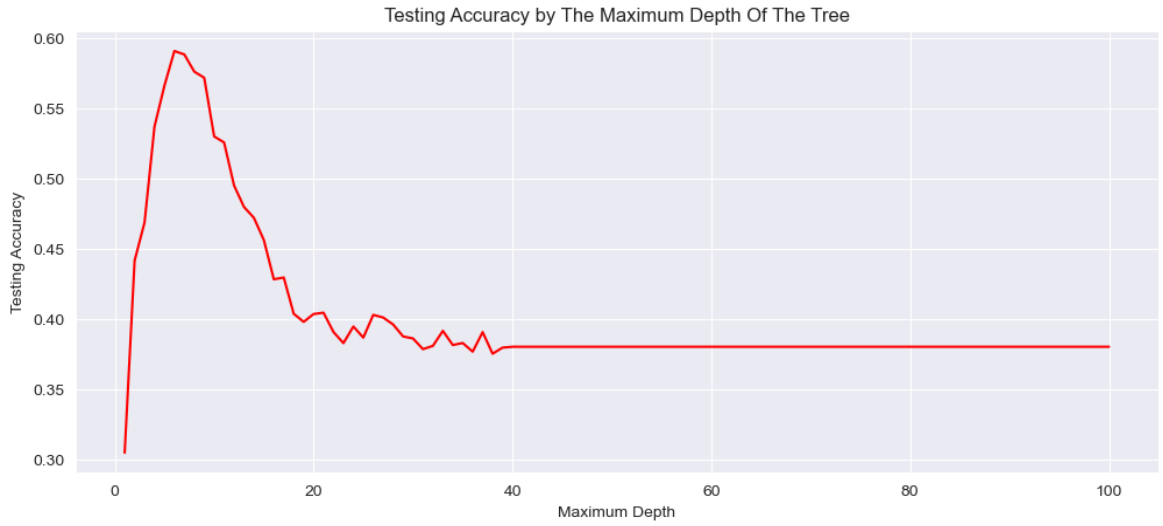
In [319]:
```
results_tree.head()
```

Out[319]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	parar
0	0.012774	0.000746	0.002602	0.000491	1	{'max_dept
1	0.020545	0.000797	0.002393	0.000489	2	{'max_dept
2	0.028352	0.001008	0.001994	0.000004	3	{'max_dept
3	0.035704	0.001467	0.002394	0.000489	4	{'max_dept
4	0.037116	0.001176	0.002192	0.000384	5	{'max_dept

In [320]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(data=results_tree,x='param_max_depth',y='mean_test_score',color=
plt.xlabel('Maximum Depth')
plt.ylabel('Testing Accuracy')
plt.title('Testing Accuracy by The Maximum Depth Of The Tree')
```

Out[320]: Text(0.5, 1.0, 'Testing Accuracy by The Maximum Depth Of The Tree')



In [321]:
```
tree = DecisionTreeRegressor(max_depth=6)
```

In [322]:
```
tree.fit(X_train,y_train)
```

Out[322]: DecisionTreeRegressor(max_depth=6)

In [323]:
```
cross_val_tree = cross_val_score(tree,X_train,y_train,verbose=3,cv=kfold)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s

[CV]  ................................................................
[CV] ................................. , score=0.624, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.536, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.578, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.600, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.542, total=   0.0s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.1s finished
```

In [324]:
```
tree_mean_score_cv = cross_val_tree.mean()
```

In [325]:
```
tree.feature_importances_
```

Out[325]:
```
array([0.        , 0.0065391 , 0.73395656, 0.00445211, 0.        ,
       0.03354512, 0.01571871, 0.        , 0.0751867 , 0.00191014,
       0.        , 0.08016478, 0.        , 0.03999626, 0.00853052])
```

In [326]:
```
output = {'Variables':X_train.columns,'Feature Importances':tree.feature_impo
```

In [327]:
```
out = pd.DataFrame(output)
```

In [328]:
```
out.head()
```

Out[328]:

	Variables	Feature Importances
0	bedrooms	0.000000
1	bathrooms	0.006539
2	sqft_living	0.733957
3	sqft_lot	0.004452
4	floors	0.000000

In [329]:
```
out =out.sort_values(by='Feature Importances',ascending =False)
```

Top 5 Variables That Have The Most Impact on the Price Of The Houses are as follows:

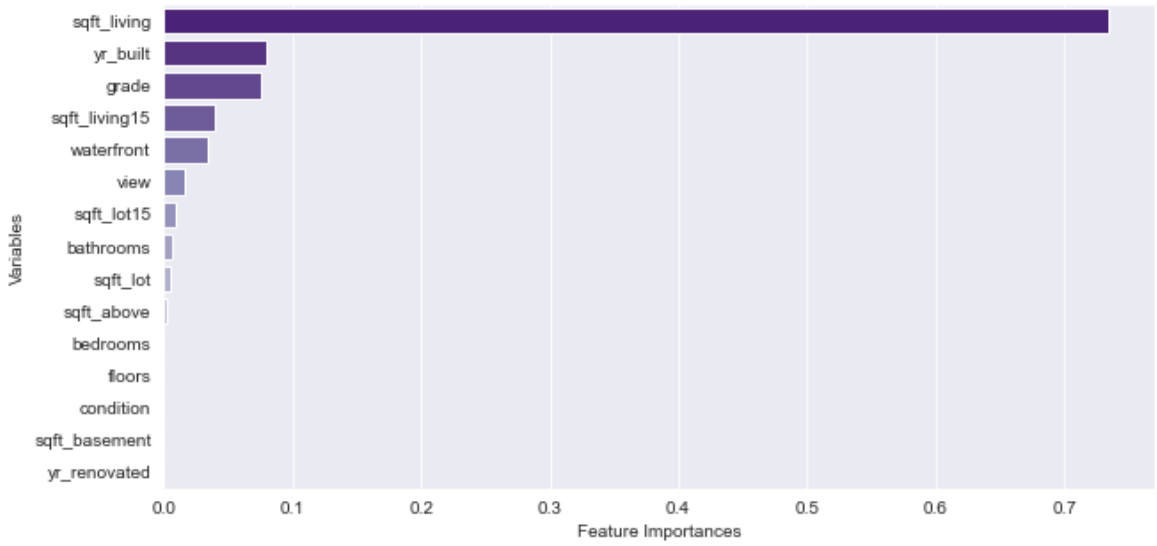
In [330]:
```
out.head()
```

Out[330]:

	Variables	Feature Importances
2	sqft_living	0.733957
11	yr_built	0.080165
8	grade	0.075187
13	sqft_living15	0.039996
5	waterfront	0.033545

In [331]:
```
fig = plt.figure(figsize=(10,5))
sns.barplot(y='Variables',x='Feature Importances',data=out,orient='h',palette
```

Out[331]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17fe1e18908>
```



Square Foot of the house has the most and very high impact on the price of the house followed by the year in which the house was built followed by the grade of the house which is essentially the type of construction that the house has had followed by the average living area of 15 houses in the neighbourhood followed by whether the house had a waterfront or not

The Variables like condition of the house, the number of floors the house had, the number of bedrooms the house has, whether or not the house was ever renovated before, area of the basement and area of the plot have little to no impact on the dependent variable which is price and hence can be ignored when trying to fit more complex models.

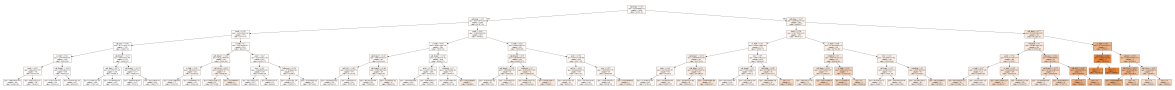
The Area of the house has very significant in determining the price of the house, It is so important that it dominates most of the other variables over a large margin.

Visualizing The Decision Tree

In [332]:
```
features = list(X_train.columns)
```

In [333]:
```
fig=plt.figure(figsize=(10,10))
dot_data = StringIO()
export_graphviz(tree, out_file=dot_data,feature_names=features,filled=True,rc

graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

Out[333]:



```
<Figure size 720x720 with 0 Axes>
```

In [334]:
```
print('The cross validated Test score (R-Squared) of the best Decision Tree R
```

```
The cross validated Test score (R-Squared) of the best Decision Tree Regres
sor model is: 0.576
```

Sampling The Data For More Complicated Datasets

In Order to implement more complicated algorithms a sample of 1500 datapoints from the training set is used to train some of the more complicated models like Random Forest, Support Vector Machines With Kernel Tricks(Linear, RBF, Poly) and LinearSVM

In [335]:
```
np.random.seed(0)
sample_train = np.random.randint(5404,16209,1500)
sample_test = np.random.randint(0,5404,1000)
```

In [336]:
```
X_train_sample = X_train.iloc[sample_train,:]
```

In [337]:
```
y_train_sample = y_train.iloc[sample_train]
```

In [338]:
```
X_test_sample = X_train.iloc[sample_test,:]
```

In [339]:
```
y_test_sample = y_test.iloc[sample_test]
```

In [340]:
```
X_train_sample.shape
```

Out[340]: (1500, 15)

In [341]:
```
y_train_sample.shape
```

Out[341]: (1500,)

In [342]:
```
X_test_sample.shape
```

Out[342]: (1000, 15)

In [343]:
```
y_test_sample.shape
```

Out[343]: (1000,)

Model 6 - Random Forest Regressor

In [344]:
```
rf = RandomForestRegressor()
```

In [345]:
```
rf.fit(X_train_sample,y_train_sample)
```

Out[345]: RandomForestRegressor()

In [346]:
```
rf.score(X_train_sample,y_train_sample)
```

Out[346]: 0.9527245786017229

In [347]:
```
rf.score(X_test_sample,y_test_sample)
```

Out[347]: -0.4706336276014731

In [348]:
```
grid_params = {'n_estimators':[400,500],'max_depth':depth[:51]}
```

In [349]:
```
grid = GridSearchCV(rf,grid_params,cv=kfold,verbose=3)
```

In [350]:
```
grid.fit(X_train_sample,y_train_sample)
```

```
Fitting 5 folds for each of 102 candidates, totalling 510 fits
[CV] max_depth=1, n_estimators=400 ....................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.

[CV] ....... max_depth=1, n_estimators=400, score=0.315, total=   0.7s
[CV] max_depth=1, n_estimators=400 ....................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remaining:
0.0s

[CV] ....... max_depth=1, n_estimators=400, score=0.327, total=   0.7s
[CV] max_depth=1, n_estimators=400 ....................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.4s remaining:
0.0s
```

In [351]:
```
grid.best_params_
```

Out[351]:
```
{'max_depth': 29, 'n_estimators': 500}
```

In [352]:
```
grid.best_score_
```

Out[352]:
```
0.6576623021409377
```

In [353]:
```
results_rf = pd.DataFrame(grid.cv_results_)
```

In [354]:
```
results_rf.head()
```

Out[354]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_
0	0.766948	0.091129	0.034108	0.004343	1	
1	0.922745	0.043632	0.041280	0.003927	1	
2	1.117818	0.111980	0.045487	0.010997	2	
3	1.009897	0.180957	0.036109	0.011015	2	
4	1.034838	0.146811	0.029907	0.004131	3	

In [355]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
#sns.heatmap(results_rf.pivot_table(index='param_n_estimators',columns='param
sns.lineplot(x=results_rf['param_max_depth'],y=results_rf['mean_test_score'],
plt.ylabel('Number Of Trees(param_n_estimators)')
plt.xlabel('Maximum Depth')
plt.title('HeatMap Of Mean Test Score With respect To param_n_estimators and
```

Out[355]:
```
Text(0.5, 1.0, 'HeatMap Of Mean Test Score With respect To param_n_estimato
rs and depth parameters')
```

In [356]:
```
rf = RandomForestRegressor(n_estimators=400,max_depth=40)
```

In [357]:
```
cross_val_rf = cross_val_score(rf,X_train_sample,y_train_sample,cv=kfold,verb
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.

[CV]  ................................................................
[CV] ................................. , score=0.674, total=   2.4s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.3s remaining:
0.0s

[CV] ................................. , score=0.628, total=   2.5s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    4.8s remaining:
0.0s

[CV] ................................. , score=0.653, total=   2.6s
[CV]  ................................................................
[CV] ................................. , score=0.620, total=   2.4s
[CV]  ................................................................
[CV] ................................. , score=0.685, total=   2.4s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   12.2s finished
```

In [358]:
```
rf_mean_score_cv = cross_val_rf.mean()
```

In [359]:
```
print('The cross validated Testing score(R-Squared) of the best Random Forest
```

```
The cross validated Testing score(R-Squared) of the best Random Forest Regr
essor model is: 0.6519
```

This states that 67.38% of the variation in the price can be explaned by our explanatory variables in the Random Forest Model.

Model 7 : Polynomial Regression

In [360]:
```
X.head()
```

Out[360]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sq
0	3.0	1.00	1180.000000	5650.0	1.0	0.0	0.0	3.0	7.0	
1	3.0	2.25	2570.000000	7242.0	2.0	0.0	0.0	3.0	7.0	
2	2.0	1.00	770.000000	10000.0	1.0	0.0	0.0	3.0	6.0	
3	4.0	3.00	1960.000000	5000.0	1.0	0.0	0.0	5.0	7.0	
4	3.0	2.00	2181.810443	8080.0	1.0	0.0	0.0	3.0	8.0	

In [361]:
```
y.head()
```

Out[361]:
```
0    221900.0
1    538000.0
2    180000.0
3    604000.0
4    510000.0
Name: price, dtype: float64
```

In [362]:
```
lis = np.random.randint(0,1000,10)
```

In [363]:
```
lis
```

Out[363]:
```
array([474, 894, 103, 396, 463,   1, 388, 203, 698, 629])
```

In [364]:
```
training_score = []
testing_score = []
deg = []

for n in range(1,3):
    for j in lis:
        X_train  , X_test , y_train , y_test = train_test_split(X,y,random_st
        scaler = MinMaxScaler()

        X_trn = scaler.fit_transform(X_train)
        X_tst = scaler.transform(X_test)



        X_train = pd.DataFrame(X_trn,columns=X_train.columns)
        X_test = pd.DataFrame(X_tst,columns=X_test.columns)

        # Polynomial Transformation
        scaler_poly = PolynomialFeatures(degree=n)
        X_poly_trn = scaler_poly.fit_transform(X_train)
        X_poly_tst =  scaler_poly.transform(X_test)

        lm_poly = LinearRegression()
        lm_poly.fit(X_poly_trn,y_train)

        pred_train = lm_poly.predict(X_poly_trn)
        pred_test = lm_poly.predict(X_poly_tst)

        training_score.append(r2_score(y_train, pred_train))
        testing_score.append(r2_score(y_test, pred_test))
        deg.append(n)
```

In [365]:
```
training_score
```

Out[365]:
```
[0.6338433756681264,
 0.6311170681612808,
 0.631947856240846,
 0.6316605449348109,
 0.6317123704851033,
 0.6306262525825439,
 0.6313424837579227,
 0.6300266166593315,
 0.6312785060435384,
 0.6296590863841547,
 0.7245705295980848,
 0.7277223892072919,
 0.7206916257130007,
 0.7190869955262119,
 0.7289250366653408,
 0.7148082799313542,
 0.7298726803372642,
 0.7311330754329675,
 0.7316296078033832,
 0.7256471844582452]
```

In [366]:
```
testing_score
```

Out[366]:
```
[0.6241709334397243,
 0.6331133890190952,
 0.627825131037675,
 0.6296453449108208,
 0.6298127497150436,
 0.6323517134845055,
 0.6322517522455535,
 0.6363530930156396,
 0.6308354253870578,
 0.6373340087585855,
 0.719943307371332,
 0.7060978667687714,
 0.7131715861303193,
 0.7333844582747209,
 0.6925735986820023,
 0.7431950693220408,
 0.702779053147597,
 0.691961976029671,
 0.6918452608573218,
 0.7145237275552888]
```

In [367]:
```
d = [1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2]
```

In [368]:
```
dic_training = {'training_score':training_score,'degree':d}
```

In [369]:
```
training = pd.DataFrame(dic_training)
```

In [370]:
```
training.head()
```

Out[370]:

	training_score	degree
0	0.633843	1
1	0.631117	1
2	0.631948	1
3	0.631661	1
4	0.631712	1

In [371]:
```
training = training.reset_index()
```

In [372]:
```
training.head()
```

Out[372]:

	index	training_score	degree
0	0	0.633843	1
1	1	0.631117	1
2	2	0.631948	1
3	3	0.631661	1
4	4	0.631712	1

In [373]:
```
dic_testing = {'testing_score':testing_score,'degree':d}
```

In [374]:
```
testing = pd.DataFrame(dic_testing)
```

In [375]:
```
testing.head()
```

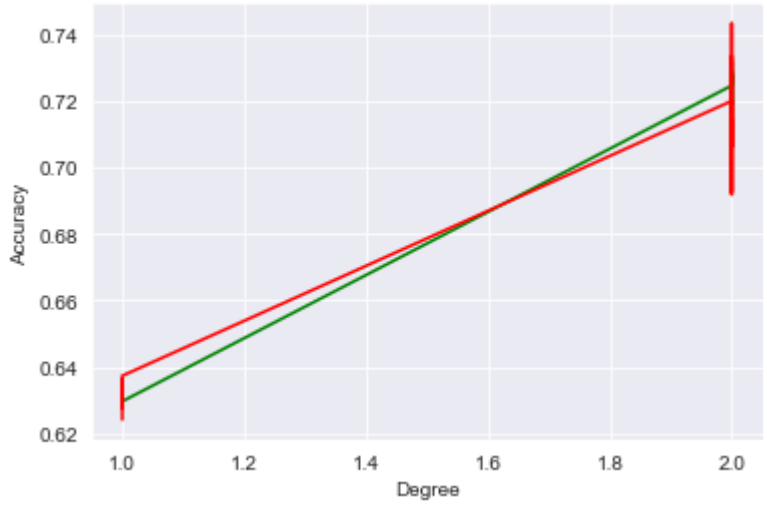
Out[375]:

	testing_score	degree
0	0.624171	1
1	0.633113	1
2	0.627825	1
3	0.629645	1
4	0.629813	1

In [376]:
```
testing = testing.reset_index()
```

In [377]:
```
plt.plot(training['degree'],training['training_score'],color='green')
plt.plot(testing['degree'],testing['testing_score'],color='red')
plt.xlabel('Degree')
plt.ylabel('Accuracy')
```

Out[377]:
```
Text(0, 0.5, 'Accuracy')
```



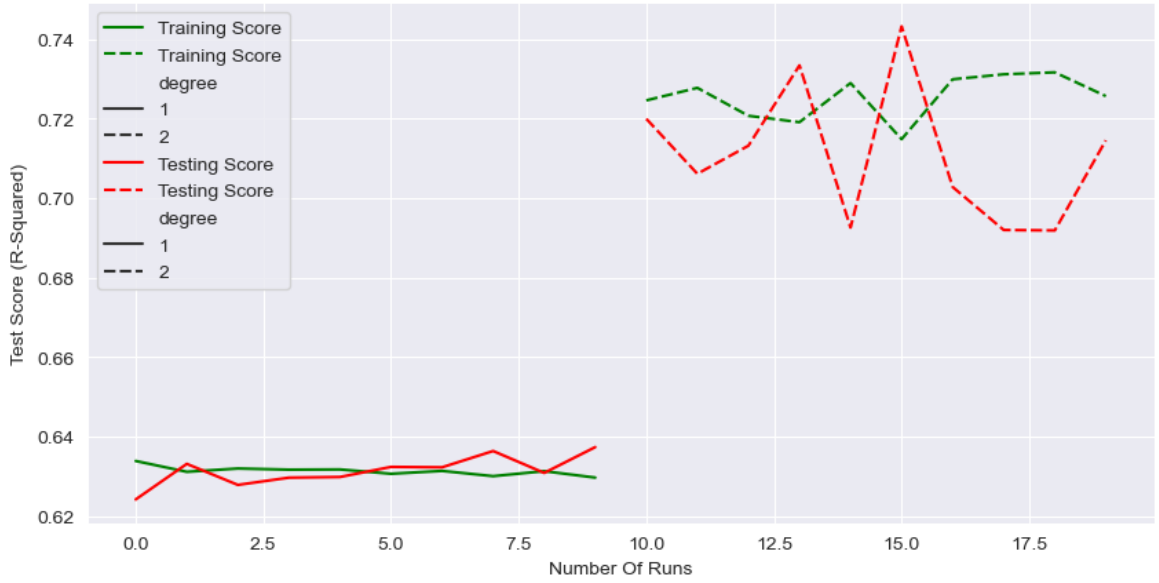
In [378]:
```
training[training['index']<=9]
```

Out[378]:

	index	training_score	degree
0	0	0.633843	1
1	1	0.631117	1
2	2	0.631948	1
3	3	0.631661	1
4	4	0.631712	1
5	5	0.630626	1
6	6	0.631342	1
7	7	0.630027	1
8	8	0.631279	1
9	9	0.629659	1

In [379]:
```
figure = plt.figure(figsize=(10,5),dpi=100)
sns.lineplot(x= training['index'],y= training['training_score'] ,color='green
sns.lineplot(x=testing['index'],y= testing['testing_score'],color='red',data=
plt.xlabel('Number Of Runs')
plt.ylabel('Test Score (R-Squared)')
```

Out[379]: Text(0, 0.5, 'Test Score (R-Squared)')



In [380]:
```
# Mean Testing Score
polynomial_reg_cv = np.mean(testing_score)
print(polynomial_reg_cv)
```

```
0.6711584722576384
```

In [381]:
```
np.mean(training_score)
```

Out[381]:
```
0.6783650782795402
```

In [382]:
```
print('The cross validated score of the best Polynomial Regressor model is: {
```

```
The cross validated score of the best Polynomial Regressor model is: 0.6712
```

One Can Clearly see from the above plot that the Regression with 2nd Degree Polynomials Outperform the linear function. The Cross Validated Test Score (R-Squared) that we are able to get form the linear regression with degree 2 is: 0.6534 which means that in this model 65.34% of the variance of the price is explaned by our explanatory variables

Gettig back the original train test split results and the scaling

In [383]:
```
X_train  , X_test , y_train , y_test = train_test_split(X,y,random_state = 0)
```

In [384]:
```
scaler = MinMaxScaler()
```

In [385]:
```
X_trn = scaler.fit_transform(X_train)
X_tst = scaler.transform(X_test)

X_train = pd.DataFrame(X_trn,columns=X_train.columns)
X_test = pd.DataFrame(X_tst,columns=X_test.columns)
```

Model 8 - Linear Support Vector Regressor

In [386]:
```
from sklearn.svm import LinearSVR
```

In [387]:
```
linear_svr = LinearSVR()
```

In [388]:
```
linear_svr.fit(X_train_sample,y_train_sample)
```

Out[388]:
```
LinearSVR()
```

In [389]:
```
grid_params = {'C':[0.001,0.01,0.1,1,10,100,1000],'epsilon':[0, 0.01, 0.1, 0.
```

In [390]:
```
grid = GridSearchCV(linear_svr,grid_params,cv=kfold,verbose=5)
```

In [391]:
```
grid.fit(X_train_sample,y_train_sample)
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
[CV] C=0.001, epsilon=0 ..............................................
[CV] ................. C=0.001, epsilon=0, score=-2.163, total=   0.0s
[CV] C=0.001, epsilon=0 ..............................................
[CV] ................. C=0.001, epsilon=0, score=-2.768, total=   0.0s
[CV] C=0.001, epsilon=0 ..............................................
[CV] ................. C=0.001, epsilon=0, score=-2.751, total=   0.0s
[CV] C=0.001, epsilon=0 ..............................................
[CV] ................. C=0.001, epsilon=0, score=-3.501, total=   0.0s
[CV] C=0.001, epsilon=0 ..............................................
[CV] ................. C=0.001, epsilon=0, score=-2.584, total=   0.0s
[CV] C=0.001, epsilon=0.01 ...........................................
[CV] .............. C=0.001, epsilon=0.01, score=-2.163, total=   0.0s
[CV] C=0.001, epsilon=0.01 ...........................................
[CV] .............. C=0.001, epsilon=0.01, score=-2.768, total=   0.0s
[CV] C=0.001, epsilon=0.01 ...........................................
[CV] .............. C=0.001, epsilon=0.01, score=-2.751, total=   0.0s
[CV] C=0.001, epsilon=0.01 ...........................................
[CV] .............. C=0.001, epsilon=0.01, score=-3.501, total=   0.0s
```

In [392]:
```
grid.best_params_
```

Out[392]:
```
{'C': 1000, 'epsilon': 2}
```

In [393]:
```
grid.best_score_
```

Out[393]:
```
0.09423368334530693
```

In [394]:
```
results_linear_svr = pd.DataFrame(grid.cv_results_)
```

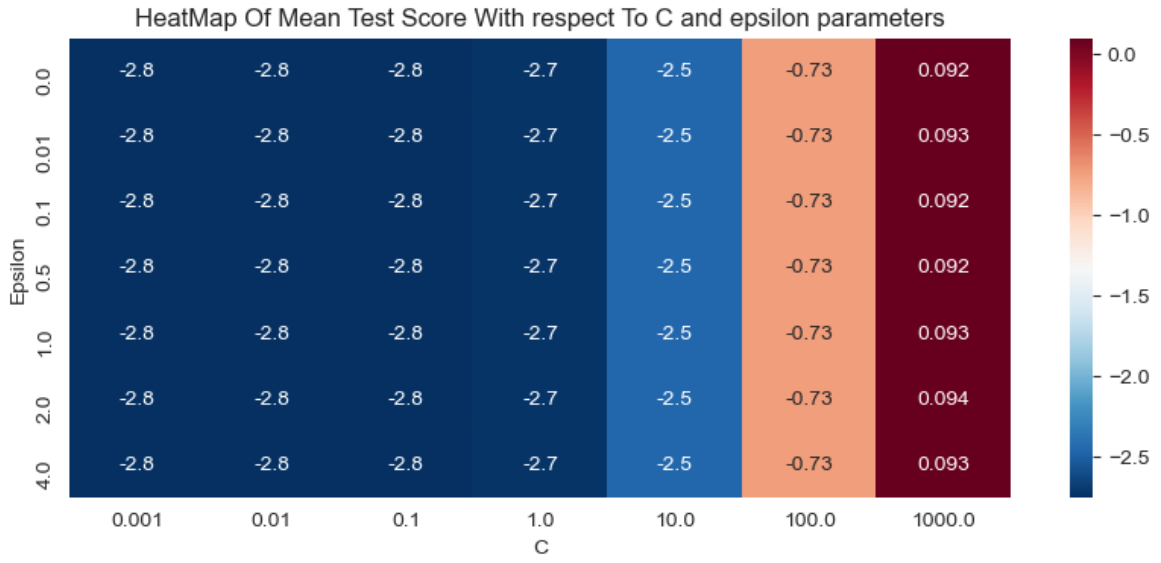
In [395]:
```
results_linear_svr.head()
```

Out[395]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_epsilon	pa
0	0.003192	3.988779e-04	0.001816	3.915267e-04	0.001	0	{'eps
1	0.003214	3.884342e-04	0.001589	4.989154e-04	0.001	0.01	{'eps
2	0.003391	4.885000e-04	0.001596	4.887142e-04	0.001	0.1	{'eps
3	0.003191	3.991128e-04	0.001596	4.888113e-04	0.001	0.5	{'eps
4	0.002992	1.784161e-07	0.001995	1.784161e-07	0.001	1	{'eps

In [396]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
sns.heatmap(results_linear_svr.pivot_table(index='param_epsilon',columns='par
plt.ylabel('Epsilon')
plt.xlabel('C')
plt.title('HeatMap Of Mean Test Score With respect To C and epsilon parameter
```

Out[396]:
```
Text(0.5, 1.0, 'HeatMap Of Mean Test Score With respect To C and epsilon pa
rameters')
```



In [397]:
```
linear_svm = LinearSVR(epsilon=0,C=1000)
```

In [398]:
```
linear_svm.fit(X_train_sample,y_train_sample)
```

Out[398]:
```
LinearSVR(C=1000, epsilon=0)
```

In [399]:
```
linear_svm_cross_val_score = cross_val_score(linear_svm,X_train_sample,y_trai
mean_cross_val_linear_svm = linear_svm_cross_val_score.mean()
```

```
[CV]  ................................................................
[CV] ................................. , score=0.101, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.028, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.088, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.098, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.153, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

In [400]:
```
print('The Final Accuracy Score That was Achived With Our Best Linear SVM Mod
```

```
The Final Accuracy Score That was Achived With Our Best Linear SVM Model wi
th the value of C being 1000 and having epsilon as 0.5 Norm is: 0.0936
```

Support Vector Machines Kernel Trick

Model 9: SVM with Linear Kernel

In [401]:
```
svm_linear_kernel = SVR(kernel='linear')
```

In [402]:
```
grid_params = {'C':[0.001,0.01,0.1,1,10,100,1000]}
```

In [403]:
```
grid = GridSearchCV(svm_linear_kernel,grid_params,cv=kfold,verbose=5)
```

In [404]:
```
grid.fit(X_train_sample,y_train_sample)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s

Fitting 5 folds for each of 7 candidates, totalling 35 fits
[CV] C=0.001 .........................................................
[CV] ........................... C=0.001, score=-0.067, total=   0.1s
[CV] C=0.001 .........................................................
[CV] ........................... C=0.001, score=-0.132, total=   0.0s
[CV] C=0.001 .........................................................
[CV] ........................... C=0.001, score=-0.073, total=   0.0s
[CV] C=0.001 .........................................................

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:
0.0s
```

In [405]:
```
grid.best_params_
```

Out[405]:
```
{'C': 1000}
```

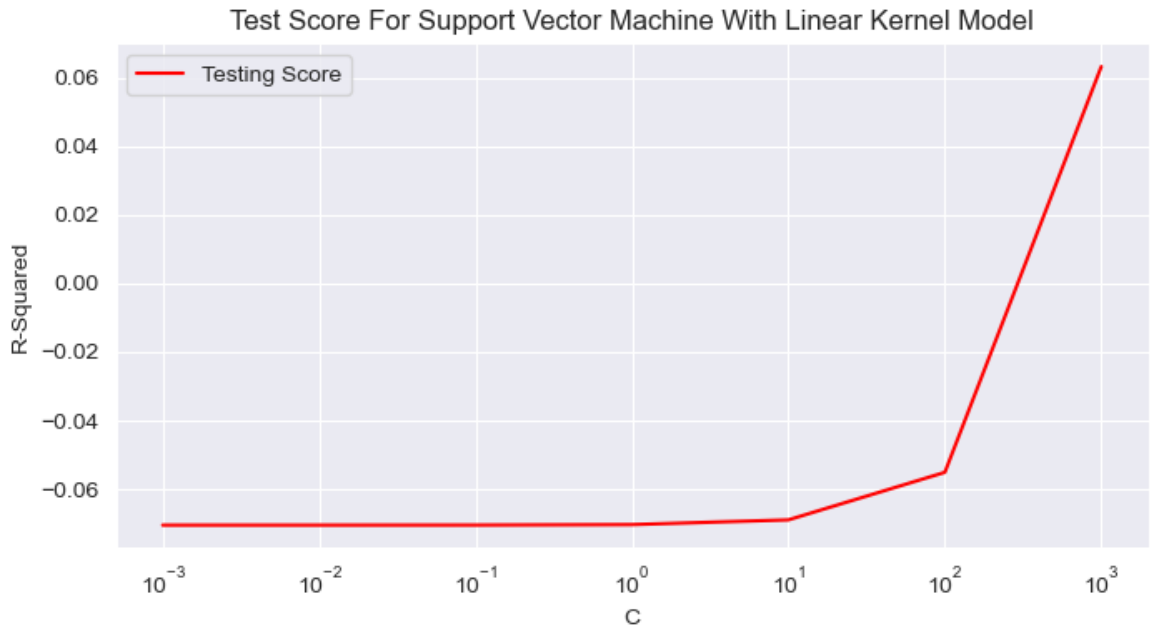
In [406]:
```
grid.best_score_
```

Out[406]:
```
0.0631502866821286
```

In [407]:
```
results_linearkernel_svm = pd.DataFrame(grid.cv_results_)
```

In [408]:
```
fig = plt.figure(figsize=(8,4),dpi=100)

plt.plot(results_linearkernel_svm['param_C'],
         results_linearkernel_svm['mean_test_score'],color='red',label ='Test
plt.xscale('log')
plt.xlabel('C')
plt.ylabel('R-Squared')
plt.title('Test Score For Support Vector Machine With Linear Kernel Model')
plt.legend()
```

Out[408]: <matplotlib.legend.Legend at 0x17fe6525128>



In [409]:
```
svm_linear_kernel = SVR(kernel='linear',C=1000)
```

In [410]:
```
svm_linear_kernel.fit(X_train_sample,y_train_sample)
```

Out[410]: SVR(C=1000, kernel='linear')

In [411]:
```
svm_linearKernel_cross_val_score = cross_val_score(svm_linear_kernel,X_train_
mean_cross_val_svm_linearKernel = svm_linearKernel_cross_val_score.mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:
0.0s

[CV]  ................................................................
[CV] ................................. , score=0.065, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.005, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.058, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.074, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.114, total=   0.1s

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.2s finished
```

In [412]:
```
print('The Final Accuracy Score That was Achived With Our Best SVM Model With
```

```
The Final Accuracy Score That was Achived With Our Best SVM Model With Line
ar Kernel with the value of C being 1000 : 0.0632
```

Model 10: SVM with RBF Kernel

In [413]:
```
svm_rbf_kernel = SVR(kernel='rbf')
```

In [414]:
```
grid_params = {'gamma':[0.001,0.01,0.1,1,10,100,1000],'C':[0.001,0.01,0.1,1,1
```

In [415]:
```
grid = GridSearchCV(svm_rbf_kernel,grid_params,cv=kfold,verbose=5)
```

In [416]:
```
grid.fit(X_train_sample,y_train_sample)
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
[CV] C=0.001, gamma=0.001 ............................................
[CV] ............... C=0.001, gamma=0.001, score=-0.067, total=   0.1s
[CV] C=0.001, gamma=0.001 ............................................

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

[CV] ............... C=0.001, gamma=0.001, score=-0.132, total=   0.1s
[CV] C=0.001, gamma=0.001 ............................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:
0.0s
```

In [417]:
```
grid.best_params_
```

Out[417]:
```
{'C': 1000, 'gamma': 1}
```

In [418]:
```
grid.best_score_
```

Out[418]:
```
0.06350594565406506
```

In [419]:
```
results_RBFkernel_svm = pd.DataFrame(grid.cv_results_)
```

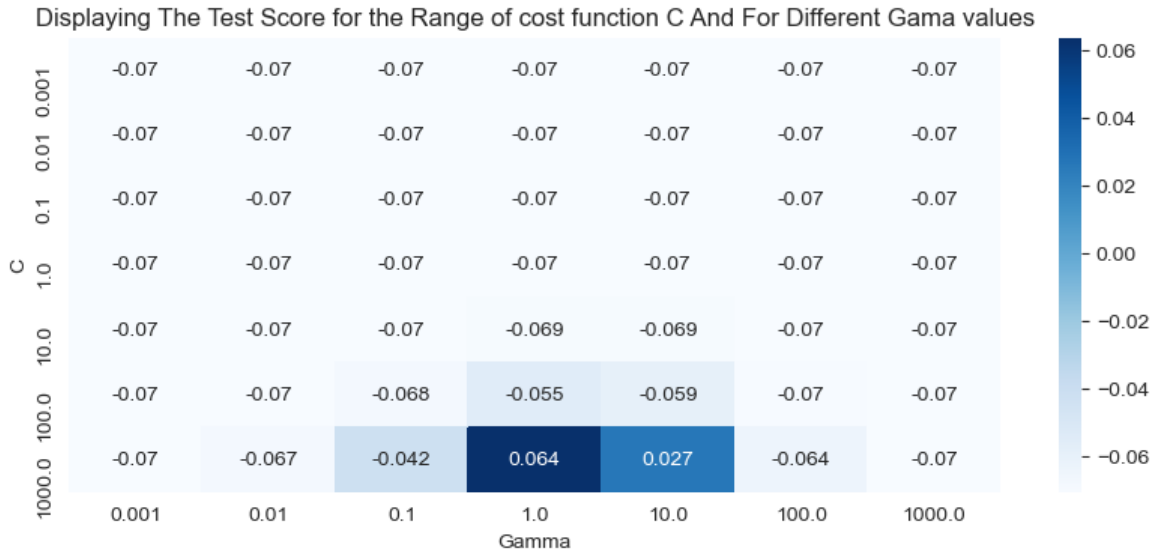
In [420]:
```
results_RBFkernel_svm.head()
```

Out[420]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	pa
0	0.065225	0.001620	0.011170	3.985885e-04	0.001	0.001	('ga (
1	0.062222	0.001616	0.011381	4.792394e-04	0.001	0.01	('ga
2	0.062430	0.001627	0.011372	4.925539e-04	0.001	0.1	('ga
3	0.060438	0.002148	0.010971	9.536743e-08	0.001	1	('ga
4	0.061045	0.002924	0.011768	3.993078e-04	0.001	10	('ga

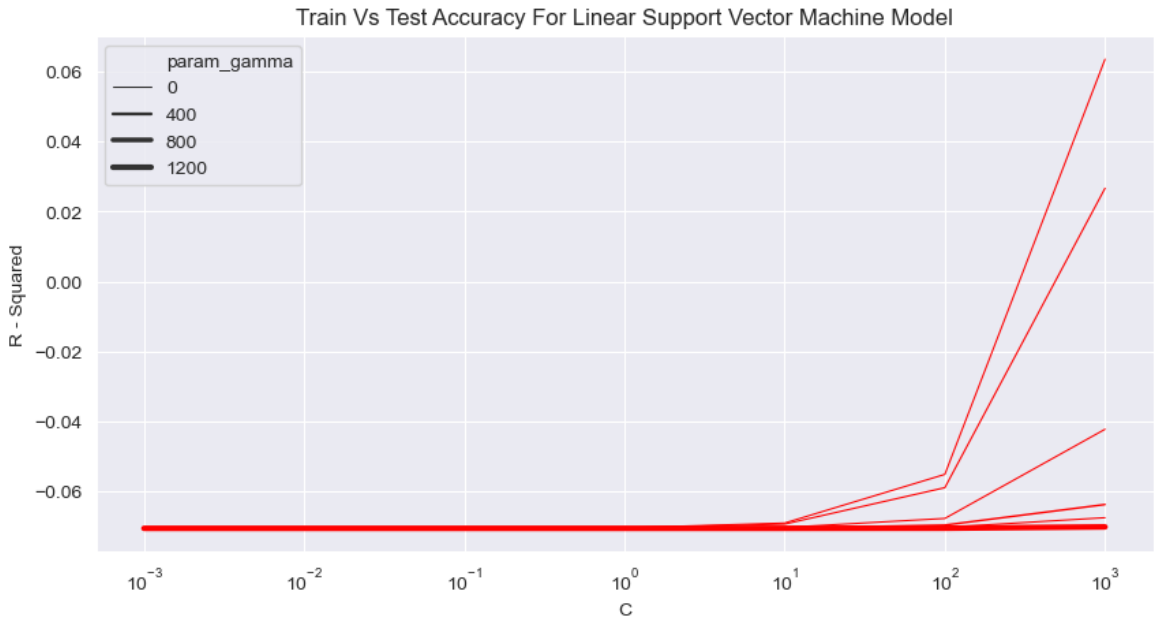
In [421]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
sns.heatmap(results_RBFkernel_svm.pivot_table(index='param_C',columns='param_
plt.xlabel('Gamma')
plt.ylabel('C')
plt.title('Displaying The Test Score for the Range of cost function C And For
```

Out[421]:
```
Text(0.5, 1.0, 'Displaying The Test Score for the Range of cost function C
And For Different Gama values')
```



In [422]:
```
fig = plt.figure(figsize=(10,5),dpi=100)

sns.lineplot(x= results_RBFkernel_svm['param_C'],y= results_RBFkernel_svm['me
size=results_RBFkernel_svm['param_gamma'],color='red')

plt.xscale('log')
plt.xlabel('C')
plt.ylabel('R - Squared')
plt.title('Train Vs Test Accuracy For Linear Support Vector Machine Model')
```

Out[422]:
```
Text(0.5, 1.0, 'Train Vs Test Accuracy For Linear Support Vector Machine Mo
del')
```



From The Above plot it can be clearly seen that the most accuracy can be obtained in the test dataset if the SVM with rbf Kernel is used with the C: 1000 and gamma: 1. The accuracy that was obtained for the test data set at this value of the cost function was 0.075 which is the R-Squared value which indicates that about only 7.5% of variation in price can be explained by our 15 explanatory.

This R-Squared Value could be considered as very poor as the value is very far from the ideal R-squared of 1. The model was very time consuming and had very high computational complexity but still cold not deliver satisfactory results

In [423]:
```
svm_rbf_kernel = SVR(kernel='rbf',C=1000,gamma=1)
```

In [424]:
```
svm_rbf_kernel.fit(X_train_sample,y_train_sample)
```

Out[424]:
```
SVR(C=1000, gamma=1)
```

In [425]:
```
svm_rbfKernel_cross_val_score = cross_val_score(svm_rbf_kernel,X_train_sample
mean_cross_val_svm_rbf_kernel = svm_rbfKernel_cross_val_score.mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:
0.0s


[CV]  ................................................................
[CV] ................................... , score=0.053, total=   0.1s
[CV]  ................................................................
[CV] ................................... , score=0.011, total=   0.1s
[CV]  ................................................................
[CV] ................................... , score=0.069, total=   0.1s
[CV]  ................................................................
[CV] ................................... , score=0.080, total=   0.1s
[CV]  ................................................................
[CV] ................................... , score=0.104, total=   0.1s


[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.3s finished
```

In [426]:
```
print('The Final Accuracy Score That was Achived With Our Best SVM Model With
```

```
The Final Accuracy Score That was Achived With Our Best SVM Model With RBF
Kernel with the value of C being 1000 and that of gamma being 1 is : 0.063
5)
```

Model 11: SVM with Polynomial Kernel

In [427]:
```
svm_poly_kernal = SVR(kernel='poly')
```

In [428]:
```
grid_params = {'gamma':[0.01,0.1,1,10,100],'C':[0.01,0.1,1,10,100,1000],'degr
```

In [429]:
```
kfold = KFold(n_splits=3,random_state=0)
```

```
C:\Users\behab\Anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

In [430]:
```
grid = GridSearchCV(svm_poly_kernal,grid_params,cv=kfold,verbose=5)
```

In [431]:
```
grid.fit(X_train_sample,y_train_sample)
```

```
Fitting 3 folds for each of 90 candidates, totalling 270 fits
[CV] C=0.01, degree=1, gamma=0.01 ....................................
[CV] ....... C=0.01, degree=1, gamma=0.01, score=-0.089, total=   0.0s
[CV] C=0.01, degree=1, gamma=0.01 ....................................
[CV] ....... C=0.01, degree=1, gamma=0.01, score=-0.087, total=   0.0s
[CV] C=0.01, degree=1, gamma=0.01 ....................................
[CV] ....... C=0.01, degree=1, gamma=0.01, score=-0.024, total=   0.0s
[CV] C=0.01, degree=1, gamma=0.1 .....................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:
0.0s
```

In [432]:
```
grid.best_params_
```

Out[432]:
```
{'C': 0.1, 'degree': 3, 'gamma': 100}
```

In [433]:
```
grid.best_score_
```

Out[433]:
```
0.631682418406088
```

In [434]:
```
results_Polynomialkernel_svm = pd.DataFrame(grid.cv_results_)
```

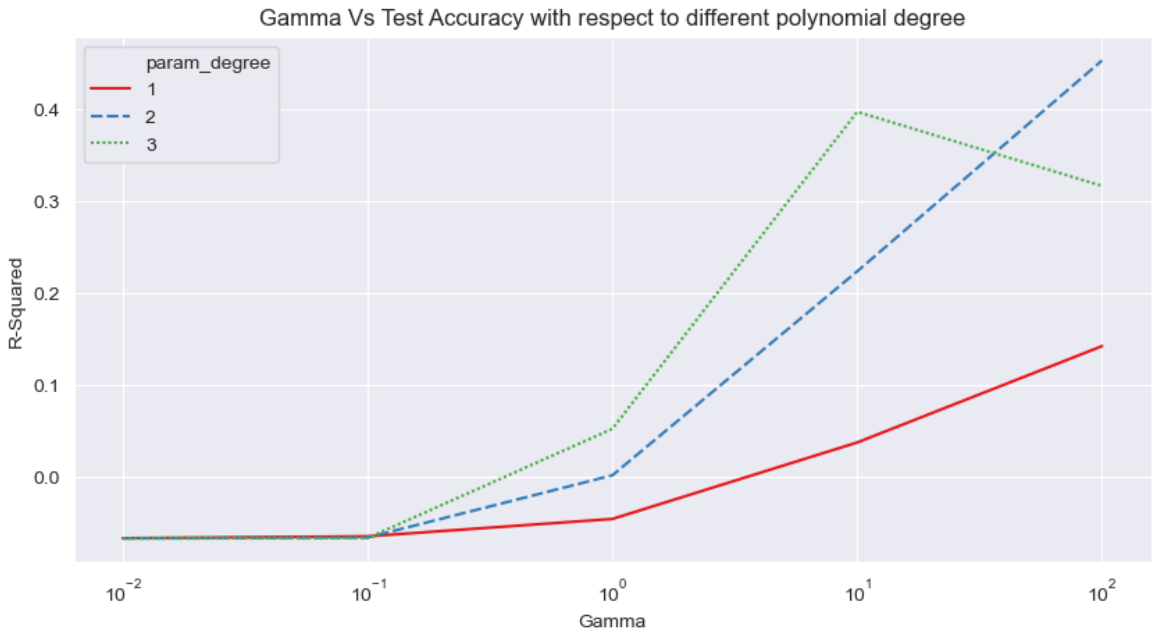
In [435]:
```
results_Polynomialkernel_svm.head()
```

Out[435]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_degree	par
0	0.032912	0.000814	0.009643	0.000467	0.01	1	
1	0.031598	0.000456	0.009641	0.000434	0.01	1	
2	0.030584	0.000475	0.008939	0.000027	0.01	1	
3	0.032261	0.001239	0.009294	0.000480	0.01	1	
4	0.031265	0.000495	0.008619	0.000480	0.01	1	

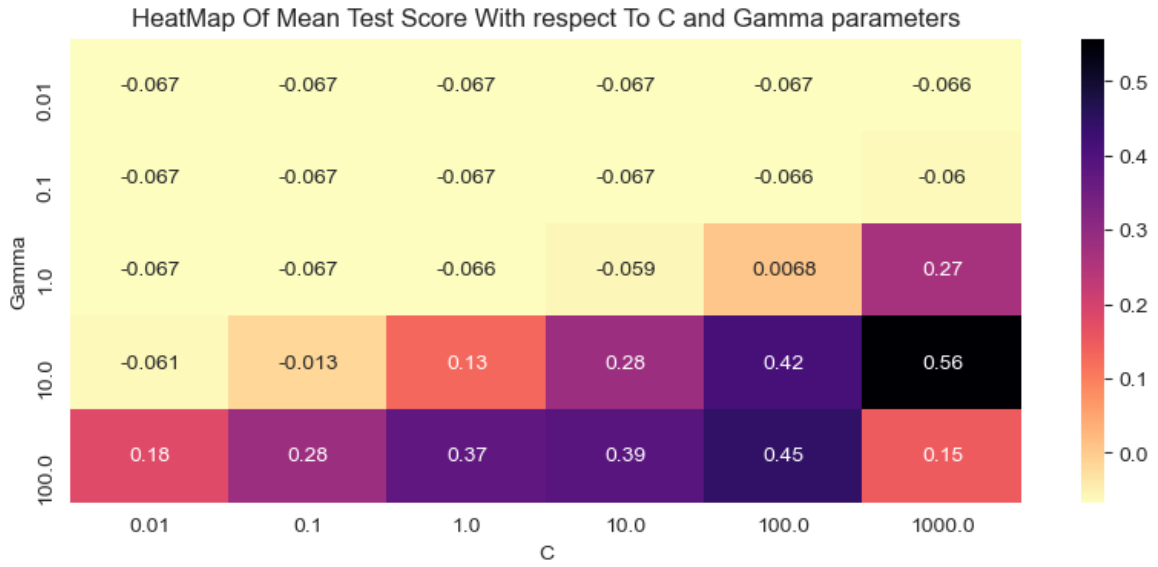
In [436]:
```
fig = plt.figure(figsize=(10,5),dpi=100)
sns.lineplot(results_Polynomialkernel_svm['param_gamma'],results_Polynomialke
             style=results_Polynomialkernel_svm['param_degree'],
             hue=results_Polynomialkernel_svm['param_degree'],ci=None,palette

plt.xscale('log')
plt.xlabel('Gamma')
plt.ylabel('R-Squared')
plt.title('Gamma Vs Test Accuracy with respect to different polynomial degree
```

Out[436]:
```
Text(0.5, 1.0, 'Gamma Vs Test Accuracy with respect to different polynomial
degree')
```



In [437]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
sns.heatmap(results_Polynomialkernel_svm.pivot_table(index='param_gamma',colu
plt.ylabel('Gamma')
plt.xlabel('C')
plt.title('HeatMap Of Mean Test Score With respect To C and Gamma parameters'
```

Out[437]:
```
Text(0.5, 1.0, 'HeatMap Of Mean Test Score With respect To C and Gamma para
meters')
```



From the Above plots it can clearly be seen that the best value of Cost parameter for the SVM with Polynomial kernel would be the 1000 and the Gamma value for the model would be 10 and the degree of the model should be 3. The test score that we get with this model is (0.675 (R-Squared)) which means that about 67.5% of the variance in the price could be explaind by our explanatory variables. This is pretty high considereing all the other previous models and the model just being trained on a sample of data.

In [438]:
```
svm_poly_kernal = SVR(kernel='poly',C=100,degree=3,gamma=10)
```

In [439]:
```
svm_poly_kernal.fit(X_train_sample,y_train_sample)
```

Out[439]:
```
SVR(C=100, gamma=10, kernel='poly')
```

In [440]:
```
kfold = KFold(n_splits=10,random_state=0)
```

```
C:\Users\behab\Anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

In [441]:
```
svm_poly_kernal_cross_val_score = cross_val_score(svm_poly_kernal,X_train_sam
mean_cross_val_svm_poly_kernal = svm_poly_kernal_cross_val_score.mean()
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:
0.0s

[CV]  ................................................................
[CV] ..................................... , score=0.740, total=   0.1s
[CV]  ................................................................
[CV] ..................................... , score=0.691, total=   0.1s
[CV]  ................................................................
[CV] ..................................... , score=0.564, total=   0.1s
[CV]  ................................................................
[CV] ..................................... , score=0.595, total=   0.1s
[CV]  ................................................................
[CV] ..................................... , score=0.585, total=   0.1s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.2s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.3s remaining:
0.0s

[CV] ..................................... , score=0.657, total=   0.1s
[CV]  ................................................................
[CV] ..................................... , score=0.556, total=   0.1s
[CV]  ................................................................
[CV] ..................................... , score=0.572, total=   0.1s
[CV]  ................................................................
[CV] ..................................... , score=0.739, total=   0.1s
[CV]  ................................................................
[CV] ..................................... , score=0.584, total=   0.1s

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.9s finished
```

In [442]:
```
mean_cross_val_svm_poly_kernal
```

Out[442]:
```
0.6283597620217782
```

In [443]:
```
print('The Final Accuracy Score That was Achived With Our Best SVM Model With
```

```
The Final Accuracy Score That was Achived With Our Best SVM Model With Poly
nomial Kernel with the value of C being 1000 and that of gamma being 10 and
degree of 3 is : 0.6284
```

Prediction Based On the Best Model.

In [444]:
```
evaluation = {'Models':['KNN Regressor','Linear Regression','Ridge Regressior
                        'Decision Tree Regressor','Random Forest Regressor','P
                        'SVM Linear Kernel Regressor','SVM RBF Kernel Regressc
              'Test Scores':[knn_mean_score_cv,lm_mean_score_cv,ridge_mean_scc
                             rf_mean_score_cv,polynomial_reg_cv,mean_cross_val_
                             mean_cross_val_svm_rbf_kernel,mean_cross_val_svm_p
```

In [445]:
```
eva = pd.DataFrame(evaluation)
```

In [446]:
```
eva.head(15)
```

Out[446]:

	Models	Test Scores
0	KNN Regressor	0.632672
1	Linear Regression	0.625564
2	Ridge Regression	0.625616
3	Lasso Regression	0.625633
4	Decision Tree Regressor	0.576007
5	Random Forest Regressor	0.651900
6	Polynomial Regression	0.671158
7	Linear SVM	0.093571
8	SVM Linear Kernel Regressor	0.063150
9	SVM RBF Kernel Regressor	0.063506
10	SVM Polynomial Kernel Regressor	0.628360

Best Model Ranked Top To Bottom

In [447]:
```
eva = eva.sort_values(by='Test Scores',ascending = False)
```

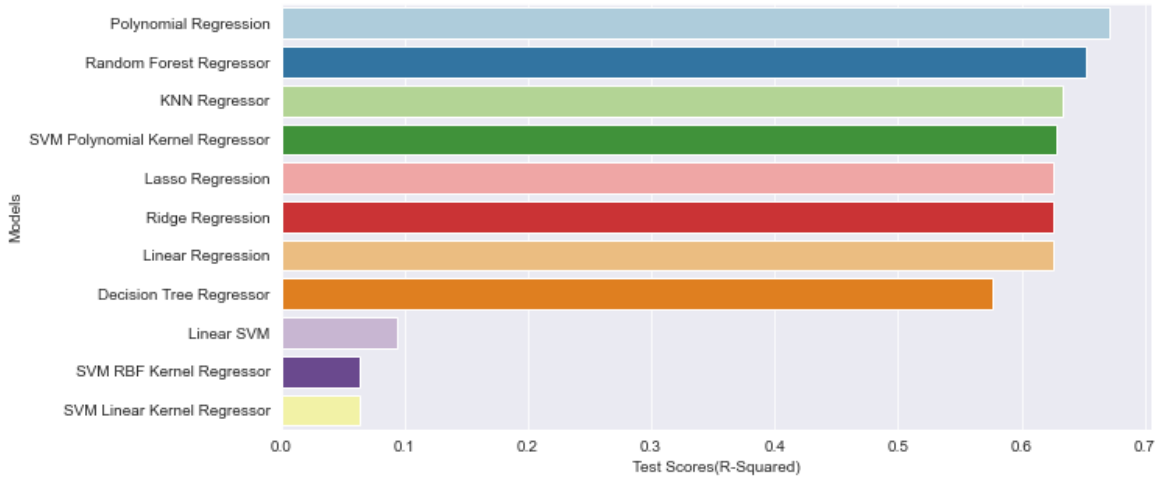
In [448]:
```
eva.head(15)
```

Out[448]:

	Models	Test Scores
6	Polynomial Regression	0.671158
5	Random Forest Regressor	0.651900
0	KNN Regressor	0.632672
10	SVM Polynomial Kernel Regressor	0.628360
3	Lasso Regression	0.625633
2	Ridge Regression	0.625616
1	Linear Regression	0.625564
4	Decision Tree Regressor	0.576007
7	Linear SVM	0.093571
9	SVM RBF Kernel Regressor	0.063506
8	SVM Linear Kernel Regressor	0.063150

In [449]:
```
fig = plt.figure(figsize=(10,5))
sns.barplot(x=eva['Test Scores'],y=eva['Models'],orient='h',palette='Paired')
plt.ylabel('Models')
plt.xlabel('Test Scores(R-Squared)')
```

Out[449]:
```
Text(0.5, 0, 'Test Scores(R-Squared)')
```



The best model is the Random Forest Regressor

As the Random Forest model is not the part of the class curriculum hence we are going to use the top 2 models for predictions which are random forest regressor and polynomial regressor

From the model that was taught in class we can say that the best model was Polynomial Regressor(Random Forest not Part of Classwork)

Predicticting With The Best Model Overall(Random Forest Regressor)

In [450]:
```
rf = RandomForestRegressor(n_estimators=400,max_depth=40)
```

In [451]:
```
rf.fit(X_train,y_train)
```

Out[451]:
```
RandomForestRegressor(max_depth=40, n_estimators=400)
```

In [452]:
```
prediction_rf = rf.predict(X_test)
```

In [453]:
```
print(prediction_rf)
```

```
[ 335752.4625     1669085.44166667  537001.1275     ...  784097.7275
  322871.025       567244.995     ]
```

In [454]:
```
# Training Score
print('The Training Score(R-Squared) of the best Random Forest Model is {}'.f
```

```
The Training Score(R-Squared) of the best Random Forest Model is 0.96
```

In [455]:
```
# Testing Score
print('The Test Score(R-Squared) of the best Random Forest Model is {}'.forma
```

```
The Test Score(R-Squared) of the best Random Forest Model is 0.74
```

In [456]:
```
# Mean Squared Error For the Random Foresst Model
metrics.mean_squared_error(y_test,prediction_rf)
```

Out[456]:
```
34548603889.80412
```

In [457]:
```
root_mean_squared_error_rf = round(metrics.mean_squared_error(y_test,predicti
print('The Root mean Squared error of the random forest model is: {}'.format(
```

```
The Root mean Squared error of the random forest model is: 185872.55
```

Predicticting With The Best Model among models discussed in class(Polynomial Regression)

In [458]:
```
scaler_poly = PolynomialFeatures(degree=2)
X_poly_trn = scaler_poly.fit_transform(X_train)
X_poly_tst =  scaler_poly.fit_transform(X_test)
```

In [459]:
```
lm_poly = LinearRegression()
lm_poly.fit(X_poly_trn,y_train)
```

Out[459]:
```
LinearRegression()
```

In [460]:
```
pred_train = lm_poly.predict(X_poly_trn)
pred_test = lm_poly.predict(X_poly_tst)
```

In [461]:
```
# Prediction For the Test Data
print(pred_test)
```

```
[ 397594.28073596 1745850.17937834  486753.15009898 ...  732060.29861499
  334884.11250541  513867.66816136]
```

In [462]:
```
print('The Training Score(R-Squared) of the best Polynomial Regression Model
```

```
The Training Score(R-Squared) of the best Polynomial Regression Model is 0.
73
```

In [463]:
```
print('The Testing Score(R-Squared) of the best Polynomial Regression Model i
```

```
The Testing Score(R-Squared) of the best Polynomial Regression Model is 0.7
1
```

Project Part 2

In [464]:
```
from sklearn.ensemble import BaggingRegressor
```

Implementing Bagging On any two Models

Models On which Bagging Would be performed: 1) K Nearest Neighbours Regressor 2) Linear Regression

Bagging with KNN

In [465]:
```
knn = KNeighborsRegressor(n_neighbors=8)
bag_reg_knn = BaggingRegressor(knn,bootstrap=True, n_jobs=-1, random_state=0)
```

In [466]:
```
kfold = KFold(n_splits=5,random_state=0)
```

```
C:\Users\behab\Anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

In [467]:
```
grid_params = {'n_estimators':[10,40,70,100],'max_samples':[0.05, 0.1, 0.2, 0
```

In [468]:
```
grid = GridSearchCV(bag_reg_knn,grid_params,cv=kfold,verbose=3)
```

In [469]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
[CV] max_samples=0.05, n_estimators=10 ..............................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.

[CV] ... max_samples=0.05, n_estimators=10, score=0.462, total=   2.9s
[CV] max_samples=0.05, n_estimators=10 ..............................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.8s remaining:
0.0s

[CV] ... max_samples=0.05, n_estimators=10, score=0.509, total=   0.3s
[CV] max_samples=0.05, n_estimators=10 ..............................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    3.2s remaining:
0.0s
```

In [470]:
```
grid.best_params_
```

Out[470]:
```
{'max_samples': 0.8, 'n_estimators': 100}
```

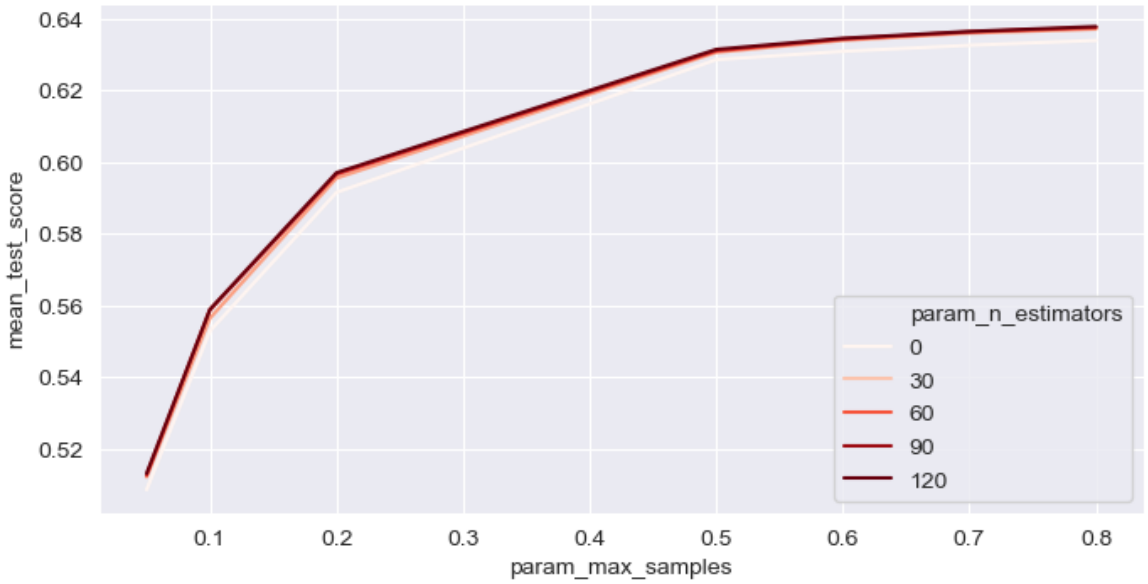
In [471]:
```
grid.best_score_
```

Out[471]:
```
0.6378345974073962
```

In [472]:
```
output = pd.DataFrame(grid.cv_results_)
```

In [473]:
```
fig = plt.figure(figsize=(8,4),dpi=100)
sns.lineplot(x=output['param_max_samples'],y=output['mean_test_score'],hue=ou
             palette='Reds')
```

Out[473]: <matplotlib.axes._subplots.AxesSubplot at 0x17fed3c9cc0>



In [474]:
```
baggaing_knn_score = grid.best_score_
```

In [475]:
```
print('The Best R Squared Value that we are getting is {}'.format(round(bagga
```

```
The Best R Squared Value that we are getting is 0.6378
```

Bagging With Linear Regression

In [476]:
```
lm = LinearRegression()
bag_reg_lm = BaggingRegressor(lm,bootstrap=True, n_jobs=-1, random_state=0)
```

In [477]:
```
grid_params = {'n_estimators':[10,40,70,100],'max_samples':[0.05, 0.1, 0.2, 0
```

In [478]:
```
grid = GridSearchCV(bag_reg_lm,grid_params,cv=kfold,verbose=3)
```

In [479]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
[CV] max_samples=0.05, n_estimators=10 ...............................
[CV] ... max_samples=0.05, n_estimators=10, score=0.632, total=   0.2s
[CV] max_samples=0.05, n_estimators=10 ...............................

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

[CV] ... max_samples=0.05, n_estimators=10, score=0.560, total=   0.2s
[CV] max_samples=0.05, n_estimators=10 ...............................
[CV] ... max_samples=0.05, n_estimators=10, score=0.629, total=   0.2s
[CV] max_samples=0.05, n_estimators=10 ...............................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.3s remaining:
0.0s
```

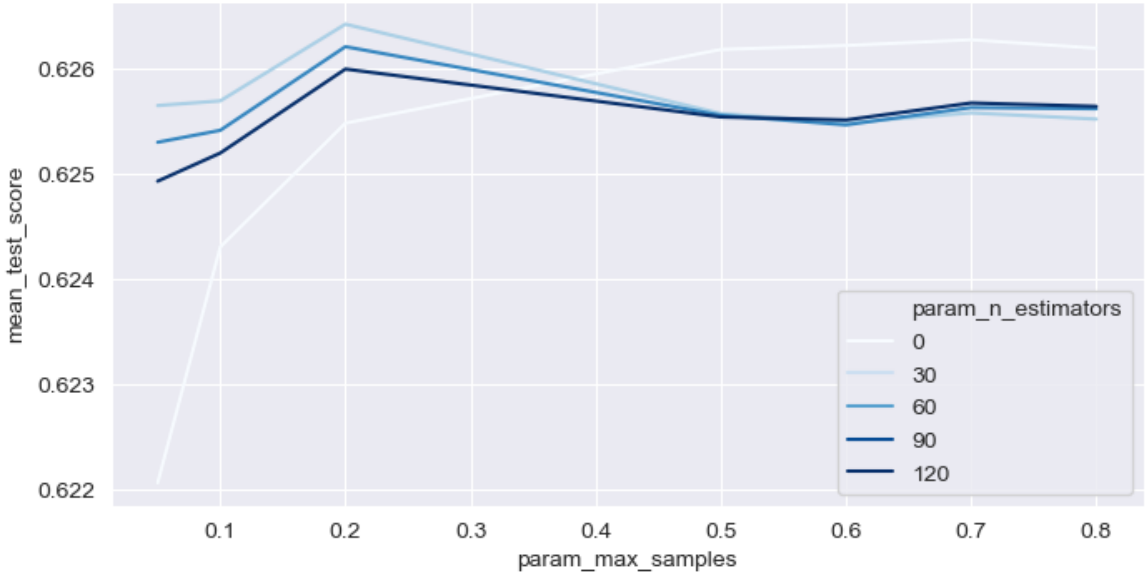
In [480]:
```
grid.best_params_
```

Out[480]:
```
{'max_samples': 0.2, 'n_estimators': 40}
```

In [481]:
```
output = pd.DataFrame(grid.cv_results_)
```

In [482]:
```
fig = plt.figure(figsize=(8,4),dpi=100)
sns.lineplot(x=output['param_max_samples'],y=output['mean_test_score'],hue=ou
             palette='Blues')
```

Out[482]: <matplotlib.axes._subplots.AxesSubplot at 0x17fa6023be0>



In [483]:
```
baggaing_lm_score = grid.best_score_
```

In [484]:
```
print('The Best R Squared Value that we are getting is {}'.format(round(bagga
```

```
The Best R Squared Value that we are getting is 0.6264
```

Pasting With Ridge

In [485]:
```
ridge = Ridge(random_state=0,alpha=1.0)
pst_reg_ridge = BaggingRegressor(ridge,bootstrap=False, n_jobs=-1, random_sta
```

In [486]:
```
grid_params = {'n_estimators':[10,40,70,100],'max_samples':[0.05, 0.1, 0.2, 0
```

In [487]:
```
grid = GridSearchCV(pst_reg_ridge,grid_params,cv=kfold,verbose=3)
```

In [488]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
[CV] max_samples=0.05, n_estimators=10 ..............................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.

[CV] ... max_samples=0.05, n_estimators=10, score=0.615, total=   0.5s
[CV] max_samples=0.05, n_estimators=10 ..............................
[CV] ... max_samples=0.05, n_estimators=10, score=0.560, total=   0.2s
[CV] max_samples=0.05, n_estimators=10 ..............................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.6s remaining:
0.0s
```

In [489]:
```
grid.best_params_
```

Out[489]:
```
{'max_samples': 0.6, 'n_estimators': 100}
```

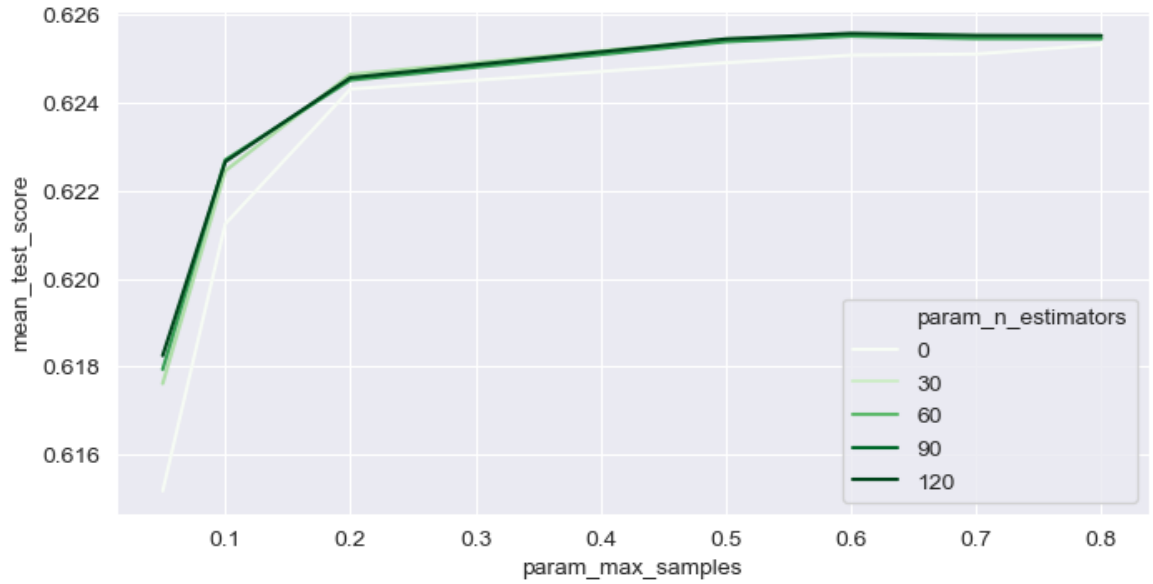
In [490]:
```
grid.best_score_
```

Out[490]:
```
0.6255672721908759
```

In [491]:
```
output = pd.DataFrame(grid.cv_results_)
```

In [492]:
```
fig = plt.figure(figsize=(8,4),dpi=100)
sns.lineplot(x=output['param_max_samples'],y=output['mean_test_score'],hue=ou
             palette='Greens')
```

Out[492]: <matplotlib.axes._subplots.AxesSubplot at 0x17fed042ac8>



In [493]:
```
pasting_ridge_score = grid.best_score_
```

In [494]:
```
print('The Best R Squared Value that we are getting is {}'.format(round(pasti
```

```
The Best R Squared Value that we are getting is 0.6256
```

Pasting With Decision Tree

In [495]:
```
tree = DecisionTreeRegressor(max_depth=6)
pst_reg_tree = BaggingRegressor(tree,bootstrap=False, n_jobs=-1, random_state
```

In [496]:
```
grid_params = {'n_estimators':[10,40,70,100],'max_samples':[0.05, 0.1, 0.2, 0
```

In [497]:
```
grid = GridSearchCV(pst_reg_tree,grid_params,cv=kfold,verbose=3)
```

In [498]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
[CV] max_samples=0.05, n_estimators=10 ..............................
[CV] ... max_samples=0.05, n_estimators=10, score=0.628, total=   0.2s
[CV] max_samples=0.05, n_estimators=10 ..............................

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

[CV] ... max_samples=0.05, n_estimators=10, score=0.593, total=   0.2s
[CV] max_samples=0.05, n_estimators=10 ..............................
[CV] ... max_samples=0.05, n_estimators=10, score=0.652, total=   0.2s
[CV] max_samples=0.05, n_estimators=10 ..............................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:
0.0s
```

In [499]:
```
grid.best_params_
```

Out[499]:
```
{'max_samples': 0.2, 'n_estimators': 100}
```

In [500]:
```
grid.best_score_
```

Out[500]:
```
0.6681545154012655
```

In [501]:
```
output = pd.DataFrame(grid.cv_results_)
```

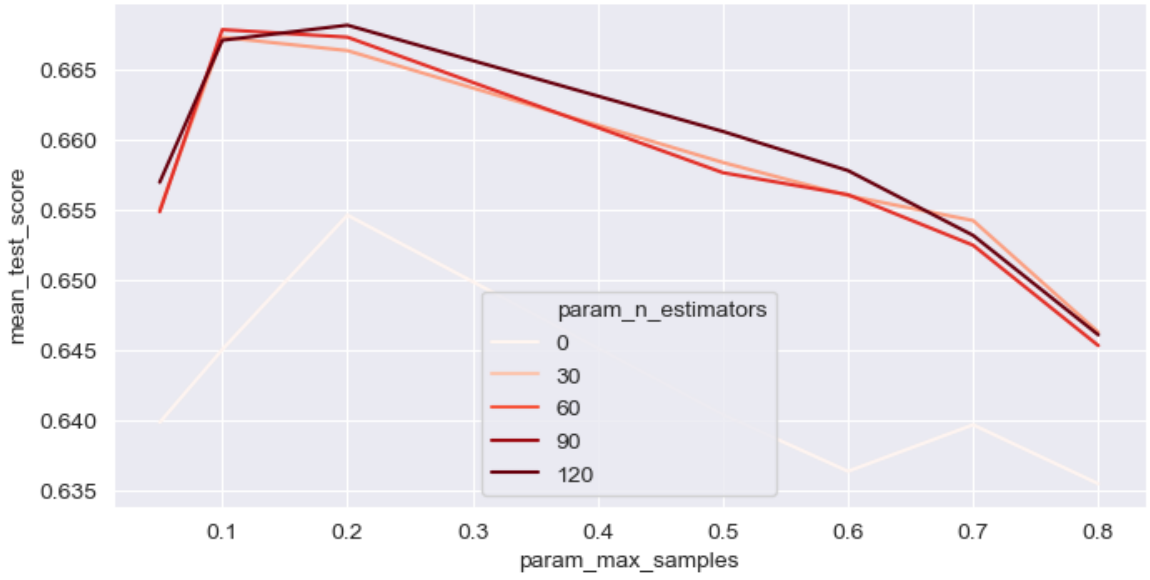
In [502]:
```
output.head()
```

Out[502]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_samples	param_
0	0.144028	0.007127	0.018425	0.001336	0.05	
1	0.185165	0.010854	0.034259	0.001887	0.05	
2	0.241210	0.064224	0.057269	0.005638	0.05	
3	0.296006	0.043199	0.060954	0.001745	0.05	
4	0.146673	0.002500	0.017758	0.001133	0.1	

In [503]:
```
fig = plt.figure(figsize=(8,4),dpi=100)
sns.lineplot(x=output['param_max_samples'],y=output['mean_test_score'],hue=ou
             palette='Reds')
```

Out[503]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17fe2154908>
```



In [504]:
```
pasting_tree_score = grid.best_score_
```

In [505]:
```
print('The Best R Squared Value that we are getting is {}'.format(round(pasti
```

```
The Best R Squared Value that we are getting is 0.6256
```

Ada Boost on Lasso And Linear Regression

Ada boost with Lasso

In [506]:
```
from sklearn.ensemble import AdaBoostRegressor

ada_lasso = AdaBoostRegressor(Lasso(alpha=100,random_state=0), random_state=0
ada_lasso.fit(X_train, y_train)
```

Out[506]:
```
AdaBoostRegressor(base_estimator=Lasso(alpha=100, random_state=0),
                  random_state=0)
```

In [507]:
```
grid_params = {'n_estimators':[10,40,70,100],'learning_rate':[0.0001, 0.001,
```

In [508]:
```
grid = GridSearchCV(ada_lasso,grid_params,cv=kfold,verbose=3)
```

In [509]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
[CV] learning_rate=0.0001, n_estimators=10 ............................
[CV]  learning_rate=0.0001, n_estimators=10, score=0.635, total=   0.2s
[CV] learning_rate=0.0001, n_estimators=10 ............................

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

[CV]  learning_rate=0.0001, n_estimators=10, score=0.561, total=   0.2s
[CV] learning_rate=0.0001, n_estimators=10 ............................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.3s remaining:
0.0s
```

In [510]:
```
grid.best_params_
```

Out[510]:
```
{'learning_rate': 0.01, 'n_estimators': 10}
```

In [511]:
```
output = pd.DataFrame(grid.cv_results_)
```

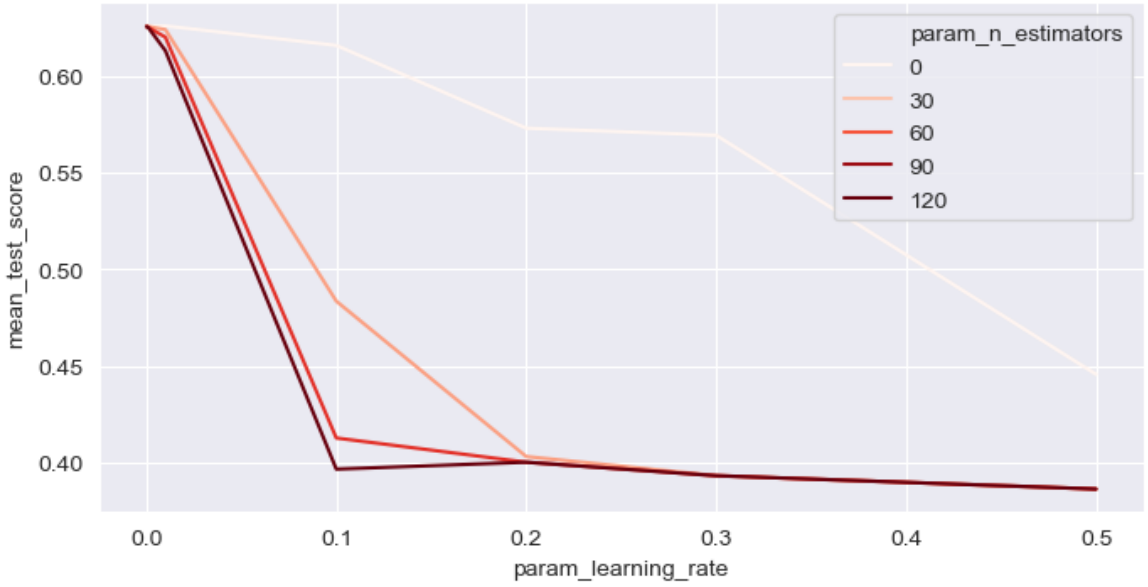
In [512]:
```
output.head()
```

Out[512]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_r
0	0.218223	0.021207	0.006377	0.001014	0.0001	
1	0.810242	0.039070	0.014960	0.001545	0.0001	
2	1.394477	0.048305	0.029112	0.003851	0.0001	
3	2.046318	0.103695	0.042691	0.003412	0.0001	
4	0.206647	0.005217	0.006183	0.000746	0.001	

In [513]:
```
fig = plt.figure(figsize=(8,4),dpi=100)
sns.lineplot(x=output['param_learning_rate'],y=output['mean_test_score'],hue=
             palette='Reds')
```

Out[513]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17fed2e4898>
```



In [514]:
```
ada_lasso_score = grid.best_score_
```

In [515]:
```
print('The Best R Squared Value that we are getting is {}'.format(round(ada_l
```

```
The Best R Squared Value that we are getting is 0.6259
```

Ada Boost with Linear Regression

In [516]:
```
ada_linearReg = AdaBoostRegressor(LinearRegression(), random_state=0)
ada_linearReg.fit(X_train, y_train)
```

Out[516]:
```
AdaBoostRegressor(base_estimator=LinearRegression(), random_state=0)
```

In [517]:
```
grid_params = {'n_estimators':[10,40,70,100],'learning_rate':[0.0001, 0.001,
```

In [518]:
```
grid = GridSearchCV(ada_linearReg,grid_params,cv=kfold,verbose=3)
```

In [519]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
[CV] learning_rate=0.0001, n_estimators=10 ............................
[CV]  learning_rate=0.0001, n_estimators=10, score=0.635, total=   0.1s
[CV] learning_rate=0.0001, n_estimators=10 ............................

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

[CV]  learning_rate=0.0001, n_estimators=10, score=0.562, total=   0.1s
[CV] learning_rate=0.0001, n_estimators=10 ............................
[CV]  learning_rate=0.0001, n_estimators=10, score=0.640, total=   0.1s
[CV] learning_rate=0.0001, n_estimators=10 ............................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:
0.0s
```

In [520]:
```
grid.best_params_
```

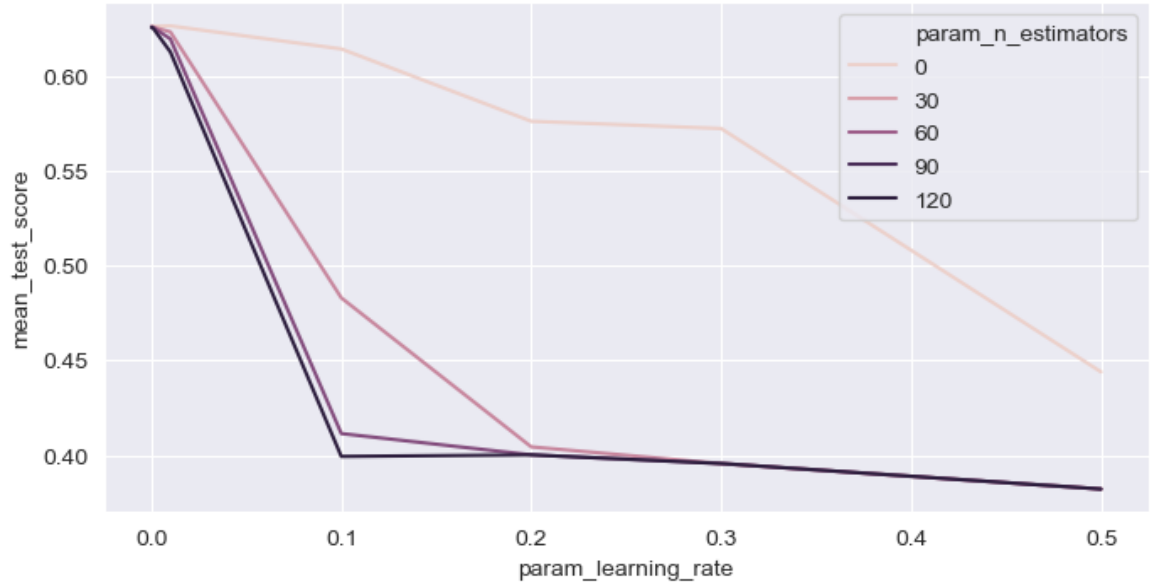
Out[520]:
```
{'learning_rate': 0.01, 'n_estimators': 10}
```

In [521]:
```
ada_linerreg_score = grid.best_score_
```

In [522]:
```
output= pd.DataFrame(grid.cv_results_)
```

In [523]:
```
fig = plt.figure(figsize=(8,4),dpi=100)
sns.lineplot(x=output['param_learning_rate'],y=output['mean_test_score'],hue=
```

Out[523]: <matplotlib.axes._subplots.AxesSubplot at 0x17fd9834748>



In [524]:
```
print('The Best R Squared Value that we are getting is {}'.format(round(ada_l
```

```
The Best R Squared Value that we are getting is 0.6263
```

Gradient Boosting

In [525]:
```
from  sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, random_state=0)
gbrt.fit(X_train, y_train)
```

Out[525]: GradientBoostingRegressor(max_depth=2, random_state=0)

In [526]:
```
grid_params = {'max_depth':[1,2,3,4,5],'n_estimators':[10,40,70,100],'learnin
```

In [527]:
```
grid = GridSearchCV(gbrt,grid_params,cv=kfold,verbose=3)
```

In [528]:
```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 140 candidates, totalling 700 fits
[CV] learning_rate=0.0001, max_depth=1, n_estimators=10 ..............
[CV]  learning_rate=0.0001, max_depth=1, n_estimators=10, score=-0.000, t
otal=   0.1s
[CV] learning_rate=0.0001, max_depth=1, n_estimators=10 ..............
[CV]  learning_rate=0.0001, max_depth=1, n_estimators=10, score=-0.001, t
otal=   0.1s
[CV] learning_rate=0.0001, max_depth=1, n_estimators=10 ..............

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
```

In [529]:
```
grid.best_params_
```

Out[529]:
```
{'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 100}
```

In [530]:
```
gb_score = grid.best_score_
```

In [531]:
```
output = pd.DataFrame(grid.cv_results_)
```

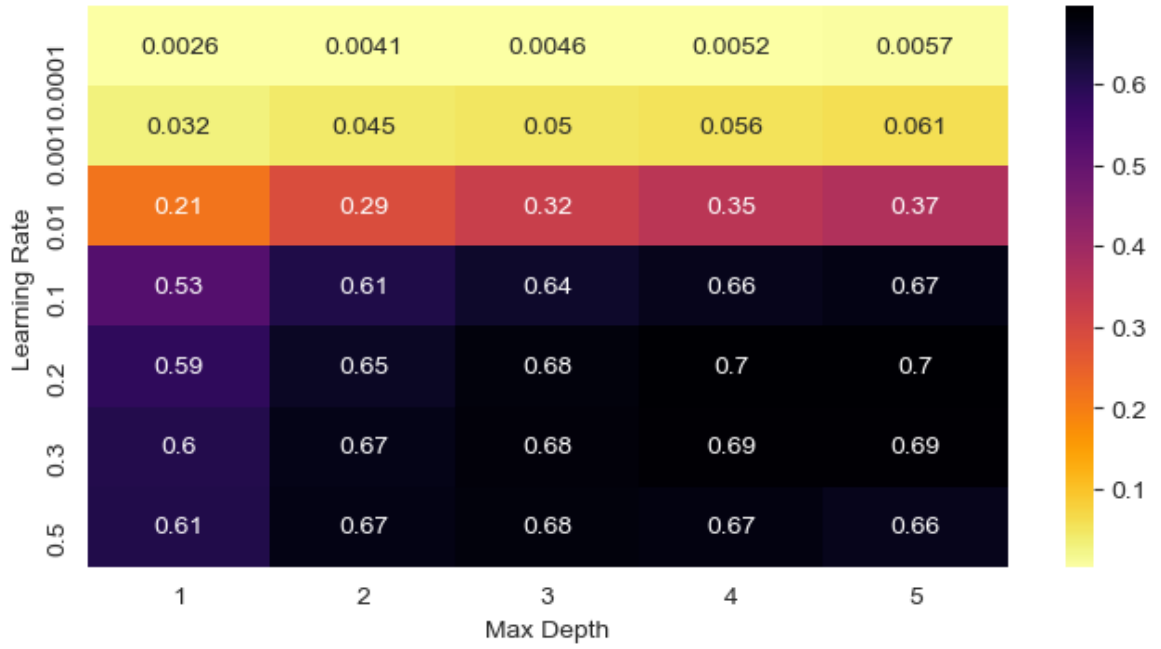
In [532]:
```
output.head()
```

Out[532]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_r
0	0.065479	0.001333	0.001996	0.000002	0.0001	
1	0.252336	0.007050	0.002795	0.000400	0.0001	
2	0.499082	0.055248	0.003192	0.000398	0.0001	
3	0.624537	0.006950	0.003790	0.000399	0.0001	
4	0.120876	0.000977	0.002394	0.000489	0.0001	

In [533]:
```
fig = plt.figure(figsize=(8,4),dpi=100)
sns.heatmap(output.pivot_table(index='param_learning_rate',columns='param_max
plt.xlabel('Max Depth')
plt.ylabel('Learning Rate')
```

Out[533]: Text(72.72222222222221, 0.5, 'Learning Rate')



In [534]:
```
print('The Best R Squared Value that we are getting is {}'.format(round(gb_sc
```

```
The Best R Squared Value that we are getting is 0.7118
```

PCA

Implementing Principal Component Analysis

In [535]:
```
from sklearn.decomposition import PCA
```

In [536]:
```
pca = PCA(n_components=0.90,random_state=0)
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)
```

In [537]:
```
X_test_reduced.shape
```

Out[537]:
```
(5404, 7)
```

We could cut down the features to lesser than half and also keep almost 90% of the variance in the dataset

Running All the previous Models in part 1 on the reduced dataset

KNN with PCA

In [538]:
```
n = list(range(1,51))
```

In [539]:
```
knn = KNeighborsRegressor()
```

In [540]:
```
grid_params  = {'n_neighbors':n}
```

In [541]:
```
kfold = KFold(n_splits=5,random_state=0)
```

```
C:\Users\behab\Anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

In [542]:
```
grid = GridSearchCV(knn,grid_params,cv=kfold,verbose=3)
```

In [543]:
```
grid.fit(X_train_reduced,y_train)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
[CV] n_neighbors=1 ...................................................
[CV] ....................... n_neighbors=1, score=0.434, total=   0.1s
[CV] n_neighbors=1 ...................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
```

In [544]:
```
grid.best_params_
```

Out[544]:
```
{'n_neighbors': 10}
```

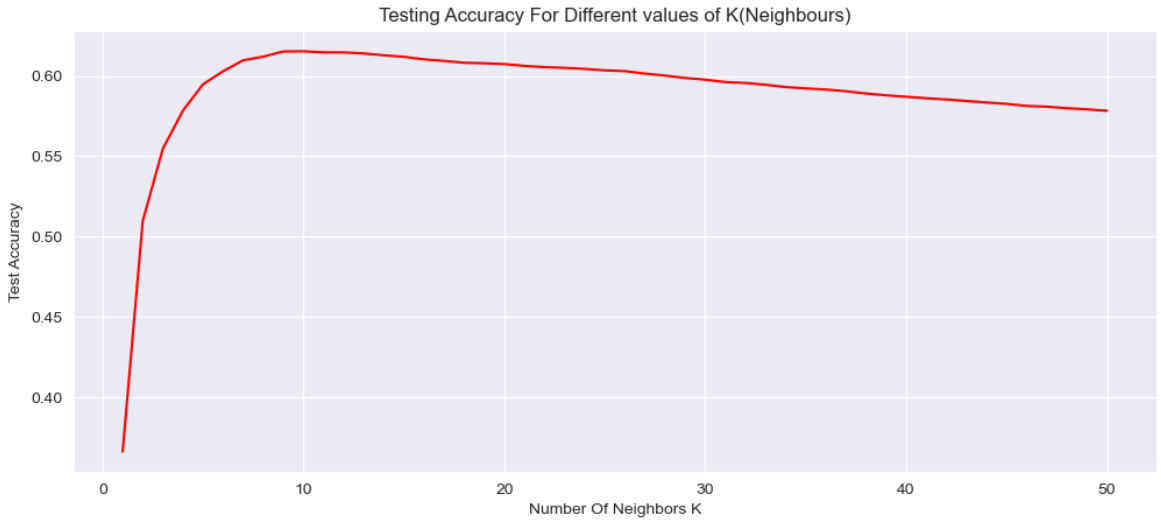
In [545]:
```
grid.best_score_
```

Out[545]:
```
0.6154049837782184
```

In [546]:
```
results_knn_pca = pd.DataFrame(grid.cv_results_)
```

In [547]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(data=results_knn_pca,x=results_knn_pca['param_n_neighbors'],y=re
plt.xlabel('Number Of Neighbors K')
plt.ylabel('Test Accuracy')
plt.title('Testing Accuracy For Different values of K(Neighbours)')
```

Out[547]: Text(0.5, 1.0, 'Testing Accuracy For Different values of K(Neighbours)')



In [548]:
```
knn = KNeighborsRegressor(n_neighbors=13)
```

In [549]:
```
cross_val_knn = cross_val_score(knn,X_train_reduced,y_train,cv=kfold,verbose=
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:
0.0s

[CV]  ................................................................
[CV] ................................. , score=0.594, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.579, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.602, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.675, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.620, total=   0.1s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.3s finished
```

In [550]:
```
knn_mean_score_cv_pca = cross_val_knn.mean()
```

In [551]:
```
print('The cross validated R-Squared score of the best knn model with 13 neig
```

```
The cross validated R-Squared score of the best knn model with 13 neighbour
s on the reduced dataset by pca is: 0.6141
```

Linear Regression with PCA

In [552]:
```
lm = LinearRegression()
```

In [553]:
```
lm.fit(X_train_reduced,y_train)
```

Out[553]:
```
LinearRegression()
```

In [554]:
```
lm.score(X_train_reduced,y_train)
```

Out[554]:
```
0.5748458975510977
```

In [555]:
```
lm.score(X_test_reduced,y_test)
```

Out[555]:
```
0.560693047850001
```

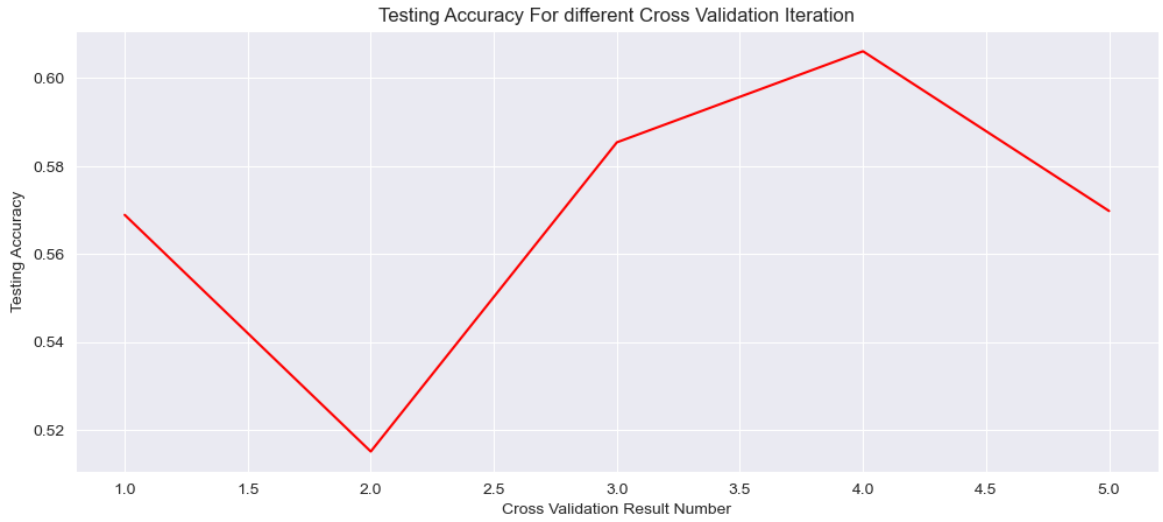
In [556]:
```
cross_val_lm = cross_val_score(lm,X_train_reduced,y_train,cv=kfold,verbose=3)
```

```
[CV]  ................................................................
[CV] ................................... , score=0.569, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.515, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.585, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.606, total=   0.0s
[CV]  ................................................................
[CV] ................................... , score=0.570, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

In [557]:
```
lm_mean_score_cv_pca= cross_val_lm.mean()
```

In [558]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
plt.plot(np.arange(1,6),cross_val_lm,c='red')
plt.xlabel('Cross Validation Result Number')
plt.ylabel('Testing Accuracy')
plt.title('Testing Accuracy For different Cross Validation Iteration')
```

Out[558]: Text(0.5, 1.0, 'Testing Accuracy For different Cross Validation Iteration')



In [559]:
```
print('The cross validated Test score(R-Squared) of the best Linear model wit
```

```
The cross validated Test score(R-Squared) of the best Linear model with red
uced dataset is: 0.5691
```

Ridge Regression with PCA

In [560]:
```
ridge = Ridge(random_state=0)
```

In [561]:
```
ridge.fit(X_train_reduced,y_train)
```

Out[561]: Ridge(random_state=0)

In [562]:
```
ridge.score(X_train_reduced,y_train)
```

Out[562]: 0.5748437522584566

In [563]:
```
ridge.score(X_test_reduced,y_test)
```

Out[563]: 0.560669811728379

In [564]:
```
ridge = Ridge(random_state=0)
```

In [565]:
```
alpha_range = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
```

In [566]:
```
grid_params = {'alpha':[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]}
```

In [567]:
```
grid = GridSearchCV(ridge,grid_params,cv=kfold,verbose=3)
```

In [568]:
```
grid.fit(X_train_reduced,y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.569, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.515, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.585, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.606, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.570, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.569, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.515, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.585, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.606, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.570, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.569, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.515, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.585, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.606, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.570, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ......................... alpha=0.1, score=0.569, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ......................... alpha=0.1, score=0.515, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ......................... alpha=0.1, score=0.585, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ......................... alpha=0.1, score=0.606, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ......................... alpha=0.1, score=0.570, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ........................... alpha=1, score=0.569, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ........................... alpha=1, score=0.516, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ........................... alpha=1, score=0.585, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ........................... alpha=1, score=0.606, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ........................... alpha=1, score=0.570, total=   0.0s
[CV] alpha=10 ........................................................
[CV] .......................... alpha=10, score=0.567, total=   0.0s
```

```
[CV] alpha=10 ........................................................
[CV] ........................... alpha=10, score=0.519, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s


[CV] alpha=10 ........................................................
[CV] ........................... alpha=10, score=0.583, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ........................... alpha=10, score=0.607, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ........................... alpha=10, score=0.570, total=   0.0s
[CV] alpha=100 .......................................................
[CV] ........................... alpha=100, score=0.538, total=   0.0s
[CV] alpha=100 .......................................................
[CV] ........................... alpha=100, score=0.523, total=   0.0s
[CV] alpha=100 .......................................................
[CV] ........................... alpha=100, score=0.553, total=   0.0s
[CV] alpha=100 .......................................................
[CV] ........................... alpha=100, score=0.595, total=   0.0s
[CV] alpha=100 .......................................................
[CV] ........................... alpha=100, score=0.556, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ........................... alpha=1000, score=0.309, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ........................... alpha=1000, score=0.346, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ........................... alpha=1000, score=0.317, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ........................... alpha=1000, score=0.367, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ........................... alpha=1000, score=0.341, total=   0.0s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.057, total=   0.0s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.069, total=   0.0s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.058, total=   0.0s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.071, total=   0.0s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.066, total=   0.0s

[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    0.2s finished
```

Out[568]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=0, shuffle=False),
             estimator=Ridge(random_state=0),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 10
00,
                                   10000]},
             verbose=3)
```

In [569]:
```
grid.best_params_
```

Out[569]:
```
{'alpha': 10}
```

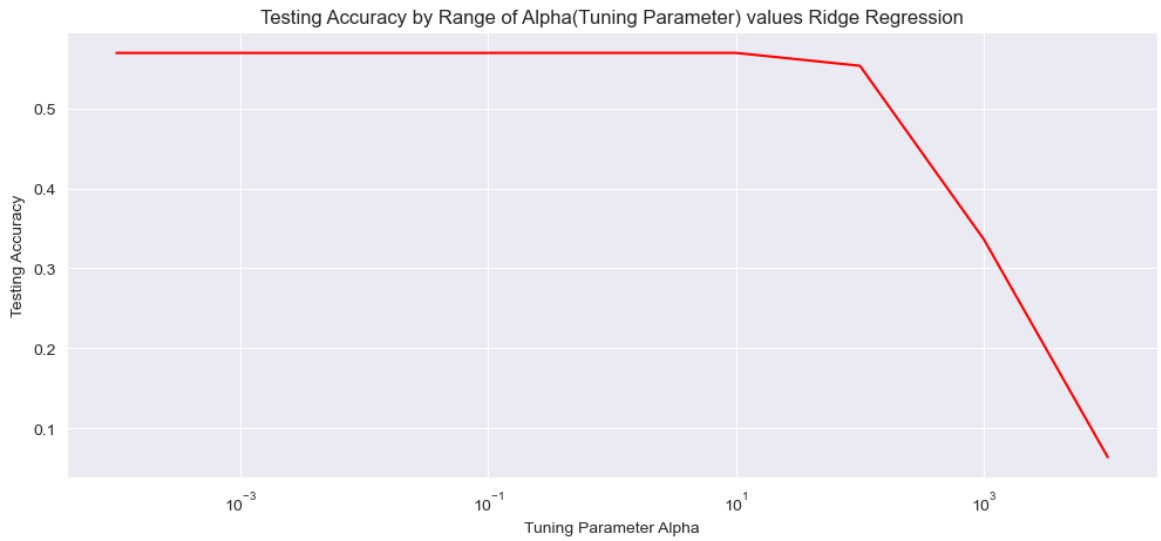
In [570]:
```
grid.best_score_
```

Out[570]:
```
0.5691137357233904
```

In [571]:
```
results_ridge_pca = pd.DataFrame(grid.cv_results_)
```

In [572]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(data=results_ridge_pca,x='param_alpha',y='mean_test_score',color
plt.xlabel('Tuning Parameter Alpha')
plt.ylabel('Testing Accuracy')
plt.xscale('log')
plt.title('Testing Accuracy by Range of Alpha(Tuning Parameter) values Ridge
```

Out[572]:
```
Text(0.5, 1.0, 'Testing Accuracy by Range of Alpha(Tuning Parameter) values
Ridge Regression')
```



In [573]:
```
ridge = Ridge(alpha=10,random_state=0)
```

In [574]:
```
cross_val_ridge = cross_val_score(ridge,X_train_reduced,y_train,cv=kfold,verb
```

```
[CV]  ................................................................
[CV] ................................. , score=0.567, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.519, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.583, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.607, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.570, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

In [575]:
```
ridge_mean_score_cv_pca= cross_val_ridge.mean()
```

In [576]:
```
print('The cross validated Testing score (R-Squared) of the best Ridge model
```

```
The cross validated Testing score (R-Squared) of the best Ridge model with
reduced dataset is: 0.5691
```

Lasso Regression with PCA

In [577]:
```
lasso = Lasso(random_state=0)
```

In [578]:
```
lasso.fit(X_train_reduced,y_train)
```

Out[578]:
```
Lasso(random_state=0)
```

In [579]:
```
lasso.score(X_train_reduced,y_train)
```

Out[579]:
```
0.5748458950768187
```

In [580]:
```
lasso.score(X_test_reduced,y_test)
```

Out[580]:
```
0.5606925605786921
```

In [581]:
```
alpha_range = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
```

In [582]:
```
grid_params = {'alpha':[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]}
```

In [583]:
```
grid = GridSearchCV(lasso,grid_params,cv=kfold,verbose=3)
```

In [584]:
```
grid.fit(X_train_reduced,y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s

Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.569, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.515, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.585, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.606, total=   0.0s
[CV] alpha=0.0001 ....................................................
[CV] ....................... alpha=0.0001, score=0.570, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.569, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.515, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.585, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.606, total=   0.0s
[CV] alpha=0.001 .....................................................
[CV] ........................ alpha=0.001, score=0.570, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.569, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.515, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.585, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.606, total=   0.0s
[CV] alpha=0.01 ......................................................
[CV] ......................... alpha=0.01, score=0.570, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.569, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.515, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.585, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.606, total=   0.0s
[CV] alpha=0.1 .......................................................
[CV] ........................... alpha=0.1, score=0.570, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ............................. alpha=1, score=0.569, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ............................. alpha=1, score=0.515, total=   0.0s
[CV] alpha=1 .........................................................
[CV] ............................. alpha=1, score=0.585, total=   0.0s
[CV] alpha=1 .........................................................
```

```
[CV] ............................ alpha=1, score=0.606, total=   0.0s
[CV] alpha=1 .....................................................
[CV] ............................ alpha=1, score=0.570, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.569, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.515, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.585, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.606, total=   0.0s
[CV] alpha=10 ....................................................
[CV] ........................... alpha=10, score=0.570, total=   0.0s
[CV] alpha=100 ...................................................
[CV] ........................... alpha=100, score=0.569, total=   0.0s
[CV] alpha=100 ...................................................
[CV] ........................... alpha=100, score=0.516, total=   0.0s
[CV] alpha=100 ...................................................
[CV] ........................... alpha=100, score=0.585, total=   0.0s
[CV] alpha=100 ...................................................
[CV] ........................... alpha=100, score=0.606, total=   0.0s
[CV] alpha=100 ...................................................
[CV] ........................... alpha=100, score=0.570, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ........................... alpha=1000, score=0.563, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ........................... alpha=1000, score=0.519, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ........................... alpha=1000, score=0.578, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ........................... alpha=1000, score=0.606, total=   0.0s
[CV] alpha=1000 ..................................................
[CV] ........................... alpha=1000, score=0.569, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................... alpha=10000, score=0.404, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................... alpha=10000, score=0.401, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................... alpha=10000, score=0.424, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................... alpha=10000, score=0.476, total=   0.0s
[CV] alpha=10000 .................................................
[CV] ........................... alpha=10000, score=0.444, total=   0.0s

[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    0.1s finished
```

Out[584]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=0, shuffle=False),
             estimator=Lasso(random_state=0),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 10
00,
                                   10000]},
             verbose=3)
```

In [585]:
```
grid.best_params_
```

Out[585]:
```
{'alpha': 100}
```

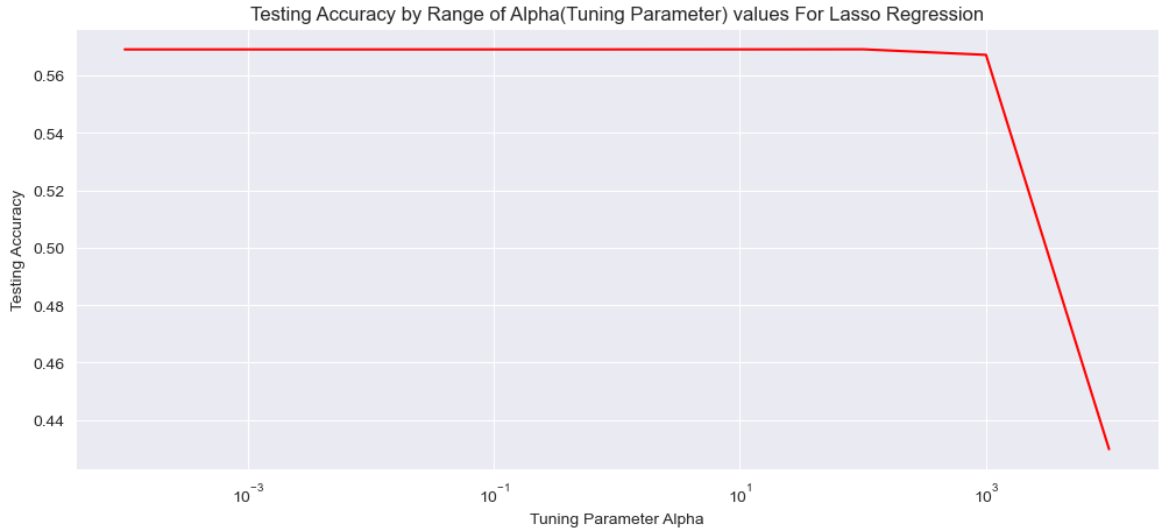
In [586]:
```
grid.best_score_
```

Out[586]:
```
0.5691025412750093
```

In [587]:
```
results_lasso_pca = pd.DataFrame(grid.cv_results_)
```

In [588]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(data=results_lasso_pca,x='param_alpha',y='mean_test_score',color
plt.xlabel('Tuning Parameter Alpha')
plt.ylabel('Testing Accuracy')
plt.xscale('log')
plt.title('Testing Accuracy by Range of Alpha(Tuning Parameter) values For La
```

Out[588]:
```
Text(0.5, 1.0, 'Testing Accuracy by Range of Alpha(Tuning Parameter) values
For Lasso Regression')
```



In [589]:
```
lasso = Lasso(alpha=100)
```

In [590]:
```
cross_val_lasso = cross_val_score(lasso,X_train_reduced,y_train,cv=kfold,verb
```

```
[CV]  ................................................................
[CV] ................................. , score=0.569, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.516, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.585, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.606, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.570, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

In [591]:
```
lasso_mean_score_cv_pca = cross_val_lasso.mean()
```

In [592]:
```
print('The cross validated Testing score(R-Squared) of the best Lasso model w
```

```
The cross validated Testing score(R-Squared) of the best Lasso model with r
educed dataset is: 0.5691
```

In [593]:
```
# Decision Tree With PCA
```

In [594]:
```
tree = DecisionTreeRegressor(random_state=0)
```

In [595]:
```
tree.fit(X_train_reduced,y_train)
```

Out[595]:
```
DecisionTreeRegressor(random_state=0)
```

In [596]:
```
tree.score(X_train_reduced,y_train)
```

Out[596]:
```
0.9999287418430224
```

In [597]:
```
tree.score(X_test_reduced,y_test)
```

Out[597]:
```
0.37229360704466685
```

In [598]:
```
depth = list(range(1,101))
```

In [599]:
```
grid_params = {'max_depth':depth}
```

In [600]:
```
grid = GridSearchCV(tree,grid_params,cv=kfold,verbose=3)
```

In [601]:
```
grid.fit(X_train_reduced,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
[CV] max_depth=1 .....................................................
[CV] ......................... max_depth=1, score=0.216, total=   0.0s
[CV] max_depth=1 .....................................................
[CV] ......................... max_depth=1, score=0.198, total=   0.0s
[CV] max_depth=1 .....................................................
[CV] ......................... max_depth=1, score=0.230, total=   0.0s
[CV] max_depth=1 .....................................................
[CV] ......................... max_depth=1, score=0.249, total=   0.0s
[CV] max_depth=1 .....................................................
[CV] ......................... max_depth=1, score=0.234, total=   0.0s
[CV] max_depth=2 .....................................................
[CV] ......................... max_depth=2, score=0.313, total=   0.0s
[CV] max_depth=2 .....................................................
[CV] ......................... max_depth=2, score=0.254, total=   0.0s
[CV] max_depth=2 .....................................................
[CV] ......................... max_depth=2, score=0.319, total=   0.0s
[CV] max_depth=2 .....................................................
[CV] ......................... max_depth=2, score=0.367, total=   0.0s
```

In [602]:
```
grid.best_params_
```

Out[602]:
```
{'max_depth': 6}
```

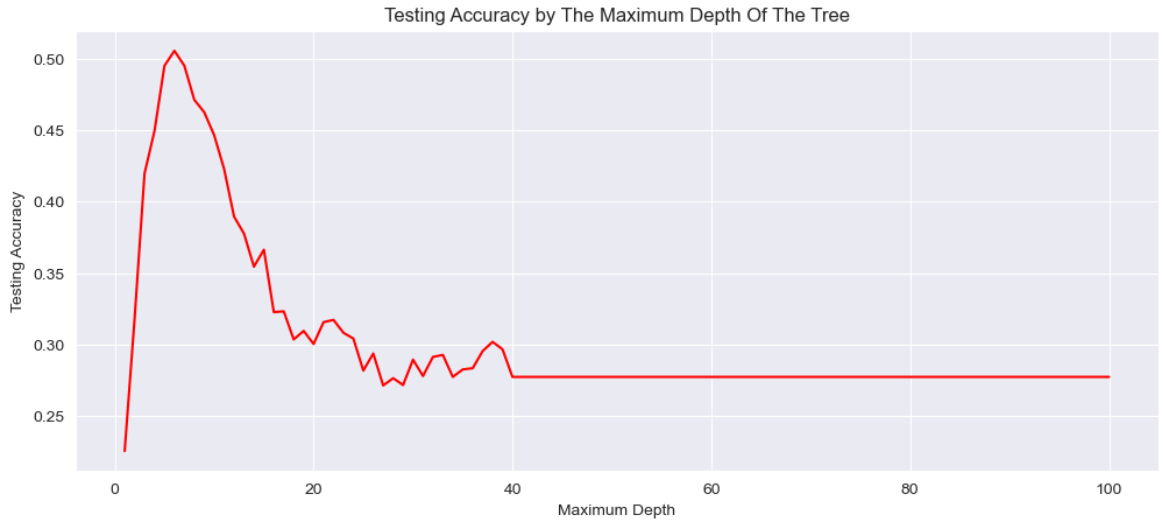
In [603]:
```
grid.best_score_
```

Out[603]:
```
0.5054241570335261
```

In [604]:
```
results_tree_pca = pd.DataFrame(grid.cv_results_)
```

In [605]:
```
fig = plt.figure(figsize=(12,5),dpi=100)
sns.lineplot(data=results_tree_pca,x='param_max_depth',y='mean_test_score',cc
plt.xlabel('Maximum Depth')
plt.ylabel('Testing Accuracy')
plt.title('Testing Accuracy by The Maximum Depth Of The Tree')
```

Out[605]:
```
Text(0.5, 1.0, 'Testing Accuracy by The Maximum Depth Of The Tree')
```



In [606]:
```
tree = DecisionTreeRegressor(max_depth=9)
```

In [607]:
```
tree.fit(X_train_reduced,y_train)
```

Out[607]:
```
DecisionTreeRegressor(max_depth=9)
```

In [608]:
```
cross_val_tree = cross_val_score(tree,X_train_reduced,y_train,verbose=3,cv=kf
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s

[CV]  ................................................................
[CV] ................................... , score=0.546, total=   0.1s
[CV]  ................................................................
[CV] ................................... , score=0.335, total=   0.1s
[CV]  ................................................................
[CV] ................................... , score=0.506, total=   0.1s
[CV]  ................................................................
[CV] ................................... , score=0.414, total=   0.1s
[CV]  ................................................................
[CV] ................................... , score=0.463, total=   0.1s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.2s finished
```

In [609]:
```
tree_mean_score_cv_pca = cross_val_tree.mean()
```

In [610]:
```
print('The cross validated Test score (R-Squared) of the best Decision Tree R
```

```
The cross validated Test score (R-Squared) of the best Decision Tree Regres
sor model on reduced dataset is: 0.4529
```

Sampling The Data For More Complicated Datasets

In [611]:
```
np.random.seed(0)
sample_train = np.random.randint(5404,16209,1500)
sample_test = np.random.randint(0,5404,1000)
```

In [612]:
```
X_train_reduced = pd.DataFrame(X_train_reduced)
X_test_reduced = pd.DataFrame(X_test_reduced)
```

In [613]:
```
X_train_sample_reduced = X_train_reduced.iloc[sample_train,:]
y_train_sample = y_train.iloc[sample_train]
```

In [614]:
```
X_test_sample_reduced = X_train_reduced.iloc[sample_test,:]
y_test_sample = y_test.iloc[sample_test]
```

Random Forrest after PCA

In [615]:
```
rf = RandomForestRegressor()
```

In [616]:
```
rf.fit(X_train_sample_reduced,y_train_sample)
```

Out[616]:
```
RandomForestRegressor()
```

In [617]:
```
rf.score(X_train_sample_reduced,y_train_sample)
```

Out[617]:
```
0.9426920974456848
```

In [618]:
```
rf.score(X_test_sample_reduced,y_test_sample)
```

Out[618]:
```
-0.3940384335805436
```

In [619]:
```
grid_params = {'n_estimators':[400,500],'max_depth':depth[:51]}
grid = GridSearchCV(rf,grid_params,cv=kfold,verbose=3)
```

In [620]:
```
grid.fit(X_train_sample_reduced,y_train_sample)
```

```
Fitting 5 folds for each of 102 candidates, totalling 510 fits
[CV] max_depth=1, n_estimators=400 ....................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.

[CV] ....... max_depth=1, n_estimators=400, score=0.310, total=   0.7s
[CV] max_depth=1, n_estimators=400 ....................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remaining:
0.0s

[CV] ....... max_depth=1, n_estimators=400, score=0.303, total=   0.6s
[CV] max_depth=1, n_estimators=400 ....................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.2s remaining:
0.0s
```

In [621]:
```
grid.best_params_
```

Out[621]:
```
{'max_depth': 13, 'n_estimators': 500}
```

In [622]:
```
grid.best_score_
```

Out[622]:
```
0.5541906463834563
```

In [623]:
```
results_rf_pca = pd.DataFrame(grid.cv_results_)
```

In [624]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
#sns.heatmap(results_rf.pivot_table(index='param_n_estimators',columns='param
sns.lineplot(x=results_rf_pca['param_max_depth'],y=results_rf_pca['mean_test_
plt.ylabel('Number Of Trees(param_n_estimators)')
plt.xlabel('Maximum Depth')
plt.title('HeatMap Of Mean Test Score With respect To param_n_estimators and
```

Out[624]:
```
Text(0.5, 1.0, 'HeatMap Of Mean Test Score With respect To param_n_estimato
rs and depth parameters')
```

In [625]:
```
rf = RandomForestRegressor(n_estimators=400,max_depth=38)
```

In [626]:
```
cross_val_rf = cross_val_score(rf,X_train_sample_reduced,y_train_sample,cv=kf
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.

[CV]  ................................................................
[CV] ..................................... , score=0.640, total=   2.1s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.0s remaining:
0.0s

[CV] ..................................... , score=0.575, total=   2.3s
[CV]  ................................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    4.4s remaining:
0.0s

[CV] ..................................... , score=0.529, total=   2.3s
[CV]  ................................................................
[CV] ..................................... , score=0.489, total=   2.1s
[CV]  ................................................................
[CV] ..................................... , score=0.489, total=   2.0s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.8s finished
```

In [627]:
```
rf_mean_score_cv_pca = cross_val_rf.mean()
```

In [628]:
```
print('The cross validated Testing score(R-Squared) of the best Random Forest
```

```
The cross validated Testing score(R-Squared) of the best Random Forest Regr
essor model on the reduced dataset is: 0.5443
```

Linear Support Vector Regressor with PCA

In [629]:
```
linear_svr = LinearSVR()
```

In [630]:
```
linear_svr.fit(X_train_sample_reduced,y_train_sample)
```

Out[630]:
```
LinearSVR()
```

In [631]:
```
grid_params = {'C':[0.001,0.01,0.1,1,10,100,1000],'epsilon':[0, 0.01, 0.1, 0.
```

In [632]:
```
grid = GridSearchCV(linear_svr,grid_params,cv=kfold,verbose=5)
```

In [633]:
```
grid.fit(X_train_sample_reduced,y_train_sample)
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
[CV] C=0.001, epsilon=0 ..............................................
[CV] .................. C=0.001, epsilon=0, score=-2.163, total=   0.0s
[CV] C=0.001, epsilon=0 ..............................................
[CV] .................. C=0.001, epsilon=0, score=-2.768, total=   0.0s
[CV] C=0.001, epsilon=0 ..............................................
[CV] .................. C=0.001, epsilon=0, score=-2.751, total=   0.0s
[CV] C=0.001, epsilon=0 ..............................................
[CV] .................. C=0.001, epsilon=0, score=-3.501, total=   0.0s
[CV] C=0.001, epsilon=0 ..............................................
[CV] .................. C=0.001, epsilon=0, score=-2.584, total=   0.0s
[CV] C=0.001, epsilon=0.01 ...........................................
[CV] ............... C=0.001, epsilon=0.01, score=-2.163, total=   0.0s
[CV] C=0.001, epsilon=0.01 ...........................................
[CV] ............... C=0.001, epsilon=0.01, score=-2.768, total=   0.0s
[CV] C=0.001, epsilon=0.01 ...........................................
[CV] ............... C=0.001, epsilon=0.01, score=-2.751, total=   0.0s
[CV] C=0.001, epsilon=0.01 ...........................................
[CV] ............... C=0.001, epsilon=0.01, score=-3.501, total=   0.0s
```

In [634]:
```
grid.best_params_
```

Out[634]:
```
{'C': 1000, 'epsilon': 1}
```

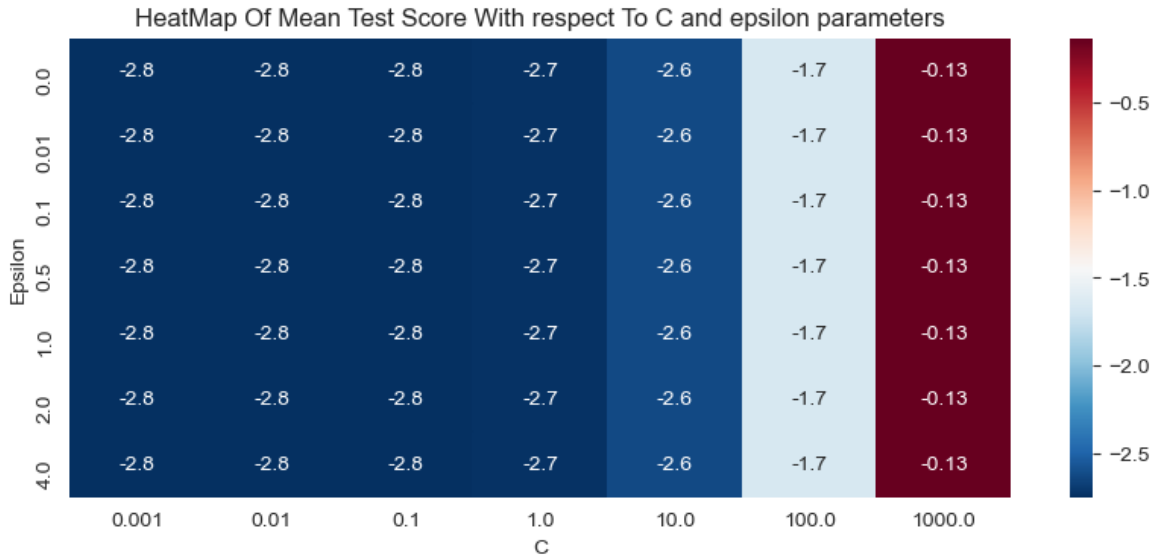
In [635]:
```
grid.best_score_
```

Out[635]:
```
-0.13213643913416434
```

In [636]:
```
results_linear_svr_pca = pd.DataFrame(grid.cv_results_)
```

In [637]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
sns.heatmap(results_linear_svr_pca.pivot_table(index='param_epsilon',columns=
plt.ylabel('Epsilon')
plt.xlabel('C')
plt.title('HeatMap Of Mean Test Score With respect To C and epsilon parameter
```

Out[637]:
```
Text(0.5, 1.0, 'HeatMap Of Mean Test Score With respect To C and epsilon pa
rameters')
```



In [638]:
```
linear_svm = LinearSVR(epsilon=0.01,C=1000)
linear_svm.fit(X_train_sample_reduced,y_train_sample)
```

Out[638]:
```
LinearSVR(C=1000, epsilon=0.01)
```

In [639]:
```
linear_svm_cross_val_score = cross_val_score(linear_svm,X_train_sample_reduce
```

```
[CV]  ................................................................
[CV] ................................. , score=-0.101, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=-0.226, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=-0.144, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=-0.171, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=-0.024, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
```

In [640]:
```
mean_cross_val_linear_svm_pca = linear_svm_cross_val_score.mean()
```

In [641]:
```
print('The Final Accuracy Score That was Achived With Our Best Linear SVM Mod
```

```
The Final Accuracy Score That was Achived With Our Best Linear SVM Model wi
th the value of C being 1000 and having epsilon as 0.01 on the reduced data
set  is: -0.1332
```

Polynomial Regression With PCA

In [642]:
```
lis = np.random.randint(0,1000,10)
```

In [643]:
```
training_score = []
testing_score = []
deg = []

for n in range(1,3):
    for j in lis:
        X_train  , X_test , y_train , y_test = train_test_split(X,y,random_st
        scaler = MinMaxScaler()

        X_trn = scaler.fit_transform(X_train)
        X_tst = scaler.transform(X_test)

        X_train_reduced = pca.fit_transform(X_trn)
        X_test_reduced = pca.transform(X_tst)

        # Polynomial Transformation
        scaler_poly = PolynomialFeatures(degree=n)
        X_poly_trn = scaler_poly.fit_transform(X_train_reduced)
        X_poly_tst =  scaler_poly.transform(X_test_reduced)

        lm_poly = LinearRegression()
        lm_poly.fit(X_poly_trn,y_train)

        pred_train = lm_poly.predict(X_poly_trn)
        pred_test = lm_poly.predict(X_poly_tst)

        training_score.append(r2_score(y_train, pred_train))
        testing_score.append(r2_score(y_test, pred_test))
        deg.append(n)
```

In [644]:
```
testing_score
```

Out[644]:
```
[0.5677523388457483,
 0.5626460001637605,
 0.5777844647513407,
 0.5586224345392403,
 0.578006088491087,
 0.5673363866103196,
 0.5726063413413547,
 0.5562158524278029,
 0.5607091700809784,
 0.5831414842815356,
 0.670466425291792,
 0.6416436428581711,
 0.6790608034826675,
 0.6239202459616526,
 0.672575421415242,
 0.6567211372033083,
 0.6458305213717356,
 0.6386755611610756,
 0.6467990868298708,
 0.6803854345025644]
```

In [645]:
```
d = [1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2]
```

In [646]:
```
dic_training = {'training_score':training_score,'degree':d}
```

In [647]:
```
training = pd.DataFrame(dic_training)
```

In [648]:
```
training = training.reset_index()
```

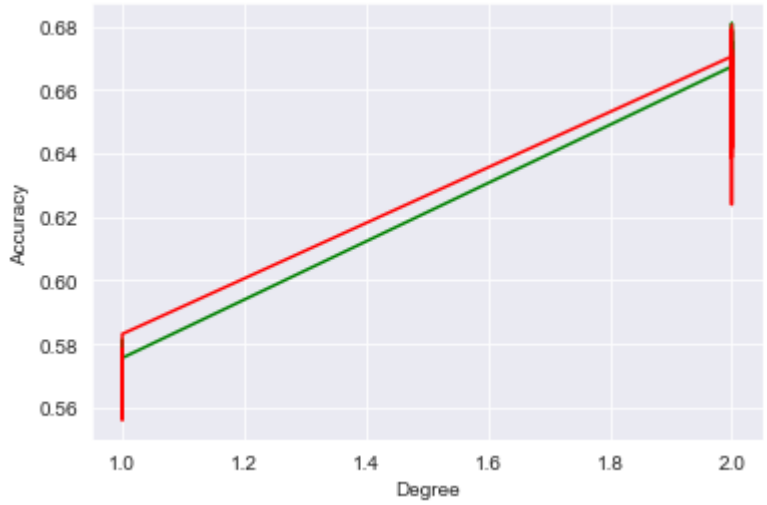
In [649]:
```
dic_testing = {'testing_score':testing_score,'degree':d}
```

In [650]:
```
testing = pd.DataFrame(dic_testing)
```

In [651]:
```
testing = testing.reset_index()
```

In [652]:
```
plt.plot(training['degree'],training['training_score'],color='green')
plt.plot(testing['degree'],testing['testing_score'],color='red')
plt.xlabel('Degree')
plt.ylabel('Accuracy')
```

Out[652]: Text(0, 0.5, 'Accuracy')



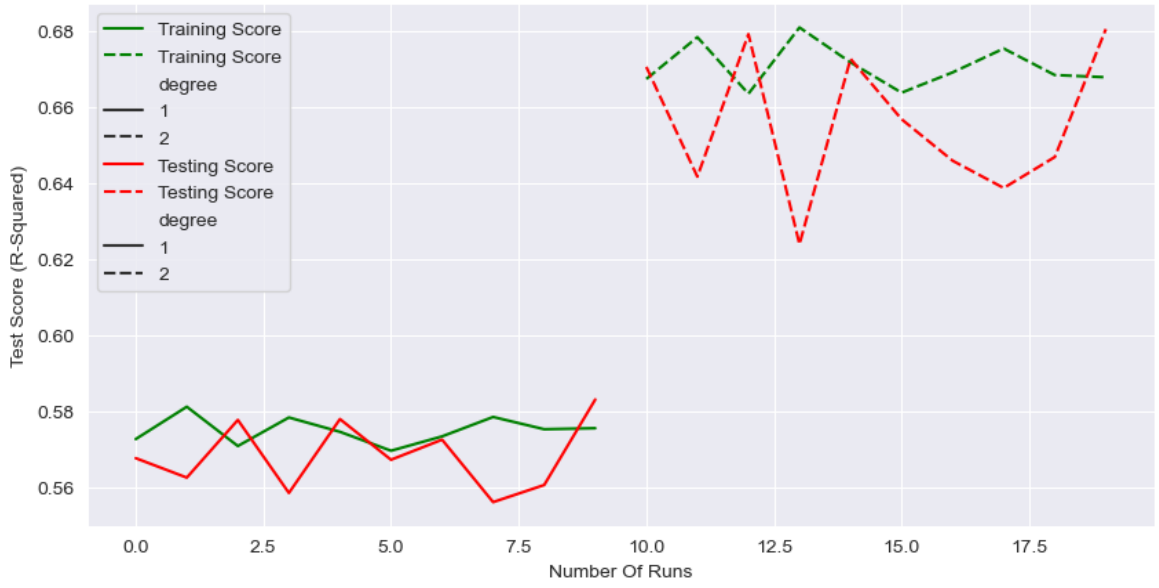
In [653]:
```
training[training['index']<=9]
```

Out[653]:

	index	training_score	degree
0	0	0.572756	1
1	1	0.581250	1
2	2	0.570939	1
3	3	0.578432	1
4	4	0.574666	1
5	5	0.569702	1
6	6	0.573467	1
7	7	0.578565	1
8	8	0.575347	1
9	9	0.575615	1

In [654]:
```
figure = plt.figure(figsize=(10,5),dpi=100)
sns.lineplot(x= training['index'],y= training['training_score'] ,color='green
sns.lineplot(x=testing['index'],y= testing['testing_score'],color='red',data=
plt.xlabel('Number Of Runs')
plt.ylabel('Test Score (R-Squared)')
```

Out[654]:
```
Text(0, 0.5, 'Test Score (R-Squared)')
```



In [655]:
```
polynomial_reg_cv_pca = np.mean(testing_score)
print(polynomial_reg_cv_pca)
```

```
0.6120449420805625
```

In [656]:
```
print('The cross validated score of the best Polynomial Regressor model is: {
```

```
The cross validated score of the best Polynomial Regressor model is: 0.612
```

In [657]:
```
X_train  , X_test , y_train , y_test = train_test_split(X,y,random_state = 0)
```

In [658]:
```
scaler = MinMaxScaler()
```

In [659]:
```
X_trn = scaler.fit_transform(X_train)
X_tst = scaler.transform(X_test)

X_train = pd.DataFrame(X_trn,columns=X_train.columns)
X_test = pd.DataFrame(X_tst,columns=X_test.columns)
```

In [660]:
```
pca = PCA(n_components=0.90,random_state=0)
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)
```

In [661]:
```
np.random.seed(0)
sample_train = np.random.randint(5404,16209,1500)
sample_test = np.random.randint(0,5404,1000)
```

In [662]:
```
X_train_reduced = pd.DataFrame(X_train_reduced)
X_test_reduced = pd.DataFrame(X_test_reduced)
```

In [663]:
```
X_train_sample_reduced = X_train_reduced.iloc[sample_train,:]
y_train_sample = y_train.iloc[sample_train]
```

In [664]:
```
X_test_sample_reduced = X_train_reduced.iloc[sample_test,:]
y_test_sample = y_test.iloc[sample_test]
```

SVM Linear Kernel with PCA

In [665]:
```
svm_linear_kernel = SVR(kernel='linear')
```

In [666]:
```
grid_params = {'C':[0.001,0.01,0.1,1,10,100,1000]}
```

In [667]:
```
grid = GridSearchCV(svm_linear_kernel,grid_params,cv=kfold,verbose=5)
```

In [668]:
```
grid.fit(X_train_sample_reduced,y_train_sample)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
[CV] C=0.001 .........................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s

[CV] ............................ C=0.001, score=-0.067, total=   0.1s
[CV] C=0.001 .........................................................
[CV] ............................ C=0.001, score=-0.132, total=   0.1s
[CV] C=0.001 .........................................................
[CV] ............................ C=0.001, score=-0.073, total=   0.1s
[CV] C=0.001 .........................................................
[CV] ............................ C=0.001, score=-0.070, total=   0.1s
[CV] C=0.001 .........................................................
[CV] ............................ C=0.001, score=-0.010, total=   0.1s
[CV] C=0.01 ..........................................................
[CV] ............................. C=0.01, score=-0.067, total=   0.0s
[CV] C=0.01 ..........................................................
[CV] ............................. C=0.01, score=-0.132, total=   0.0s
[CV] C=0.01 ..........................................................

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:
0.0s

[CV] ............................. C=0.01, score=-0.073, total=   0.1s
[CV] C=0.01 ..........................................................
[CV] ............................. C=0.01, score=-0.070, total=   0.0s
[CV] C=0.01 ..........................................................
[CV] ............................. C=0.01, score=-0.010, total=   0.0s
[CV] C=0.1 ...........................................................
[CV] .............................. C=0.1, score=-0.067, total=   0.0s
[CV] C=0.1 ...........................................................
[CV] .............................. C=0.1, score=-0.132, total=   0.0s
[CV] C=0.1 ...........................................................
[CV] .............................. C=0.1, score=-0.073, total=   0.0s
[CV] C=0.1 ...........................................................
[CV] .............................. C=0.1, score=-0.070, total=   0.0s
[CV] C=0.1 ...........................................................
[CV] .............................. C=0.1, score=-0.010, total=   0.0s
[CV] C=1 .............................................................
[CV] ................................ C=1, score=-0.067, total=   0.0s
[CV] C=1 .............................................................
[CV] ................................ C=1, score=-0.132, total=   0.0s
[CV] C=1 .............................................................
[CV] ................................ C=1, score=-0.073, total=   0.0s
[CV] C=1 .............................................................
[CV] ................................ C=1, score=-0.069, total=   0.0s
```

```
[CV] C=1 ..............................................................
[CV] ............................... C=1, score=-0.010, total=   0.0s
[CV] C=10 .............................................................
[CV] .............................. C=10, score=-0.066, total=   0.0s
[CV] C=10 .............................................................
[CV] .............................. C=10, score=-0.130, total=   0.0s
[CV] C=10 .............................................................
[CV] .............................. C=10, score=-0.072, total=   0.0s
[CV] C=10 .............................................................
[CV] .............................. C=10, score=-0.068, total=   0.0s
[CV] C=10 .............................................................
[CV] .............................. C=10, score=-0.009, total=   0.0s
[CV] C=100 ............................................................
[CV] ............................. C=100, score=-0.052, total=   0.0s
[CV] C=100 ............................................................
[CV] ............................. C=100, score=-0.115, total=   0.0s
[CV] C=100 ............................................................
[CV] ............................. C=100, score=-0.059, total=   0.0s
[CV] C=100 ............................................................
[CV] ............................. C=100, score=-0.053, total=   0.0s
[CV] C=100 ............................................................
[CV] ............................. C=100, score=0.003, total=   0.0s
[CV] C=1000 ...........................................................
[CV] ............................. C=1000, score=0.064, total=   0.0s
[CV] C=1000 ...........................................................
[CV] ............................. C=1000, score=0.005, total=   0.0s
[CV] C=1000 ...........................................................
[CV] ............................. C=1000, score=0.058, total=   0.0s
[CV] C=1000 ...........................................................
[CV] ............................. C=1000, score=0.074, total=   0.0s
[CV] C=1000 ...........................................................
[CV] ............................. C=1000, score=0.113, total=   0.0s

[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    1.5s finished
```

Out[668]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=0, shuffle=False),
             estimator=SVR(kernel='linear'),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}, verbos
e=5)
```

In [669]:
```
grid.best_params_
```

Out[669]:
```
{'C': 1000}
```

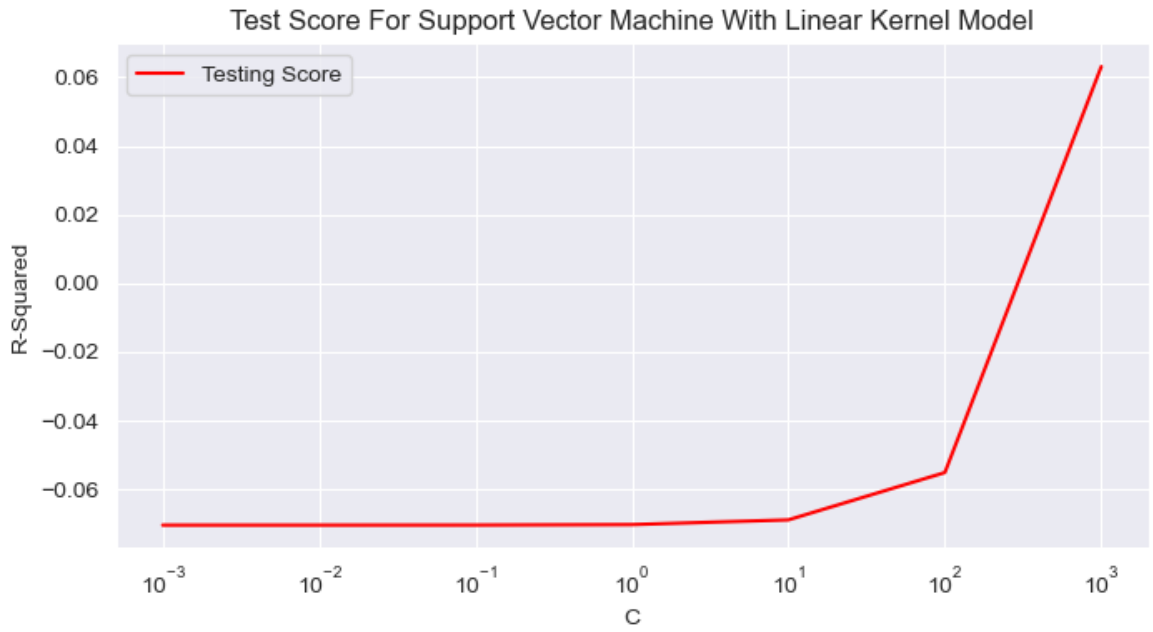
In [670]:
```
grid.best_score_
```

Out[670]:
```
0.06307925257964878
```

In [671]:
```
results_linearkernel_svm_pca = pd.DataFrame(grid.cv_results_)
```

In [672]:
```
fig = plt.figure(figsize=(8,4),dpi=100)

plt.plot(results_linearkernel_svm_pca['param_C'],
         results_linearkernel_svm_pca['mean_test_score'],color='red',label ='
plt.xscale('log')
plt.xlabel('C')
plt.ylabel('R-Squared')
plt.title('Test Score For Support Vector Machine With Linear Kernel Model')
plt.legend()
```

Out[672]: <matplotlib.legend.Legend at 0x17fe1285fd0>



In [673]:
```
svm_linear_kernel = SVR(kernel='linear',C=1000)
svm_linear_kernel.fit(X_train_sample_reduced,y_train_sample)
```

Out[673]: SVR(C=1000, kernel='linear')

In [674]:
```
svm_linearKernel_cross_val_score = cross_val_score(svm_linear_kernel,X_train_
```

```
[CV]  ................................................................
[CV] ................................. , score=0.064, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.005, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.058, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.074, total=   0.0s
[CV]  ................................................................
[CV] ................................. , score=0.113, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.1s finished
```

In [675]:
```
mean_cross_val_svm_linearKernel_pca = svm_linearKernel_cross_val_score.mean()
```

In [676]:
```
print('The corss validated score for SVM with Linear kernel on reduced datase
```

```
The corss validated score for SVM with Linear kernel on reduced dataset is
0.06
```

SVM RBF Kernel With PCA

In [677]:
```
svm_rbf_kernel = SVR(kernel='rbf')
```

In [678]:
```
grid_params = {'gamma':[0.001,0.01,0.1,1,10,100,1000],'C':[0.001,0.01,0.1,1,1
```

In [679]:
```
grid = GridSearchCV(svm_rbf_kernel,grid_params,cv=kfold,verbose=5)
```

In [680]:
```
grid.fit(X_train_sample_reduced,y_train_sample)
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
[CV] C=0.001, gamma=0.001 ............................................
[CV] ............... C=0.001, gamma=0.001, score=-0.067, total=   0.1s
[CV] C=0.001, gamma=0.001 ............................................
[CV] ............... C=0.001, gamma=0.001, score=-0.132, total=   0.1s
[CV] C=0.001, gamma=0.001 ............................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:
0.0s

[CV] ............... C=0.001, gamma=0.001, score=-0.073, total=   0.1s
[CV] C=0.001, gamma=0.001 ............................................
[CV] ............... C=0.001, gamma=0.001, score=-0.070, total=   0.1s
[CV] C=0.001, gamma=0.001 ............................................
[CV] ............... C=0.001, gamma=0.001, score=-0.010, total=   0.1s
[CV] C=0.001, gamma=0.01 .............................................
```

In [681]:
```
grid.best_params_
```

Out[681]:
```
{'C': 1000, 'gamma': 1}
```

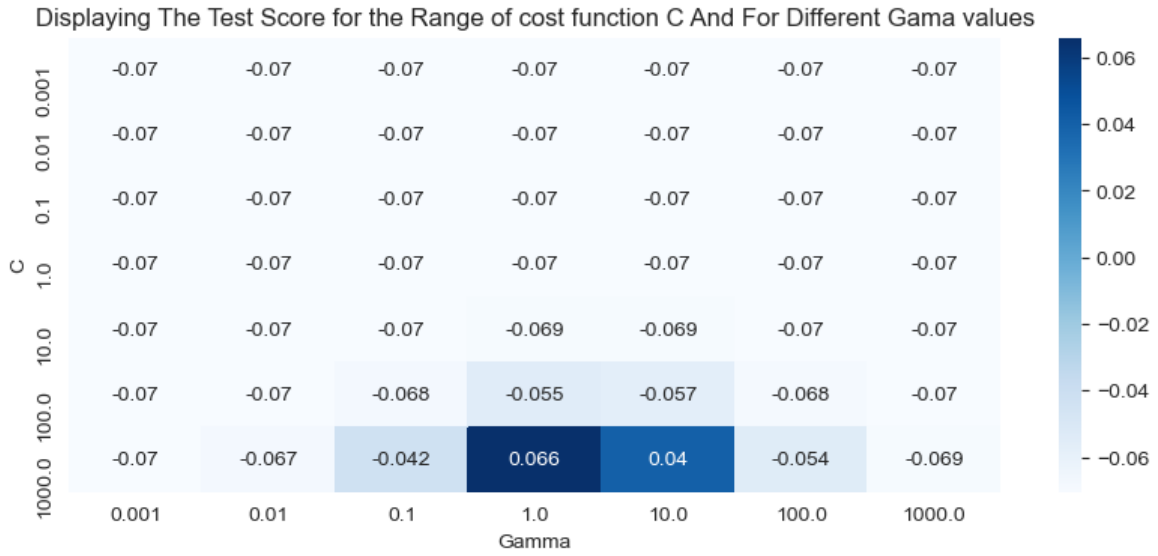
In [682]:
```
grid.best_score_
```

Out[682]:
```
0.06597207475678812
```

In [683]:
```
results_RBFkernel_svm_pca = pd.DataFrame(grid.cv_results_)
```

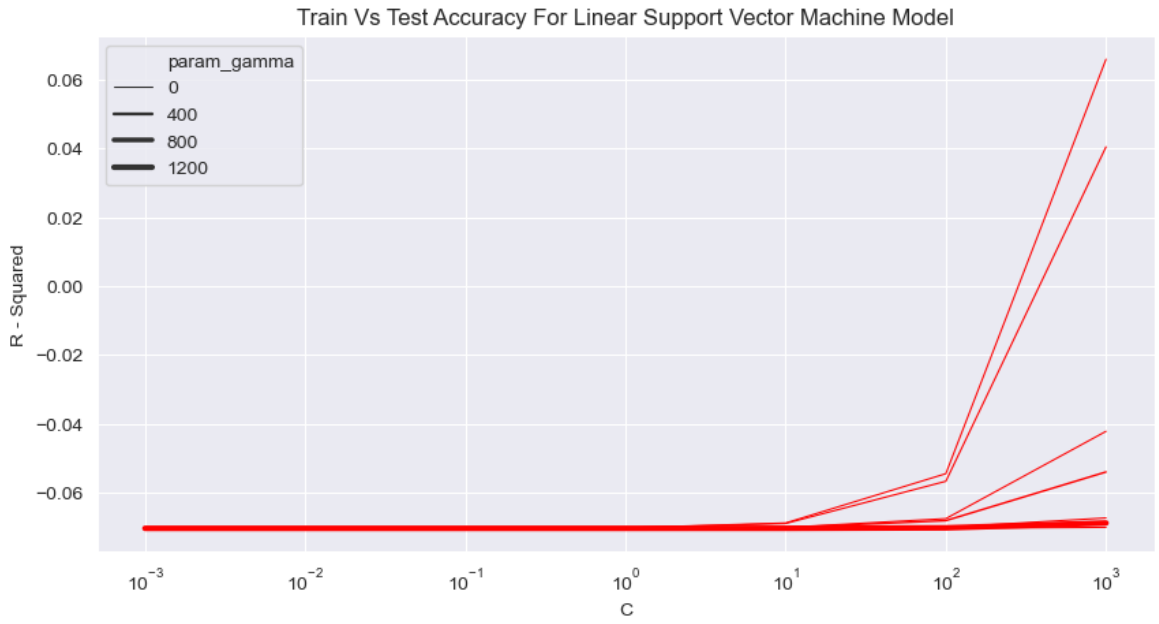
In [684]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
sns.heatmap(results_RBFkernel_svm_pca.pivot_table(index='param_C',columns='pa
plt.xlabel('Gamma')
plt.ylabel('C')
plt.title('Displaying The Test Score for the Range of cost function C And For
```

Out[684]:
```
Text(0.5, 1.0, 'Displaying The Test Score for the Range of cost function C
And For Different Gama values')
```



In [685]:
```
fig = plt.figure(figsize=(10,5),dpi=100)

sns.lineplot(x= results_RBFkernel_svm_pca['param_C'],y= results_RBFkernel_svm
size=results_RBFkernel_svm_pca['param_gamma'],color='red')

plt.xscale('log')
plt.xlabel('C')
plt.ylabel('R - Squared')
plt.title('Train Vs Test Accuracy For Linear Support Vector Machine Model')
```

Out[685]:
```
Text(0.5, 1.0, 'Train Vs Test Accuracy For Linear Support Vector Machine Mo
del')
```



In [686]:
```
svm_rbf_kernel = SVR(kernel='rbf',C=1000,gamma=1)
```

In [687]:
```
svm_rbf_kernel.fit(X_train_sample_reduced,y_train_sample)
```

Out[687]:
```
SVR(C=1000, gamma=1)
```

In [688]:
```
svm_rbfKernel_cross_val_score = cross_val_score(svm_rbf_kernel,X_train_sample
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:
0.0s

[CV]  ................................................................
[CV] ................................. , score=0.054, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.015, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.071, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.082, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.107, total=   0.1s

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.2s finished
```

In [689]:
```
mean_cross_val_svm_rbf_kernel_pca = svm_rbfKernel_cross_val_score.mean()
```

In [690]:
```
print('The Final Accuracy Score That was Achived With Our Best SVM Model With
```

```
The Final Accuracy Score That was Achived With Our Best SVM Model With RBF
Kernel with the value of C being 1000 and that of gamma being 1 for the red
uced dataset is : 0.066)
```

SVM Polynomial Kernel with PCA

In [691]:
```
svm_poly_kernal = SVR(kernel='poly')
```

In [692]:
```
grid_params = {'gamma':[0.01,0.1,1,10,100],'C':[0.01,0.1,1,10,100,1000],'degr
```

In [693]:
```
kfold = KFold(n_splits=3,random_state=0)
```

```
C:\Users\behab\Anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

In [694]:
```
grid = GridSearchCV(svm_poly_kernal,grid_params,cv=kfold,verbose=5)
```

In [695]:
```
grid.fit(X_train_sample_reduced,y_train_sample)
```

```
Fitting 3 folds for each of 90 candidates, totalling 270 fits
[CV] C=0.01, degree=1, gamma=0.01 ....................................
[CV] ....... C=0.01, degree=1, gamma=0.01, score=-0.089, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.


[CV] C=0.01, degree=1, gamma=0.01 ....................................
[CV] ....... C=0.01, degree=1, gamma=0.01, score=-0.087, total=   0.0s
[CV] C=0.01, degree=1, gamma=0.01 ....................................
[CV] ....... C=0.01, degree=1, gamma=0.01, score=-0.024, total=   0.0s
[CV] C=0.01, degree=1, gamma=0.1 .....................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:
0.0s
```

In [696]:
```
grid.best_params_
```

Out[696]:
```
{'C': 1000, 'degree': 1, 'gamma': 100}
```

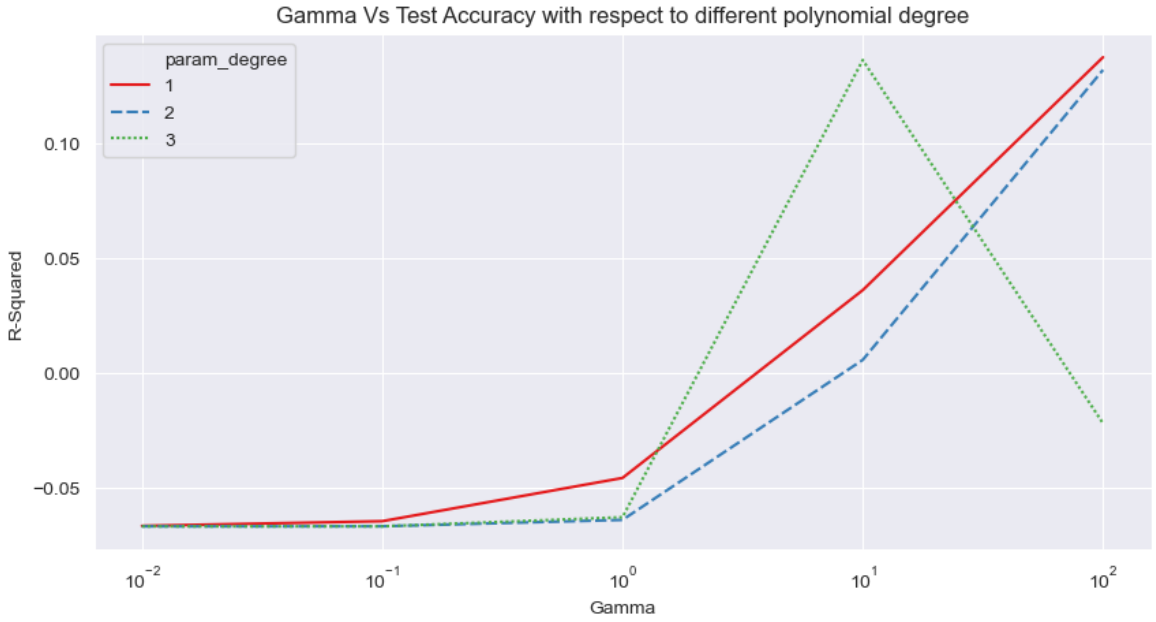
In [697]:
```
grid.best_score_
```

Out[697]:
```
0.5428007132034981
```

In [698]:
```
results_Polynomialkernel_svm_pca = pd.DataFrame(grid.cv_results_)
```

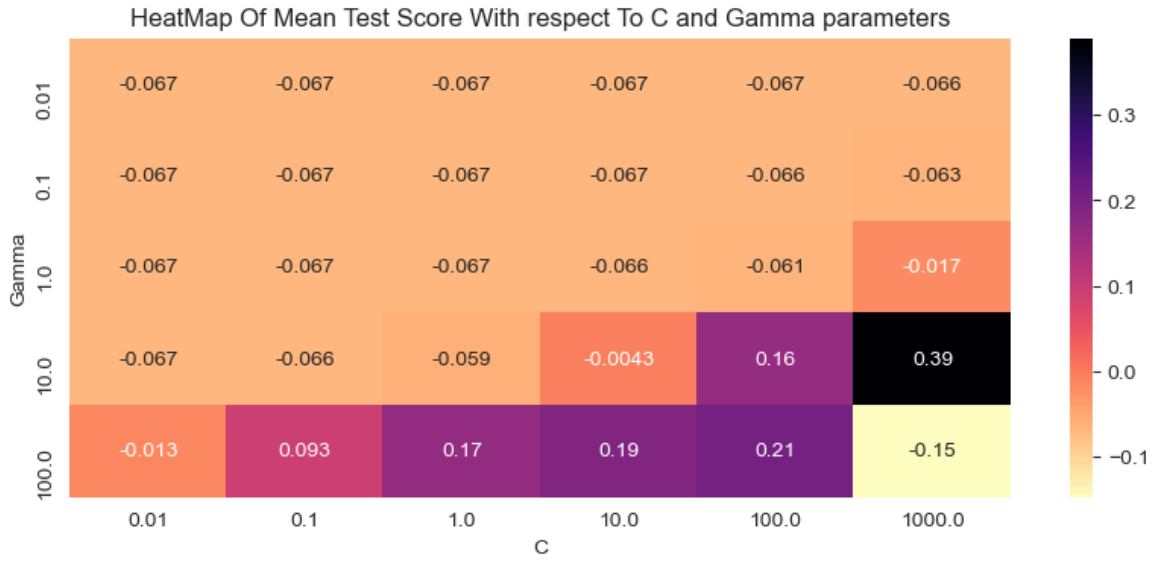
In [699]:
```
fig = plt.figure(figsize=(10,5),dpi=100)
sns.lineplot(results_Polynomialkernel_svm_pca['param_gamma'],results_Polynomi
             style=results_Polynomialkernel_svm_pca['param_degree'],
             hue=results_Polynomialkernel_svm_pca['param_degree'],ci=None,pal

plt.xscale('log')
plt.xlabel('Gamma')
plt.ylabel('R-Squared')
plt.title('Gamma Vs Test Accuracy with respect to different polynomial degree
```

Out[699]:
```
Text(0.5, 1.0, 'Gamma Vs Test Accuracy with respect to different polynomial
degree')
```



In [700]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
sns.heatmap(results_Polynomialkernel_svm_pca.pivot_table(index='param_gamma',
plt.ylabel('Gamma')
plt.xlabel('C')
plt.title('HeatMap Of Mean Test Score With respect To C and Gamma parameters'
```

Out[700]:
```
Text(0.5, 1.0, 'HeatMap Of Mean Test Score With respect To C and Gamma para
meters')
```



In [701]:
```
svm_poly_kernal = SVR(kernel='poly',C=1000,degree=3,gamma=10)
```

In [702]:
```
svm_poly_kernal.fit(X_train_sample_reduced,y_train_sample)
```

Out[702]:
```
SVR(C=1000, gamma=10, kernel='poly')
```

In [703]:
```
kfold = KFold(n_splits=10,random_state=0)
```

```
C:\Users\behab\Anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

In [704]:
```
svm_poly_kernal_cross_val_score = cross_val_score(svm_poly_kernal,X_train_sam
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s

[CV]  ................................................................
[CV] ................................. , score=0.575, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.444, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.457, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.517, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.394, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.541, total=   0.1s
[CV]  ................................................................


[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.2s remaining:
0.0s

[CV] ................................. , score=0.393, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.551, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.583, total=   0.1s
[CV]  ................................................................
[CV] ................................. , score=0.507, total=   0.1s

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.5s finished
```

In [705]:
```
mean_cross_val_svm_poly_kernal_pca = svm_poly_kernal_cross_val_score.mean()
```

In [706]:
```
print('The Final Accuracy Score That was Achived With Our Best SVM Model With
```

```
The Final Accuracy Score That was Achived With Our Best SVM Model With Poly
nomial Kernel with the value of C being 1000 and that of gamma being 10 and
degree of 3 is : 0.4962
```

Comparing The Results OF the Models With PCA and Models Without PCA

In [707]:
```
evaluation_PCA = {'Models':['KNN Regressor PCA','Linear Regression PCA','Ridg
                            'Decision Tree Regressor PCA','Random Forest Regressor
                            'SVM Linear Kernel Regressor PCA','SVM RBF Kernel Regr
                  'Test Scores':[knn_mean_score_cv_pca,lm_mean_score_cv_pca,ridge_
                                 rf_mean_score_cv_pca,polynomial_reg_cv_pca,mean_cr
                                 mean_cross_val_svm_rbf_kernel_pca,mean_cross_val_s
```

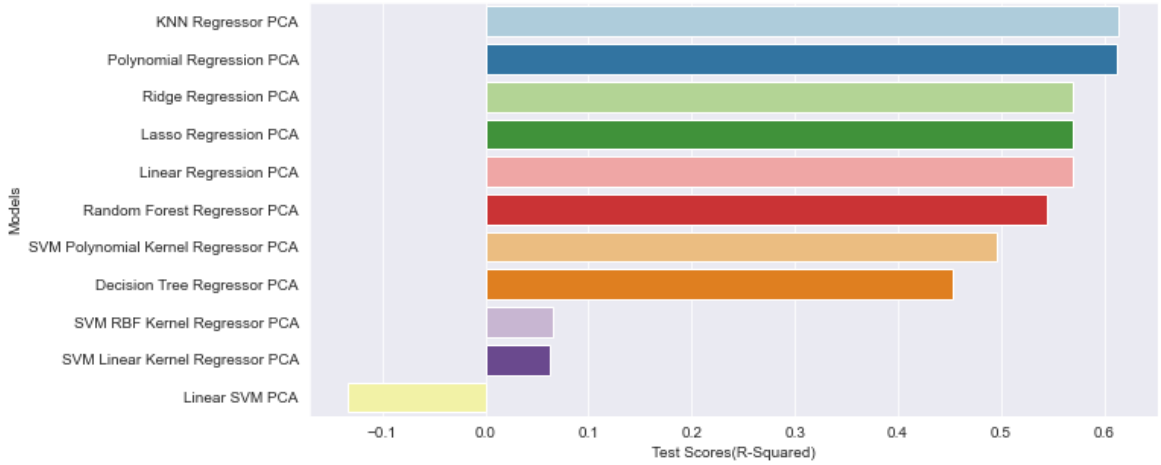
In [708]:
```
eva_pca = pd.DataFrame(evaluation_PCA)
```

In [709]:
```
eva_pca = eva_pca.sort_values(by='Test Scores',ascending = False)
```

Results For Models With PCA

In [710]:
```
fig = plt.figure(figsize=(10,5))
sns.barplot(x=eva_pca['Test Scores'],y=eva_pca['Models'],orient='h',palette='
plt.ylabel('Models')
plt.xlabel('Test Scores(R-Squared)')
```

Out[710]: Text(0.5, 0, 'Test Scores(R-Squared)')



In [747]:
```
evaluation = {'Models':['KNN Regressor','Linear Regression','Ridge Regression
                        'Decision Tree Regressor','Random Forest Regressor','P
                        'SVM Linear Kernel Regressor','SVM RBF Kernel Regressc
                        'Bagging KNN','Bagging Linear Model','Pasting Ridge','
                        'AdaBoost Linear Regression','Gradient Descent'],
              'Test Scores':[knn_mean_score_cv,lm_mean_score_cv,ridge_mean_scc
                             rf_mean_score_cv,polynomial_reg_cv,mean_cross_val_
                             mean_cross_val_svm_rbf_kernel,mean_cross_val_svm_p
                             pasting_ridge_score,pasting_tree_score,ada_lasso_s
```

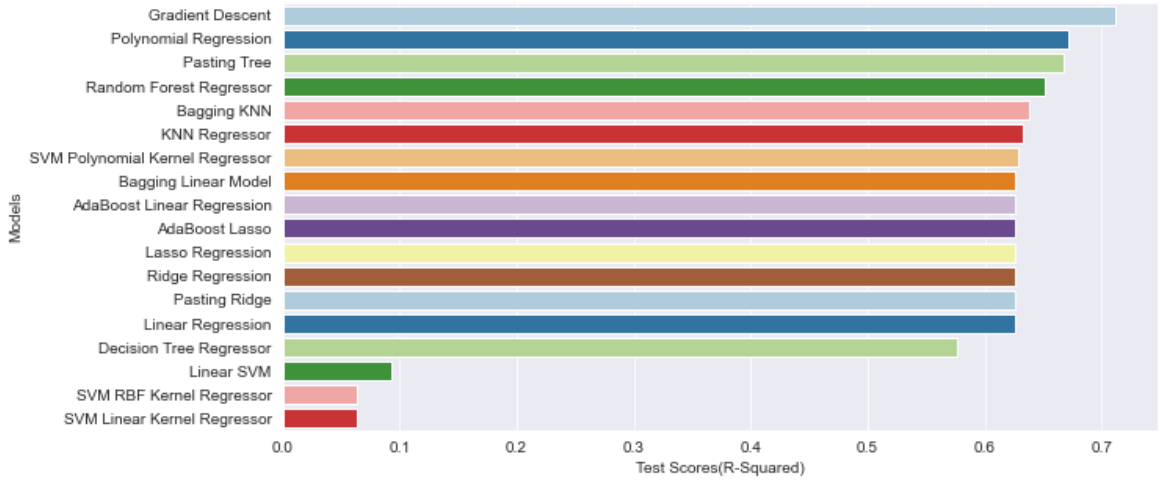
In [748]:
```
eva = pd.DataFrame(evaluation)
```

In [749]:
```
eva = eva.sort_values(by='Test Scores',ascending = False)
```

Results For Models Without PCA

In [750]:
```
fig = plt.figure(figsize=(10,5))
sns.barplot(x=eva['Test Scores'],y=eva['Models'],orient='h',palette='Paired')
plt.ylabel('Models')
plt.xlabel('Test Scores(R-Squared)')
```

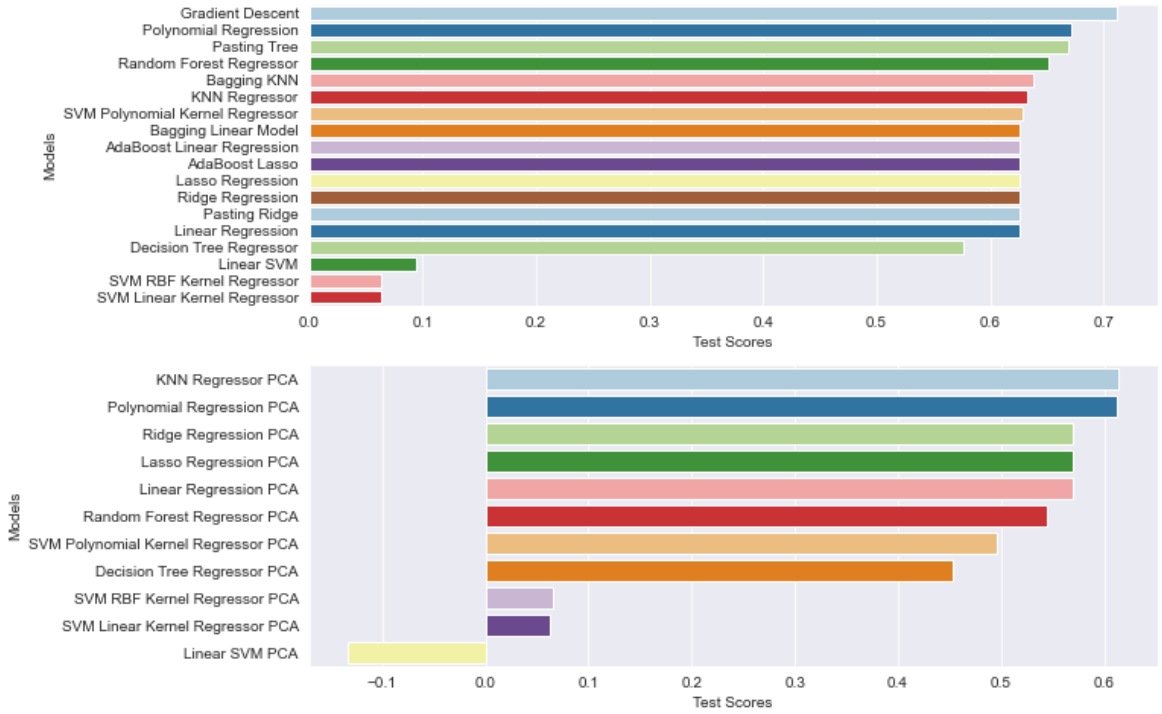
Out[750]: Text(0.5, 0, 'Test Scores(R-Squared)')



In [751]:
```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 10
fig_size[1] = 8
plt.rcParams["figure.figsize"] = fig_size
```

In [752]:
```
fig, ax =plt.subplots(2,1)
plt.figure(figsize=(10,5))

sns.barplot(x=eva['Test Scores'],y=eva['Models'],orient='h',palette='Paired',

sns.barplot(x=eva_pca['Test Scores'],y=eva_pca['Models'],orient='h',palette='
```

Out[752]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1803eff4048>
```

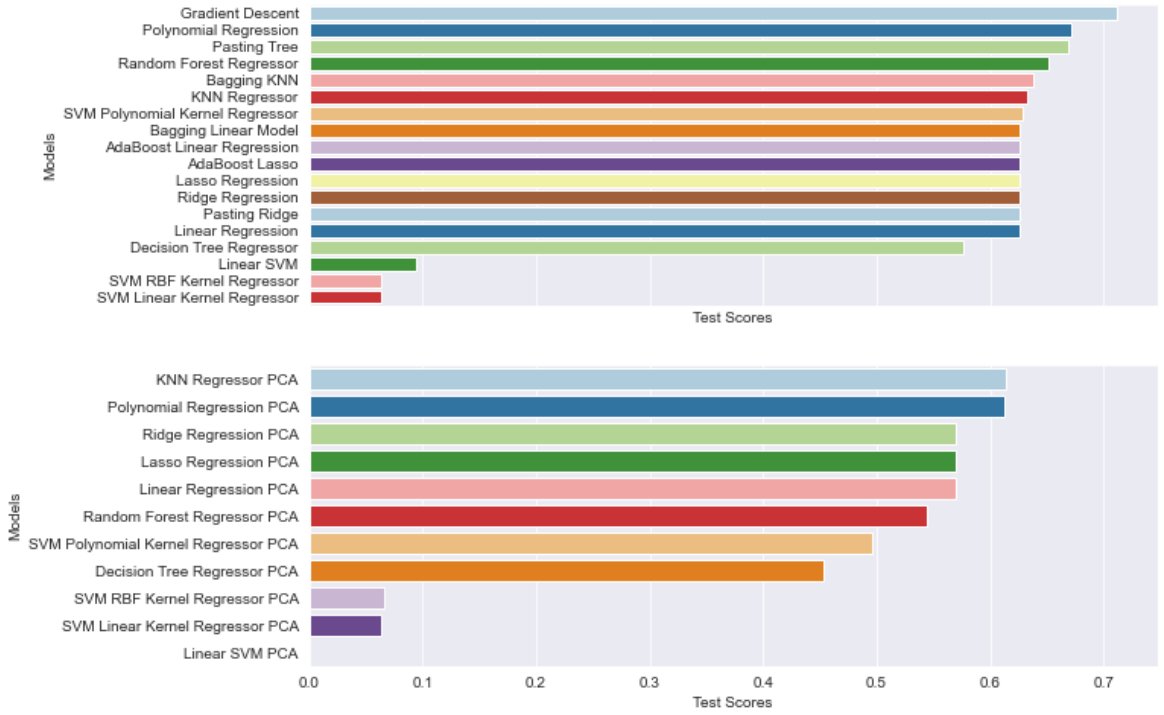



```
<Figure size 720x360 with 0 Axes>
```

As the Linear SVM PCA is performing very poorly we would like to remove it so that we have a common axis to compare the non pca to pca models

In [755]:
```
fig, ax =plt.subplots(2,1,sharex=True)
plt.figure(figsize=(10,10),dpi=100)
sns.barplot(x=eva['Test Scores'],y=eva['Models'],orient='h',palette='Paired',
sns.barplot(x=eva_pca['Test Scores'][:-1],y=eva_pca['Models'],orient='h',pale
```

Out[755]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x180408cf4a8>
```




```
<Figure size 1000x1000 with 0 Axes>
```

As we can see that after implementing the PCA algorithm we could reduce the attributes form 15 to 7 that is more than 50% reduction in the number of variables while keeping upto 90% of the variance intact. The PCA algorithm works very well we can see that even after reducing the number of attributes the models with PCA algorithm do pretty well, the accuracy they provide are less when compared to the models that were run on the full data. But the closeness in accuracy and the reduction in computation resource and the time to train the data makes PCA a viable option.

Deep Learning Model

In [715]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
```

```
C:\Users\behab\Anaconda3\lib\site-packages\tensorflow\python\framework\dt
ypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorflow\python\framework\dt
ypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorflow\python\framework\dt
ypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorflow\python\framework\dt
ypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorflow\python\framework\dt
ypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorflow\python\framework\dt
ypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_
stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a syno
nym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_
stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a syno
nym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_
stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a syno
nym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_
stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a syno
nym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_
stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a syno
```

```
nym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\behab\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_
stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a syno
nym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [716]:
```
X_train.shape
```

Out[716]:
```
(16209, 15)
```

In [717]:
```
def create_model():
    model = Sequential()

    model.add(Dense(15,activation='relu'))


    model.add(Dense(7,activation='relu'))


    model.add(Dense(3,activation='relu'))


    model.add(Dense(1))

    model.compile(optimizer = 'adam',loss='mse')
    return model
```

In [718]:
```
seed = 10
np.random.seed(10)
```

In [719]:
```
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
```

In [720]:
```
kfold = KFold(n_splits=5,random_state=0)
```

```
C:\Users\behab\Anaconda3\lib\site-packages\sklearn\model_selection\_split.p
y:297: FutureWarning: Setting a random_state has no effect since shuffle is
False. This will raise an error in 0.24. You should leave random_state to i
ts default (None), or set shuffle=True.
  FutureWarning
```

In [721]:
```
model = KerasRegressor(build_fn= create_model,verbose=0)
param_grid = {'batch_size':[32,64,128,256],'epochs':[100,250,500]}
grid_search = GridSearchCV(estimator=model,param_grid=param_grid,cv=kfold,ver
```

In [722]:
```
grid_search.fit(X_train.values,y_train.values)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

Fitting 5 folds for each of 12 candidates, totalling 60 fits

[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  8.1min

WARNING:tensorflow:From C:\Users\behab\Anaconda3\lib\site-packages\tensorfl
ow\python\ops\init_ops.py:1251: calling VarianceScaling.__init__ (from tens
orflow.python.ops.init_ops) with dtype is deprecated and will be removed in
a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to
the constructor


[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed: 16.4min finished
```

Out[722]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=0, shuffle=False),
             estimator=<tensorflow.python.keras.wrappers.scikit_learn.Keras
Regressor object at 0x0000017FAEED9CC0>,
             n_jobs=-1,
             param_grid={'batch_size': [32, 64, 128, 256],
                         'epochs': [100, 250, 500]},
             verbose=3)
```

In [723]:
```
grid_search.best_params_
```

Out[723]:
```
{'batch_size': 128, 'epochs': 500}
```

In [724]:
```
grid_search.best_score_
```

Out[724]:
```
-49147472556.73611
```

In [725]:
```
dnn_results = pd.DataFrame(grid_search.cv_results_)
```

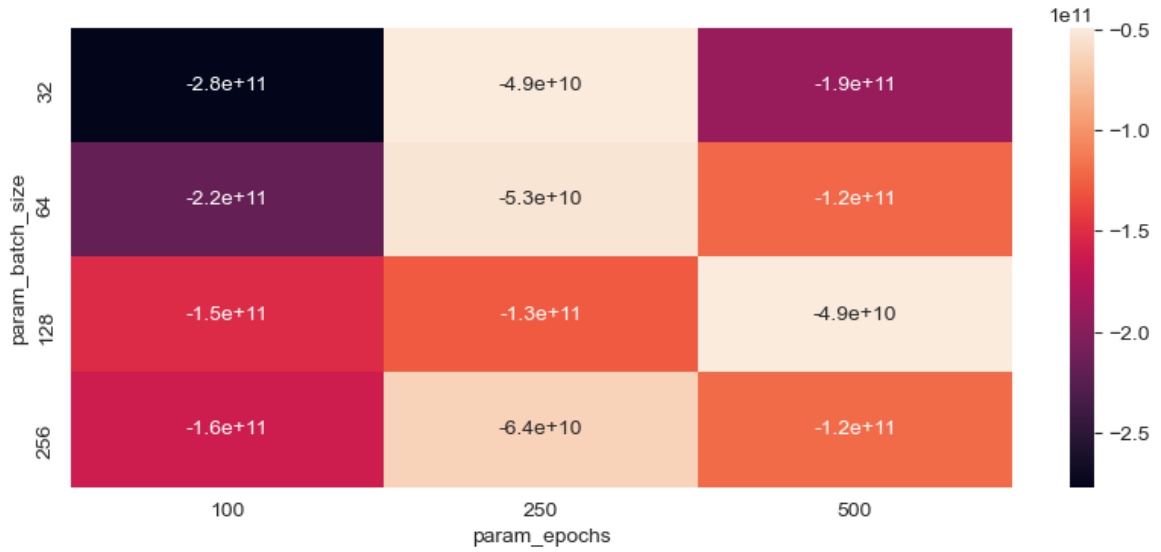
In [726]:
```
dnn_results.head()
```

Out[726]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_batch_size	param_epc
0	85.651386	0.309602	0.203655	0.015118	32	
1	220.219871	2.190051	0.349664	0.050495	32	
2	448.007845	2.691757	0.339092	0.071187	32	
3	47.543368	0.892554	0.375595	0.086887	64	
4	123.486020	6.403342	0.523399	0.064531	64	

In [727]:
```
fig = plt.figure(figsize=(10,4),dpi=100)
sns.heatmap(dnn_results.pivot_table(index='param_batch_size',columns='param_e
```

Out[727]: <matplotlib.axes._subplots.AxesSubplot at 0x17faeee4cf8>



In [728]:

```
model = Sequential()

model.add(Dense(15,activation='relu'))


model.add(Dense(7,activation='relu'))


model.add(Dense(3,activation='relu'))


model.add(Dense(1))

model.compile(optimizer = 'adam',loss='mse')
```

In [734]:
```
model.fit(X_train.values,y_train.values,epochs=500,batch_size=128,validation_
```

```
Train on 16209 samples, validate on 5404 samples
Epoch 1/500
16209/16209 [==============================] - 0s 13us/sample - loss: 435
08584886.6857 - val_loss: 42864714061.5011
Epoch 2/500
16209/16209 [==============================] - 0s 16us/sample - loss: 435
05767532.5974 - val_loss: 42870255224.8941
Epoch 3/500
16209/16209 [==============================] - 0s 14us/sample - loss: 435
04843581.2322 - val_loss: 42858927652.0030
Epoch 4/500
16209/16209 [==============================] - 0s 15us/sample - loss: 435
01305909.0668 - val_loss: 42860992662.0755
Epoch 5/500
16209/16209 [==============================] - 0s 16us/sample - loss: 435
00969580.6922 - val_loss: 42857244303.6329
Epoch 6/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
97713939.6947 - val_loss: 42856902116.3346
Epoch 7/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
99459333.5120 - val_loss: 42869732675.6477
Epoch 8/500
16209/16209 [==============================] - 0s 15us/sample - loss: 434
92793697.3365 - val_loss: 42859496497.2672
Epoch 9/500
16209/16209 [==============================] - 0s 15us/sample - loss: 434
89960814.6348 - val_loss: 42845731539.0910
Epoch 10/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
86964505.7279 - val_loss: 42846928714.0903
Epoch 11/500
16209/16209 [==============================] - 0s 19us/sample - loss: 434
82539138.4559 - val_loss: 42842815426.6055
Epoch 12/500
16209/16209 [==============================] - 0s 15us/sample - loss: 434
86939103.5282 - val_loss: 42855756808.3375
Epoch 13/500
16209/16209 [==============================] - 0s 15us/sample - loss: 434
80841084.4069 - val_loss: 42846931460.9267
Epoch 14/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
80163888.2971 - val_loss: 42836365562.1258
Epoch 15/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
76146285.5451 - val_loss: 42843078243.6714
Epoch 16/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
70234801.0789 - val_loss: 42829003587.2687
Epoch 17/500
16209/16209 [==============================] - 0s 15us/sample - loss: 434
80728421.5377 - val_loss: 42830813080.9178
Epoch 18/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
```

```
63681563.7021 - val_loss: 42837570595.6240
Epoch 19/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
66047371.5057 - val_loss: 42826020231.1059
Epoch 20/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
69535048.6668 - val_loss: 42813818484.3464
Epoch 21/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
57947117.2055 - val_loss: 42829852135.3664
Epoch 22/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
51002231.8268 - val_loss: 42843601925.3057
Epoch 23/500
16209/16209 [==============================] - ETA: 0s - loss: 4362099756
3.733 - 0s 17us/sample - loss: 43447638352.7216 - val_loss: 42824955312.7
935
Epoch 24/500
16209/16209 [==============================] - 0s 19us/sample - loss: 434
44438873.3449 - val_loss: 42822575812.6899
Epoch 25/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
38017281.0898 - val_loss: 42833799877.4478
Epoch 26/500
16209/16209 [==============================] - 0s 18us/sample - loss: 434
38522714.3241 - val_loss: 42815852450.0133
Epoch 27/500
16209/16209 [==============================] - 0s 20us/sample - loss: 434
31786010.9440 - val_loss: 42809974849.1843
Epoch 28/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
27707545.2304 - val_loss: 42798551770.6706
Epoch 29/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
25557114.9065 - val_loss: 42807738017.8238
Epoch 30/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
24205970.1233 - val_loss: 42808382628.4767
Epoch 31/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
19672363.4800 - val_loss: 42801205513.2850
Epoch 32/500
16209/16209 [==============================] - 0s 18us/sample - loss: 434
13839271.0815 - val_loss: 42793271217.1725
Epoch 33/500
16209/16209 [==============================] - 0s 16us/sample - loss: 434
09192228.8151 - val_loss: 42793493753.3679
Epoch 34/500
16209/16209 [==============================] - 0s 17us/sample - loss: 434
08310063.7759 - val_loss: 42808939391.9053
Epoch 35/500
16209/16209 [==============================] - 0s 18us/sample - loss: 433
99365075.7461 - val_loss: 42797529909.6255
Epoch 36/500
16209/16209 [==============================] - 0s 18us/sample - loss: 433
95520634.4327 - val_loss: 42781498876.5892
Epoch 37/500
```

```
16209/16209 [==============================] - 0s 17us/sample - loss: 433
91503067.0585 - val_loss: 42788490589.4182
Epoch 38/500
16209/16209 [==============================] - 0s 17us/sample - loss: 433
91297911.3845 - val_loss: 42769577216.9474
Epoch 39/500
16209/16209 [==============================] - 0s 20us/sample - loss: 433
85627922.1154 - val_loss: 42794761222.8216
Epoch 40/500
16209/16209 [==============================] - 0s 20us/sample - loss: 433
81278051.9899 - val_loss: 42788120010.5640
Epoch 41/500
16209/16209 [==============================] - 0s 18us/sample - loss: 433
79282173.9626 - val_loss: 42783957063.2480
Epoch 42/500
16209/16209 [==============================] - 0s 19us/sample - loss: 433
71423273.9796 - val_loss: 42770807910.3242
Epoch 43/500
16209/16209 [==============================] - 0s 19us/sample - loss: 433
72100286.6299 - val_loss: 42767556269.9511
Epoch 44/500
16209/16209 [==============================] - 0s 18us/sample - loss: 433
67838108.6577 - val_loss: 42762994206.6973
Epoch 45/500
16209/16209 [==============================] - 0s 18us/sample - loss: 433
68351140.4282 - val_loss: 42779095902.5551
Epoch 46/500
16209/16209 [==============================] - 0s 20us/sample - loss: 433
59891697.8331 - val_loss: 42767109381.4952
Epoch 47/500
16209/16209 [==============================] - 0s 22us/sample - loss: 433
55057743.6318 - val_loss: 42763723973.0688
Epoch 48/500
16209/16209 [==============================] - 0s 18us/sample - loss: 433
50636650.2600 - val_loss: 42753073876.6070
Epoch 49/500
16209/16209 [==============================] - 0s 18us/sample - loss: 433
41681157.2751 - val_loss: 42754789072.8172
Epoch 50/500
16209/16209 [==============================] - 0s 19us/sample - loss: 433
46646483.9356 - val_loss: 42742460619.1325
Epoch 51/500
16209/16209 [==============================] - 0s 19us/sample - loss: 433
38034519.7340 - val_loss: 42756093747.3516
Epoch 52/500
16209/16209 [==============================] - 0s 18us/sample - loss: 433
35206004.8102 - val_loss: 42741302727.5322
Epoch 53/500
16209/16209 [==============================] - 0s 21us/sample - loss: 433
34655369.0103 - val_loss: 42752382310.5137
Epoch 54/500
16209/16209 [==============================] - 0s 22us/sample - loss: 433
31453286.3273 - val_loss: 42718770087.3190
Epoch 55/500
16209/16209 [==============================] - 1s 70us/sample - loss: 433
21837667.6898 - val_loss: 42720945439.2657
Epoch 56/500
```

```
16209/16209 [==============================] - 0s 19us/sample - loss: 433
20947669.7361 - val_loss: 42742783739.2628
Epoch 57/500
16209/16209 [==============================] - 0s 29us/sample - loss: 433
11743989.5762 - val_loss: 42724252422.6321
Epoch 58/500
16209/16209 [==============================] - 0s 27us/sample - loss: 433
10821510.5623 - val_loss: 42714361763.5292
Epoch 59/500
16209/16209 [==============================] - 0s 19us/sample - loss: 433
04333861.9996 - val_loss: 42707175680.1895
Epoch 60/500
16209/16209 [==============================] - 0s 20us/sample - loss: 433
02646210.5625 - val_loss: 42703620982.8098
Epoch 61/500
16209/16209 [==============================] - 0s 21us/sample - loss: 433
01113413.2711 - val_loss: 42703328108.1984
Epoch 62/500
16209/16209 [==============================] - 0s 21us/sample - loss: 432
97402199.7340 - val_loss: 42676060702.6973
Epoch 63/500
16209/16209 [==============================] - 0s 21us/sample - loss: 432
92455409.8173 - val_loss: 42676754912.5448

Epoch 64/500
16209/16209 [==============================] - 0s 20us/sample - loss: 432
87095535.4324 - val_loss: 42682762292.2990
Epoch 65/500
16209/16209 [==============================] - 0s 23us/sample - loss: 432
84494202.7012 - val_loss: 42680742636.8616
Epoch 66/500
16209/16209 [==============================] - 0s 20us/sample - loss: 432
79689310.2884 - val_loss: 42681946078.6499
Epoch 67/500
16209/16209 [==============================] - 0s 19us/sample - loss: 432
78095013.9917 - val_loss: 42674570155.1088
Epoch 68/500
16209/16209 [==============================] - 0s 22us/sample - loss: 432
80509395.6197 - val_loss: 42670073860.5477
Epoch 69/500
16209/16209 [==============================] - 0s 20us/sample - loss: 432
68014367.0662 - val_loss: 42661432287.4078
Epoch 70/500
16209/16209 [==============================] - 0s 26us/sample - loss: 432
68580875.7821 - val_loss: 42649602252.6484
Epoch 71/500
16209/16209 [==============================] - 0s 29us/sample - loss: 432
65020596.9010 - val_loss: 42653151623.1059
Epoch 72/500
16209/16209 [==============================] - 0s 19us/sample - loss: 432
61400556.7001 - val_loss: 42640278819.8135
Epoch 73/500
16209/16209 [==============================] - 0s 21us/sample - loss: 432
60482537.1939 - val_loss: 42642092703.5500
Epoch 74/500
16209/16209 [==============================] - 0s 23us/sample - loss: 432
51302940.5234 - val_loss: 42640749486.8986
```

```
Epoch 75/500
16209/16209 [==============================] - 0s 21us/sample - loss: 432
49097508.2149 - val_loss: 42642456782.1643
Epoch 76/500
16209/16209 [==============================] - 0s 18us/sample - loss: 432
47879056.7492 - val_loss: 42634321280.2842
Epoch 77/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
45261520.1925 - val_loss: 42634001011.5885
Epoch 78/500
16209/16209 [==============================] - 0s 18us/sample - loss: 432
42103160.0163 - val_loss: 42625775080.8823
Epoch 79/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
33027097.7121 - val_loss: 42619163328.1421
Epoch 80/500
16209/16209 [==============================] - 0s 18us/sample - loss: 432
37885386.0803 - val_loss: 42618503689.4745
Epoch 81/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
31470003.6533 - val_loss: 42619808879.4197
Epoch 82/500
16209/16209 [==============================] - 0s 19us/sample - loss: 432
27628489.8592 - val_loss: 42605326211.6950
Epoch 83/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
26471174.5228 - val_loss: 42606656004.9267
Epoch 84/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
21297680.6466 - val_loss: 42596948847.9882
Epoch 85/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
17709782.6363 - val_loss: 42605492657.5514
Epoch 86/500
16209/16209 [==============================] - 0s 19us/sample - loss: 432
15730485.9671 - val_loss: 42588252561.7172
Epoch 87/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
08533987.4450 - val_loss: 42598457719.9467
Epoch 88/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
08346087.3302 - val_loss: 42599384152.6810
Epoch 89/500
16209/16209 [==============================] - 0s 17us/sample - loss: 432
03937849.2679 - val_loss: 42585981251.6477
Epoch 90/500
16209/16209 [==============================] - 0s 19us/sample - loss: 432
00508772.5269 - val_loss: 42574167146.8719
Epoch 91/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
96224452.5525 - val_loss: 42572857113.5811
Epoch 92/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
92966870.6995 - val_loss: 42566591682.7950
Epoch 93/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
94981084.9538 - val_loss: 42565204672.9001
```

```
Epoch 94/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
85076928.6515 - val_loss: 42573191004.2813
Epoch 95/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
82695550.5391 - val_loss: 42575349673.5929
Epoch 96/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
76551127.7735 - val_loss: 42558936813.6195
Epoch 97/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
79744764.4938 - val_loss: 42560728367.9408
Epoch 98/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
74196240.8677 - val_loss: 42546469717.4597
Epoch 99/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
75563883.6182 - val_loss: 42539949252.3109
Epoch 100/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
69646384.1550 - val_loss: 42535676190.5078
Epoch 101/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
66185188.0925 - val_loss: 42537380011.2983
Epoch 102/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
66019881.6796 - val_loss: 42538236583.8875
Epoch 103/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
59001940.2278 - val_loss: 42521211275.6536
Epoch 104/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
54496530.2575 - val_loss: 42528840378.8364
Epoch 105/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
59722313.6460 - val_loss: 42519176083.6121
Epoch 106/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
52965602.9396 - val_loss: 42517931308.9090
Epoch 107/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
50872900.9711 - val_loss: 42515793800.2428
Epoch 108/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
47813948.5846 - val_loss: 42522794278.0873
Epoch 109/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
47184536.9935 - val_loss: 42498990110.3183
Epoch 110/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
43049880.8514 - val_loss: 42502969977.6521
Epoch 111/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
42095870.1679 - val_loss: 42510236907.7246
Epoch 112/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
32612671.9329 - val_loss: 42507097337.3679
```

```
Epoch 113/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
26755943.7014 - val_loss: 42497727741.9156
Epoch 114/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
26017868.3309 - val_loss: 42507400562.6410
Epoch 115/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
29504706.0255 - val_loss: 42497083493.5662
Epoch 116/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
20124445.2815 - val_loss: 42502047177.0481
Epoch 117/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
20115821.9241 - val_loss: 42472005647.1591
Epoch 118/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
12565190.4794 - val_loss: 42481873817.6758
Epoch 119/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
11027420.5747 - val_loss: 42481429964.0799
Epoch 120/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
07177682.6879 - val_loss: 42493248036.0030
Epoch 121/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
05696649.5473 - val_loss: 42473144921.8179
Epoch 122/500
16209/16209 [==============================] - 0s 20us/sample - loss: 431
04177432.9382 - val_loss: 42475019087.3960
Epoch 123/500
16209/16209 [==============================] - 0s 17us/sample - loss: 431
00823974.0075 - val_loss: 42470019634.4041
Epoch 124/500
16209/16209 [==============================] - 0s 18us/sample - loss: 431
01603055.2429 - val_loss: 42490022927.9171
Epoch 125/500
16209/16209 [==============================] - 0s 19us/sample - loss: 430
97776969.2354 - val_loss: 42461799890.1436
Epoch 126/500
16209/16209 [==============================] - 0s 23us/sample - loss: 430
89478810.8414 - val_loss: 42453212790.6203
Epoch 127/500
16209/16209 [==============================] - 0s 19us/sample - loss: 430
91811551.1965 - val_loss: 42454589644.6484
Epoch 128/500
16209/16209 [==============================] - 0s 20us/sample - loss: 430
85513924.0313 - val_loss: 42454697973.3886
Epoch 129/500
16209/16209 [==============================] - 0s 20us/sample - loss: 430
88159962.9006 - val_loss: 42447617179.3812
Epoch 130/500
16209/16209 [==============================] - 0s 20us/sample - loss: 430
82775339.2905 - val_loss: 42450630925.8327
Epoch 131/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
77513672.2483 - val_loss: 42439122343.6980
```

```
Epoch 132/500
16209/16209 [==============================] - 0s 19us/sample - loss: 430
78579034.1662 - val_loss: 42442192097.1133
Epoch 133/500
16209/16209 [==============================] - 0s 18us/sample - loss: 430
74955251.2387 - val_loss: 42429969426.9489
Epoch 134/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
71265684.6661 - val_loss: 42439361379.1029
Epoch 135/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
69946592.2389 - val_loss: 42430459133.9156
Epoch 136/500
16209/16209 [==============================] - 0s 18us/sample - loss: 430
67613566.8392 - val_loss: 42419161463.9467
Epoch 137/500
16209/16209 [==============================] - 0s 18us/sample - loss: 430
61548083.9928 - val_loss: 42416696103.9822
Epoch 138/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
66188445.9054 - val_loss: 42414638377.8771
Epoch 139/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
63914521.5226 - val_loss: 42415058828.7905
Epoch 140/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
54673368.8632 - val_loss: 42410963377.5514
Epoch 141/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
55046401.9110 - val_loss: 42396680654.3538
Epoch 142/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
53978984.2858 - val_loss: 42392214166.4545
Epoch 143/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
46477100.3961 - val_loss: 42406489974.0518
Epoch 144/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
49099770.4090 - val_loss: 42393133246.2472
Epoch 145/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
46015183.2133 - val_loss: 42378173277.0392
Epoch 146/500
16209/16209 [==============================] - 0s 18us/sample - loss: 430
42623594.3863 - val_loss: 42399350636.9563
Epoch 147/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
41754666.2323 - val_loss: 42379546786.2028
Epoch 148/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
34012520.0647 - val_loss: 42381208785.9541
Epoch 149/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
31527974.8841 - val_loss: 42371602166.7150
Epoch 150/500
16209/16209 [==============================] - 0s 18us/sample - loss: 430
32609139.7204 - val_loss: 42372117280.4027
```

```
Epoch 151/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
30178740.1271 - val_loss: 42382699561.6876
Epoch 152/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
26076864.8726 - val_loss: 42371420408.6099
Epoch 153/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
32627684.0452 - val_loss: 42369765921.7291
Epoch 154/500
16209/16209 [==============================] - 0s 18us/sample - loss: 430
18938056.5483 - val_loss: 42358951732.8675
Epoch 155/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
22879234.2427 - val_loss: 42370073097.4745
Epoch 156/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
17220743.1624 - val_loss: 42350202366.1051
Epoch 157/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
15095140.4005 - val_loss: 42363245050.3153
Epoch 158/500
16209/16209 [==============================] - 0s 18us/sample - loss: 430
07343400.7635 - val_loss: 42352434573.9275
Epoch 159/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
22496898.0769 - val_loss: 42367296879.6092
Epoch 160/500
16209/16209 [==============================] - 0s 16us/sample - loss: 430
14540740.0155 - val_loss: 42350028298.2324
Epoch 161/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
06751118.1275 - val_loss: 42342449361.1962
Epoch 162/500
16209/16209 [==============================] - 0s 19us/sample - loss: 430
05932395.6024 - val_loss: 42345195997.5130
Epoch 163/500
16209/16209 [==============================] - 0s 17us/sample - loss: 430
02214401.3898 - val_loss: 42340409245.4656
Epoch 164/500
16209/16209 [==============================] - 0s 17us/sample - loss: 429
99220527.3653 - val_loss: 42347150699.0614
Epoch 165/500
16209/16209 [==============================] - 0s 17us/sample - loss: 429
98608423.3895 - val_loss: 42330455550.1051
Epoch 166/500
16209/16209 [==============================] - 0s 19us/sample - loss: 429
94585933.5628 - val_loss: 42332742107.9970
Epoch 167/500
16209/16209 [==============================] - 0s 17us/sample - loss: 429
98042105.8405 - val_loss: 42332913345.6580
Epoch 168/500
16209/16209 [==============================] - 0s 18us/sample - loss: 429
89758714.0458 - val_loss: 42319059021.3116
Epoch 169/500
16209/16209 [==============================] - 0s 20us/sample - loss: 429
85937122.2921 - val_loss: 42316580598.7150
```

```
Epoch 170/500
16209/16209 [==============================] - 0s 21us/sample - loss: 429
87277172.8260 - val_loss: 42316498384.6277
Epoch 171/500
16209/16209 [==============================] - 0s 18us/sample - loss: 429
80524133.4587 - val_loss: 42318919338.9193
Epoch 172/500
16209/16209 [==============================] - 0s 17us/sample - loss: 429
78495265.4352 - val_loss: 42311352216.1599
Epoch 173/500
16209/16209 [==============================] - 0s 18us/sample - loss: 429
74269096.0765 - val_loss: 42308657661.3472
Epoch 174/500
16209/16209 [==============================] - 0s 19us/sample - loss: 429
74776088.0854 - val_loss: 42301815848.1717
Epoch 175/500
16209/16209 [==============================] - 0s 21us/sample - loss: 429
72529671.0124 - val_loss: 42307385562.2916
Epoch 176/500
16209/16209 [==============================] - 0s 19us/sample - loss: 429
77885961.8711 - val_loss: 42299903869.6314
Epoch 177/500
16209/16209 [==============================] - 0s 19us/sample - loss: 429
64613396.8319 - val_loss: 42313459367.8875
Epoch 178/500
16209/16209 [==============================] - 0s 19us/sample - loss: 429
69495909.6956 - val_loss: 42298630747.3338
Epoch 179/500
16209/16209 [==============================] - 0s 19us/sample - loss: 429
63647896.6777 - val_loss: 42306171944.1717
Epoch 180/500
16209/16209 [==============================] - 0s 21us/sample - loss: 429
60533412.9020 - val_loss: 42289667271.3427
Epoch 181/500
16209/16209 [==============================] - 1s 56us/sample - loss: 429
57890307.4904 - val_loss: 42304336009.9482
Epoch 182/500
16209/16209 [==============================] - 0s 22us/sample - loss: 429
54882997.4538 - val_loss: 42294105528.3731
Epoch 183/500
16209/16209 [==============================] - 0s 21us/sample - loss: 429
56030781.8639 - val_loss: 42294660084.6306
Epoch 184/500
16209/16209 [==============================] - 0s 19us/sample - loss: 429
53854394.5393 - val_loss: 42313434312.8586
Epoch 185/500
16209/16209 [==============================] - 0s 18us/sample - loss: 429
46770221.4069 - val_loss: 42290406467.4582
Epoch 186/500
16209/16209 [==============================] - 0s 19us/sample - loss: 429
46437545.0399 - val_loss: 42298637995.6773
Epoch 187/500
16209/16209 [==============================] - 0s 17us/sample - loss: 429
43303713.3563 - val_loss: 42271828170.3745
Epoch 188/500
16209/16209 [==============================] - 0s 20us/sample - loss: 429
40352013.5510 - val_loss: 42285201773.3353
```

```
Epoch 189/500
16209/16209 [==============================] - 1s 44us/sample - loss: 429
36829819.7120 - val_loss: 42280333460.5596
Epoch 190/500

16209/16209 [==============================] - 0s 20us/sample - loss: 42932
360817.1144 - val_loss: 42270376452.9267
Epoch 191/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42936
717088.3613 - val_loss: 42264396109.5011
Epoch 192/500
16209/16209 [==============================] - 0s 30us/sample - loss: 42930
947114.3903 - val_loss: 42265837349.7084
Epoch 193/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42930
794780.1602 - val_loss: 42260086104.8705
Epoch 194/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42927
838299.4139 - val_loss: 42267930561.8475
Epoch 195/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42925
430755.0660 - val_loss: 42269417561.4389
Epoch 196/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42937
478936.9067 - val_loss: 42267417946.3864
Epoch 197/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42917
772091.7160 - val_loss: 42256713728.0000
Epoch 198/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42924
152966.0569 - val_loss: 42252437297.8357
Epoch 199/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42913
804472.9758 - val_loss: 42253304500.7728
Epoch 200/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42914
409637.0757 - val_loss: 42258148440.6810
Epoch 201/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42914
652108.1967 - val_loss: 42259922975.8342
Epoch 202/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42911
288194.4717 - val_loss: 42247069597.4656
Epoch 203/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42907
232260.1695 - val_loss: 42242024885.3412
Epoch 204/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42903
099332.4894 - val_loss: 42241688117.4360
Epoch 205/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42904
943378.5260 - val_loss: 42249118946.6292
Epoch 206/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42899
114554.4683 - val_loss: 42238878549.4597
Epoch 207/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42895
```

```
708081.4106 - val_loss: 42236189370.8364
Epoch 208/500
16209/16209 [==============================] - 0s 25us/sample - loss: 42892
546339.8675 - val_loss: 42240756989.9156
Epoch 209/500
16209/16209 [==============================] - 0s 25us/sample - loss: 42891
137474.0571 - val_loss: 42234604397.7143
Epoch 210/500
16209/16209 [==============================] - 0s 25us/sample - loss: 42893
820062.9477 - val_loss: 42217118331.1680
Epoch 211/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42887
827991.9748 - val_loss: 42229713978.3627
Epoch 212/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42882
549215.5598 - val_loss: 42232892979.1621
Epoch 213/500
16209/16209 [==============================] - 0s 27us/sample - loss: 42880
790706.6583 - val_loss: 42227189141.5070
Epoch 214/500
16209/16209 [==============================] - 1s 32us/sample - loss: 42875
934147.5102 - val_loss: 42226563977.0007
Epoch 215/500
16209/16209 [==============================] - 0s 27us/sample - loss: 42876
155086.7711 - val_loss: 42234903029.7676
Epoch 216/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42877
127500.3941 - val_loss: 42228234652.3286
Epoch 217/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42872
899761.7738 - val_loss: 42217535510.7387
Epoch 218/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42873
217200.0049 - val_loss: 42221038485.1281
Epoch 219/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42876
462167.6234 - val_loss: 42208926664.6691
Epoch 220/500
16209/16209 [==============================] - 0s 30us/sample - loss: 42866
368876.8975 - val_loss: 42219646719.8105
Epoch 221/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42868
686921.0300 - val_loss: 42230420263.9822
Epoch 222/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42864
833399.9847 - val_loss: 42220737406.3893
Epoch 223/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42864
945583.4837 - val_loss: 42200812103.6269
Epoch 224/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42859
063318.6166 - val_loss: 42206068769.3501
Epoch 225/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42853
314703.5331 - val_loss: 42210734910.7209
Epoch 226/500
16209/16209 [==============================] - 0s 25us/sample - loss: 42855
```

```
529494.3007 - val_loss: 42206745130.8246
Epoch 227/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42850
699398.8150 - val_loss: 42203405321.8534
Epoch 228/500
16209/16209 [==============================] - 0s 26us/sample - loss: 42844
554346.9549 - val_loss: 42198150173.5603
Epoch 229/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42849
317563.7239 - val_loss: 42199071712.1658
Epoch 230/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42844
094183.1249 - val_loss: 42198676362.5167
Epoch 231/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42843
643124.4232 - val_loss: 42189522745.4152
Epoch 232/500
16209/16209 [==============================] - 0s 27us/sample - loss: 42840
236329.6796 - val_loss: 42199835751.8401
Epoch 233/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42834
019118.0702 - val_loss: 42183001498.0548
Epoch 234/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42833
435277.6063 - val_loss: 42177319951.9171
Epoch 235/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42832
887298.3059 - val_loss: 42171338197.9334
Epoch 236/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42829
253774.8382 - val_loss: 42178531468.2221
Epoch 237/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42827
482329.0054 - val_loss: 42176119610.1732
Epoch 238/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42827
055729.0513 - val_loss: 42164561675.9378
Epoch 239/500
16209/16209 [==============================] - 0s 26us/sample - loss: 42823
158630.1694 - val_loss: 42173197312.7580
Epoch 240/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42821
612295.0282 - val_loss: 42168328786.2383
Epoch 241/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42815
306130.6760 - val_loss: 42171782385.7883
Epoch 242/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42817
055338.7338 - val_loss: 42166044247.5440
Epoch 243/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42817
408168.1081 - val_loss: 42155736411.9023
Epoch 244/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42809
432452.5249 - val_loss: 42174406005.6728
Epoch 245/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42808
```

```
375492.2209 - val_loss: 42160872726.1702
Epoch 246/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42809
497491.8448 - val_loss: 42162540383.3131
Epoch 247/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42800
547510.6067 - val_loss: 42160317155.7661
Epoch 248/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42812
338688.5370 - val_loss: 42149310272.9948
Epoch 249/500
16209/16209 [==============================] - 0s 27us/sample - loss: 42804
783265.6011 - val_loss: 42174062325.1991
Epoch 250/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42801
086381.3674 - val_loss: 42152932780.2457
Epoch 251/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42796
725190.1951 - val_loss: 42142060413.6314
Epoch 252/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42795
304656.7611 - val_loss: 42139889048.5389
Epoch 253/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42797
173029.0046 - val_loss: 42130877042.0725
Epoch 254/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42787
238442.1692 - val_loss: 42139793984.0474
Epoch 255/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42789
204792.5572 - val_loss: 42126245183.0999
Epoch 256/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42782
761633.3168 - val_loss: 42136118196.9622
Epoch 257/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42779
183872.3317 - val_loss: 42137445228.1984
Epoch 258/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42780
201900.5461 - val_loss: 42126371896.8468
Epoch 259/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42776
982734.1393 - val_loss: 42128268184.1599
Epoch 260/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42777
081123.2989 - val_loss: 42124199209.8771
Epoch 261/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42774
004925.4612 - val_loss: 42129099620.6188
Epoch 262/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42767
763488.3455 - val_loss: 42129652927.7631
Epoch 263/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42772
262715.9055 - val_loss: 42113262306.2502
Epoch 264/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42764
```

```
155112.6095 - val_loss: 42108125195.3694
Epoch 265/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42769
203197.9784 - val_loss: 42129017270.0992
Epoch 266/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42761
818894.7355 - val_loss: 42104754576.2013
Epoch 267/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42757
150556.6300 - val_loss: 42111023631.5381
Epoch 268/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42755
567962.1978 - val_loss: 42107041347.8372
Epoch 269/500
16209/16209 [==============================] - 0s 27us/sample - loss: 42756
803460.4933 - val_loss: 42096817969.8357
Epoch 270/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42752
755908.2209 - val_loss: 42093443135.6684
Epoch 271/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42751
863318.8377 - val_loss: 42088202421.9097
Epoch 272/500
16209/16209 [==============================] - 0s 24us/sample - loss: 42747
678817.0365 - val_loss: 42080330498.0844
Epoch 273/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42746
859970.3098 - val_loss: 42090644530.7831
Epoch 274/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42746
154965.7992 - val_loss: 42072258928.3671
Epoch 275/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42741
945032.6115 - val_loss: 42074411994.1021
Epoch 276/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42740
367983.5351 - val_loss: 42064369693.5603
Epoch 277/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42739
264104.7122 - val_loss: 42060077726.0340
Epoch 278/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42735
794814.9497 - val_loss: 42056894580.7254
Epoch 279/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42729
875502.7493 - val_loss: 42060298646.2650
Epoch 280/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42732
875688.8188 - val_loss: 42062687375.2539
Epoch 281/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42730
823792.1984 - val_loss: 42069867786.0429
Epoch 282/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42727
965539.7688 - val_loss: 42053943359.6684
Epoch 283/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42724
```

```
002529.2497 - val_loss: 42050437932.5300
Epoch 284/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42723
140539.7081 - val_loss: 42069245260.7432
Epoch 285/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42723
534740.9819 - val_loss: 42056308145.5514
Epoch 286/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42714
900095.5183 - val_loss: 42039522086.4663
Epoch 287/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42718
585668.9395 - val_loss: 42056233257.8771
Epoch 288/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42715
723478.1941 - val_loss: 42032489654.6677
Epoch 289/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42719
645754.3735 - val_loss: 42037949784.8705
Epoch 290/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42716
779267.7431 - val_loss: 42032163084.3168
Epoch 291/500
16209/16209 [==============================] - 0s 20us/sample - loss: 42710
916786.5635 - val_loss: 42023462627.0081
Epoch 292/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42705
810020.7954 - val_loss: 42027020588.1510
Epoch 293/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42705
623587.3463 - val_loss: 42020803768.1836
Epoch 294/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42704
868783.0415 - val_loss: 42016474735.7987
Epoch 295/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42702
182191.9023 - val_loss: 42011855845.4715
Epoch 296/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42695
935395.6701 - val_loss: 42009793611.7957
Epoch 297/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42697
721730.0295 - val_loss: 42003591569.7172
Epoch 298/500
16209/16209 [==============================] - 0s 21us/sample - loss: 42694
084559.1027 - val_loss: 42010934571.3930
Epoch 299/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42689
899101.8461 - val_loss: 42017873189.3294
Epoch 300/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42691
206139.7673 - val_loss: 42010708615.2953
Epoch 301/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42691
887079.4882 - val_loss: 42007221821.7735
Epoch 302/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42685
```

```
836187.2362 - val_loss: 41990262976.5211
Epoch 303/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42686
862339.0956 - val_loss: 41995804050.4752
Epoch 304/500
16209/16209 [==============================] - 0s 19us/sample - loss: 42684
296148.3462 - val_loss: 41993876301.8801
Epoch 305/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42681
934764.5461 - val_loss: 41983437135.7750
Epoch 306/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42683
179290.7703 - val_loss: 42000501722.8601
Epoch 307/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42676
707326.2311 - val_loss: 41992918180.4767
Epoch 308/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42675
326197.6867 - val_loss: 41983527348.5833
Epoch 309/500
16209/16209 [==============================] - 0s 22us/sample - loss: 42671
220163.8892 - val_loss: 41985525282.4870
Epoch 310/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42669
005276.9696 - val_loss: 41981317996.1984
Epoch 311/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42667
610802.8794 - val_loss: 41971074004.7964
Epoch 312/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42666
220721.3316 - val_loss: 41979208880.6040
Epoch 313/500
16209/16209 [==============================] - 0s 23us/sample - loss: 42667
230843.7278 - val_loss: 41980823659.6299
Epoch 314/500
16209/16209 [==============================] - 0s 18us/sample - loss: 42669
442388.8279 - val_loss: 41968885241.5574
Epoch 315/500
16209/16209 [==============================] - 0s 17us/sample - loss: 42662
761594.7486 - val_loss: 41969025230.1643
Epoch 316/500

16209/16209 [==============================] - 0s 18us/sample - loss: 426
62508381.8935 - val_loss: 41963467297.7291
Epoch 317/500
16209/16209 [==============================] - 0s 22us/sample - loss: 426
56308843.6814 - val_loss: 41951020147.9674
Epoch 318/500
16209/16209 [==============================] - 0s 18us/sample - loss: 426
54729205.3235 - val_loss: 41952647851.6773
Epoch 319/500
16209/16209 [==============================] - 0s 20us/sample - loss: 426
51921629.9330 - val_loss: 41960267982.1643
Epoch 320/500
16209/16209 [==============================] - 0s 28us/sample - loss: 426
59052168.5523 - val_loss: 41954151712.7816
Epoch 321/500
```

```
16209/16209 [==============================] - 0s 20us/sample - loss: 426
52085994.7891 - val_loss: 41953179792.0118
Epoch 322/500
16209/16209 [==============================] - 0s 21us/sample - loss: 426
48449320.9215 - val_loss: 41956401655.2835
Epoch 323/500
16209/16209 [==============================] - 0s 23us/sample - loss: 426
46465246.4700 - val_loss: 41958220934.1584
Epoch 324/500
16209/16209 [==============================] - 0s 20us/sample - loss: 426
46064727.9709 - val_loss: 41942095175.4375
Epoch 325/500
16209/16209 [==============================] - 0s 19us/sample - loss: 426
42382620.9498 - val_loss: 41940218560.9001
Epoch 326/500
16209/16209 [==============================] - 0s 19us/sample - loss: 426
42767712.2942 - val_loss: 41945206112.4500
Epoch 327/500
16209/16209 [==============================] - 0s 21us/sample - loss: 426
46107660.0348 - val_loss: 41945541773.7380
Epoch 328/500
16209/16209 [==============================] - 0s 19us/sample - loss: 426
37982638.1887 - val_loss: 41934463717.2820
Epoch 329/500
16209/16209 [==============================] - 0s 17us/sample - loss: 426
40788094.4443 - val_loss: 41934587941.8979
Epoch 330/500
16209/16209 [==============================] - 0s 19us/sample - loss: 426
34385286.3254 - val_loss: 41931821016.5862
Epoch 331/500
16209/16209 [==============================] - 0s 22us/sample - loss: 426
37172209.1223 - val_loss: 41930803162.1021
Epoch 332/500
16209/16209 [==============================] - 0s 21us/sample - loss: 426
32069209.0764 - val_loss: 41943039978.7772
Epoch 333/500
16209/16209 [==============================] - 0s 18us/sample - loss: 426
32845024.5547 - val_loss: 41927896627.1621
Epoch 334/500
16209/16209 [==============================] - 0s 25us/sample - loss: 426
30076502.5494 - val_loss: 41938731590.1110
Epoch 335/500
16209/16209 [==============================] - 0s 17us/sample - loss: 426
25805724.7208 - val_loss: 41908973436.1155
Epoch 336/500
16209/16209 [==============================] - 0s 18us/sample - loss: 426
24347094.8732 - val_loss: 41923976979.5174
Epoch 337/500
16209/16209 [==============================] - 0s 18us/sample - loss: 426
24459801.4594 - val_loss: 41928144961.1843
Epoch 338/500
16209/16209 [==============================] - 0s 22us/sample - loss: 426
18541696.0237 - val_loss: 41910232104.1717
Epoch 339/500
16209/16209 [==============================] - 0s 17us/sample - loss: 426
18458754.1085 - val_loss: 41917059751.1295
Epoch 340/500
```

```
16209/16209 [==============================] - 0s 17us/sample - loss: 426
19377659.5778 - val_loss: 41905981088.3079
Epoch 341/500
16209/16209 [==============================] - 0s 17us/sample - loss: 426
22634843.3034 - val_loss: 41904055004.1865
Epoch 342/500
16209/16209 [==============================] - 0s 22us/sample - loss: 426
17939656.6747 - val_loss: 41923919729.5041
Epoch 343/500
16209/16209 [==============================] - 0s 21us/sample - loss: 426
11584644.9513 - val_loss: 41912788102.1584
Epoch 344/500
16209/16209 [==============================] - 0s 24us/sample - loss: 426
10898827.7584 - val_loss: 41905348059.2391
Epoch 345/500
16209/16209 [==============================] - 0s 25us/sample - loss: 426
11207358.3456 - val_loss: 41906662080.9001
Epoch 346/500
16209/16209 [==============================] - 0s 23us/sample - loss: 426
11856578.2624 - val_loss: 41893783514.1021
Epoch 347/500
16209/16209 [==============================] - 1s 37us/sample - loss: 426
05262608.8203 - val_loss: 41896937373.4656
Epoch 348/500
16209/16209 [==============================] - 0s 25us/sample - loss: 426
06552604.8393 - val_loss: 41904830594.3686
Epoch 349/500
16209/16209 [==============================] - 0s 19us/sample - loss: 426
04878983.0045 - val_loss: 41900895082.6825
Epoch 350/500
16209/16209 [==============================] - 0s 21us/sample - loss: 426
03846862.3920 - val_loss: 41889100703.7394
Epoch 351/500
16209/16209 [==============================] - 0s 27us/sample - loss: 426
00718849.9900 - val_loss: 41890892088.2783
Epoch 352/500
16209/16209 [==============================] - 0s 22us/sample - loss: 425
98155714.7521 - val_loss: 41899226649.3916
Epoch 353/500
16209/16209 [==============================] - 0s 18us/sample - loss: 426
02477961.8316 - val_loss: 41895947196.5418
Epoch 354/500
16209/16209 [==============================] - 0s 21us/sample - loss: 425
94886001.8252 - val_loss: 41883315520.6158
Epoch 355/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
94558683.4376 - val_loss: 41895707682.1081
Epoch 356/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
97324012.5895 - val_loss: 41867767767.8283
Epoch 357/500
16209/16209 [==============================] - 0s 20us/sample - loss: 425
88761249.5379 - val_loss: 41881409676.9800
Epoch 358/500
16209/16209 [==============================] - 0s 21us/sample - loss: 425
91622350.8974 - val_loss: 41878847839.6921
Epoch 359/500
```

```
16209/16209 [==============================] - 0s 18us/sample - loss: 425
84887319.2483 - val_loss: 41901004119.3546
Epoch 360/500
16209/16209 [==============================] - 0s 21us/sample - loss: 425
81180704.3297 - val_loss: 41885628863.1947
Epoch 361/500
16209/16209 [==============================] - 0s 22us/sample - loss: 425
80454157.7247 - val_loss: 41878153441.1133
Epoch 362/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
84228969.8178 - val_loss: 41867703527.1769
Epoch 363/500
16209/16209 [==============================] - 0s 22us/sample - loss: 425
75577964.1078 - val_loss: 41870238150.7742
Epoch 364/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
83079458.0828 - val_loss: 41861513434.2916
Epoch 365/500
16209/16209 [==============================] - 0s 29us/sample - loss: 425
72029765.5080 - val_loss: 41868139713.2791
Epoch 366/500
16209/16209 [==============================] - 0s 21us/sample - loss: 425
79962728.0647 - val_loss: 41863665580.6247
Epoch 367/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
70493801.6440 - val_loss: 41862214949.3294
Epoch 368/500
16209/16209 [==============================] - 0s 27us/sample - loss: 425
68517156.5466 - val_loss: 41870206108.8971
Epoch 369/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
75676700.4760 - val_loss: 41868013140.5122
Epoch 370/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
69561547.5965 - val_loss: 41862851643.8786
Epoch 371/500
16209/16209 [==============================] - 0s 21us/sample - loss: 425
67738804.3482 - val_loss: 41864882340.4767
Epoch 372/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
75831511.0312 - val_loss: 41842021995.2509
Epoch 373/500
16209/16209 [==============================] - 0s 20us/sample - loss: 425
63915091.4381 - val_loss: 41866300993.5633
Epoch 374/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
60264160.5390 - val_loss: 41849421435.9260
Epoch 375/500
16209/16209 [==============================] - 0s 21us/sample - loss: 425
58657439.4058 - val_loss: 41848922627.4108
Epoch 376/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
56122300.9558 - val_loss: 41854970630.6321
Epoch 377/500
16209/16209 [==============================] - 0s 20us/sample - loss: 425
53757749.3195 - val_loss: 41840587155.2332
Epoch 378/500
```

```
16209/16209 [==============================] - 0s 20us/sample - loss: 425
55970132.1172 - val_loss: 41845020306.6647
Epoch 379/500
16209/16209 [==============================] - 0s 21us/sample - loss: 425
48026780.5945 - val_loss: 41838931576.8941
Epoch 380/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
48021238.5238 - val_loss: 41843305709.2406
Epoch 381/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
48776403.4302 - val_loss: 41835342777.5100
Epoch 382/500
16209/16209 [==============================] - 0s 25us/sample - loss: 425
44905346.9613 - val_loss: 41835767491.1739
Epoch 383/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
45618859.3458 - val_loss: 41830434058.0429
Epoch 384/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
41676050.8577 - val_loss: 41830059501.4301
Epoch 385/500
16209/16209 [==============================] - 0s 17us/sample - loss: 425
40642448.7966 - val_loss: 41825827340.5063
Epoch 386/500
16209/16209 [==============================] - 0s 23us/sample - loss: 425
37855486.3259 - val_loss: 41817576651.8904
Epoch 387/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
38783596.7396 - val_loss: 41824813698.7476
Epoch 388/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
36585696.0020 - val_loss: 41840691944.3138
Epoch 389/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
35156483.7431 - val_loss: 41827026702.9697
Epoch 390/500
16209/16209 [==============================] - 0s 21us/sample - loss: 425
38977516.6527 - val_loss: 41819330608.1303
Epoch 391/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
36683015.4862 - val_loss: 41826224156.8024
Epoch 392/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
30629338.2373 - val_loss: 41816275214.5907
Epoch 393/500
16209/16209 [==============================] - 0s 24us/sample - loss: 425
26317600.2823 - val_loss: 41813550856.9060
Epoch 394/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
27121996.1572 - val_loss: 41813787594.1851
Epoch 395/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
24469110.4053 - val_loss: 41819684701.7972
Epoch 396/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
26812406.5080 - val_loss: 41819295280.1303
Epoch 397/500
```

```
16209/16209 [==============================] - 0s 24us/sample - loss: 425
23970798.8638 - val_loss: 41811849690.4811
Epoch 398/500
16209/16209 [==============================] - 0s 20us/sample - loss: 425
19621961.2670 - val_loss: 41796768456.4796
Epoch 399/500
16209/16209 [==============================] - 0s 20us/sample - loss: 425
25097418.7121 - val_loss: 41804781402.7654
Epoch 400/500
16209/16209 [==============================] - 0s 22us/sample - loss: 425
13847517.4276 - val_loss: 41798941861.9926
Epoch 401/500
16209/16209 [==============================] - 0s 19us/sample - loss: 425
15520672.4008 - val_loss: 41807451697.6462
Epoch 402/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
17492193.3918 - val_loss: 41799180030.2946
Epoch 403/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
11000511.7513 - val_loss: 41800360933.4715
Epoch 404/500
16209/16209 [==============================] - 0s 23us/sample - loss: 425
10135101.1690 - val_loss: 41801089674.3272
Epoch 405/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
07972944.5952 - val_loss: 41813464740.8557
Epoch 406/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
08272300.8146 - val_loss: 41803651037.1340
Epoch 407/500
16209/16209 [==============================] - 0s 22us/sample - loss: 425
06795329.5596 - val_loss: 41815928177.1251
Epoch 408/500
16209/16209 [==============================] - 0s 18us/sample - loss: 425
15293849.2936 - val_loss: 41803612498.0489
Epoch 409/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
98969087.2735 - val_loss: 41795106919.0822
Epoch 410/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
97490839.8880 - val_loss: 41797948253.0392
Epoch 411/500
16209/16209 [==============================] - 0s 22us/sample - loss: 424
96710787.6562 - val_loss: 41816660057.4389
Epoch 412/500
16209/16209 [==============================] - 0s 22us/sample - loss: 424
99855773.3526 - val_loss: 41799021657.4389
Epoch 413/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
94567725.4069 - val_loss: 41800050289.3146
Epoch 414/500
16209/16209 [==============================] - 0s 22us/sample - loss: 424
95267060.7391 - val_loss: 41786118177.3501
Epoch 415/500
16209/16209 [==============================] - 0s 23us/sample - loss: 424
90962490.0892 - val_loss: 41767306919.1295
Epoch 416/500
```

```
16209/16209 [==============================] - 0s 23us/sample - loss: 424
89274014.4739 - val_loss: 41781767340.0563
Epoch 417/500
16209/16209 [==============================] - 0s 20us/sample - loss: 424
89986491.8028 - val_loss: 41786687984.4619
Epoch 418/500
16209/16209 [==============================] - 0s 22us/sample - loss: 424
84555181.3990 - val_loss: 41779212115.1858
Epoch 419/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
86131999.0030 - val_loss: 41776981314.1318
Epoch 420/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
86660919.3569 - val_loss: 41774912876.5773
Epoch 421/500
16209/16209 [==============================] - 0s 23us/sample - loss: 424
80959219.1281 - val_loss: 41781797115.6417
Epoch 422/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
78708976.0642 - val_loss: 41776468938.1851
Epoch 423/500
16209/16209 [==============================] - 0s 20us/sample - loss: 424
74375244.6942 - val_loss: 41792838097.3856
Epoch 424/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
74862965.6156 - val_loss: 41782356815.3960
Epoch 425/500
16209/16209 [==============================] - 0s 21us/sample - loss: 424
72497822.9162 - val_loss: 41770846882.5818
Epoch 426/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
70819979.3636 - val_loss: 41769855874.9371
Epoch 427/500
16209/16209 [==============================] - 0s 27us/sample - loss: 424
70197675.4406 - val_loss: 41776227057.4093
Epoch 428/500
16209/16209 [==============================] - 0s 29us/sample - loss: 424
69826584.7645 - val_loss: 41764755339.2746
Epoch 429/500
16209/16209 [==============================] - 0s 22us/sample - loss: 424
68710096.3820 - val_loss: 41758839802.6943
Epoch 430/500
16209/16209 [==============================] - 0s 22us/sample - loss: 424
64066308.8802 - val_loss: 41751644135.7454
Epoch 431/500
16209/16209 [==============================] - 0s 29us/sample - loss: 424
71426326.0954 - val_loss: 41760709161.3087
Epoch 432/500
16209/16209 [==============================] - 0s 25us/sample - loss: 424
63215010.5961 - val_loss: 41760709623.6625
Epoch 433/500
16209/16209 [==============================] - 0s 25us/sample - loss: 424
63010818.5270 - val_loss: 41756303078.0400
Epoch 434/500
16209/16209 [==============================] - 0s 28us/sample - loss: 424
60531717.8753 - val_loss: 41753337971.2095
Epoch 435/500
```

```
16209/16209 [==============================] - 0s 26us/sample - loss: 424
61359992.5533 - val_loss: 41753157835.1325
Epoch 436/500
16209/16209 [==============================] - 0s 31us/sample - loss: 424
53995704.2177 - val_loss: 41766944143.4434
Epoch 437/500
16209/16209 [==============================] - 0s 24us/sample - loss: 424
49831636.6779 - val_loss: 41756851726.7802
Epoch 438/500
16209/16209 [==============================] - 0s 24us/sample - loss: 424
47770619.6409 - val_loss: 41754722698.1377
Epoch 439/500
16209/16209 [==============================] - 0s 29us/sample - loss: 424
49824944.6999 - val_loss: 41739373882.5522
Epoch 440/500
16209/16209 [==============================] - 0s 20us/sample - loss: 424
48747756.8422 - val_loss: 41742275822.7565
Epoch 441/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
43827060.8891 - val_loss: 41740726835.1621
Epoch 442/500
16209/16209 [==============================] - 1s 34us/sample - loss: 424
44275779.9761 - val_loss: 41747533695.9053
Epoch 443/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
52482120.3983 - val_loss: 41721630357.6965
Epoch 444/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
42784757.8920 - val_loss: 41734371100.6129
Epoch 445/500
16209/16209 [==============================] - 0s 20us/sample - loss: 424
45680304.6683 - val_loss: 41726049746.1436
Epoch 446/500
16209/16209 [==============================] - 0s 23us/sample - loss: 424
40093662.5174 - val_loss: 41725924101.1162
Epoch 447/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
40487937.7057 - val_loss: 41740025343.6210
Epoch 448/500
16209/16209 [==============================] - 0s 17us/sample - loss: 424
32432744.4595 - val_loss: 41733970353.5514
Epoch 449/500
16209/16209 [==============================] - 0s 21us/sample - loss: 424
31006189.2055 - val_loss: 41723305447.3664
Epoch 450/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
29356639.3623 - val_loss: 41726934591.2894
Epoch 451/500
16209/16209 [==============================] - 0s 17us/sample - loss: 424
26518455.9176 - val_loss: 41720018328.5389
Epoch 452/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
31851846.8031 - val_loss: 41732757607.0822
Epoch 453/500
16209/16209 [==============================] - 0s 25us/sample - loss: 424
24848867.9188 - val_loss: 41726091121.5041
Epoch 454/500
```

```
16209/16209 [==============================] - 0s 22us/sample - loss: 424
21833978.2985 - val_loss: 41723677910.5019
Epoch 455/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
27188840.2700 - val_loss: 41717245680.6514
Epoch 456/500
16209/16209 [==============================] - 0s 24us/sample - loss: 424
21678240.9693 - val_loss: 41729875254.0044
Epoch 457/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
20904030.0041 - val_loss: 41723513497.4863
Epoch 458/500
16209/16209 [==============================] - 0s 18us/sample - loss: 424
13552462.5420 - val_loss: 41718648765.2998
Epoch 459/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
15958250.8838 - val_loss: 41731307795.8964
Epoch 460/500
16209/16209 [==============================] - 0s 24us/sample - loss: 424
19742665.2906 - val_loss: 41714123070.3420
Epoch 461/500
16209/16209 [==============================] - 0s 22us/sample - loss: 424
09519921.1658 - val_loss: 41708757584.7224
Epoch 462/500
16209/16209 [==============================] - 0s 23us/sample - loss: 424
08668912.7275 - val_loss: 41700030492.8024
Epoch 463/500
16209/16209 [==============================] - 0s 25us/sample - loss: 424
17981134.3604 - val_loss: 41696727231.0052
Epoch 464/500
16209/16209 [==============================] - 0s 20us/sample - loss: 424
09831582.0001 - val_loss: 41719061917.0866
Epoch 465/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
06823706.9283 - val_loss: 41725200612.9030
Epoch 466/500
16209/16209 [==============================] - 0s 22us/sample - loss: 424
05544854.0559 - val_loss: 41710416917.9808
Epoch 467/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
08208559.6259 - val_loss: 41711721218.8423
Epoch 468/500
16209/16209 [==============================] - 0s 20us/sample - loss: 423
97595398.0806 - val_loss: 41711093320.3849
Epoch 469/500
16209/16209 [==============================] - 0s 19us/sample - loss: 424
04950768.2853 - val_loss: 41699247668.6780
Epoch 470/500
16209/16209 [==============================] - 0s 22us/sample - loss: 423
97447501.1206 - val_loss: 41692957830.1584
Epoch 471/500
16209/16209 [==============================] - 0s 20us/sample - loss: 423
97244244.7332 - val_loss: 41697336532.9859
Epoch 472/500
16209/16209 [==============================] - 0s 18us/sample - loss: 423
94226941.8363 - val_loss: 41695543657.5455
Epoch 473/500
```

```
16209/16209 [==============================] - 0s 20us/sample - loss: 423
95795413.0412 - val_loss: 41697706375.1059
Epoch 474/500
16209/16209 [==============================] - 0s 20us/sample - loss: 423
90618514.6760 - val_loss: 41700371036.8497
Epoch 475/500
16209/16209 [==============================] - 0s 19us/sample - loss: 423
89591556.7065 - val_loss: 41698643960.4204
Epoch 476/500
16209/16209 [==============================] - 0s 18us/sample - loss: 423
92294862.1235 - val_loss: 41699516693.4123
Epoch 477/500
16209/16209 [==============================] - 0s 23us/sample - loss: 423
85464632.3993 - val_loss: 41709960166.9874
Epoch 478/500
16209/16209 [==============================] - 0s 28us/sample - loss: 423
83916112.8005 - val_loss: 41690029419.0614
Epoch 479/500
16209/16209 [==============================] - 0s 22us/sample - loss: 423
85341817.0903 - val_loss: 41695563084.7432
Epoch 480/500
16209/16209 [==============================] - 1s 35us/sample - loss: 423
81924138.4692 - val_loss: 41688276444.7550
Epoch 481/500
16209/16209 [==============================] - 0s 21us/sample - loss: 423
88600149.8387 - val_loss: 41692202002.9489
Epoch 482/500
16209/16209 [==============================] - 0s 18us/sample - loss: 423
81948311.5405 - val_loss: 41688644143.3723
Epoch 483/500
16209/16209 [==============================] - 0s 23us/sample - loss: 423
81960629.3590 - val_loss: 41684112142.2117
Epoch 484/500
16209/16209 [==============================] - 0s 20us/sample - loss: 423
74262597.1290 - val_loss: 41666150104.3967
Epoch 485/500
16209/16209 [==============================] - 0s 19us/sample - loss: 423
76775959.3589 - val_loss: 41673595848.6691
Epoch 486/500
16209/16209 [==============================] - 0s 30us/sample - loss: 423
70823592.2186 - val_loss: 41670924219.7839
Epoch 487/500
16209/16209 [==============================] - 0s 22us/sample - loss: 423
68871491.4075 - val_loss: 41668379274.3272
Epoch 488/500
16209/16209 [==============================] - 0s 20us/sample - loss: 423
67519073.6524 - val_loss: 41680955907.4108
Epoch 489/500
16209/16209 [==============================] - 0s 19us/sample - loss: 423
66321438.5292 - val_loss: 41673437123.3634
Epoch 490/500
16209/16209 [==============================] - 0s 22us/sample - loss: 423
65342115.6069 - val_loss: 41659476616.8112
Epoch 491/500
16209/16209 [==============================] - 0s 19us/sample - loss: 423
64501584.1372 - val_loss: 41670621373.4893
Epoch 492/500
```

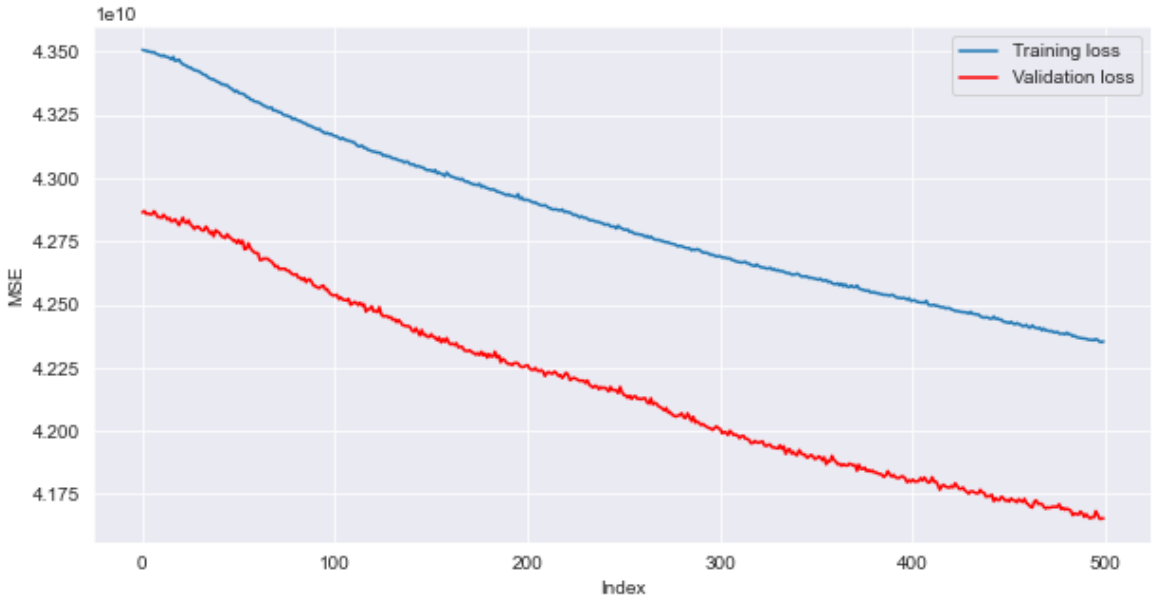
```
16209/16209 [==============================] - 0s 18us/sample - loss: 423
61422132.4193 - val_loss: 41660284300.4115
Epoch 493/500
16209/16209 [==============================] - 0s 28us/sample - loss: 423
60062455.7557 - val_loss: 41653671050.7061
Epoch 494/500
16209/16209 [==============================] - 0s 30us/sample - loss: 423
60707777.2832 - val_loss: 41658584748.4352
Epoch 495/500
16209/16209 [==============================] - 0s 22us/sample - loss: 423
58533509.2198 - val_loss: 41657240894.3420
Epoch 496/500
16209/16209 [==============================] - 0s 25us/sample - loss: 423
63552048.2497 - val_loss: 41682316544.9474
Epoch 497/500
16209/16209 [==============================] - 0s 19us/sample - loss: 423
59917057.6110 - val_loss: 41667492456.9771
Epoch 498/500
16209/16209 [==============================] - 0s 18us/sample - loss: 423
53179441.6080 - val_loss: 41652027246.4722
Epoch 499/500
16209/16209 [==============================] - 0s 22us/sample - loss: 423
51074237.8560 - val_loss: 41652582128.6514
Epoch 500/500
16209/16209 [==============================] - 0s 21us/sample - loss: 423
54201394.2397 - val_loss: 41654101832.5744
```

Out[734]: <tensorflow.python.keras.callbacks.History at 0x17faeeee630>

In [735]:
```
output = pd.DataFrame(model.history.history)
```

In [736]:
```
fig = plt.figure(figsize=(10,5))
sns.lineplot(x= output.index,y=output['loss'],label='Training loss')
sns.lineplot(x= output.index,y=output['val_loss'],label='Validation loss',col
plt.ylabel('MSE')
plt.xlabel('Index')
```

Out[736]: Text(0.5, 0, 'Index')



In [737]:
```
X_train.head()
```

Out[737]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade
0	0.090909	0.12500	0.050566	0.024488	0.0	0.0	0.0	1.00	0.500000
1	0.272727	0.21875	0.092075	0.005573	0.0	0.0	0.0	0.50	0.500000
2	0.272727	0.28125	0.104151	0.003570	0.0	0.0	0.0	0.75	0.583333
3	0.272727	0.25000	0.194351	0.116049	0.0	0.0	0.0	0.75	0.666667
4	0.272727	0.12500	0.105660	0.007863	0.0	0.0	0.0	0.75	0.500000

In [738]:
```
print('The Prediction Accuracy for the Deep Neural Network on the testing dat
```

```
The Prediction Accuracy for the Deep Neural Network on the testing dataset
is 0.69
```

In [740]:
```
dnn_accuracy = metrics.r2_score(y_test,model.predict(X_test))
```

In [741]:
```
dnn_accuracy
```

Out[741]: 0.6864354280503161

Comparing All the Models That Have been

Created

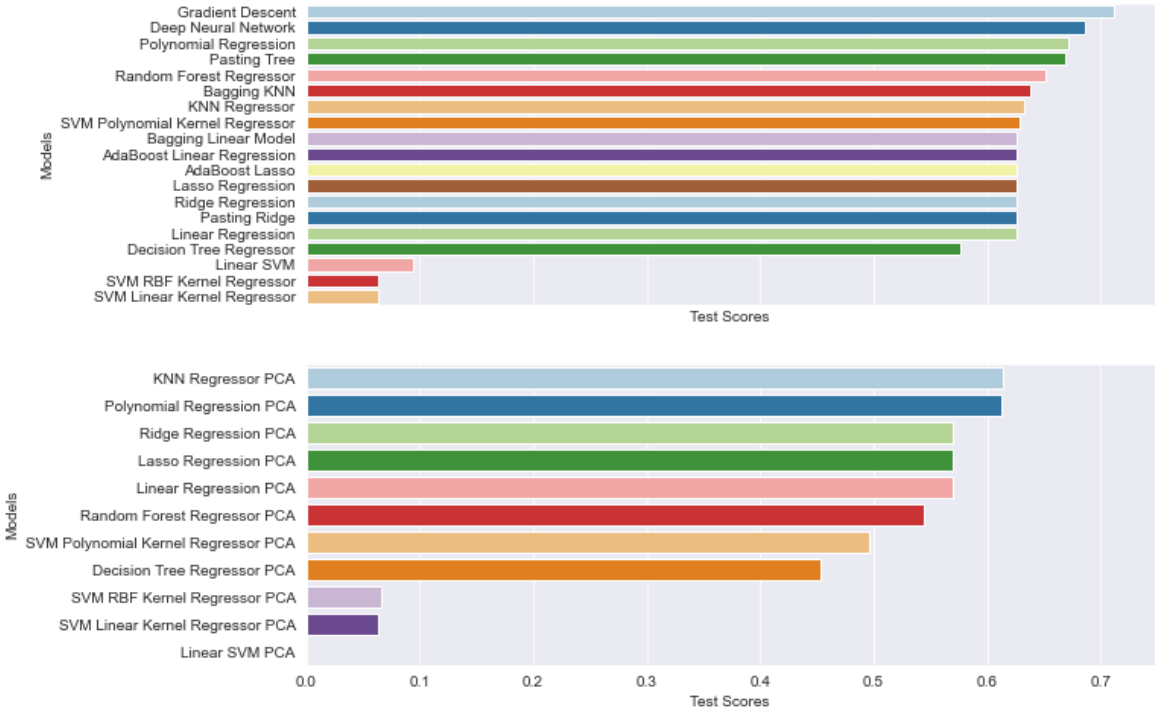
In [760]:
```
evaluation = {'Models':['KNN Regressor','Linear Regression','Ridge Regression
                        'Decision Tree Regressor','Random Forest Regressor','P
                        'SVM Linear Kernel Regressor','SVM RBF Kernel Regressc
                        'Bagging KNN','Bagging Linear Model','Pasting Ridge','
                        'AdaBoost Linear Regression','Gradient Descent','Deep
              'Test Scores':[knn_mean_score_cv,lm_mean_score_cv,ridge_mean_scc
                             rf_mean_score_cv,polynomial_reg_cv,mean_cross_val_
                             mean_cross_val_svm_rbf_kernel,mean_cross_val_svm_p
                             pasting_ridge_score,pasting_tree_score,ada_lasso_s
```

In [761]:
```
eva = pd.DataFrame(evaluation)
```

In [762]:
```
eva = eva.sort_values(by='Test Scores',ascending = False)
```

In [763]:
```
fig, ax =plt.subplots(2,1,sharex=True)
plt.figure(figsize=(10,10),dpi=100)
sns.barplot(x=eva['Test Scores'],y=eva['Models'],orient='h',palette='Paired',
sns.barplot(x=eva_pca['Test Scores'][:-1],y=eva_pca['Models'],orient='h',pale
```

Out[763]: <matplotlib.axes._subplots.AxesSubplot at 0x1803f07a748>



<Figure size 1000x1000 with 0 Axes>

The plot Above makes it very clear that the Gradient Descent Algorithm On trees actually provides us with the best score or the best R squared which is about 0.71

Hence we would like to make us of the Gradient Descent Algorithm to make the predictions for the

test data

Prediction

In [764]:
```
gbrt = GradientBoostingRegressor(learning_rate=0.2,max_depth=4,n_estimators=1
```

In [765]:
```
gbrt.fit(X_train,y_train)
```

Out[765]:
```
GradientBoostingRegressor(learning_rate=0.2, max_depth=4, random_state=0)
```

In [811]:
```
prediction = gbrt.predict(X_test)
```

Predictions For the Test Set

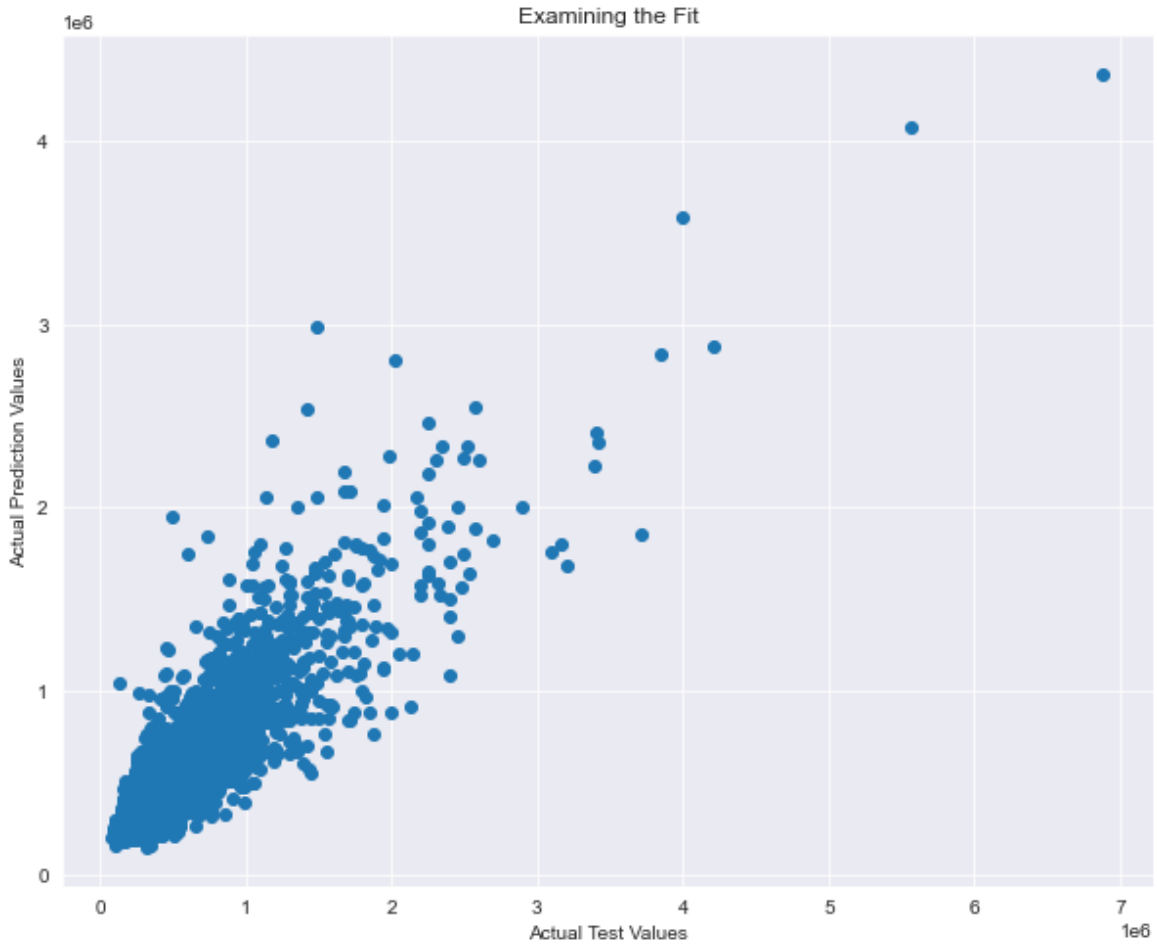
In [770]:
```
print(list(gbrt.predict(X_test)))
```

```
[391694.4628908604, 1450567.793271134, 505127.8354101141, 653901.08112559
1, 696532.2963082253, 385273.77122629643, 372215.6004149924, 448499.21246
950934, 586238.2294266961, 1948082.5303039704, 724583.6543836907, 395456.
9346012529, 378270.22699316393, 410147.829426596, 751184.0918978811, 3768
83.6199281304, 415657.6911124771, 683889.2933568554, 191015.17416581605,
414636.1757956521, 303125.0231792115, 406968.0339042072, 516944.310638321
74, 278949.8343654908, 525414.1925881532, 401295.0233934811, 352231.70753
210934, 398700.3066344121, 868609.0404709959, 483896.5498061563, 850679.4
473091619, 384832.7538296443, 525309.2537049463, 780376.2544050686, 75090
1.0701809343, 419006.550236517, 342119.27147707087, 367903.112790368, 383
000.375118268, 238967.18160387586, 342902.18505432416, 265192.8668421055,
469157.1545281147, 603936.848055994, 515744.74103303463, 658878.977532090
2, 374868.58857605024, 695197.890737423, 600535.6487302675, 785220.492778
8172, 330974.5720983239, 450579.0201237603, 866861.1097457795, 299149.337
43002714, 817614.6108859775, 447973.10596270056, 374005.61401801475, 3591
49.44018644193, 462185.5536264747, 729558.430180722, 399589.2146786551, 3
62329.6551151341, 521972.8122931022, 704308.9493807111, 503064.214957641
2, 936305.4358850726, 446072.2209340517, 447750.2342489209, 1127698.48725
88925, 391117.72221818805, 850396.6448567372, 465438.39062254864, 479565.
```

In [808]:
```
x=np.array([100])
y = np.array([100])
```

In [810]:
```
plt.scatter(y_test,gbrt.predict(X_test))
plt.xlabel('Actual Test Values')
plt.ylabel('Actual Prediction Values')
plt.title('Examining the Fit')
```

Out[810]: Text(0.5, 1.0, 'Examining the Fit')



The above plot actually means that that we have significantly good predictions as the test actual scores and the predicted scores actually line up

In [814]:
```
print('The R squared value we get after implementing the final model on the t
```

```
The R squared value we get after implementing the final model on the testin
g dataset is 0.74 which also means that about 74% of the variance in the ho
use prices is explained by our model which if you take the nature of proble
m into account is a very good score to have. The score is 0.74
```

Additional fit metrics

In [816]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
```

In [819]:
```
print('The Mean Absolute Error is {}'.format(mean_absolute_error(y_test,predi
```

```
The Mean Absolute Error is 121537.34656403525
```

In [821]:
```
print('The Mean Squared Error is {}'.format(mean_squared_error(y_test,predict
```

```
The Mean Squared Error is 34580837874.11784
```

End Of The Project