```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
```

Importing the Boston House Price Dataset

```
house_df = pd.read_csv('/content/home_data.csv')
```

```
house_df.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...
0	7129300520	20141013T000000	221900	3	1.00	1180	5650	1.0	0	0	...
1	6414100192	20141209T000000	538000	3	2.25	2570	7242	2.0	0	0	...
2	5631500400	20150225T000000	180000	2	1.00	770	10000	1.0	0	0	...
3	2487200875	20141209T000000	604000	4	3.00	1960	5000	1.0	0	0	...
4	1954400510	20150218T000000	510000	3	2.00	1680	8080	1.0	0	0	...

5 rows × 21 columns

```
house_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  int64  
 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(4), int64(16), object(1)
memory usage: 3.5+ MB
```

```
house_df.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfron
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.00000
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.00754
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.08651
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.00000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.00000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.00000

```
# checking the number of rows and Columns in the data frame
house_df.shape
```

```
(21613, 21)
```

```
house_df.nunique()
```

	0
id	21436
date	372
price	4032
bedrooms	13
bathrooms	30
sqft_living	1038
sqft_lot	9782
floors	6
waterfront	2
view	5
condition	5
grade	12
sqft_above	946
sqft_basement	306
yr_built	116
yr_renovated	70
zipcode	70
lat	5034
long	752
sqft_living15	777
sqft_lot15	8689

dtype: int64

```
# check for missing values
house_df.isnull().sum()
```

	0
id	0
date	0
price	0
bedrooms	0
bathrooms	0
sqft_living	0
sqft_lot	0
floors	0
waterfront	0
view	0
condition	0
grade	0
sqft_above	0
sqft_basement	0
yr_built	0
yr_renovated	0
zipcode	0
lat	0
long	0

dtype: int64

```
# statistical measures of the dataset
house_price_dataframe.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000

Data Preprocessing

```
# changing float to integer
house_df['bathrooms'] = house_df['bathrooms'].astype(int)
house_df['floors'] = house_df['floors'].astype(int)
# renaming the column yr_built to age and changing the values to age
house_df.rename(columns={'yr_built':'age'},inplace=True)
house_df['age'] = 2023 - house_df['age']
# changing the column yr_renovated to renovated and changing the values to 0 and 1
house_df.rename(columns={'yr_renovated':'renovated'},inplace=True)
house_df['renovated'] = house_df['renovated'].apply(lambda x: 0 if x == 0 else 1)
```

```
# using simple feature scaling
house_df['sqft_living'] = house_df['sqft_living']/house_df['sqft_living'].max()
house_df['sqft_living15'] = house_df['sqft_living15']/house_df['sqft_living15'].max()
house_df['sqft_lot'] = house_df['sqft_lot']/house_df['sqft_lot'].max()
house_df['sqft_above'] = house_df['sqft_above']/house_df['sqft_above'].max()
house_df['sqft_basement'] = house_df['sqft_basement']/house_df['sqft_basement'].max()
house_df['sqft_lot15'] = house_df['sqft_lot15']/house_df['sqft_lot15'].max()
```

```
house_df.head()
```

:	view	...
)	0	...
)	0	...
)	0	...
)	0	...
)	0	...

Exploratory Data Analysis

Correlation Matrix to find the relationship between the variables

Understanding the correlation between various features in the dataset

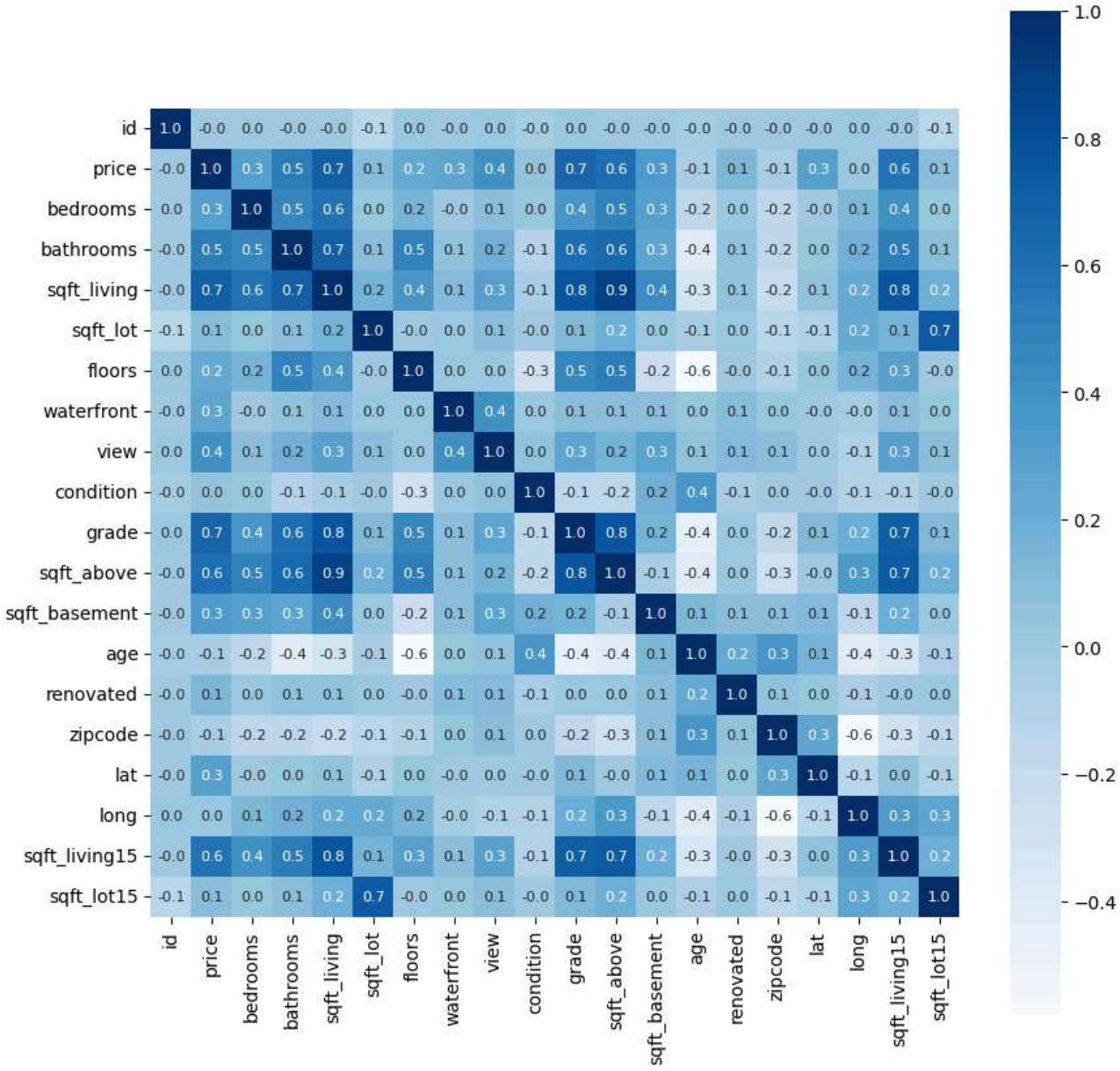
1. Positive Correlation
2. Negative Correlation

```
# using correlation statistical method to find the relation between the price and other features
# The 'date' column is a string and cannot be converted to a float for correlation calculation.
# drop it before calculating the correlation matrix.
correlation = house_df.drop('date', axis=1).corr()
correlation['price'].sort_values(ascending=False)
```

	price
price	1.000000
sqft_living	0.702035
grade	0.667434
sqft_above	0.605567
sqft_living15	0.585379
bathrooms	0.510072

```
# constructing a heatmap to nderstand the correlation
plt.figure(figsize=(10,10))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot=True, annot_kws={'size':8}, cmap='Blues')
```

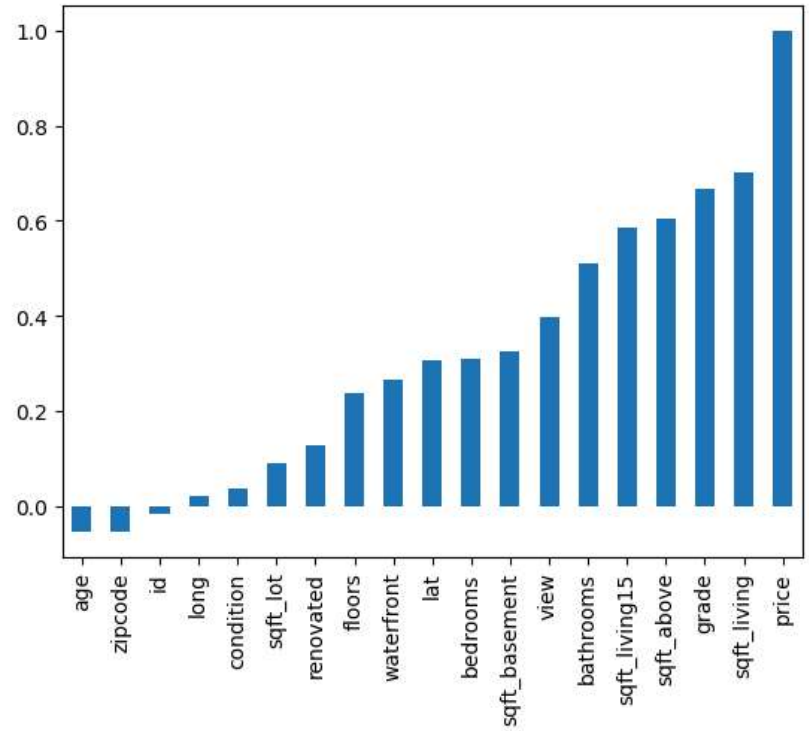
```
<Axes: >
```



Visualizing the coorelation with price

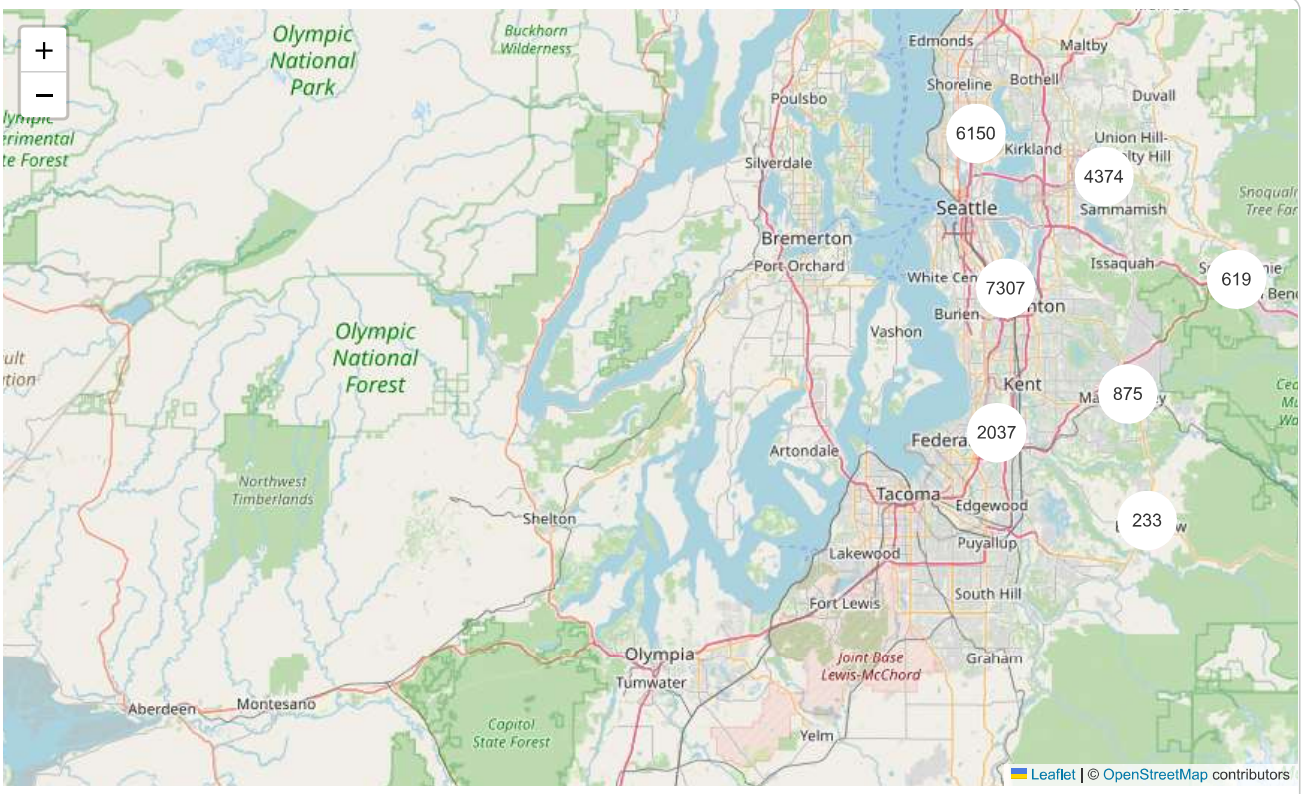
```
house_df.drop('date', axis=1).corr()['price'][:-1].sort_values().plot(kind='bar')
```

```
<Axes: >
```




```
# adding a new column price_range and categorizing the price into 4 categories
# The 'price' column values are already scaled (e.g., 4.526 means $452,600).
# The original bins were for absolute dollar values, so they are too large for the scaled 'price' column.
# Let's define new bins based on the quantiles of the 'price' column to create balanced categories.

q1 = house_df['price'].quantile(0.25)
q2 = house_df['price'].quantile(0.50)
q3 = house_df['price'].quantile(0.75)
max_price = house_df['price'].max()

# Define bins using quantiles
bins = [0, q1, q2, q3, max_price + 0.00001] # Adding a small epsilon to ensure max value is included
labels = ['Low', 'Medium', 'High', 'Very High']

house_df['price_range'] = pd.cut(house_df['price'], bins=bins, labels=labels, include_lowest=True)
```

```
import folium
from folium.plugins import FastMarkerCluster

# Use the mean of Latitude and Longitude for initial map centering
map = folium.Map(location=[house_df['lat'].mean(), house_df['long'].mean()], zoom_start=8)
marker_cluster = FastMarkerCluster(house_df[['lat', 'long']].values.tolist()).add_to(map)
map
```



Train/Test Split

```
X = house_df.drop(['price', 'price_range', 'date'], axis=1)
Y = house_df['price']
```

```
print(X)
print(Y)
```

```
                id  bedrooms  bathrooms  sqft_living  sqft_lot  floors  \
0       7129300520         3          1     0.087149  0.003421       1
1       6414100192         3          2     0.189808  0.004385       2
2       5631500400         2          1     0.056869  0.006056       1
3       2487200875         4          3     0.144756  0.003028       1
4       1954400510         3          2     0.124077  0.004893       1
...            ...       ...        ...          ...       ...     ...
21608    263000018         3          2     0.112999  0.000685       3
21609   6600060120         4          2     0.170606  0.003520       2
21610   1523300141         2          0     0.075332  0.000818       2
21611    291310100         3          2     0.118168  0.001446       2
21612   1523300157         2          0     0.075332  0.000652       2

       waterfront  view  condition  grade  sqft_above  sqft_basement  age  \
0               0     0          3      7    0.125399       0.000000   68
1               0     0          3      7    0.230606       0.082988   72
2               0     0          3      6    0.081828       0.000000   90
3               0     0          5      7    0.111583       0.188797   58
4               0     0          3      8    0.178533       0.000000   36
...           ...   ...        ...    ...         ...            ...  ...
21608           0     0          3      8    0.162593       0.000000   14
21609           0     0          3      8    0.245484       0.000000    9
21610           0     0          3      7    0.108395       0.000000   14
21611           0     0          3      8    0.170032       0.000000   19
21612           0     0          3      7    0.108395       0.000000   15

       renovated  zipcode      lat     long  sqft_living15  sqft_lot15
0              0    98178  47.5112 -122.257       0.215781    0.006485
1              1    98125  47.7210 -122.319       0.272142    0.008768
2              0    98028  47.7379 -122.233       0.438003    0.009254
```

```
3             0    98136  47.5208 -122.393       0.219002    0.005739
4             0    98074  47.6168 -122.045       0.289855    0.008612
...         ...      ...      ...      ...            ...         ...
21608         0    98103  47.6993 -122.346       0.246377    0.001732
21609         0    98146  47.5107 -122.362       0.294686    0.008264
21610         0    98144  47.5944 -122.299       0.164251    0.002304
21611         0    98027  47.5345 -122.069       0.227053    0.001477
21612         0    98144  47.5941 -122.299       0.164251    0.001558

[21613 rows x 19 columns]
0        221900
1        538000
2        180000
3        604000
4        510000
          ...
21608    360000
21609    400000
21610    402101
21611    400000
21612    325000
Name: price, Length: 21613, dtype: int64
```

Splitting the data into Training data and Test data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(21613, 19) (17290, 19) (4323, 19)
```

Model Training

XGBoost Regressor

```
# loading the model
model = XGBRegressor()
```

```
# training the model with X_train
model.fit(X_train, Y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, ...)
```

```
# accuracy for prediction on training data
training_data_prediction = model.predict(X_train)
```

```
print(training_data_prediction)
```

```
[1740651.2   364077.75  667344.25 ... 1013368.06  320587.1   718365.56]
```

```
# R squared error
score_1 = metrics.r2_score(Y_train, training_data_prediction)
```

```
# Mean Absolute Error
score_2 = metrics.mean_absolute_error(Y_train, training_data_prediction)

print("R squared error : ", score_1)
print('Mean Absolute Error : ', score_2)
```

```
R squared error :  0.9772560000419617
Mean Absolute Error :  39741.98046875
```

```
#training the model
model.fit(X_train, Y_train)
model.score(X_test,Y_test)
```

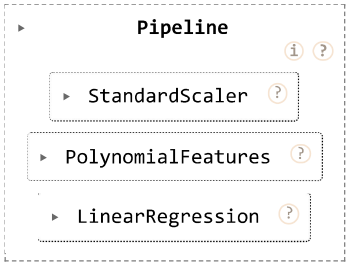
```
0.8970620036125183
```

```
#testing the model
pipe_pred = model.predict(X_test)
r2_score(Y_test, pipe_pred)
```

```
0.8970620036125183
```

Using pipeline to combine the transformers and estimators and fit the model

```
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline

input = [('scale',StandardScaler()),('polynomial', PolynomialFeatures(degree=2)),('model',LinearRegression())]
pipe = Pipeline(input)
pipe
```




```
#training the model
pipe.fit(X_train,Y_train)
pipe.score(X_test,Y_test)
```

```
0.8243963806325882
```

```
from sklearn.metrics import r2_score

#testing the model
pipe_pred = pipe.predict(X_test)
r2_score(Y_test, pipe_pred)
```

```
0.8243963806325882
```

Ridge Regression

```
from sklearn.linear_model import Ridge
Ridgemodel = Ridge(alpha = 0.001)
Ridgemodel
```

```
Ridge
Ridge(alpha=0.001)
```

```
# training the model
Ridgemodel.fit(X_train,Y_train)
Ridgemodel.score(X_test,Y_test)

#testing the model
r_pred = Ridgemodel.predict(X_test)
r2_score(Y_test,r_pred)
```

```
/usr/local/lib/python3.12/dist-packages/scipy/_lib/_util.py:1233: LinAlgWarning: Ill-conditioned matrix (rcond=5.
  return f(*arrays, *other_args, **kwargs)
0.7178709138577588
```

Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100, random_state=0)
regressor
```

```
RandomForestRegressor
RandomForestRegressor(random_state=0)
```

```
# training the model
regressor.fit(X_train,Y_train)
regressor.score(X_test,Y_test)
```

```
0.8758830375179375
```

```
#testing the model
yhat = regressor.predict(X_test)
r2_score(Y_test,yhat)
```

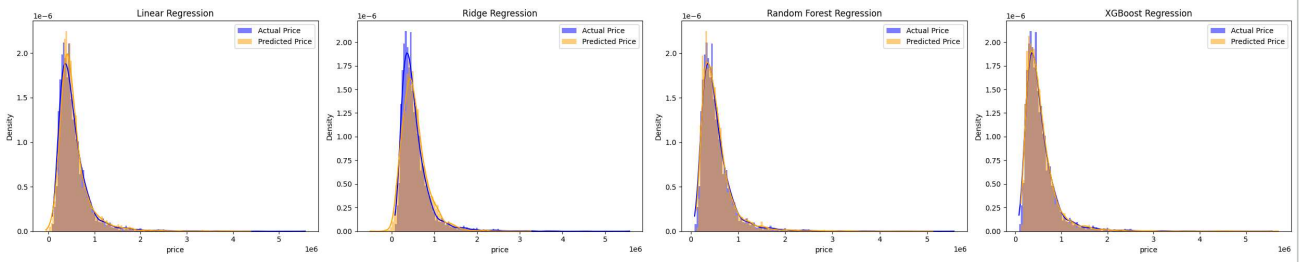
```
0.8758830375179375
```

Model Evalution

Distribution plot from the models predictions and the actual values

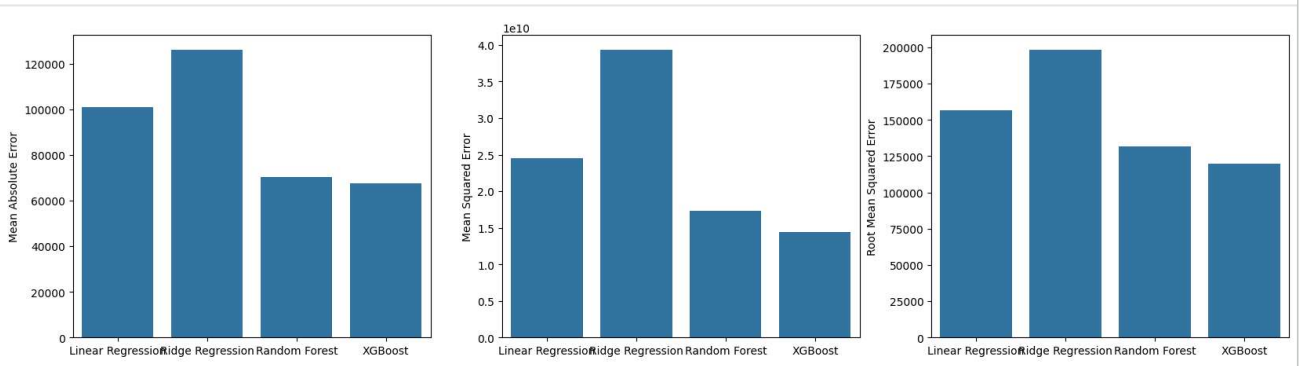
```
# displot of the actual price and predicted price for all models
fig, ax = plt.subplots(1,4,figsize=(25,5))
sns.histplot(Y_test,ax=ax[0], color='blue', label='Actual Price', kde=True, stat='density', linewidth=0)
sns.histplot(pipe_pred,ax=ax[0], color='orange', label='Predicted Price', kde=True, stat='density', linewidth=0)
sns.histplot(Y_test,ax=ax[1], color='blue', label='Actual Price', kde=True, stat='density', linewidth=0)
sns.histplot(r_pred,ax=ax[1], color='orange', label='Predicted Price', kde=True, stat='density', linewidth=0)
sns.histplot(Y_test,ax=ax[2], color='blue', label='Actual Price', kde=True, stat='density', linewidth=0)
sns.histplot(yhat,ax=ax[2], color='orange', label='Predicted Price', kde=True, stat='density', linewidth=0)
sns.histplot(Y_test,ax=ax[3], color='blue', label='Actual Price', kde=True, stat='density', linewidth=0)
sns.histplot(test_data_prediction,ax=ax[3], color='orange', label='Predicted Price', kde=True, stat='density', l

# legends
ax[0].legend()
ax[1].legend()
ax[2].legend()
ax[3].legend()

#model name as title
ax[0].set_title('Linear Regression')
ax[1].set_title('Ridge Regression')
```

```
ax[2].set_title('Random Forest Regression')
ax[3].set_title('XGBoost Regression')

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



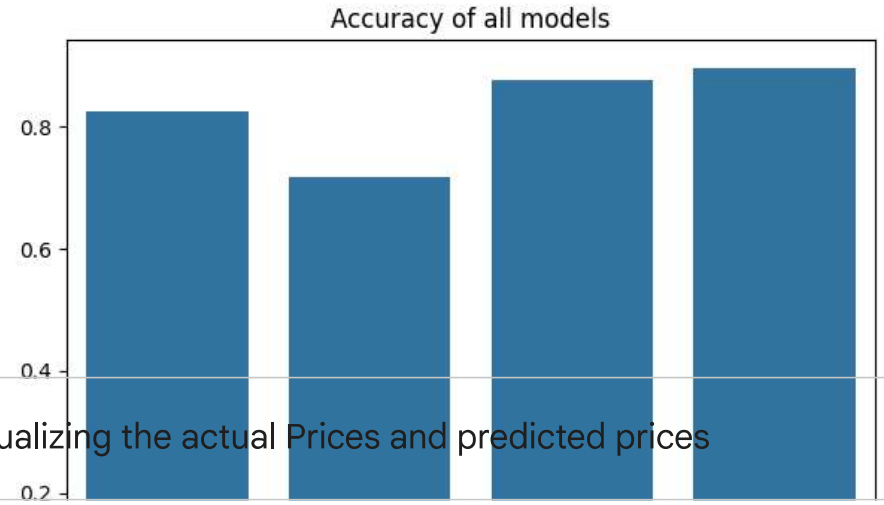
Error Evaluation

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

#plot the graph to compare mae, mse, rmse for all models
fig, ax = plt.subplots(1,3,figsize=(20,5))
sns.barplot(x=['Linear Regression','Ridge Regression','Random Forest', 'XGBoost'],y=[mean_absolute_error(Y_test,
sns.barplot(x=['Linear Regression','Ridge Regression','Random Forest', 'XGBoost'],y=[mean_squared_error(Y_test,p
sns.barplot(x=['Linear Regression','Ridge Regression','Random Forest', 'XGBoost'],y=[np.sqrt(mean_squared_error(
# label for the graph
ax[0].set_ylabel('Mean Absolute Error')
ax[1].set_ylabel('Mean Squared Error')
ax[2].set_ylabel('Root Mean Squared Error')
plt.show()
```



Accuracy Evaluation

```
# plot accuracy of all models in the same graph
fig, ax = plt.subplots(figsize=(7,5))
sns.barplot(x=['Linear Regression','Ridge Regression','Random Forest Regression', 'XGBoost'],y=[metrics.r2_score
ax.set_title('Accuracy of all models')
plt.show()
```



Visualizing the actual Prices and predicted prices

```
plt.scatter(Y_train, training_data_prediction)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Price vs Predicted Price")
plt.show()
```


```
# accuracy for prediction on test data
test_data_prediction = model.predict(X_test)
```

```
# R squared error
score_1 = metrics.r2_score(Y_test, test_data_prediction)

# Mean Absolute Error
score_2 = metrics.mean_absolute_error(Y_test, test_data_prediction)

print("R squared error : ", score_1)
print('Mean Absolute Error : ', score_2)
```

```
R squared error :  0.8970620036125183
Mean Absolute Error :  67419.4765625
```