Palestine Technical University – Kadoorie
College of Engineering and Technology
Department of Computer Systems Engineering

Course name:

Artificial Intelligence

Project title:

PREDICT HOUSE PRICE FOR HOUSES IN KING COUNTRY, USA DATASET USING ML ALGORITHMS

By:

Tasneem Ghazal – 202010412

Supervisor:

Dr. Osama Hamed

Tulkarm, Palestine

26 Dec. 2023

Introduction

Machine learning (ML) is a subdomain of artificial intelligence (AI) that focuses on developing systems that learn—or improve performance—based on the data they ingest. Artificial intelligence is a broad word that refers to systems or machines that resemble human intelligence. Machine learning and AI are frequently discussed together, and the terms are occasionally used interchangeably, although they do not signify the same thing. A crucial distinction is that, while all machine learning is AI, not all AI is machine learning.

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that makes it more similar to humans: The ability to learn. Machine learning is actively being used today, perhaps in many more places than one would expect.

Machine learning algorithms are computational models that allow computers to understand patterns and forecast or make judgments based on data without the need for explicit programming. These algorithms form the foundation of modern artificial intelligence and are used in a wide range of applications, including image and speech recognition, natural language processing, recommendation systems, fraud detection, autonomous cars etc.

In this project we will use machine learning algorithms to predict house prices, like DecisionTreeRegressor and XGBoost.

Decision Tree Regression:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

XGBoost Algorithm:

XGBoost is a robust machine-learning algorithm that can help you understand your data and make better decisions.

XGBoost is an implementation of gradient-boosting decision trees. It has been used by data scientists and researchers worldwide to optimize their machine-learning models.

About Dataset

Abstract: House Sales in King County, USA.

Dataset Information: This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015.

Dataset Attributes:

- **id** : Unique notation for each house sold (primary key of the dataset)
- **date**: Date house was sold
- **price**: Price of the house (our prediction target)
- **bedrooms**: Number of bedrooms
- **bathrooms**: Number of bathrooms
- **sqft_living**: Square footage of the home
- **sqft_lot**: Square footage of the lot
- **floors** :Total floors (levels) in house
- **waterfront** :House which has a view to a waterfront
- **view**: boolean feature (True (1) if the house has been viewed, False (0) if the house has not been viewed)
- **condition** :How good the condition is overall
- **grade**: overall grade given to the housing unit, based on King County grading system
- **sqft_above** : Square footage of house apart from basement
- **sqft_basement:** Square footage of the basement
- **yr_built** : year the house was built
- **yr_renovated** : Year when the house was renovated
- **zipcode**: Zip code
- **lat**: Latitude coordinate
- **long**: Longitude coordinate
- **sqft_living15** : Living room area in 2015(implies-- some renovations) This might or might not have affected the lotsize area
- **sqft_lot15** : LotSize area in 2015(implies-- some renovations)

Dataset link from Kaggle

Data Understanding

1- import required libraries.

Data Understanding

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor,plot_tree
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score
import xgboost as xgb
import graphviz
import seaborn as sns
```

2- Read data from csv file.

The dataset consists of 21613 rows and 21 columns.

```
df = pd.read_csv("https://raw.githubusercontent.com/mGalarnyk/Tutorial_Data/master/King_County/kingCountyHouseData.csv")
```

```
df
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_l
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530	0	2
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	0	0	...	8	2310	0	2
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	0	0	...	7	1020	0	2
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600	0	2
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	0	0	...	7	1020	0	2

21613 rows × 21 columns

3- Use head(10) to display the first 10 rows.

```
df.head(10)
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_buil
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987
5	7237550310	20140512T000000	1225000.0	4	4.50	5420	101930	1.0	0	0	...	11	3890	1530	2001
6	1321400060	20140627T000000	257500.0	3	2.25	1715	6819	2.0	0	0	...	7	1715	0	1995
7	2008000270	20150115T000000	291850.0	3	1.50	1060	9711	1.0	0	0	...	7	1060	0	1963
8	2414600126	20150415T000000	229500.0	3	1.00	1780	7470	1.0	0	0	...	7	1050	730	1960
9	3793500160	20150312T000000	323000.0	3	2.50	1890	6560	2.0	0	0	...	7	1890	0	2003

10 rows × 21 columns

4- Use describe() to get some statistical measures.

```
df.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	g
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.00
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.65
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.17
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.00
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.00
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.00
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.00
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.00

5- Use isnull().sum() to count the null values in each column.

```
df.isnull().sum()
```

```
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
sqft_living15    0
sqft_lot15       0
dtype: int64
```

6- Use dtypes to get the data type of each column.

```
df.dtypes
```

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

7- Use info() method prints information about the DataFrame.

The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

```
df.info()
```

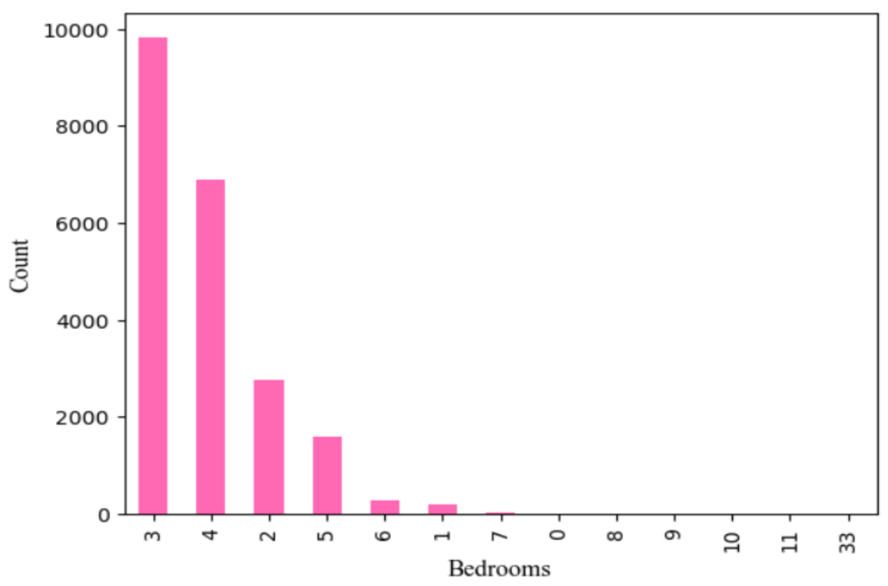
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          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
```

8- Count the occurrence of each value in bedrooms column.

```
room_count=df["bedrooms"].value_counts()
```

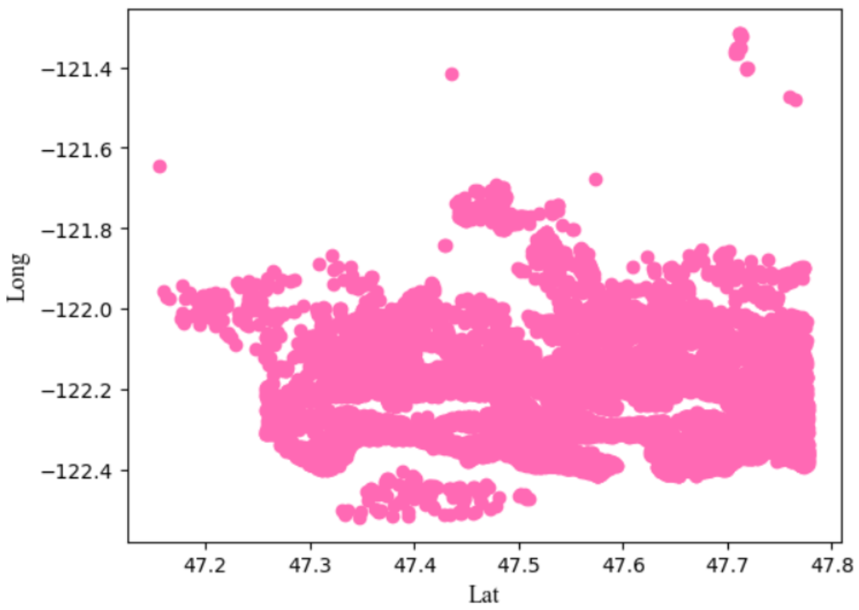
9- Plot the room_count

```
room_count.plot(kind="bar",color="hotpink")
plt.ylabel("Count",fontname=fontFamily ,fontsize=labelsFontSize)
plt.xlabel("Bedrooms",fontname=fontFamily,fontsize=labelsFontSize)
plt.show()
```



10- Scatter plot for latitude and longitude coordinates.

```
plt.scatter(df["lat"],df["long"],color="hotpink")
plt.xlabel("Lat",fontname = fontFamily , fontsize = labelsFontSize)
plt.ylabel("Long",fontname = fontFamily , fontsize = labelsFontSize)
plt.show()
```



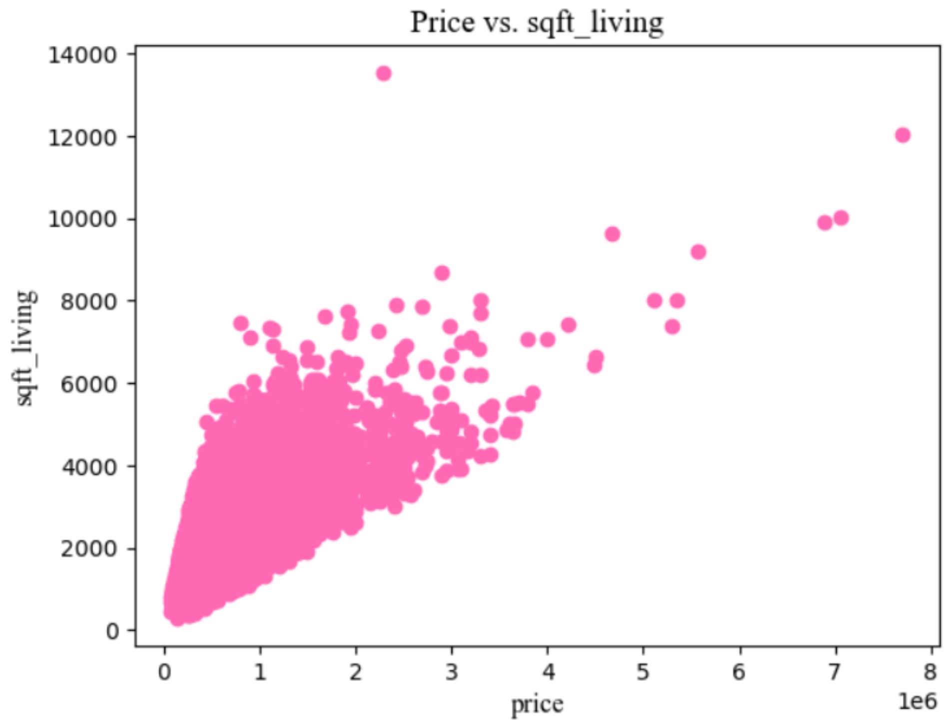
11- Define a function to plot a scatter plot for price vs. a column passing to it.

```
def scatter_plt(column):
    plt.scatter(df["price"],df[column],color="hotpink")
    plt.xlabel("price", fontname = fontFamily , fontsize = labelsFontSize)
    plt.ylabel(column,fontname = fontFamily , fontsize = labelsFontSize)
    plt.title(f"Price vs. {column}" , fontname = fontFamily , fontsize = titlesFontSize)
    plt.show()
```

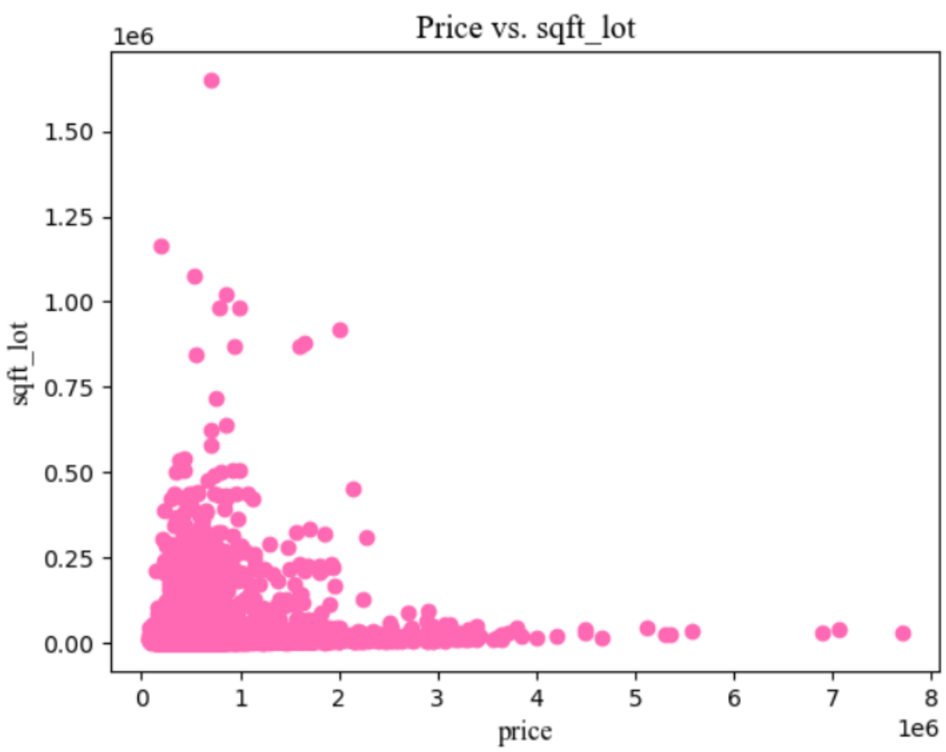
12- Call the function with different arguments.

```
scatter_plt("bedrooms")
```

```
scatter_plt("sqft_living")
```

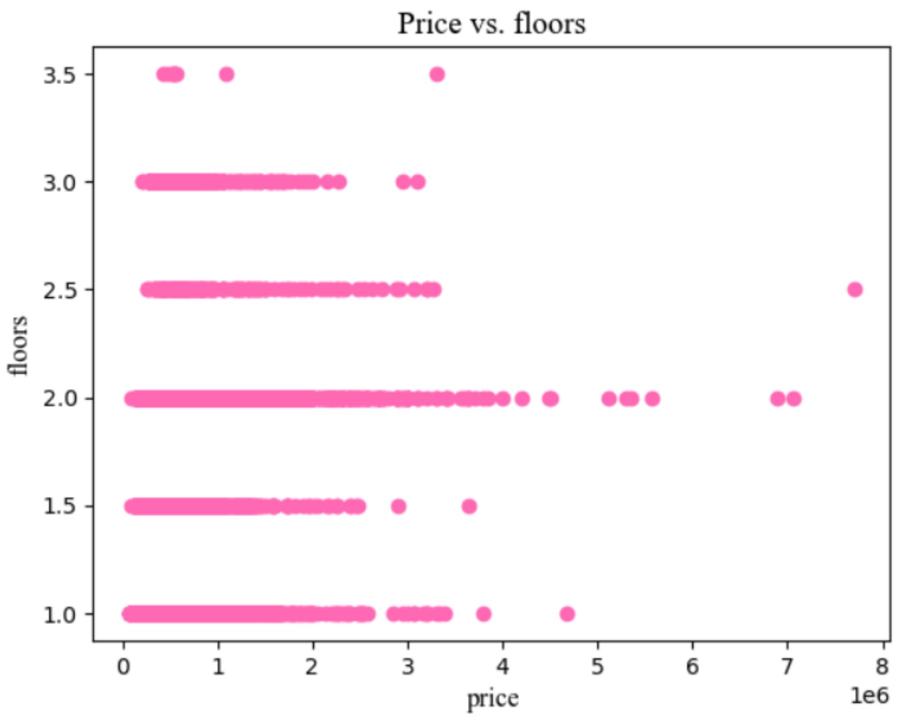



```
scatter_plt("sqft_lot")
```



```
scatter_plt("bathrooms")
```


```
scatter_plt("floors")
```



13- Define a function to find the correlation between price and other attributes.

```python
def correlation(column):
    return df["price"].corr(df[column])
```

14- Call the function with different arguments.

```python
correlation("bedrooms")
```

```
0.3083495981456382
```

```python
correlation("bathrooms")
```

```
0.5251375054139615
```

```python
correlation("sqft_living")
```

```
0.7020350546118004
```

```python
correlation("sqft_lot")
```

```
0.08966086058710013
```

```python
correlation("floors")
```

```
0.25679388755071847
```

Train and Test

1- Create a data frame with only 6 columns appear in columns set and with all rows.

Train and Test

```
columns=["bedrooms","bathrooms","sqft_living","sqft_lot","floors","price"]
```

```
df_predection=df.loc[:,columns]
```

```
df_predection
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	price
0	3	1.00	1180	5650	1.0	221900.0
1	3	2.25	2570	7242	2.0	538000.0
2	2	1.00	770	10000	1.0	180000.0
3	4	3.00	1960	5000	1.0	604000.0
4	3	2.00	1680	8080	1.0	510000.0
...	...	...	...	...	...	...
21608	3	2.50	1530	1131	3.0	360000.0
21609	4	2.50	2310	5813	2.0	400000.0
21610	2	0.75	1020	1350	2.0	402101.0
21611	3	2.50	1600	2388	2.0	400000.0
21612	2	0.75	1020	1076	2.0	325000.0

21613 rows × 6 columns

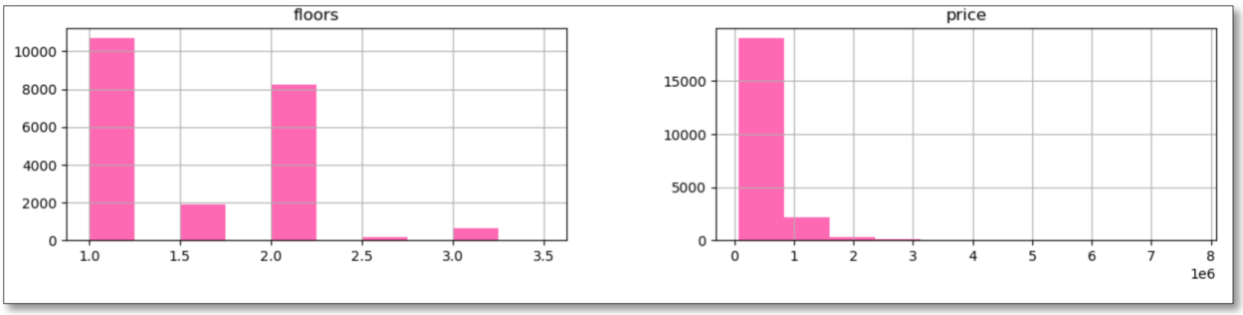
```
df_predection.head()
```

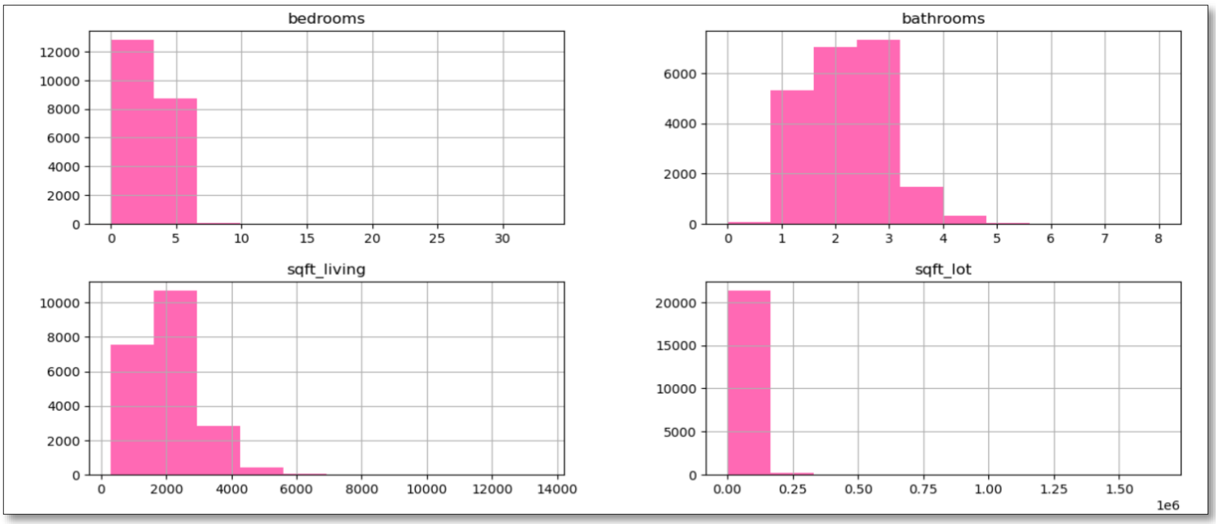
	bedrooms	bathrooms	sqft_living	sqft_lot	floors	price
0	3	1.00	1180	5650	1.0	221900.0
1	3	2.25	2570	7242	2.0	538000.0
2	2	1.00	770	10000	1.0	180000.0
3	4	3.00	1960	5000	1.0	604000.0
4	3	2.00	1680	8080	1.0	510000.0

2- Plot histogram for each attribute

```
df_predection.hist(figsize=(15,10),color="hotpink")
```

```
array([[<Axes: title={'center': 'bedrooms'}>,
        <Axes: title={'center': 'bathrooms'}>],
       [<Axes: title={'center': 'sqft_living'}>,
        <Axes: title={'center': 'sqft_lot'}>],
       [<Axes: title={'center': 'floors'}>,
        <Axes: title={'center': 'price'}>]], dtype=object)
```

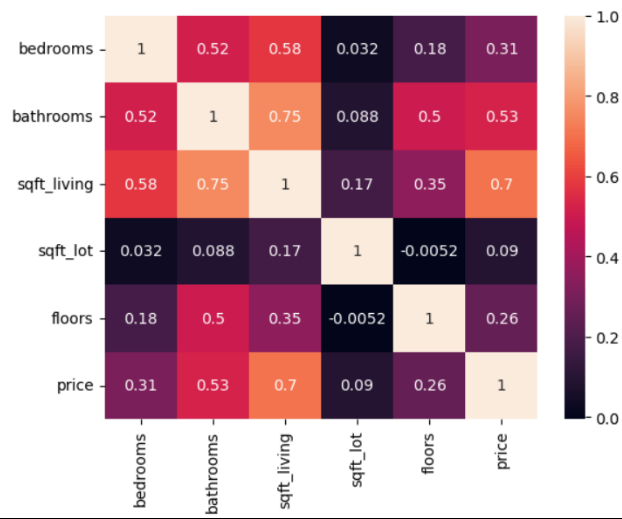




3- Plot heatmap demonstrates the correlation between price and each attribute.

```
sns.heatmap(df_predection.corr(),annot=True)
```

<Axes: >



4- Arrange data into Features and Target

Arrange data into "Features" and "Target"

```
features=["bedrooms","bathrooms","sqft_living","sqft_lot","floors"]
```

```
x = df_predection.loc[:,features]
x
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	3	1.00	1180	5650	1.0
1	3	2.25	2570	7242	2.0
2	2	1.00	770	10000	1.0
3	4	3.00	1960	5000	1.0
4	3	2.00	1680	8080	1.0
...	...	...	...	...	...
21608	3	2.50	1530	1131	3.0
21609	4	2.50	2310	5813	2.0
21610	2	0.75	1020	1350	2.0
21611	3	2.50	1600	2388	2.0
21612	2	0.75	1020	1076	2.0

21613 rows × 5 columns

```
y = df_predection.loc[:,["price"]]
y
```

	price
0	221900.0
1	538000.0
2	180000.0
3	604000.0
4	510000.0
...	...
21608	360000.0
21609	400000.0
21610	402101.0
21611	400000.0
21612	325000.0

21613 rows × 1 columns

5- Split data into train and test using train_test_split.

```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state = 0 )
```

```
X.shape
```

```
(21613, 5)
```

```
y.shape
```

```
(21613, 1)
```

```
X_train.shape
```

```
(16209, 5)
```

```
X_test.shape
```

```
(5404, 5)
```

```
y_train.shape
```

```
(16209, 1)
```

```
y_test.shape
```

```
(5404, 1)
```

Prediction

1- **Predict labels on unseen test data using DecisionTreeRegressor Algorithm.**

Make an instance of DecisionTreeRegressor

```
reg = DecisionTreeRegressor(max_depth = 5, random_state = 0)
```

Train the model on the data

```
reg.fit(X_train,y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, random_state=0)
```

Predict labels on unseen test data using DecisionTreeRegressor

```
# predict multiple observations/houses
reg.predict(X_test[0:10])
```

```
array([ 417038.32831325, 1016098.95714286,  417038.32831325,
        417038.32831325,  739397.58833333,  474303.437014  ,
        393266.79097299,  555258.34883721,  555258.34883721,
       1342344.828125  ])
```

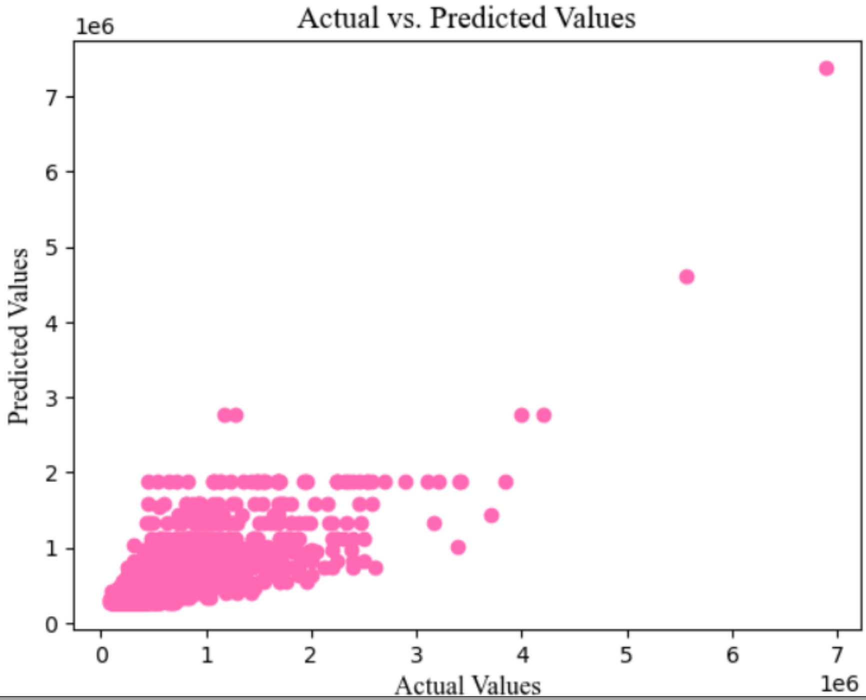
```
# prdict one observation/house
reg.predict(X_test.iloc[0].values.reshape(1,-1))
```

```
C:\Users\DELL G3\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature names, but Decisio
nTreeRegressor was fitted with feature names
  warnings.warn(
```

```
array([417038.32831325])
```

2- Plot a scatter plot between y_predict and y_test.

```
plt.scatter(y_test, y_predict,color="hotpink")
plt.xlabel("Actual Values",fontname="Times New Roman", fontsize=12)
plt.ylabel("Predicted Values",fontname="Times New Roman",fontsize=12)
plt.title("Actual vs. Predicted Values", fontname="Times New Roman" ,fontsize=14)
plt.show()
```

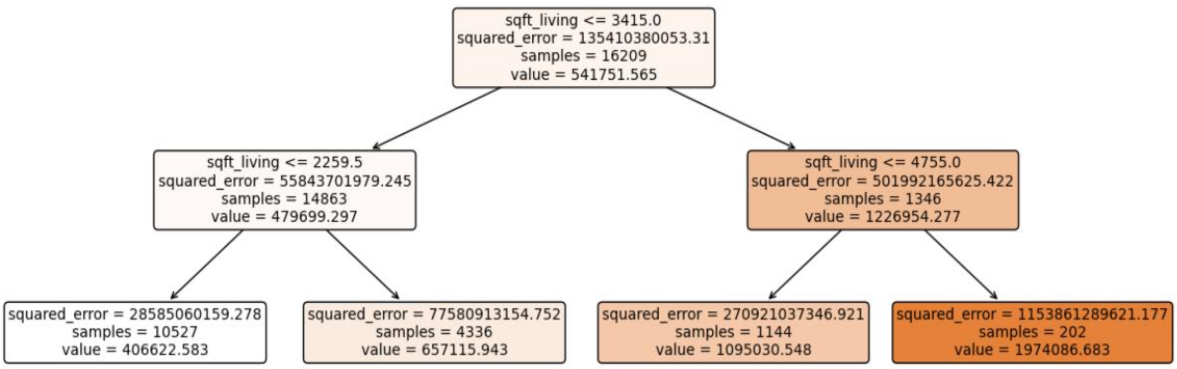


3- Plot the decision tree with max depth = 2.

```
# max_depth=2
reg_2 = DecisionTreeRegressor(max_depth = 2, random_state = 0)
reg_2.fit(X_train,y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=2, random_state=0)
```

```
plt.figure(figsize=(15, 5))
plot_tree(reg_2, filled=True, feature_names=X.columns, rounded=True, fontsize=10)
plt.show()
```



4- Predict labels on unseen test data using XGBRegressor.

```
xg_reg = xgb.XGBRegressor(max_depth=5, random_state=0)
```

```
xg_reg.fit(X_train, y_train)
```

```
XGBRegressor
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=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=5, 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, random_state=0, ...)
```

```
y_pred = xg_reg.predict(X_test)
```

```
xg_reg.predict(X_test[0:10])
```

```
array([ 435814.25,  996494.7 ,  469985.84,  417881.8 ,  689574.94,
        356917.53,  384925.25,  589488.8 ,  546086.44, 1519243.9 ],
      dtype=float32)
```

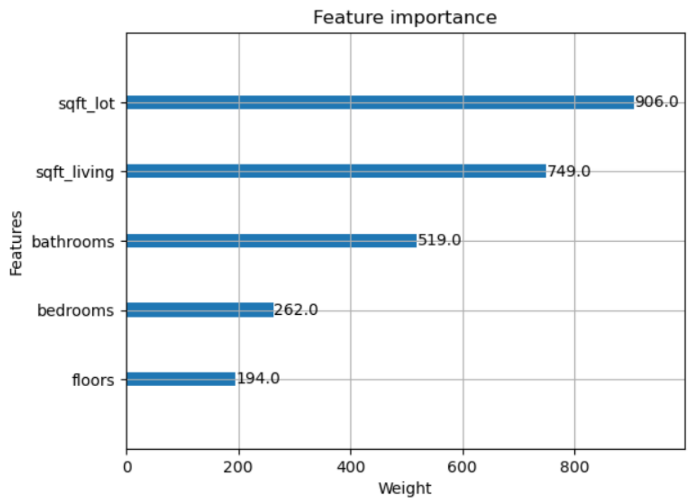
```
xg_reg.predict(X_test.iloc[0].values.reshape(1,-1))
```

```
array([435814.25], dtype=float32)
```

5- Show the importance of each attribute.

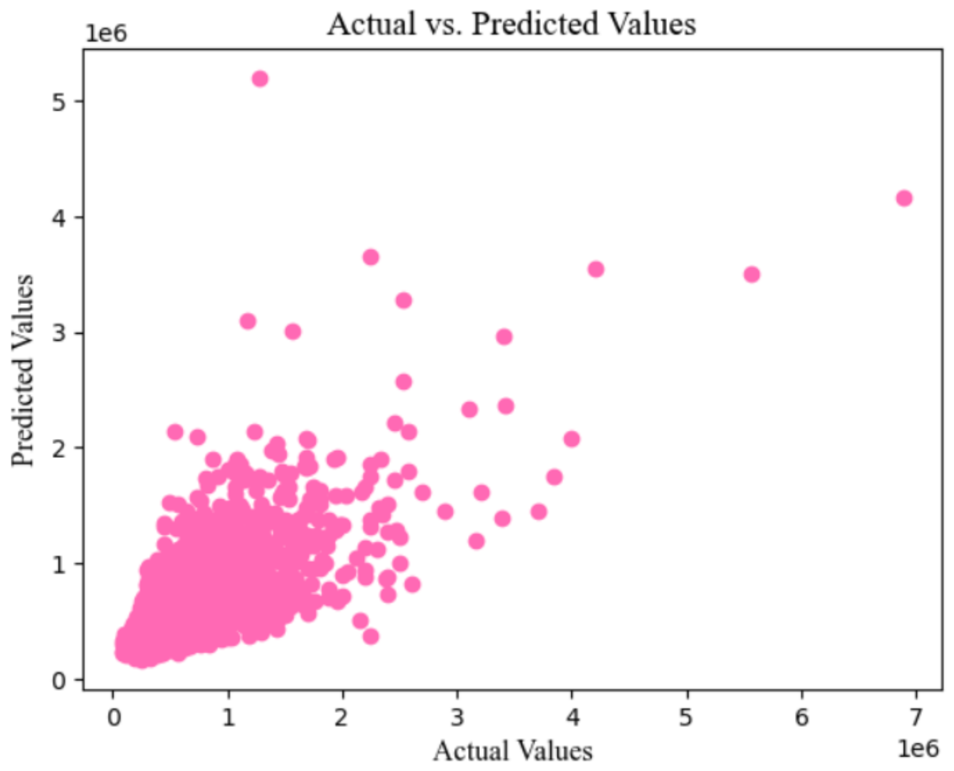
```
#show the importance of each attribute
plt.figure(figsize=(10, 6))
xgb.plot_importance(xg_reg, importance_type='weight', xlabel='Weight')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



6- **Plot a scatter plot between y_predict and y_test.**

```
plt.scatter(y_test, y_pred,color="hotpink")
plt.xlabel("Actual Values",fontname="Times New Roman", fontsize=12)
plt.ylabel("Predicted Values",fontname="Times New Roman",fontsize=12)
plt.title("Actual vs. Predicted Values", fontname="Times New Roman" ,fontsize=14)
plt.show()
```



Evaluate the results using different evaluation metrics

- **For DecisionTreeRegressor.**

```
#R^2
r2_DesReg = reg.score(X_test, y_test)
r2_DesReg
```

```
0.5558073822490773
```

```
y_predict = reg.predict(X_test)
```

```
#mean_squared_error
mean_squared_DesReg = mean_squared_error (y_test,y_predict)
mean_squared_DesReg
```

```
59006808469.74107
```

```
#mean_absolute_error
mean_abs_DesReg = mean_absolute_error (y_test,y_predict)
mean_abs_DesReg
```

```
159699.5412733098
```

- **For XGBRegressor.**

```
#mean_squared_error
mean_squared_xgb = mean_squared_error (y_test,y_pred)
mean_squared_xgb

58863436630.214516

#mean_abolute_error
mean_abs_xgb = mean_absolute_error(y_test, y_pred)
mean_abs_xgb

150507.71617667467

#R^2
r2_xgb = r2_score(y_test, y_pred)
r2_xgb

0.5568866596132094
```

Discuss the results reported from both algorithms.

We note that R^2 in the first algorithm is less than the second, and as for the other two metrics, they are less in the second algorithm than the first, so the second algorithm can be preferred.

In general, If simplicity and interpretability are critical, and you have a relatively simple dataset, a Decision Tree might be sufficient.

If you are dealing with a more complex dataset, want higher predictive accuracy, and are willing to invest in hyperparameter tuning, XGBoost is often a strong choice.

References:

- https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article
- https://www.geeksforgeeks.org/machine-learning/
- https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/
- https://www.kaggle.com/datasets/harlfoxem/housesalesprediction
- https://builtin.com/data-science/train-test-split
- raw.githubusercontent.com/mGalarnyk/Tutorial_Data/master/King_County/kingCountyHouseData.csv