House Sales in King County, USA

Submitted on February 20, 2022

Shareable Link

PROMPT

Question 1) Display the data types of each column using the attribute *dtypes*, then take a screenshot and submit it, include your code in the image.

Display the data types of each column

Display the data types of each column using the function dtypes,

```
df.dtypes
```

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

df.dtypes

RUBRIC

1 a) Does the assignment use the data attribute **dtypes** to get the data type and display the following image

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

0 points

No

1 point

LJ

Yes

PROMPT

Question 2) Drop the columns *"id"* and *"Unnamed: 0*" from axis 1 using the method *drop()*, then use the method *describe()* to obtain a statistical summary of the data. Take a screenshot and submit it, make sure the *inplace* parameter is set to *True.* Your output should look like this:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_bas
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.0
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	1788.390691	291.5
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	828.090978	442.5
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	290.000000	0.0
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	1190.000000	0.0
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	1560.000000	0.0
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	2210.000000	560.0
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	9410.000000	4820.0

Drop the columns "id" and "Unnamed: 0" from axis 1

Drop the columns `"id"` and `"Unnamed: 0"` from axis 1 using the method `drop()`, then use the method `describe()` to obtain a statistical summary of the data. Take a screenshot and submit it, make sure the `inplace` parameter is set to `True`

```
[9] df.drop(["Unnamed: 0","id"], axis = 1, inplace = True)
    df.describe()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	1788.390691	291.509045
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	828.090978	442.575043
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	290.000000	0.000000
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	1190.000000	0.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	1560.000000	0.000000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	2210.000000	560.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	9410.000000	4820.000000

df.drop(["Unnamed: 0","id"], axis = 1, inplace = True) df.describe()

RUBRIC

Question 2) Does the assignment drop the colunms *"id"* and *"Unnamed: 0"*

use the method **describe()** to obtain a statistical summary of the dataframe and produce the result. Note the missing collumns

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_bas
count	2.161300e+04	21600.000000	21603.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	21613.0
mean	5.400881e+05	3.372870	2.115736	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	1788.390691	291.5
std	3.671272e+05	0.926657	0.768996	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	828.090978	442.5
min	7.500000e+04	1.000000	0.500000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	290.000000	0.0
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	1190.000000	0.0
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	1560.000000	0.0
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	2210.000000	560.0
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	9410.000000	4820.0

- ○ 0 points

  No

- ◉ **2 points** LJ

  Yes

- ○ 1 point

  Partially complete

PROMPT

Question 3) use the method **value_counts** to count the number of houses with unique floor values, use the method .to_frame() to convert it to a dataframe. Your output should look like this :

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

Y

use the method value_counts to count the number of houses with unique floor values

Use the method `value_counts` to count the number of houses with unique floor values, dataframe.

```
df['floors'].value_counts().to_frame()
```

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

df['floors'].value_counts().to_frame()

RUBRIC

Question 3) does the assignment use the method **value_counts** to count the number of houses with unique floor values, then produce the following plot:

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

0 points

No

1 point

LJ

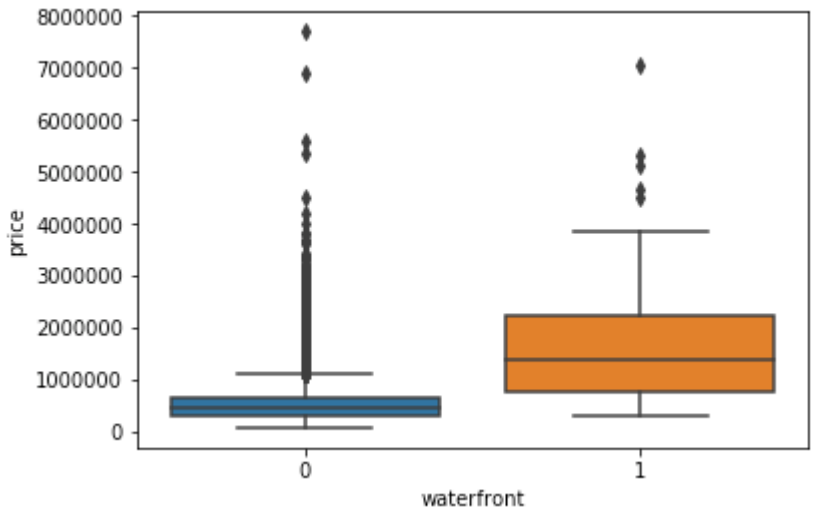
Yes

0.5 points

They used the method **value_counts** on the correct column but did not produce the plot.

PROMPT

Question 4) use the function *boxplot* in the seaborn library to produce a plot that can be used to determine whether houses with a waterfront view or without a waterfront view have more price outliers. Your output should look like this with the code that produced it (the colors may be different) :

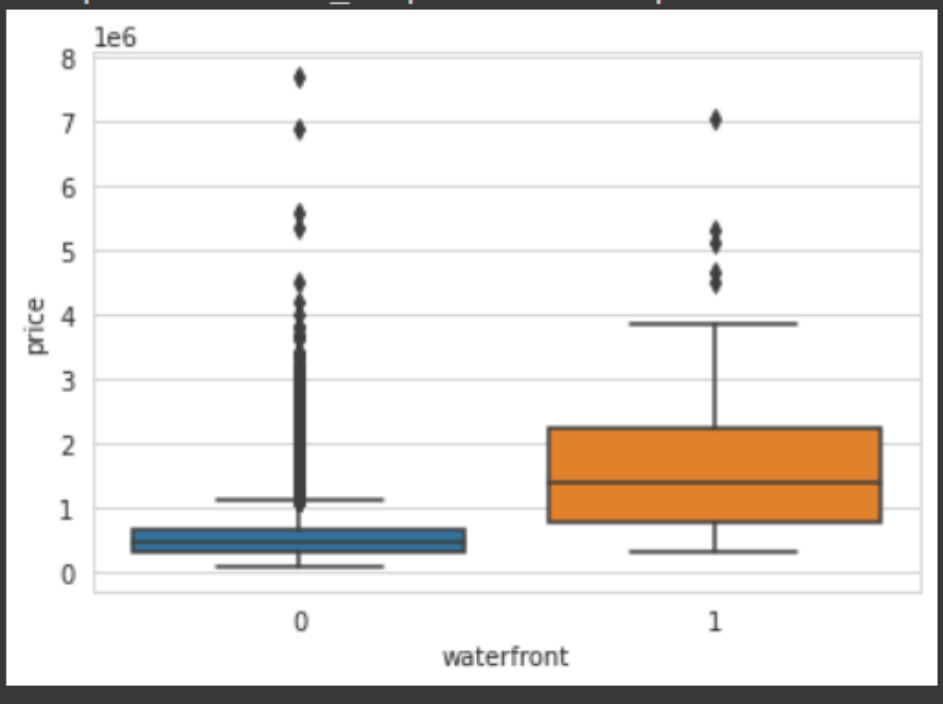


use the function boxplot in the seaborn library to produce a plot

Use the function `boxplot` in the seaborn library to determine whether houses price outliers.

```
sns.set_style("whitegrid")
sns.boxplot(x = 'waterfront', y = 'price', data = df)
```

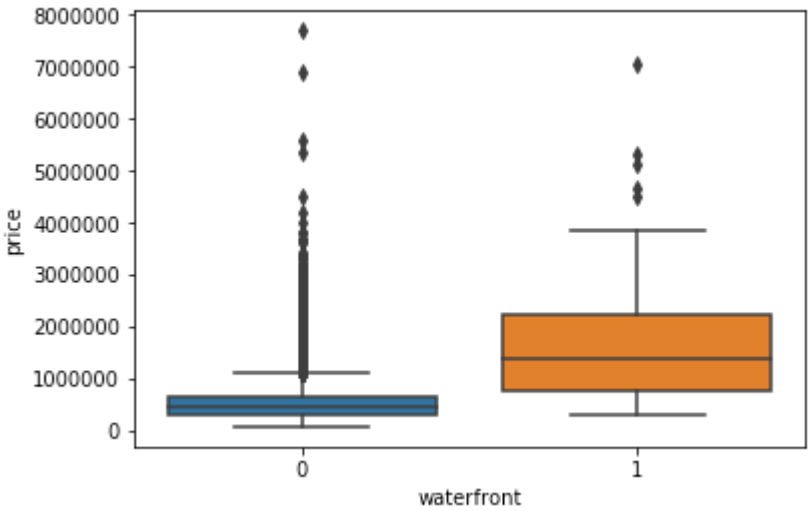
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f970595a090>
```



sns.set_style("whitegrid") sns.boxplot(x = 'waterfront', y = 'price', data = df)

RUBRIC

Question 4) was the following plot produced:



0 points

No

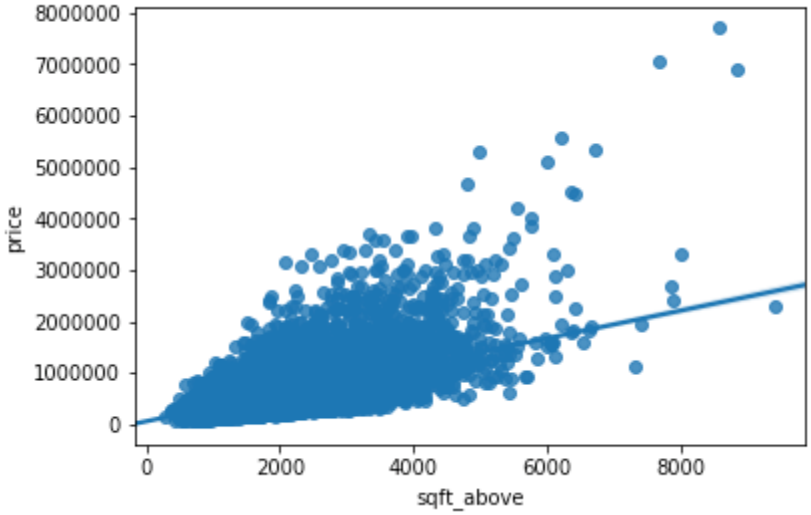
1 point LJ

Yes

PROMPT

Question 5) Use the function *regplot i*n the seaborn library to determine if the feature sqft_above is negatively or positively correlated with price. Take a screenshot of the plot and the code used to generate it.

Your output should look like this with the code that produced it :

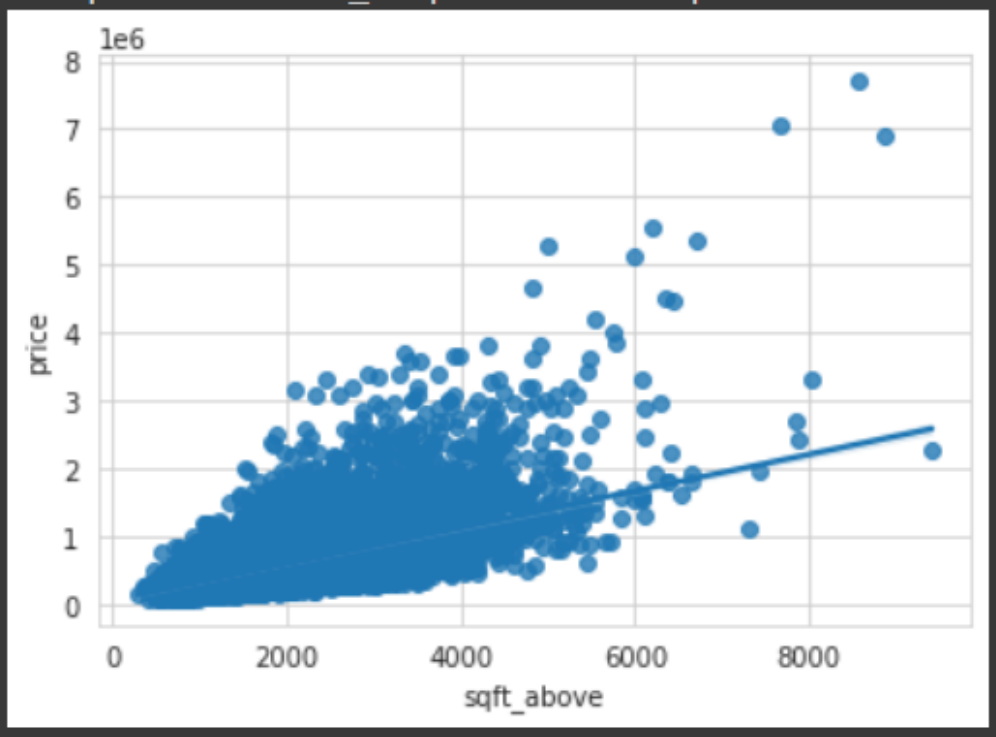


Use the function regplot in the seaborn library to determine

Use the function `regplot` in the seaborn library to determine if the feature

```
sns.regplot(x = 'sqft_above', y = 'price', data = df)
```

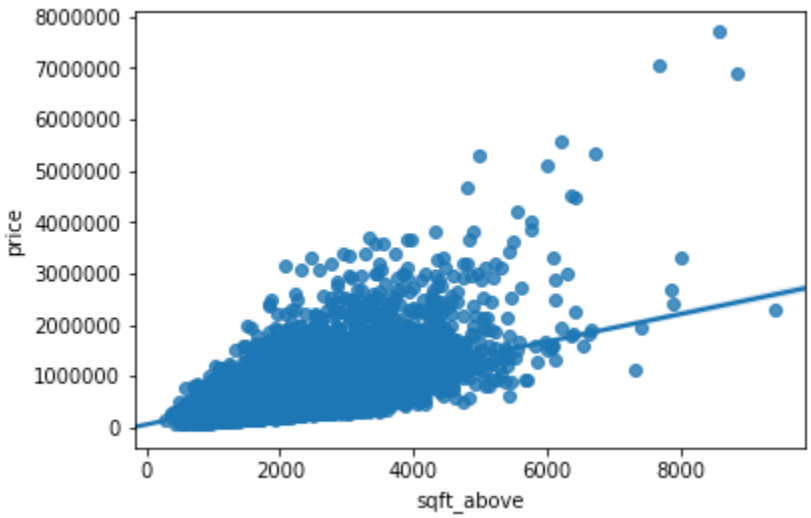
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9705404c10>
```



sns.regplot(x = 'sqft_above', y = 'price', data = df)

RUBRIC

Question 5) Was the following plot produced?



0 points

No

1 point LJ

Yes

PROMPT

Question 6) Fit a linear regression model to predict the price using the feature 'sqft_living' then calculate the R^2. Take a screenshot of your code and the value of the R^2.

Fit a linear regression model to predict the price

Fit a linear regression model to predict the `'price'` using the feature `'sqft_living'` and the value of the R^2.

```
[25] X = df[['sqft_living']]
     Y = df['price']
     lm.fit(X,Y)
     lm.score(X,Y)

     0.4928532179037931
```

X = df[['sqft_living']] Y = df['price'] lm.fit(X,Y) lm.score(X,Y)

RUBRIC

Question 6) Was a linear regression model fit and a R^2 of approximately 0.49285 calculated?

0 points

No

2 points LJ

Yes

1 point

Partially complete

PROMPT

Question 7) Fit a linear regression model to predict the 'price' using the list of features:

- "floors"
- "waterfront"
- "lat"
- "bedrooms"
- "sqft_basement"
- "view"
- "bathrooms"
- "sqft_living15"
- "sqft_above"
- "grade"
- "sqft_living"

The calculate the R^2. Take a screenshot of your code and the value of the R^2.

Fit a linear regression model to predict the 'price' using the list of features

Fit a linear regression model to predict the `'price'` using the list of features:

```
[18] features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view" ,"bathrooms","sqft_living15","sqft_above","grade","sqft_living"]
```

Then calculate the R^2. Take a screenshot of your code.

```
X = df[features]
Y = df['price']
lm.fit(X,Y)
lm.score(X,Y)
```

```
0.6576537175949295
```

X = df[features] Y = df['price'] lm.fit(X,Y) lm.score(X,Y)

RUBRIC

Question 7) Was a linear regression model fit and a R^2 of approximately 0.657695 calculated?

- 0 points

  No

- **2 points** LJ

  Yes

- 1 point

  Partially complete

PROMPT

Question 8) Create a pipeline object that scales the data performs a polynomial transform and fits a linear regression model. Fit the object using the features in the question above, then fit the model and calculate the R^2. Take a screenshot of your code and the R^2.

There are some hints in the notebook

Create a pipeline object that scales the data performs a polynomial transform and fits a linear regression model

Question 8

Use the list to create a pipeline object to predict the 'price', fit the object using the features in the list `features`, and calculate the R^2.

```
[26] pipe=Pipeline(Input)
     pipe.fit(X,Y)
     pipe.score(X,Y)

     0.7513413874579267
```

pipe=Pipeline(Input) pipe.fit(X,Y) pipe.score(X,Y)

RUBRIC

Question 8) Was an R^2 of approximately 0.75133 calculated?

- ○ 0 points

  No

- ◉ **2 points** LJ

  Yes

- ○ 1 point

  Partially complete

PROMPT

Question 9) Create and fit a Ridge regression object using the training data, setting the regularization parameter to 0.1 and calculate the R^2 using the test data. Take a screenshot for your code and the R^2

Create and fit a Ridge regression object

Question 9

Create and fit a Ridge regression object using the training data, set the regularization parameter to 0.1, and calculate the R^2 using the test data.

```
[29] from sklearn.linear_model import Ridge
```

```
[30] RigeModel=Ridge(alpha=0.1)
     RigeModel.fit(x_train, y_train)
     RigeModel.score(x_test, y_test)
```

```
0.6478759163939114
```

RigeModel=Ridge(alpha=0.1) RigeModel.fit(x_train, y_train) RigeModel.score(x_test, y_test)

RUBRIC

Question 9) Was the R^2 of approximately 0.647?

- 0 points

  No

- 2 points

  Yes

- **1 point** LJ

  Partially complete

PROMPT

Question 10) Perform a second order polynomial transform on both the training data and testing data. Create and fit a Ridge regression object using the training data, setting the regularisation parameter to 0.1. Calculate the R^2 utilising the test data provided. Take a screenshot of your code and the R^2.

Perform a second order polynomial transform on both the training data and testing data

Question 10

Perform a second order polynomial transform on both the training data and testing data. Create and fit a Ridge regression object using the training data, set the regularisation parameter to 0.1, and calculate the R^2 utilising the test data provided. Take a screenshot of your code and the R^2.

```
[31] pr = PolynomialFeatures(degree=2)
     x_train_pr = pr.fit_transform(x_train)
     x_test_pr = pr.fit_transform(x_test)

     RigeModel=Ridge(alpha=0.1)
     RigeModel.fit(x_train_pr, y_train)
     RigeModel.score(x_test_pr, y_test)
```

```
0.7002744296219889
```

pr = PolynomialFeatures(degree=2) x_train_pr = pr.fit_transform(x_train) x_test_pr = pr.fit_transform(x_test) RigeModel=Ridge(alpha=0.1) RigeModel.fit(x_train_pr, y_train) RigeModel.score(x_test_pr, y_test)

RUBRIC

Question 10) Was the R^2 of approximately 0.7?

- 0 points

  No

- **1 point** LJ

  Yes

PROMPT

Share the link for your notebook

[House Sales in King County, USA]

I don't have credit card for IBM Cloud. so I did the project on Google Collaboratory. Please feel free to check the link on here: https://colab.research.google.com/drive/17fR_KgeEst5GIUC742bHAjE-LbjZfzKK?usp=sharing

RUBRIC

Did the user share their notebook?

- 0 points

  No

- **3 points** LJ

  Yes