



















[](https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDA0101ENSkillsNetwork971-2022-01-01)

**House Sales in King County, USA**

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

| **Variable** | **Description** |
| --- | --- |
| id | A notation for a house |
| date | Date house was sold |
| price | Price is 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 | Has 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 | Built Year |
| yr\_renovated | Year when 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) |

If you run the lab locally using Anaconda, you can load the correct library and versions by uncommenting the following:

[2]:



*# All Libraries required for this lab are listed below. The libraries pre-installed on Skills Network Labs are commented.*

**!**pip install **-**qy pandas**==**1.3.4 numpy**==**1.21.4 seaborn**==**0.9.0 matplotlib**==**3.5.0 scikit**-**learn**==**0.20.1

*# Note: If your environment doesn't support "!mamba install", use "!pip install"*

---------------------------------------------------------------------------

AttributeError Traceback (most recent call last)

Cell In[2], line 2

**1** # All Libraries required for this lab are listed below. The libraries pre-installed on Skills Network Labs are commented.

----> 2 get\_ipython().system('pip install -qy pandas==1.3.4 numpy==1.21.4 seaborn==0.9.0 matplotlib==3.5.0 scikit-learn==0.20.1')

**3** # Note: If your environment doesn't support "!mamba install", use "!pip install"

File /lib/python3.10/site-packages/IPython/core/interactiveshell.py:2542, in InteractiveShell.system\_piped(self, cmd)

**2537** **raise** **OSError**("Background processes not supported.")

**2539** # we explicitly do NOT return the subprocess status code, because

**2540** # a non-None value would trigger :func:`sys.displayhook` calls.

**2541** # Instead, we store the exit\_code in user\_ns.

-> 2542 self.user\_ns['\_exit\_code'] = system(self.var\_expand(cmd, depth=1))

File /lib/python3.10/site-packages/IPython/utils/\_process\_posix.py:129, in ProcessHandler.system(self, cmd)

**125** enc = DEFAULT\_ENCODING

**127** # Patterns to match on the output, for pexpect. We read input and

**128** # allow either a short timeout or EOF

--> 129 patterns = [pexpect.TIMEOUT, pexpect.EOF]

**130** # the index of the EOF pattern in the list.

**131** # even though we know it's 1, this call means we don't have to worry if

**132** # we change the above list, and forget to change this value:

**133** EOF\_index = patterns.index(pexpect.EOF)

AttributeError: module 'pexpect' has no attribute 'TIMEOUT'

[32]:



*# Surpress warnings:*

**def** warn(**\***args, **\*\***kwargs):

**pass**

**import** warnings

warnings.warn **=** warn

You will require the following libraries:

[33]:



**import** piplite

**await** piplite.install(['pandas','matplotlib','scikit-learn','seaborn', 'numpy'])

​

[34]:



**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**import** seaborn **as** sns

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.preprocessing **import** StandardScaler,PolynomialFeatures

**from** sklearn.linear\_model **import** LinearRegression

**%**matplotlib inline

**Module 1: Importing Data Sets**

The functions below will download the dataset into your browser:

[35]:



**from** pyodide.http **import** pyfetch

​

**async** **def** download(url, filename):

response **=** **await** pyfetch(url)

**if** response.status **==** 200:

**with** open(filename, "wb") **as** f:

f.write(**await** response.bytes())

[36]:



file\_name**=**'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/FinalModule\_Coursera/data/kc\_house\_data\_NaN.csv'

You will need to download the dataset; if you are running locally, please comment out the following code:

[37]:



**await** download(file\_name, "kc\_house\_data\_NaN.csv")

file\_name**=**"kc\_house\_data\_NaN.csv"

Use the Pandas method **read\_csv()** to load the data from the web address.

[38]:



df **=** pd.read\_csv(file\_name)

We use the method head to display the first 5 columns of the dataframe.

[39]:



df.head()

[39]:

|  | **Unnamed: 0** | **id** | **date** | **price** | **bedrooms** | **bathrooms** | **sqft\_living** | **sqft\_lot** | **floors** | **waterfront** | **...** | **grade** | **sqft\_above** | **sqft\_basement** | **yr\_built** | **yr\_renovated** | **zipcode** | **lat** | **long** | **sqft\_living15** | **sqft\_lot15** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 7129300520 | 20141013T000000 | 221900.0 | 3.0 | 1.00 | 1180 | 5650 | 1.0 | 0 | ... | 7 | 1180 | 0 | 1955 | 0 | 98178 | 47.5112 | -122.257 | 1340 | 5650 |
| **1** | 1 | 6414100192 | 20141209T000000 | 538000.0 | 3.0 | 2.25 | 2570 | 7242 | 2.0 | 0 | ... | 7 | 2170 | 400 | 1951 | 1991 | 98125 | 47.7210 | -122.319 | 1690 | 7639 |
| **2** | 2 | 5631500400 | 20150225T000000 | 180000.0 | 2.0 | 1.00 | 770 | 10000 | 1.0 | 0 | ... | 6 | 770 | 0 | 1933 | 0 | 98028 | 47.7379 | -122.233 | 2720 | 8062 |
| **3** | 3 | 2487200875 | 20141209T000000 | 604000.0 | 4.0 | 3.00 | 1960 | 5000 | 1.0 | 0 | ... | 7 | 1050 | 910 | 1965 | 0 | 98136 | 47.5208 | -122.393 | 1360 | 5000 |
| **4** | 4 | 1954400510 | 20150218T000000 | 510000.0 | 3.0 | 2.00 | 1680 | 8080 | 1.0 | 0 | ... | 8 | 1680 | 0 | 1987 | 0 | 98074 | 47.6168 | -122.045 | 1800 | 7503 |

5 rows × 22 columns

**Question 1**

Display the data types of each column using the function dtypes, then take a screenshot and submit it, include your code in the image.

[40]:



*# Display the data types of each column*

print(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

We use the method describe to obtain a statistical summary of the dataframe.

[41]:



df.describe()

[41]:

|  | **Unnamed: 0** | **id** | **price** | **bedrooms** | **bathrooms** | **sqft\_living** | **sqft\_lot** | **floors** | **waterfront** | **view** | **...** | **grade** | **sqft\_above** | **sqft\_basement** | **yr\_built** | **yr\_renovated** | **zipcode** | **lat** | **long** | **sqft\_living15** | **sqft\_lot15** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 21613.00000 | 2.161300e+04 | 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 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 |
| **mean** | 10806.00000 | 4.580302e+09 | 5.400881e+05 | 3.372870 | 2.115736 | 2079.899736 | 1.510697e+04 | 1.494309 | 0.007542 | 0.234303 | ... | 7.656873 | 1788.390691 | 291.509045 | 1971.005136 | 84.402258 | 98077.939805 | 47.560053 | -122.213896 | 1986.552492 | 12768.455652 |
| **std** | 6239.28002 | 2.876566e+09 | 3.671272e+05 | 0.926657 | 0.768996 | 918.440897 | 4.142051e+04 | 0.539989 | 0.086517 | 0.766318 | ... | 1.175459 | 828.090978 | 442.575043 | 29.373411 | 401.679240 | 53.505026 | 0.138564 | 0.140828 | 685.391304 | 27304.179631 |
| **min** | 0.00000 | 1.000102e+06 | 7.500000e+04 | 1.000000 | 0.500000 | 290.000000 | 5.200000e+02 | 1.000000 | 0.000000 | 0.000000 | ... | 1.000000 | 290.000000 | 0.000000 | 1900.000000 | 0.000000 | 98001.000000 | 47.155900 | -122.519000 | 399.000000 | 651.000000 |
| **25%** | 5403.00000 | 2.123049e+09 | 3.219500e+05 | 3.000000 | 1.750000 | 1427.000000 | 5.040000e+03 | 1.000000 | 0.000000 | 0.000000 | ... | 7.000000 | 1190.000000 | 0.000000 | 1951.000000 | 0.000000 | 98033.000000 | 47.471000 | -122.328000 | 1490.000000 | 5100.000000 |
| **50%** | 10806.00000 | 3.904930e+09 | 4.500000e+05 | 3.000000 | 2.250000 | 1910.000000 | 7.618000e+03 | 1.500000 | 0.000000 | 0.000000 | ... | 7.000000 | 1560.000000 | 0.000000 | 1975.000000 | 0.000000 | 98065.000000 | 47.571800 | -122.230000 | 1840.000000 | 7620.000000 |
| **75%** | 16209.00000 | 7.308900e+09 | 6.450000e+05 | 4.000000 | 2.500000 | 2550.000000 | 1.068800e+04 | 2.000000 | 0.000000 | 0.000000 | ... | 8.000000 | 2210.000000 | 560.000000 | 1997.000000 | 0.000000 | 98118.000000 | 47.678000 | -122.125000 | 2360.000000 | 10083.000000 |
| **max** | 21612.00000 | 9.900000e+09 | 7.700000e+06 | 33.000000 | 8.000000 | 13540.000000 | 1.651359e+06 | 3.500000 | 1.000000 | 4.000000 | ... | 13.000000 | 9410.000000 | 4820.000000 | 2015.000000 | 2015.000000 | 98199.000000 | 47.777600 | -121.315000 | 6210.000000 | 871200.000000 |

8 rows × 21 columns

**Module 2: Data Wrangling**

**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

[42]:



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

df.describe()

​

[42]:

|  | **price** | **bedrooms** | **bathrooms** | **sqft\_living** | **sqft\_lot** | **floors** | **waterfront** | **view** | **condition** | **grade** | **sqft\_above** | **sqft\_basement** | **yr\_built** | **yr\_renovated** | **zipcode** | **lat** | **long** | **sqft\_living15** | **sqft\_lot15** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 | 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 | 1971.005136 | 84.402258 | 98077.939805 | 47.560053 | -122.213896 | 1986.552492 | 12768.455652 |
| **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 | 29.373411 | 401.679240 | 53.505026 | 0.138564 | 0.140828 | 685.391304 | 27304.179631 |
| **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 | 1900.000000 | 0.000000 | 98001.000000 | 47.155900 | -122.519000 | 399.000000 | 651.000000 |
| **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 | 1951.000000 | 0.000000 | 98033.000000 | 47.471000 | -122.328000 | 1490.000000 | 5100.000000 |
| **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 | 1975.000000 | 0.000000 | 98065.000000 | 47.571800 | -122.230000 | 1840.000000 | 7620.000000 |
| **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 | 1997.000000 | 0.000000 | 98118.000000 | 47.678000 | -122.125000 | 2360.000000 | 10083.000000 |
| **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 | 2015.000000 | 2015.000000 | 98199.000000 | 47.777600 | -121.315000 | 6210.000000 | 871200.000000 |

We can see we have missing values for the columns  bedrooms and  bathrooms

[43]:



print("number of NaN values for the column bedrooms :", df['bedrooms'].isnull().sum())

print("number of NaN values for the column bathrooms :", df['bathrooms'].isnull().sum())

​

number of NaN values for the column bedrooms : 13

number of NaN values for the column bathrooms : 10

We can replace the missing values of the column 'bedrooms' with the mean of the column 'bedrooms'  using the method replace(). Don't forget to set the inplace parameter to True

[45]:



mean**=**df['bedrooms'].mean()

df['bedrooms'].replace(np.nan,mean, inplace**=True**)

We also replace the missing values of the column 'bathrooms' with the mean of the column 'bathrooms'  using the method replace(). Don't forget to set the  inplace  parameter top  True

[46]:



mean**=**df['bathrooms'].mean()

df['bathrooms'].replace(np.nan,mean, inplace**=True**)

[47]:



print("number of NaN values for the column bedrooms :", df['bedrooms'].isnull().sum())

print("number of NaN values for the column bathrooms :", df['bathrooms'].isnull().sum())

number of NaN values for the column bedrooms : 0

number of NaN values for the column bathrooms : 0

**Module 3: Exploratory Data Analysis**

**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.

[48]:



floor\_counts **=** df['floors'].value\_counts().to\_frame()

floor\_counts

[48]:

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

**Question 4**

Use the function boxplot in the seaborn library to determine whether houses with a waterfront view or without a waterfront view have more price outliers.

[49]:



**import** seaborn **as** sns

​

*# create a new dataframe with only the waterfront and price columns*

waterfront\_price **=** df[['waterfront', 'price']]

​

*# create the boxplot*

sns.boxplot(x**=**'waterfront', y**=**'price', data**=**waterfront\_price)

[49]:

<AxesSubplot:xlabel='waterfront', ylabel='price'>

**Question 5**

Use the function regplot in the seaborn library to determine if the feature sqft\_above is negatively or positively correlated with price.

[50]:



**import** seaborn **as** sns

​

sns.regplot(x**=**'sqft\_above', y**=**'price', data**=**df)

[50]:

<AxesSubplot:xlabel='sqft\_above', ylabel='price'>

We can use the Pandas method corr() to find the feature other than price that is most correlated with price.

[51]:



df.corr()['price'].sort\_values()

[51]:

zipcode -0.053203

long 0.021626

condition 0.036362

yr\_built 0.054012

sqft\_lot15 0.082447

sqft\_lot 0.089661

yr\_renovated 0.126434

floors 0.256794

waterfront 0.266369

lat 0.307003

bedrooms 0.308797

sqft\_basement 0.323816

view 0.397293

bathrooms 0.525738

sqft\_living15 0.585379

sqft\_above 0.605567

grade 0.667434

sqft\_living 0.702035

price 1.000000

Name: price, dtype: float64

**Module 4: Model Development**

We can Fit a linear regression model using the longitude feature 'long' and caculate the R^2.

[52]:



X **=** df[['long']]

Y **=** df['price']

lm **=** LinearRegression()

lm.fit(X,Y)

lm.score(X, Y)

[52]:

0.00046769430149007363

**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.

[53]:



**from** sklearn.linear\_model **import** LinearRegression

​

X **=** df[['sqft\_living']]

Y **=** df['price']

​

lm **=** LinearRegression()

lm.fit(X,Y)

​

print('R^2:', lm.score(X, Y))

R^2: 0.4928532179037931

**Question 7**

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

[54]:



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.

[55]:



**from** sklearn.linear\_model **import** LinearRegression

​

X **=** df[features]

y **=** df['price']

​

lm **=** LinearRegression()

lm.fit(X, y)

​

print("R^2:", lm.score(X, y))

R^2: 0.6576890354915759

**This will help with Question 8**

Create a list of tuples, the first element in the tuple contains the name of the estimator:

'scale'

'polynomial'

'model'

The second element in the tuple contains the model constructor

StandardScaler()

PolynomialFeatures(include\_bias=False)

LinearRegression()

[56]:



Input**=**[('scale',StandardScaler()),('polynomial', PolynomialFeatures(include\_bias**=False**)),('model',LinearRegression())]

**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.

[57]:



**from** sklearn.pipeline **import** Pipeline

**from** sklearn.preprocessing **import** StandardScaler, PolynomialFeatures

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.metrics **import** r2\_score

​

*# Create a pipeline object*

Input **=** [('scale', StandardScaler()), ('polynomial', PolynomialFeatures(include\_bias**=False**)), ('model', LinearRegression())]

pipe **=** Pipeline(Input)

​

*# Fit the pipeline object using the features in the list 'features'*

pipe.fit(X[features], y)

​

*# Calculate the R^2*

r\_squared **=** pipe.score(X[features], y)

print("R-squared:", r\_squared)

R-squared: 0.7512398529081656

**Module 5: Model Evaluation and Refinement**

Import the necessary modules:

[58]:



**from** sklearn.model\_selection **import** cross\_val\_score

**from** sklearn.model\_selection **import** train\_test\_split

print("done")

done

We will split the data into training and testing sets:

[67]:



features **=**["floors", "waterfront","lat" ,"bedrooms" ,"sqft\_basement" ,"view" ,"bathrooms","sqft\_living15","sqft\_above","grade","sqft\_living"]

X **=** df[features]

Y **=** df['price']

​

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(X, Y, test\_size**=**0.15, random\_state**=**1)

​

​

print("number of test samples:", x\_test.shape[0])

print("number of training samples:",x\_train.shape[0])

number of test samples: 3242

number of training samples: 18371

**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.

[68]:



**from** sklearn.linear\_model **import** Ridge

[71]:



**from** sklearn.model\_selection **import** train\_test\_split

​

X **=** df[features]

y **=** df['price']

​

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**0)

[72]:



**from** sklearn.linear\_model **import** Ridge

​

ridge **=** Ridge(alpha**=**0.1)

ridge.fit(X\_train, y\_train)

[72]:

Ridge(alpha=0.1)

[73]:



**from** sklearn.metrics **import** r2\_score

​

y\_pred **=** ridge.predict(X\_test)

r2 **=** r2\_score(y\_test, y\_pred)

​

print("R^2:", r2)

R^2: 0.6513705109371565

**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.

[74]:



**from** sklearn.preprocessing **import** PolynomialFeatures

**from** sklearn.linear\_model **import** Ridge

**from** sklearn.metrics **import** r2\_score

​

poly **=** PolynomialFeatures(degree**=**2)

X\_train\_poly **=** poly.fit\_transform(X\_train)

X\_test\_poly **=** poly.transform(X\_test)

​

ridge **=** Ridge(alpha**=**0.1)

ridge.fit(X\_train\_poly, y\_train)

​

y\_test\_pred **=** ridge.predict(X\_test\_poly)

r2 **=** r2\_score(y\_test, y\_test\_pred)

print("R^2:", r2)

R^2: 0.7161272723631096