AI534 Warm up excercises 0

This is a warm-up assignment (individual) for you to get familiar with some basics:

1. Using google colab and python notebook to complete implememtation assignments
2. Basic packages and functions for working with data, performing simple analysis and plotting.
3. Walk you through some such basic steps for getting an intuitive understanding of what your data looks like, which is the first step to tackling any machine learning problem.

We will use a data set that contains historic data on houses sold between May 2014 to May 2015. Each house in the data set is described by a set of 20 descriptors of the house (referred to as features or attributes, denoted by $\mathbf{x}$ mathematically) and taged with the selling price of the house (referred to as the target variable or label, denoted as y).

Let's get started by importing the necessary packages.

```
!pip install nbconvert > /dev/null 2>&1
!pip install pdfkit > /dev/null 2>&1
!apt-get install -y wkhtmltopdf > /dev/null 2>&1
import os
import pdfkit
import contextlib
import sys
from google.colab import files
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Follow along step 1: accessing and loading the data

First, you need to download the file ia0_train.csv (provided on canvas) to your google drive. To allow the colab to access your google drive, you need to mount Google Drive from your notebook:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
os.chdir('/content/drive/My Drive')
```

Set the path to the data file:

```
file_path='/content/drive/My Drive/AI534/ia0_train.csv' #please use the same path to store your data file to avoid needing modification
```

```
from google.colab import files
import io
```

```
uploaded = files.upload()
```

Choose files ia0_train.csv

- **ia0_train.csv**(text/csv) - 807654 bytes, last modified: 30/09/2024 - 100% done

Now load the csv data into a DataFrame, and take a look to see what it looks like.

```
raw_data = pd.read_csv(io.BytesIO(uploaded['ia0_train (2).csv']))
raw_data
```

	id	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	...	sqft_above	sqft_l
0	7972604355	5/21/2014	3	1.00	1020	7874	1.0	0	0	3	...	1020	
1	8731951130	6/9/2014	3	2.25	2210	8000	2.0	0	0	4	...	2210	
2	7885800740	2/18/2015	4	2.50	2350	5835	2.0	0	0	3	...	2350	
3	4232900940	5/22/2014	3	1.50	1660	4800	2.0	0	0	3	...	1660	
4	3275850190	9/5/2014	3	2.50	2410	9916	2.0	0	0	4	...	2410	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7995	4222500410	2/26/2015	4	1.75	2000	7350	1.0	0	0	3	...	1100	
7996	1150700170	9/26/2014	4	2.25	1870	6693	2.0	0	0	3	...	1870	
7997	1959702045	11/19/2014	2	1.00	1240	5500	1.0	0	0	3	...	1240	
7998	7234601221	10/14/2014	3	1.50	1280	2114	1.5	0	0	3	...	1280	
7999	3275740030	5/7/2014	3	2.25	1770	8165	2.0	0	0	3	...	1770	

8000 rows × 21 columns

```
#you can see the data type for each column
raw_data.dtypes
```

	0
id	int64
date	object
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
price	float64

Follow along step 2: Understanding and preprocessing the data

As you can see from the output of the previous cell, there are 10k examples, each with 21 columns in this csv file. The column 'price' stores the price of the house, which we hope our model can learn to predict. The other columns are considered the input features (or attributes). Before feeding this data to a machine learning algorithm to learn a model, it is always a good idea to examine the features, as **features are not always useful** and also they might be in a format that is not well suited for our learning algorithm to consume. Here are two immediate issues in this regard:

1. The ID feature is a unique identifyer assigned to each example, hence it carries no useful information for generalization and should not be included as a feature for machine learning. You should drop this column from the data before feeding to the learning algorithm.

2. The date feature is currently in the object format, which means it is string. Most of ML algorithms assume numerical inputs, hence we want to change it into a numerical feautre. Here please break the date string into three separate numerical values representing the Month, day and year of sale respectively.

```
#1. drop the ID column
data_without_id = raw_data.drop(columns=['id'])
data_without_id.dtypes
```

	0
date	object
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
price	float64

dtype: object

```
#2. handle the date feature and convert it to datetime
data_without_id['date']=pd.to_datetime(data_without_id['date'], format='%m/%d/%Y')
#extract month, day, and year into separate columns
data_without_id['SaleMonth'] = data_without_id['date'].dt.month
data_without_id['SaleDay'] = data_without_id['date'].dt.day
data_without_id['SaleYear'] = data_without_id['date'].dt.year
#drop the original date column
data_without_id=data_without_id.drop(columns=['date'])
data_without_id.dtypes
```

	0
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
price	float64
SaleMonth	int32
SaleDay	int32
SaleYear	int32

Follow along step 3: check out some specific features

The first thing coming to mind when buying a house is the number of rooms, bedrooms, bathrooms, these are going to be among the most important factors deciding the price of a house. So let's check these features out. Specifically, let's take a look at the statistics of these features.

```
# Group the data by the 'bedrooms' column and calculate statistics for 'price'
bedroom_stats = data_without_id.groupby('bedrooms')['price'].agg(['mean', 'median', 'min', 'max', 'count'])
bedroom_stats
```

bedrooms	mean	median	min	max	count
1	3.340914	3.146	0.89950	12.50	80
2	3.917504	3.700	0.82500	17.00	1035
3	4.667360	4.170	0.82000	38.00	3579
4	6.305826	5.500	1.39000	40.00	2600
5	7.582359	6.190	1.58550	53.50	591
6	8.652208	6.700	2.30000	68.90	95
7	9.530048	5.650	3.10000	24.50	12
8	6.915000	6.915	5.75000	8.08	2
9	7.446663	7.000	5.99999	9.34	3
10	6.600000	6.600	6.60000	6.60	1
11	5.200000	5.200	5.20000	5.20	1
33	6.400000	6.400	6.40000	6.40	1

Next steps: Generate code with bedroom_stats | View recommended plots | New interactive sheet

```
# Group the data by the 'bathrooms' column and calculate statistics for 'price'
```

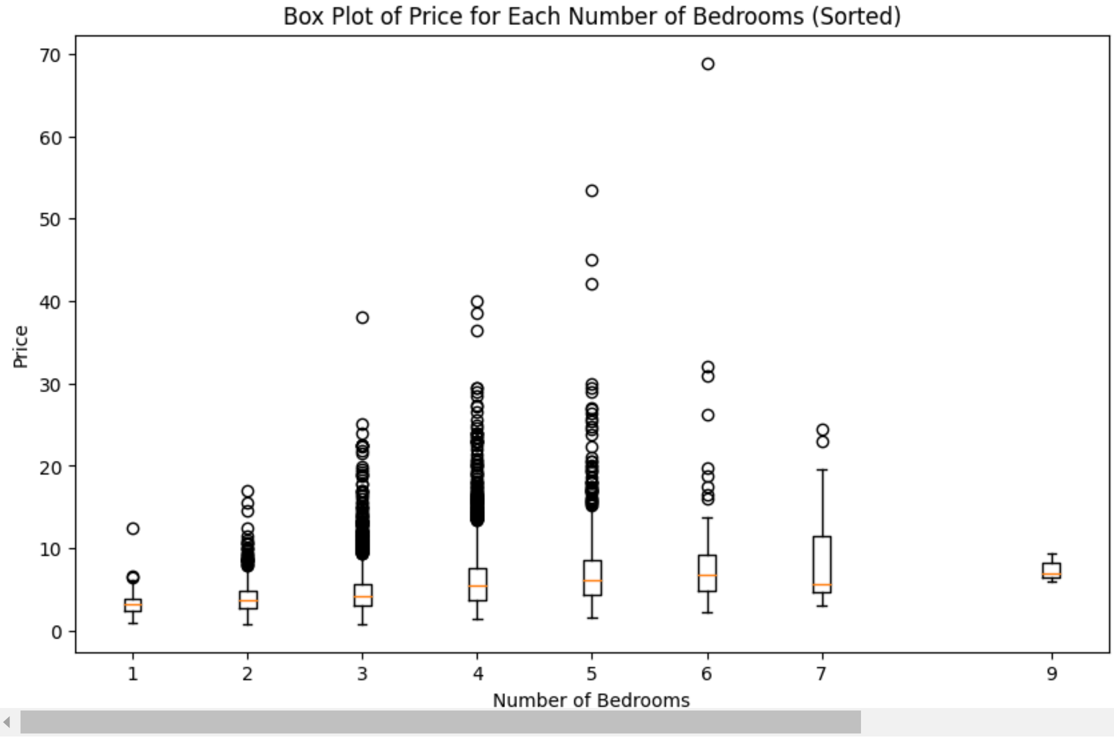
```
bathroom_stats = data_without_id.groupby('bathrooms')['price'].agg(['mean', 'median', 'min', 'max', 'count'])
bathroom_stats
```

bathrooms	mean	median	min	max	count
0.50	2.640000	2.64000	2.5500	2.730	2
0.75	3.218479	2.90000	1.0000	5.621	23
1.00	3.482674	3.24400	0.8200	13.000	1404
1.25	6.156500	3.21950	2.7500	12.500	3
1.50	4.136978	3.75000	1.3400	13.800	542
1.75	4.554531	4.30000	1.2075	15.000	1131
2.00	4.522037	4.10000	1.1500	17.000	723
2.25	5.301349	4.65000	1.6000	24.000	746
2.50	5.557118	5.00000	1.5800	29.000	2000
2.75	6.411762	5.89975	1.9995	19.000	432
3.00	6.937321	5.94866	1.9995	26.800	267
3.25	9.139617	8.25500	1.7600	36.400	236
3.50	9.000317	8.10000	2.4800	29.500	286
3.75	11.952167	11.70000	3.4510	24.800	63
4.00	10.771943	9.37500	2.6500	30.000	47
4.25	16.945345	14.30000	4.9000	38.500	29
4.50	12.339184	10.05000	2.9000	29.500	38
4.75	18.234143	23.00000	5.9900	27.000	7
5.00	17.821667	14.30000	4.8000	53.500	9
5.25	13.650000	13.50000	3.0000	24.600	4
5.50	25.050000	16.00000	9.2500	45.000	5
6.00	42.100000	42.10000	42.1000	42.100	1
6.50	22.400000	22.40000	22.4000	22.400	1
7.75	68.900000	68.90000	68.9000	68.900	1

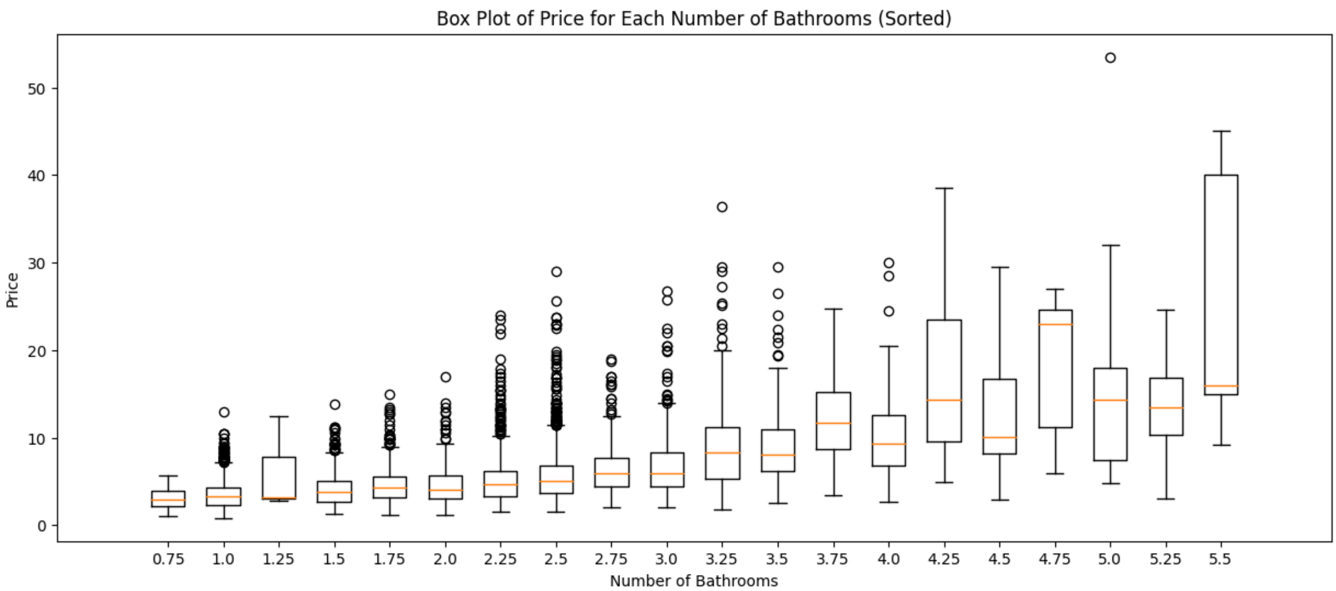
Next steps: Generate code with bathroom_stats View recommended plots New interactive sheet

You can see there are a lot more unique values than one might expect (what is .75 bathroom? I wonder about that too). Now to verify our intuition that more bedroom and bath room leads to higher pricing, we can further visualize the price distribution for each bedroom and bathroom number. This can be achived by grouping price data by the different values of bedrooms, and bathrooms, then use box plots to visualize how prices are distributed, given specific values for the numbers of bedrooms / bathrooms:

```
# find the unique number of bedrooms in the data
unique_bedrooms = sorted(data_without_id['bedrooms'].unique())

# Create a box plot of 'price' for each unique number of bedrooms with at least 3 examples
plt.figure(figsize=(10, 6))  # Adjust the figure size if needed

for num in unique_bedrooms:
    bedroom_data = data_without_id[data_without_id['bedrooms'] == num]['price']

    # Skip plotting if there are less than 3 examples with this number of bedrooms. you can remove the skipping and see the effect.
    if len(bedroom_data) >= 3:
        plt.boxplot(bedroom_data, positions=[num], labels=[num], showfliers=True)

# Add labels and a title to the plot
plt.xlabel('Number of Bedrooms')
plt.ylabel('Price')
plt.title('Box Plot of Price for Each Number of Bedrooms (Sorted)')

# Show the plot
plt.show()
```

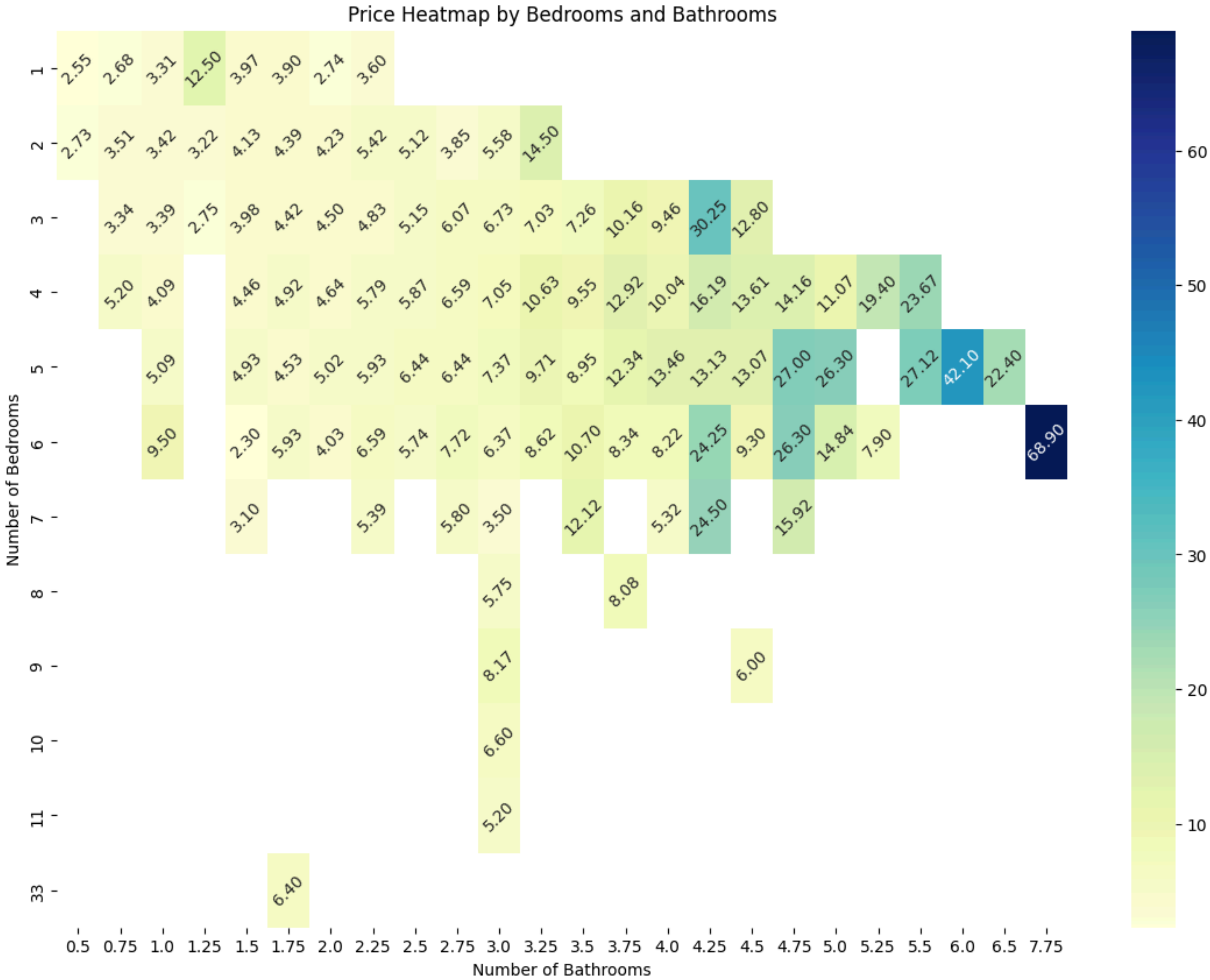



```
# find the unique number of bathrooms in the data
unique_bathrooms = sorted(data_without_id['bathrooms'].unique())

# Create a box plot of 'price' for each unique number of bedrooms with at least 3 examples
plt.figure(figsize=(15, 6))  # Adjust the figure size if needed

for num in unique_bathrooms:
    bathroom_data = data_without_id[data_without_id['bathrooms'] == num]['price']

    # Skip plotting if there are less than 3 examples with this number of bedrooms
    if len(bathroom_data) >= 3:
        plt.boxplot(bathroom_data, positions=[num], labels=[num])

# Add labels and a title to the plot
plt.xlabel('Number of Bathrooms')
plt.ylabel('Price')
plt.title('Box Plot of Price for Each Number of Bathrooms (Sorted)')

# Show the plot
plt.show()
```



As can be seen from the results above, the price does appear to adhere to the "more rooms -> more expensive" trend. We can also create a heatmap to show the price of the house as a function of the # of bathroom and # of bedrooms using the seaborn package.

```
import seaborn as sns

# Create a pivot table to prepare data for the heatmap
pivot_table = data_without_id.pivot_table(index='bedrooms', columns='bathrooms', values='price', aggfunc='mean')

# Create a heatmap using seaborn
plt.figure(figsize=(14, 10))  # Adjust the figure size if needed
heatmap = sns.heatmap(pivot_table, cmap='YlGnBu', annot=True, fmt='.2f', cbar=True)

for text in heatmap.texts:
    text.set(rotation=45)

# Add labels and a title to the plot
plt.xlabel('Number of Bathrooms')
plt.ylabel('Number of Bedrooms')
plt.title('Price Heatmap by Bedrooms and Bathrooms')

# Show the plot
plt.show()
```



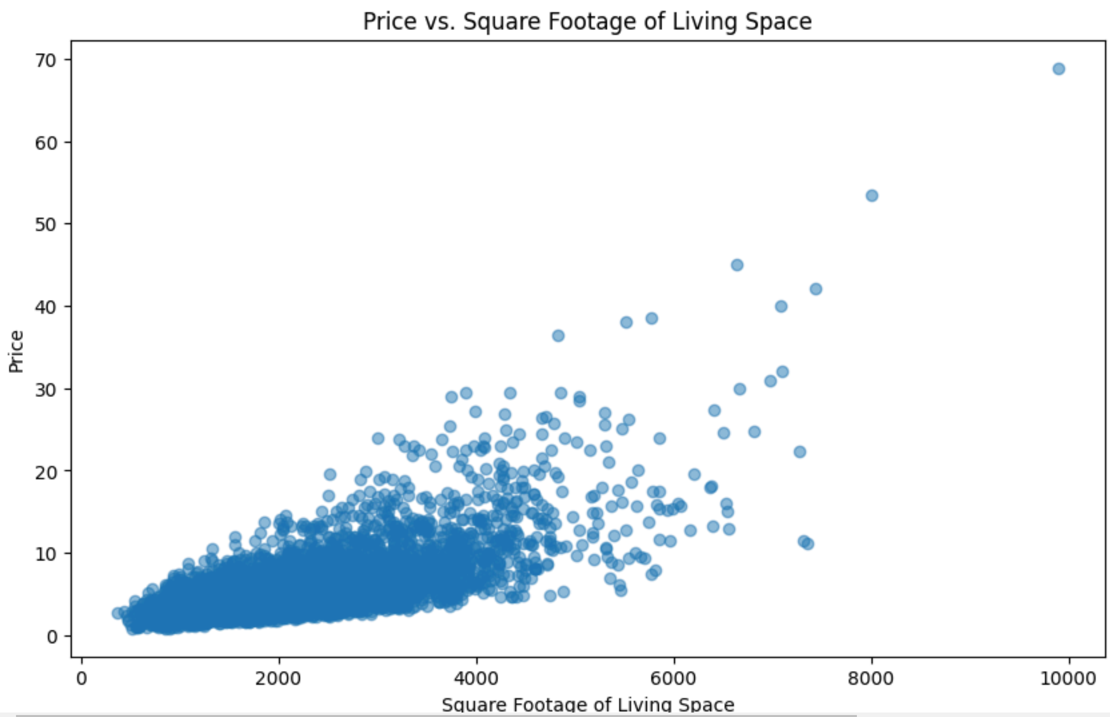
Does the trend follow your expection? Any outliers?

It follows our expectation of more bedroom and bath room leads to higher, but we have few outliers like 33 bedroom and 1.75 bathroom has price is only 6.40 and 9 bedrooms and 4.5 bathrooms is 6.00 which is less than all the bedrooms below 9

Another intuitively important feature for a house is the square footage of the house. We can plot the price of the house agaist the square footage of the house and see if there is a clear trend as expected.

```
plt.figure(figsize=(10, 6))  # Adjust the figure size if needed
```

```
plt.scatter(data_without_id['sqft_living'], data_without_id['price'], alpha=0.5)

# Add labels and a title to the plot
plt.xlabel('Square Footage of Living Space')
plt.ylabel('Price')
plt.title('Price vs. Square Footage of Living Space')

# Show the plot
plt.show()
```



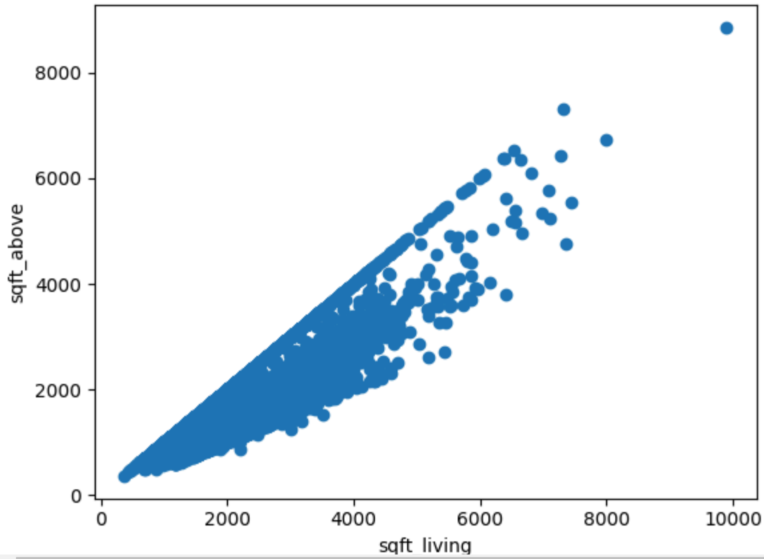
Closer inspection reveals that there are several features associated with square footage. Let's see how strongly correlated they are with one another.

```
data_without_id[["sqft_living", "sqft_lot", "sqft_living15", "sqft_lot15", "sqft_above", "sqft_basement"]].corr()
```

	sqft_living	sqft_lot	sqft_living15	sqft_lot15	sqft_above	sqft_basement
sqft_living	1.000000	0.164651	0.762189	0.178888	0.878699	0.416699
sqft_lot	0.164651	1.000000	0.139211	0.774133	0.176956	0.007146
sqft_living15	0.762189	0.139211	1.000000	0.171446	0.737738	0.188109
sqft_lot15	0.178888	0.774133	0.171446	1.000000	0.190612	0.010897
sqft_above	0.878699	0.176956	0.737738	0.190612	1.000000	-0.067804
sqft_basement	0.416699	0.007146	0.188109	0.010897	-0.067804	1.000000

Sqft_living and sqft_above are the two most correlated feautres. We can visualize their relationship by using a scatter plot:

```
plt.scatter(data_without_id['sqft_living'].values, data_without_id['sqft_above'].values)
plt.xlabel("sqft_living")
plt.ylabel("sqft_above")
plt.show()
```

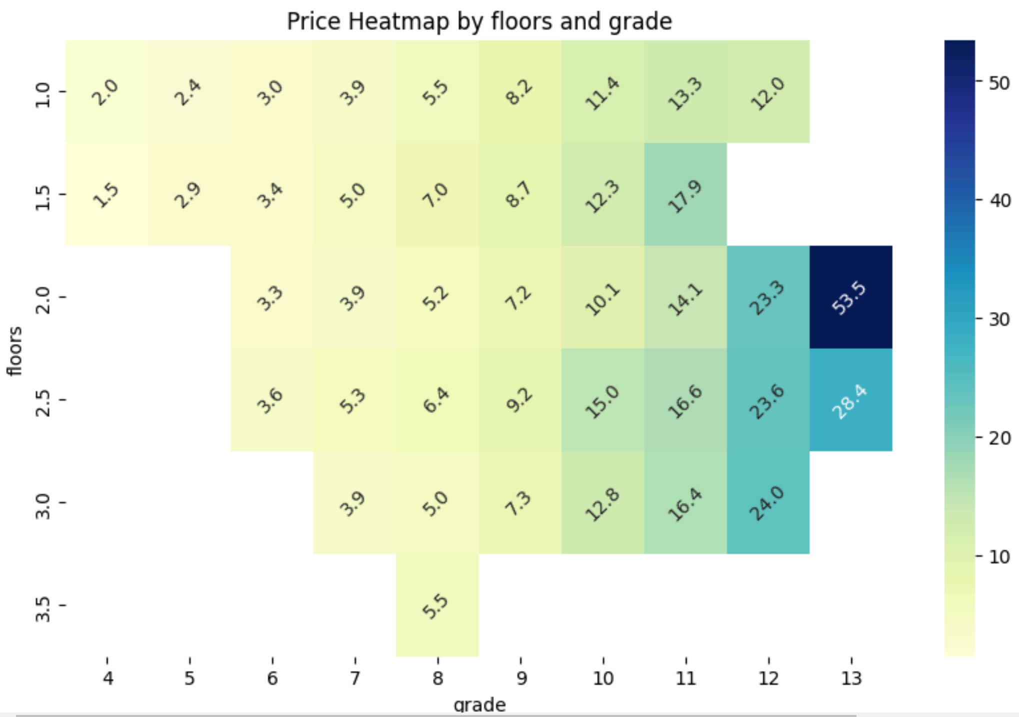


When we have features that are highly redundant, it is important to understand the impact of such redundant features to the learning algorithm. We will explore more on this in later assignments.

TO DO 1: do a bit exploration of other features on our own (5 pts)

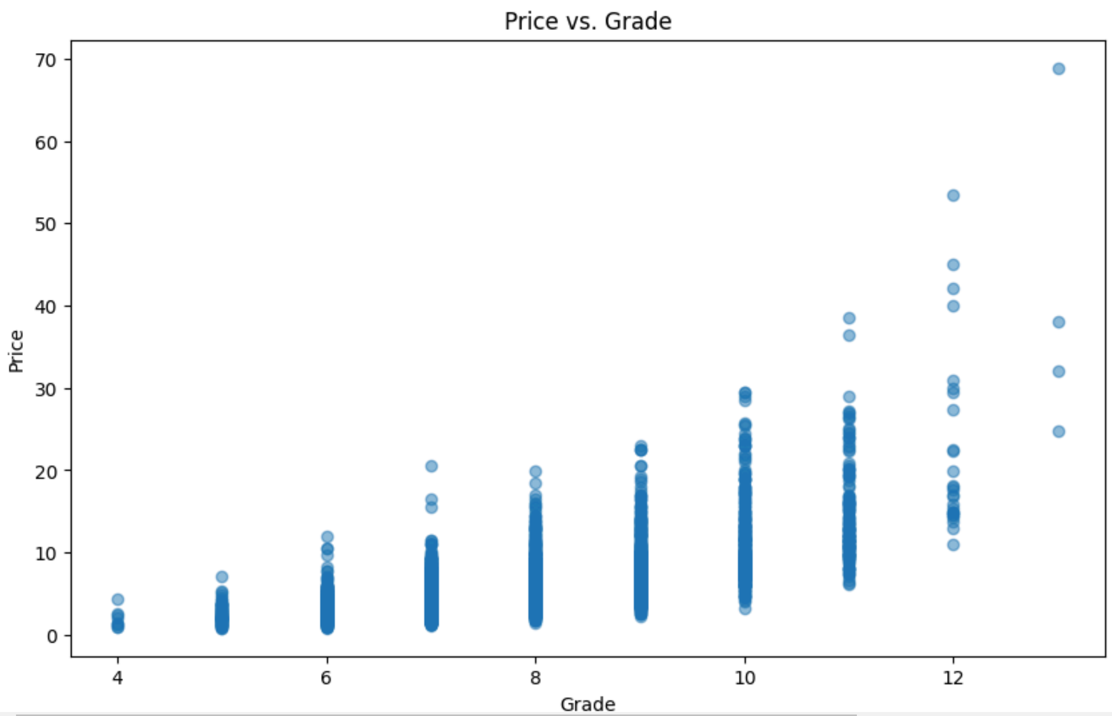
TO DO: perform similar analysis to at least two other features of your choice. Use a text box to report your observations and understanding of these features.

```
import seaborn as sns

# Create a pivot table to prepare data for the heatmap
pivot_table = data_without_id.pivot_table(index='floors', columns='grade', values='price', aggfunc='mean')

# Create a heatmap using seaborn
plt.figure(figsize=(10, 6))  # Adjust the figure size if needed
heatmap = sns.heatmap(pivot_table, cmap='YlGnBu', annot=True, fmt='.1f', cbar=True)

for text in heatmap.texts:
    text.set(rotation=45)

# Add labels and a title to the plot
plt.xlabel('grade')
plt.ylabel('floors')
plt.title('Price Heatmap by floors and grade')

# Show the plot
plt.show()
```



if More Number of floors and Grade of construction higher the price can be high,seen in the heatmap

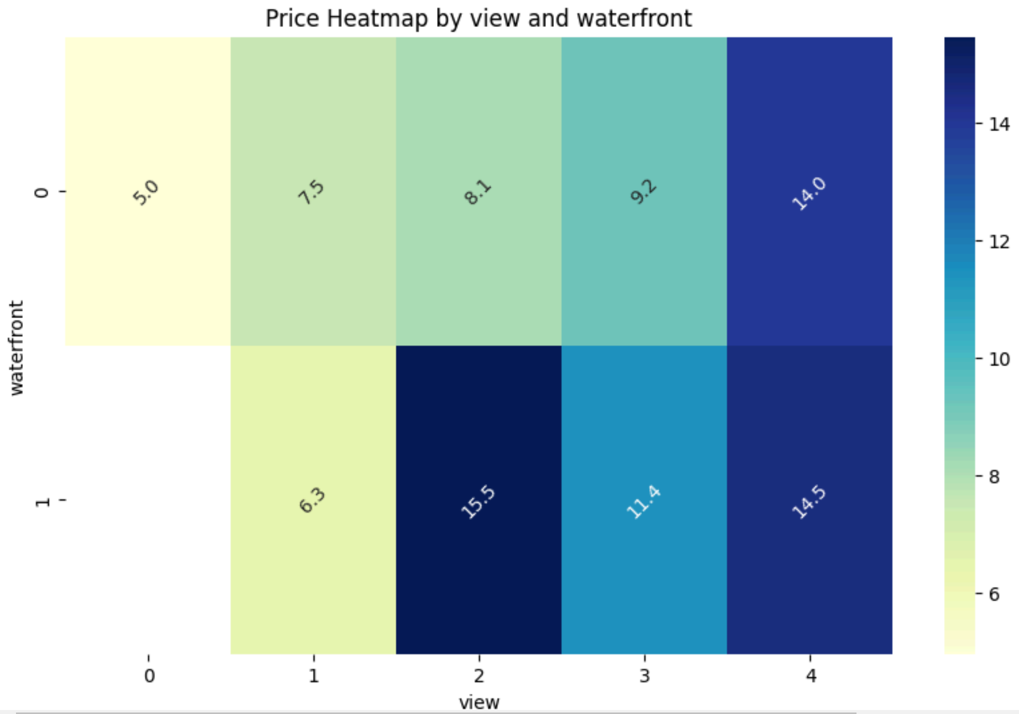
```
# put your code here for exploring other feautres. Feel free to use more blocks of text and code
plt.figure(figsize=(10, 6))  # Adjust the figure size if needed
plt.scatter(data_without_id['grade'], data_without_id['price'], alpha=0.5)

# Add labels and a title to the plot
plt.xlabel('Grade')
plt.ylabel('Price')
plt.title('Price vs. Grade')

# Show the plot
plt.show()
```



Price of Home vs Grade of Homes

Higher-grade homes,high quality level of construction and design tend to have higher prices.

```
import seaborn as sns

# Create a pivot table to prepare data for the heatmap
pivot_table = data_without_id.pivot_table(index='waterfront', columns='view', values='price', aggfunc='mean')

# Create a heatmap using seaborn
plt.figure(figsize=(10, 6))  # Adjust the figure size if needed
heatmap = sns.heatmap(pivot_table, cmap='YlGnBu', annot=True, fmt='.1f', cbar=True)

for text in heatmap.texts:
    text.set(rotation=45)

# Add labels and a title to the plot
plt.xlabel('view')
plt.ylabel('waterfront')
plt.title('Price Heatmap by view and waterfront')

# Show the plot
plt.show()
```



Not a major difference in price when you have view and waterfront than not having a waterfront but view

```
import matplotlib.pyplot as plt


# line 1 points
x1 = (data_without_id['waterfront'] == 1)
y1 = data_without_id['view']
# plotting the line 1 points
plt.plot(x1, y1, label = "with waterfront")

# line 2 points
x2 = (data_without_id['waterfront']== 0)
y2 = data_without_id['view']
# plotting the line 2 points
plt.plot(x2, y2, label = "without waterfront")

# naming the x axis
plt.xlabel('')
# naming the y axis
plt.ylabel('view')
# giving a title to my graph
plt.title('View and Waterfront!')

# show a legend on the plot
plt.legend()
```

NO relation between view and waterfront is available or not

```
# put your code here for exploring other feautres. Feel free to use more blocks of text and code
plt.figure(figsize=(10, 6))  # Adjust the figure size if needed
plt.scatter(data_without_id['grade'], data_without_id['yr_built'], alpha=0.5)

# Add labels and a title to the plot
plt.xlabel('Grade')
plt.ylabel('Year Built')
plt.title('yr built vs. Grade')

# Show the plot
plt.show()
```