Chapter 1 Dataframes and Datasets

Introduction to Data Wrangling

"Data wrangling is the process of transforming and structuring data from its raw form into a desired format with the intent of improving data quality and making it more consumable and useful for analytics or machine learning. It is also sometimes called data munging."

In Python, data wrangling is achieved using the 'pandas' library.

To install the pandas library, use the following command:

In command prompt: pip install pandas

In a notebook environment: !pip install pandas

In [1]:

```
# Install pandas
!pip install pandas
```

```
Requirement already satisfied: pandas in c:\python310\lib\site-packages (2.0.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\python310\lib\site-packa
ges (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\python310\lib\site-packages (from
pandas) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in c:\python310\lib\site-packages (fro
m pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\python310\lib\site-packages (from
pandas) (1.24.2)
Requirement already satisfied: six>=1.5 in c:\python310\lib\site-packages (from pyth
on-dateutil>=2.8.2->pandas) (1.16.0)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
```

Pandas

"Pandas (commonly imported as pd) is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool, built on top of the Python programming language."

Pandas works with datasets and helps analyze them in detail. Therefore, it is essential to provide pandas with a dataset.

Official documentation for pandas: https://pandas.pydata.org/docs/reference/index.html

Datasets

Textual datasets can be stored in many different formats, including:

1. CSV (Comma-Separated Values)
2. JSON (JavaScript Object Notation)
3. SQL (Structured Query Language) Relations
4. And many others

By far, the most commonly used format is CSV.

Comma-Seperated Values

As the name suggests, the values are separated by commas. The dataset is divided into rows and columns, with the first row defining all the columns.

Ex:

```
"Pokedex Number", "Name", "Type"
1, "Bulbasaur", "Grass"
2, "Ivysaur", "Grass"
4, "Charmander", "Fire"
7, "Squirtle", "Water"
```

More readable format:

Pokedex Number	Name	Type
1	Bulbasaur	Grass
2	Ivysaur	Grass
4	Charmander	Fire
7	Squirtle	Water

However, it's worth noting that when reading CSV files, pandas is flexible enough to separate values by any special character as specified by the user.

Best source for datasets, Kaggle: https://www.kaggle.com/datasets

Dataframes

Pandas works only upon a special data structure called Dataframes (commonly referred as df). To work with any dataset in any form, first it needs to be converted into a Dataframe.

"A Dataframe is a two dimensional, size-mutable, potentially heterogeneous tabular data structure with labeled axes."

To read a dataset and convert into a Dataframe we use the method **read_extension()**

Input a .csv dataset into Pandas

To read a .csv file, we make use of the Pandas.read_csv() method. It converts the .csv file into a dataframe with which pandas can perform data analysis or manipulation.

Syntax: pandas.read_csv('Path to file')

Example:

In [2]:

```python
# Import pandas
import pandas as pd

# Path to the file
path = "./datasets/sales.csv"

# Read the csv file and convert it into a dataframe
pd.read_csv(path)

# Capture the dataframe into a variable
df_sales = pd.read_csv(path)

# Print the dataframe
df_sales
```

Out[2]:

	rating	shipping_zip	billing_zip
0	5.0	NaN	81220.0
1	4.5	94931.0	94931.0
2	NaN	92625.0	92625.0
3	4.5	10003.0	10003.0
4	4.0	NaN	92660.0
5	NaN	NaN	NaN
6	NaN	60007.0	60007.0

In [3]:

```python
# Try with a much bigger dataset "kc_house_data.csv" in the "datasets" folder.

df_houses = pd.read_csv("./datasets/kc_house_data.csv")

df_houses
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	20141013T000000	221900.0	3	1.00	1180	565
1	6414100192	20141209T000000	538000.0	3	2.25	2570	724
2	5631500400	20150225T000000	180000.0	2	1.00	770	1000
3	2487200875	20141209T000000	604000.0	4	3.00	1960	500
4	1954400510	20150218T000000	510000.0	3	2.00	1680	808
...	...	...	...	...	...	...	.
21608	263000018	20140521T000000	360000.0	3	2.50	1530	113
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	581
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	135
21611	291310100	20150116T000000	400000.0	3	2.50	1600	238
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	107

21613 rows × 21 columns

Simple properties/methods defined on DataFrames

Official documentation for all the various methods defined on dataframes: https://pandas.pydata.org/docs/reference/frame.html

1. **Dataframe.columns:** Returns the column/attribute labels of the dataframe.

Ex: df_sales.columns

In [4]:
```
# Columns in sales dataframe
df_sales.columns
```

Out[4]:
```
Index(['rating', 'shipping_zip', 'billing_zip'], dtype='object')
```

In [5]:
```
# Columns in houses dataframe
df_houses.columns
```

Out[5]:
```
Index(['id', 'date', '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'],
      dtype='object')
```

2. **len(Dataframe):** Returns the no. of rows/records/tuples in the dataframe.

Ex: len(df_sales)

In [6]:
```
# Return the no. of rows in sales dataframe
```

```
len(df_sales)
```

Out[6]: 7

In [7]:

```
# Return the no. of rows in houses dataframe
len(df_houses)
```

Out[7]: 21613

3. **Dataframe.shape:** Returns the no. of rows and cols in the dataframe.

Ex: df_sales.shape

In [8]:

```
# Return the shape of sales dataframe
df_sales.shape

# The output (7, 3) indicates that the dataframe has 7 rows and 3 columns
```

Out[8]: (7, 3)

In [9]:

```
# Return the shape of houses dataframe
df_houses.shape
```

Out[9]: (21613, 21)

4. **Dataframe.size:** Returns the area/size (i.e, rows * cols) of dataframe.

Ex: df_sales.size

In [10]:

```
# Return the size of sales dataframe
df_sales.size
```

Out[10]: 21

In [11]:

```
# Return the size of houses dataframe
df_houses.size
```

Out[11]: 453873

Display property of Pandas and Subsetting of Dataframes by no. of rows

1. **pd.options.display.min_rows:** Max no. of rows to be displayed.

Ex: pd.options.display.min_rows = 10

In [12]:

```
# Display at max 10 rows
pd.options.display.min_rows = 10
```

In [13]:
```
# Now try printing sales dataframe
df_sales
```

Out[13]:

	rating	shipping_zip	billing_zip
0	5.0	NaN	81220.0
1	4.5	94931.0	94931.0
2	NaN	92625.0	92625.0
3	4.5	10003.0	10003.0
4	4.0	NaN	92660.0
5	NaN	NaN	NaN
6	NaN	60007.0	60007.0

In [14]:
```
# Now try printing houses dataframe
df_houses
```

Out[14]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
0	7129300520	20141013T000000	221900.0	3	1.00	1180	565
1	6414100192	20141209T000000	538000.0	3	2.25	2570	724
2	5631500400	20150225T000000	180000.0	2	1.00	770	1000
3	2487200875	20141209T000000	604000.0	4	3.00	1960	500
4	1954400510	20150218T000000	510000.0	3	2.00	1680	808
...	...	...	...	...	...	...	.
21608	263000018	20140521T000000	360000.0	3	2.50	1530	113
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	581
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	135
21611	291310100	20150116T000000	400000.0	3	2.50	1600	238
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	107

21613 rows × 21 columns

2. **Dataframe.head(count):** Returns the first 'count' no. of rows of the dataframe. By default the value of count is 5.

Ex: df_sales.head()

In [15]:
```
# Print the first five rows in sales dataframe
df_sales.head()
```

Out[15]:

	rating	shipping_zip	billing_zip
0	5.0	NaN	81220.0
1	4.5	94931.0	94931.0
2	NaN	92625.0	92625.0
3	4.5	10003.0	10003.0
4	4.0	NaN	92660.0

In [16]:

```
# Print the first five rows in houses dataframe
df_houses.head()
```

Out[16]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	

5 rows × 21 columns

In [17]:

```
# Store the first 15 rows in houses dataframe into a new dataframe called 'df_15hou
df_15houses = df_houses.head(15)

df_15houses
```

Out[17]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080
5	7237550310	20140512T000000	1225000.0	4	4.50	5420	101930
6	1321400060	20140627T000000	257500.0	3	2.25	1715	6819
7	2008000270	20150115T000000	291850.0	3	1.50	1060	9711
8	2414600126	20150415T000000	229500.0	3	1.00	1780	7470
9	3793500160	20150312T000000	323000.0	3	2.50	1890	6560
10	1736800520	20150403T000000	662500.0	3	2.50	3560	9796
11	9212900260	20140527T000000	468000.0	2	1.00	1160	6000
12	114101516	20140528T000000	310000.0	3	1.00	1430	19901
13	6054650070	20141007T000000	400000.0	3	1.75	1370	9680
14	1175000570	20150312T000000	530000.0	5	2.00	1810	4850

15 rows × 21 columns

3. **Dataframe.tail(count):** Returns the last 'count' no. of rows of the dataframe. By default the value of count is 5.

Ex: df_sales.tail()

In [18]:

```
# Print the last five rows in sales dataframe
df_sales.tail()
```

Out[18]:

	rating	shipping_zip	billing_zip
2	NaN	92625.0	92625.0
3	4.5	10003.0	10003.0
4	4.0	NaN	92660.0
5	NaN	NaN	NaN
6	NaN	60007.0	60007.0

In [19]:

```
# Print the last five rows in houses dataframe
df_houses.tail()
```

Out[19]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lo
21608	263000018	20140521T000000	360000.0	3	2.50	1530	113
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	581
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	135
21611	291310100	20150116T000000	400000.0	3	2.50	1600	238
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	107

5 rows × 21 columns

In [20]:

```
# Store the last 15 rows in houses dataframe into a new dataframe called 'df_15last
df_15lasthouses = df_houses.tail(15)

df_15lasthouses
```

Out[20]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_l
21598	8956200760	20141013T000000	541800.0	4	2.50	3118	78
21599	7202300110	20140915T000000	810000.0	4	3.00	3990	78
21600	249000205	20141015T000000	1537000.0	5	3.75	4470	80
21601	5100403806	20150407T000000	467000.0	3	2.50	1425	11
21602	844000965	20140626T000000	224000.0	3	1.75	1500	119
21603	7852140040	20140825T000000	507250.0	3	2.50	2270	55
21604	9834201367	20150126T000000	429000.0	3	2.00	1490	11
21605	3448900210	20141014T000000	610685.0	4	2.50	2520	60
21606	7936000429	20150326T000000	1007500.0	4	3.50	3510	72
21607	2997800021	20150219T000000	475000.0	3	2.50	1310	12
21608	263000018	20140521T000000	360000.0	3	2.50	1530	11
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	58
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	13
21611	291310100	20150116T000000	400000.0	3	2.50	1600	23
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	10

15 rows × 21 columns

Datatypes of columns in Dataframes

Pandas by default while converting the dataset into a dataframe, analyzes the type of each column in the dataset and assigns appropriate datatype to it.

1. **Dataframe.dtypes:** To know the datatypes assigned to each column.

Ex: df_sales.dtypes

```
In [21]: # Information about datatypes in sales dataframe
         df_sales.dtypes
```

```
Out[21]: rating          float64
         shipping_zip    float64
         billing_zip     float64
         dtype: object
```

```
In [22]: # Information about datatypes in houses dataframe
         df_houses.dtypes
```

```
Out[22]: 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
```

2. **Dataframe.info():** To know the datatypes assigned to each column in the dataframe, the no. of non-null records of the specified column and the total memory occupied by the dataframe.

Ex: df_sales.info()

```
In [23]: # Print sales dataset
         df_sales.head()
```

Out[23]:

	rating	shipping_zip	billing_zip
0	5.0	NaN	81220.0
1	4.5	94931.0	94931.0
2	NaN	92625.0	92625.0
3	4.5	10003.0	10003.0
4	4.0	NaN	92660.0

In [24]:

```
# Information about datatypes, no. of non-null records in sales dataframe
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   rating        4 non-null      float64
 1   shipping_zip  4 non-null      float64
 2   billing_zip   6 non-null      float64
dtypes: float64(3)
memory usage: 296.0 bytes
```

In [25]:

```
# Print houses dataset
df_houses.head()
```

Out[25]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	

5 rows × 21 columns

In [26]:

```
# Information about datatypes, no. of non-null records in houses dataframe
df_houses.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
```

Basic Dataframe Analysis Methods

Method	Description
sum	Returns the sum of values in the DataFrame.
min	Returns the minimum value in the DataFrame.
max	Returns the maximum value in the DataFrame.
count	Returns the count of non-null values in the DataFrame.
mean	Returns the mean of values in the DataFrame.
median	Returns the median of values in the DataFrame.
mode	Returns the mode of values in the DataFrame.
describe	Returns a DataFrame with statistical information like mean, standard deviation, minimum, maximum, and quartiles.

In [27]:

```
# Print houses dataset
df_houses.head()
```

Out[27]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	

5 rows × 21 columns

In [28]:

```
# Find the sum of all the columns in houses dataset
df_houses.sum()

# Note: Peforms string concatenation for Date/String datatypes
```

Out[28]:

```
id                                                98994056770455
date             20141013T00000020141209T00000020150225T0000002...
price                                              11672925008.0
bedrooms                                                   72854
bathrooms                                               45706.25
sqft_living                                             44952873
sqft_lot                                               326506890
floors                                                   32296.5
waterfront                                                   163
view                                                        5064
condition                                                  73688
grade                                                     165488
sqft_above                                              38652488
sqft_basement                                            6300385
yr_built                                                42599334
yr_renovated                                             1824186
zipcode                                               2119758513
lat                                                 1027915.4151
long                                                -2641408.943
sqft_living15                                           42935359
sqft_lot15                                             275964632
dtype: object
```

In [29]:

```
# Find the sum of all the numeric columns in houses dataset
df_houses.sum(numeric_only=True)
```

Out[29]:

```
id               9.899406e+13
price            1.167293e+10
bedrooms         7.285400e+04
bathrooms        4.570625e+04
sqft_living      4.495287e+07
sqft_lot         3.265069e+08
floors           3.229650e+04
waterfront       1.630000e+02
view             5.064000e+03
condition        7.368800e+04
grade            1.654880e+05
sqft_above       3.865249e+07
sqft_basement    6.300385e+06
yr_built         4.259933e+07
yr_renovated     1.824186e+06
zipcode          2.119759e+09
lat              1.027915e+06
long            -2.641409e+06
sqft_living15    4.293536e+07
sqft_lot15       2.759646e+08
dtype: float64
```

In [30]:

```
# Find the min among all the columns in houses dataset
df_houses.min()
```

Out[30]:

```
id                       1000102
date             20140502T000000
price                    75000.0
bedrooms                       0
bathrooms                    0.0
sqft_living                  290
sqft_lot                     520
floors                       1.0
waterfront                     0
view                           0
condition                      1
grade                          1
sqft_above                   290
sqft_basement                  0
yr_built                    1900
yr_renovated                   0
zipcode                    98001
lat                      47.1559
long                    -122.519
sqft_living15                399
sqft_lot15                   651
dtype: object
```

In [31]:

```
# Find the max among all the columns in houses dataset
df_houses.max()
```

```
Out[31]: id                    9900000190
         date             20150527T000000
         price                  7700000.0
         bedrooms                      33
         bathrooms                    8.0
         sqft_living                13540
         sqft_lot                 1651359
         floors                       3.5
         waterfront                     1
         view                           4
         condition                      5
         grade                         13
         sqft_above                  9410
         sqft_basement               4820
         yr_built                    2015
         yr_renovated                2015
         zipcode                    98199
         lat                      47.7776
         long                    -121.315
         sqft_living15               6210
         sqft_lot15                871200
         dtype: object
```

```
In [32]: # Find the count of all the columns (NA values excluded) in houses dataset
         df_houses.count()
```

```
Out[32]: id               21613
         date             21613
         price            21613
         bedrooms         21613
         bathrooms        21613
         sqft_living      21613
         sqft_lot         21613
         floors           21613
         waterfront       21613
         view             21613
         condition        21613
         grade            21613
         sqft_above       21613
         sqft_basement    21613
         yr_built         21613
         yr_renovated     21613
         zipcode          21613
         lat              21613
         long             21613
         sqft_living15    21613
         sqft_lot15       21613
         dtype: int64
```

```
In [33]: # Print sales dataset
         df_sales.head()
```

Out[33]:

	rating	shipping_zip	billing_zip
0	5.0	NaN	81220.0
1	4.5	94931.0	94931.0
2	NaN	92625.0	92625.0
3	4.5	10003.0	10003.0
4	4.0	NaN	92660.0

In [34]:

```python
# Find the count of all the columns (NA values excluded) in sales dataset
df_sales.count()
```

Out[34]:

```
rating          4
shipping_zip    4
billing_zip     6
dtype: int64
```

In [35]:

```python
# Find the mean of all the columns in houses dataset
# df_houses.mean()
```

Important Note: The mean, median and mode of a dataset could be found only for numerical data.

The columns that contain non-numeric data are known as nuisance columns, which need to be removed before calculating them. Fortunately, pandas by default returns only for non-numeric data, but it may not in the near future.

Ex:

df_numeric_sales = df_sales.select_dtypes(include="number")

df_numeric_sales.mean()

In [36]:

```python
# Extract numeric columns from dataset
df_numeric_houses = df_houses.select_dtypes(include="number")

# Calculate the mean
df_numeric_houses.mean()
```

Out[36]:

```
id               4.580302e+09
price            5.400881e+05
bedrooms         3.370842e+00
bathrooms        2.114757e+00
sqft_living      2.079900e+03
sqft_lot         1.510697e+04
floors           1.494309e+00
waterfront       7.541757e-03
view             2.343034e-01
condition        3.409430e+00
grade            7.656873e+00
sqft_above       1.788391e+03
sqft_basement    2.915090e+02
yr_built         1.971005e+03
yr_renovated     8.440226e+01
zipcode          9.807794e+04
lat              4.756005e+01
long            -1.222139e+02
sqft_living15    1.986552e+03
sqft_lot15       1.276846e+04
dtype: float64
```

In [37]:

```python
# Find the median of all the columns in houses dataset
df_numeric_houses.median()
```

Out[37]:

```
id               3.904930e+09
price            4.500000e+05
bedrooms         3.000000e+00
bathrooms        2.250000e+00
sqft_living      1.910000e+03
sqft_lot         7.618000e+03
floors           1.500000e+00
waterfront       0.000000e+00
view             0.000000e+00
condition        3.000000e+00
grade            7.000000e+00
sqft_above       1.560000e+03
sqft_basement    0.000000e+00
yr_built         1.975000e+03
yr_renovated     0.000000e+00
zipcode          9.806500e+04
lat              4.757180e+01
long            -1.222300e+02
sqft_living15    1.840000e+03
sqft_lot15       7.620000e+03
dtype: float64
```

In [38]:

```python
# Find the mode of all the columns in houses dataset
df_numeric_houses.mode()
```

Out[38]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	795000620.0	350000.0	3.0	2.5	1300.0	5000.0	1.0	0.0
1	NaN	450000.0	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [39]:

```
# Describe the statistical information of houses dataset - Always runs only on nume
df_houses.describe()
```

Out[39]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lo
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+0
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+0
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+0
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+0
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+0
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+0
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+0
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+0