Introduction

Welcome to **CS148 - Data Science Fundamentals!** As we're planning to move through topics aggressively in this course, to start out, we'll look to do an end-to-end walkthrough of a datascience project, and then ask you to replicate the code yourself for a new dataset.

Please note: We don't expect you to fully grasp everything happening here in either code or theory. This content will be reviewed throughout the quarter. Rather we hope that by giving you the full perspective on a data science project it will better help to contextualize the pieces as they're covered in class

In that spirit, we will first work through an example project from end to end to give you a feel for the steps involved.

Here are the main steps:

1. Get the data
2. Visualize the data for insights
3. Preprocess the data for your machine learning algorithm
4. Select a machine learning model and train it
5. Evaluate its performance

Working with Real Data

It is best to experiment with real-data as opposed to aritifical datasets.

There are many different open datasets depending on the type of problems you might be interested in!

Here are a few data repositories you could check out:

- UCI Datasets
- Kaggle Datasets
- AWS Datasets

Below we will run through an California Housing example collected from the 1990's.

Setup

We'll start by importing a series of libraries we'll be using throughout the project.

```
[1] import sys
    assert sys.version_info >= (3, 5) # python>=3.5
    import sklearn
    #assert sklearn.__version__ >= "0.20" # sklearn >= 0.20

    import numpy as np #numerical package in python
    %matplotlib inline
    import matplotlib.pyplot as plt #plotting package

    # to make this notebook's output identical at every run
    np.random.seed(42)

    #matplotlib magic for inline figures
    %matplotlib inline
    import matplotlib # plotting library
    import matplotlib.pyplot as plt
```

Intro to Data Exploration Using Pandas

In this section we will load the dataset, and visualize different features using different types of plots.

Packages we will use:

- **Pandas**: is a fast, flexibile and expressive data structure widely used for tabular and multidimensional datasets.
- **Matplotlib**: is a 2d python plotting library which you can use to create quality figures (you can plot almost anything if you're willing to code it out!)
    - other plotting libraries:seaborn, ggplot2

Note: If you're working in CoLab for this project, the CSV file first has to be loaded into the environment. This can be done manually using the sidebar menu option, or using the following code here.

If you're running this notebook locally on your device, simply proceed to the next step.

```
[ ] from google.colab import files
    files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving housing.csv to housing (1).csv
{'housing.csv': b'longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,median_income,median_house_value,ocean_proximity\n-122.23,37.88,41.0,880.0,129.0,322.0,126.0,8.3252,452600.0,NEAR
BAY\n-122.22,37.86,21.0,7099.0,1106.0,2401.0,1138.0,8.3014,358500.0,NEAR BAY\n-122.24,37.85,52.0,1467.0,190.0,496.0,177.0,7.2574,352100.0,NEAR BAY\n-122.25,37.85,52.0,1274.0,235.0,558.0,219.0,5.6431,341300.0,NEAR BAY\n-
122.25,37.85,52.0,1627.0,280.0,565.0,259.0,3.8462,342200.0,NEAR BAY\n-122.25,37.85,52.0,919.0,213.0,413.0,193.0,4.0368,269700.0,NEAR BAY\n-122.25,37.84,52.0,2535.0,489.0,1094.0,514.0,3.6591,299200.0,NEAR BAY\n-
122.25,37.84,52.0,3104.0,687.0,1157.0,647.0,3.12,241400.0,NEAR BAY\n-122.26,37.84,42.0,2555.0,665.0,1206.0,595.0,2.0804,226700.0,NEAR BAY\n-122.25,37.84,52.0,3549.0,707.0,1551.0,714.0,3.6912,261100.0,NEAR BAY\n-
122.26,37.85,52.0,2202.0,434.0,910.0,402.0,3.2031,281500.0,NEAR BAY\n-122.26,37.85,52.0,3503.0,752.0,1504.0,734.0,3.2705,241800.0,NEAR BAY\n-122.26,37.85,52.0,2491.0,474.0,1098.0,468.0,3.075,213500.0,NEAR BAY\n-
122.26,37.84,52.0,696.0,191.0,345.0,174.0,2.6736,191300.0,NEAR BAY\n-122.26,37.85,52.0,2643.0,626.0,1212.0,620.0,1.9167,159200.0,NEAR BAY\n-122.26,37.85,50.0,1120.0,283.0,697.0,264.0,2.125,140000.0,NEAR BAY\n-
122.27,37.85,52.0,1966.0,347.0,793.0,331.0,2.775,152500.0,NEAR BAY\n-122.27,37.85,52.0,1228.0,293.0,648.0,303.0,2.1202,155500.0,NEAR BAY\n-122.26,37.84,50.0,2239.0,455.0,990.0,419.0,1.9911,158700.0,NEAR BAY\n-
122.27,37.84,52.0,1503.0,298.0,690.0,275.0,2.6033,162900.0,NEAR BAY\n-122.27,37.85,40.0,751.0,184.0,409.0,166.0,1.3578,147500.0,NEAR BAY\n-122.27,37.85,42.0,1639.0,367.0,929.0,366.0,1.7135,159800.0,NEAR BAY\n-
122.27,37.84,52.0,2436.0,541.0,1015.0,478.0,1.725,113900.0,NEAR BAY\n-122.27,37.84,52.0,1688.0,337.0,853.0,325.0,2.1806,99700.0,NEAR BAY\n-122.27,37.84,52.0,2224.0,437.0,1006.0,422.0,2.6,132600.0,NEAR BAY\n-
122.28,37.85,41.0,535.0,123.0,317.0,119.0,2.4038,107500.0,NEAR BAY\n-122.28,37.85,49.0,1130.0,244.0,607.0,239.0,2.4597,93800.0,NEAR BAY\n-122.28,37.85,52.0,1898.0,421.0,1102.0,397.0,1.808,105500.0,NEAR BAY\n-
122.28,37.84,50.0,2082.0,492.0,1131.0,473.0,1.6424,108900.0,NEAR BAY\n-122.28,37.84,52.0,729.0,160.0,395.0,155.0,1.6875,132000.0,NEAR BAY\n-122.28,37.84,49.0,1916.0,447.0,863.0,378.0,1.9274,122300.0,NEAR BAY\n-
122.28,37.84,52.0,2153.0,481.0,1168.0,441.0,1.9615,115200.0,NEAR BAY\n-122.27,37.84,48.0,1922.0,409.0,1026.0,335.0,1.7969,110400.0,NEAR BAY\n-122.27,37.83,49.0,1655.0,366.0,754.0,329.0,1.375,104900.0,NEAR BAY\n-
122.27,37.83,51.0,2665.0,574.0,1258.0,536.0,2.7303,109700.0,NEAR BAY\n-122.27,37.83,49.0,1215.0,282.0,570.0,264.0,1.4861,97200.0,NEAR BAY\n-122.27,37.83,48.0,1798.0,432.0,987.0,374.0,1.0972,104500.0,NEAR BAY\n-
122.28,37.83,52.0,1511.0,390.0,901.0,403.0,1.4103,103900.0,NEAR BAY\n-122.26,37.83,52.0,1470.0,330.0,689.0,309.0,3.48,191400.0,NEAR BAY\n-122.26,37.83,52.0,2432.0,715.0,1377.0,696.0,2.5898,176000.0,NEAR BAY\n-
122.26,37.83,52.0,1665.0,419.0,946.0,395.0,2.0978,155400.0,NEAR BAY\n-122.26,37.83,51.0,936.0,311.0,517.0,249.0,1.2852,150000.0,NEAR BAY\n-122.26,37.84,49.0,713.0,202.0,462.0,189.0,1.025,118800.0,NEAR BAY\n-
122.26,37.84,52.0,950.0,202.0,467.0,198.0,3.9643,188800.0,NEAR BAY\n-122.26,37.83,52.0,1443.0,311.0,660.0,292.0,3.0125,184400.0,NEAR BAY\n-122.26,37.83,52.0,1656.0,420.0,718.0,382.0,2.6768,182300.0,NEAR BAY\n-
122.26,37.83,50.0,1125.0,322.0,616.0,304.0,2.026,142500.0,NEAR BAY\n-122.27,37.82,43.0,1007.0,312.0,558.0,253.0,1.7348,137500.0,NEAR BAY\n-122.26,37.82,40.0,624.0,195.0,423.0,160.0,0.9506,187500.0,NEAR BAY\n-
122.27,37.82,40.0,946.0,375.0,700.0,352.0,1.775,112500.0,NEAR BAY\n-122.27,37.82,21.0,896.0,453.0,735.0,438.0,0.9218,171900.0,NEAR BAY\n-122.27,37.82,43.0,1868.0,456.0,1061.0,407.0,1.5045,93800.0,NEAR BAY\n-
122.27,37.82,41.0,3221.0,853.0,1959.0,720.0,1.1108,97500.0,NEAR BAY\n-122.27,37.82,52.0,1630.0,456.0,1162.0,400.0,1.2475,104200.0,NEAR BAY\n-122.28,37.82,52.0,1170.0,235.0,701.0,233.0,1.6098,87500.0,NEAR BAY\n-
122.28,37.82,52.0,945.0,243.0,576.0,220.0,1.4113,83100.0,NEAR BAY\n-122.28,37.82,52.0,1238.0,288.0,622.0,259.0,1.5057,87500.0,NEAR BAY\n-122.28,37.82,52.0,1489.0,335.0,728.0,244.0,0.8172,85300.0,NEAR BAY\n-
122.28,37.82,52.0,1387.0,341.0,1074.0,304.0,1.2171,80300.0,NEAR BAY\n-122.29,37.82,2.0,158.0,43.0,94.0,57.0,2.5625,60000.0,NEAR BAY\n-122.29,37.83,52.0,1121.0,211.0,554.0,187.0,3.3929,75700.0,NEAR BAY\n-
122.29,37.82,49.0,135.0,29.0,86.0,23.0,6.1183,75000.0,NEAR BAY\n-122.29,37.81,50.0,760.0,190.0,377.0,122.0,0.9011,86100.0,NEAR BAY\n-122.3,37.81,52.0,1224.0,237.0,521.0,159.0,1.191,76100.0,NEAR BAY\n-
122.3,37.81,48.0,828.0,182.0,392.0,133.0,2.5938,73500.0,NEAR BAY\n-122.3,37.81,52.0,1010.0,209.0,604.0,187.0,1.1667,78400.0,NEAR BAY\n-122.3,37.81,48.0,1455.0,354.0,788.0,332.0,0.8056,84400.0,NEAR BAY\n-
122.29,37.8,52.0,1027.0,244.0,492.0,147.0,2.6094,81300.0,NEAR BAY\n-122.3,37.81,52.0,572.0,109.0,274.0,82.0,1.8516,85000.0,NEAR BAY\n-122.29,37.81,46.0,2801.0,644.0,1823.0,611.0,0.9802,129200.0,NEAR BAY\n-
122.29,37.81,26.0,768.0,152.0,392.0,127.0,1.7719,82500.0,NEAR BAY\n-122.29,37.81,46.0,935.0,297.0,582.0,277.0,0.7286,95200.0,NEAR BAY\n-122.29,37.81,49.0,844.0,204.0,560.0,152.0,1.75,75000.0,NEAR BAY\n-
122.29,37.81,46.0,12.0,4.0,18.0,7.0,0.4999,67500.0,NEAR BAY\n-122.29,37.81,20.0,835.0,161.0,290.0,133.0,2.483,137500.0,NEAR BAY\n-122.28,37.81,17.0,1237.0,462.0,762.0,439.0,0.9241,177500.0,NEAR BAY\n-
122.28,37.81,36.0,2914.0,562.0,1236.0,509.0,2.4464,102100.0,NEAR BAY\n-122.28,37.81,19.0,1207.0,243.0,721.0,207.0,1.1111,108300.0,NEAR BAY\n-122.29,37.81,23.0,1745.0,374.0,1054.0,325.0,0.8026,112500.0,NEAR BAY\n-
122.28,37.8,38.0,684.0,176.0,344.0,155.0,2.0114,131300.0,NEAR BAY\n-122.28,37.81,17.0,924.0,289.0,609.0,289.0,1.5,162500.0,NEAR BAY\n-122.27,37.81,52.0,210.0,56.0,183.0,56.0,1.1667,112500.0,NEAR BAY\n-
122.28,37.81,52.0,340.0,97.0,200.0,87.0,1.5208,112500.0,NEAR BAY\n-122.28,37.81,52.0,386.0,164.0,346.0,155.0,0.8075,137500.0,NEAR BAY\n-122.28,37.81,35.0,948.0,184.0,467.0,169.0,1.8088,118800.0,NEAR BAY\n-
122.28,37.81,52.0,773.0,143.0,377.0,115.0,2.4083,98200.0,NEAR BAY\n-122.27,37.81,40.0,880.0,451.0,582.0,380.0,0.977,118800.0,NEAR BAY\n-122.27,37.81,10.0,875.0,348.0,546.0,330.0,0.76,162500.0,NEAR BAY\n-
122.27,37.8,10.0,105.0,42.0,125.0,39.0,0.9722,137500.0,NEAR BAY\n-122.27,37.8,52.0,249.0,78.0,396.0,85.0,1.2434,500001.0,NEAR BAY\n-122.27,37.8,16.0,994.0,392.0,800.0,362.0,2.0938,162500.0,NEAR BAY\n-
122.28,37.8,52.0,215.0,87.0,904.0,88.0,0.8668,137500.0,NEAR BAY\n-122.28,37.8,52.0,96.0,31.0,191.0,34.0,0.75,162500.0,NEAR BAY\n-122.27,37.79,27.0,1055.0,347.0,718.0,302.0,2.6354,187500.0,NEAR BAY\n-
122.27,37.8,39.0,1715.0,623.0,1327.0,467.0,1.8477,179200.0,NEAR BAY\n-122.26,37.8,36.0,5329.0,2477.0,3469.0,2323.0,2.0096,130000.0,NEAR BAY\n-122.26,37.82,31.0,4596.0,1331.0,2048.0,1180.0,2.8345,183800.0,NEAR BAY\n-
122.26,37.81,29.0,335.0,107.0,202.0,91.0,2.0062,125000.0,NEAR BAY\n-122.26,37.82,22.0,3682.0,1270.0,2024.0,1250.0,1.2185,170000.0,NEAR BAY\n-122.26,37.82,37.0,3633.0,1085.0,1838.0,980.0,2.6104,193100.0,NEAR BAY\n-
122.25,37.81,29.0,4656.0,1414.0,2304.0,1250.0,2.4912,257800.0,NEAR BAY\n-122.25,37.81,28.0,5806.0,1603.0,2563.0,1497.0,3.2177,273400.0,NEAR BAY\n-122.25,37.81,39.0,854.0,242.0,389.0,228.0,3.125,237500.0,NEAR BAY\n-
122.25,37.81,52.0,2155.0,701.0,895.0,613.0,2.5795,350000.0,NEAR BAY\n-122.26,37.81,34.0,5871.0,1914.0,2689.0,1789.0,2.8406,335700.0,NEAR BAY\n-122.24,37.82,52.0,1509.0,225.0,674.0,244.0,4.9306,313400.0,NEAR BAY\n-
122.24,37.81,52.0,2026.0,482.0,709.0,456.0,3.2727,268500.0,NEAR BAY\n-122.25,37.81,52.0,1758.0,460.0,686.0,422.0,3.1691,259400.0,NEAR BAY\n-122.24,37.82,52.0,3481.0,751.0,1444.0,718.0,3.9,275700.0,NEAR BAY\n-
122.25,37.82,28.0,3337.0,855.0,1520.0,802.0,3.9063,225000.0,NEAR BAY\n-122.25,37.82,52.0,1424.0,289.0,550.0,253.0,5.0917,262500.0,NEAR BAY\n-122.25,37.82,32.0,3809.0,1098.0,1806.0,1022.0,2.6429,218500.0,NEAR BAY\n-
122.25,37.82,26.0,3959.0,1196.0,1749.0,1217.0,3.0233,255000.0,NEAR BAY\n-122.25,37.83,52.0,2376.0,559.0,939.0,519.0,3.1484,224100.0,NEAR BAY\n-122.25,37.83,35.0,1613.0,428.0,675.0,422.0,3.4722,243100.0,NEAR BAY\n-
122.25,37.83,52.0,1279.0,287.0,534.0,291.0,3.1429,231600.0,NEAR BAY\n-122.25,37.83,28.0,5022.0,1750.0,2558.0,1661.0,2.4234,218500.0,NEAR BAY\n-122.25,37.83,52.0,4190.0,1105.0,1786.0,1037.0,3.0897,234100.0,NEAR BAY\n-
122.23,37.84,50.0,2515.0,399.0,970.0,373.0,5.8596,327600.0,NEAR BAY\n-122.23,37.84,47.0,3175.0,454.0,1098.0,485.0,5.2868,347600.0,NEAR BAY\n-122.24,37.83,41.0,2576.0,406.0,794.0,376.0,5.956,366100.0,NEAR BAY\n-
122.24,37.85,37.0,334.0,54.0,98.0,47.0,4.9643,335000.0,NEAR BAY\n-122.23,37.85,52.0,2800.0,411.0,1061.0,403.0,6.3434,373600.0,NEAR BAY\n-122.24,37.84,52.0,3529.0,574.0,1177.0,555.0,5.1773,389500.0,NEAR BAY\n-
122.24,37.85,52.0,2612.0,365.0,901.0,367.0,7.2354,391100.0,NEAR BAY\n-122.22,37.85,28.0,5287.0,1048.0,2031.0,956.0,5.457,337300.0,NEAR BAY\n-122.22,37.84,50.0,2935.0,473.0,1031.0,479.0,7.5,295200.0,NEAR BAY\n-
122.21,37.84,44.0,3424.0,597.0,1358.0,597.0,6.0194,292300.0,NEAR BAY\n-122.21,37.83,40.0,4991.0,674.0,1616.0,654.0,7.5544,411500.0,NEAR BAY\n-122.2,37.84,30.0,2211.0,346.0,844.0,343.0,6.0666,311500.0,NEAR BAY\n-
122.21,37.84,34.0,3038.0,490.0,1140.0,496.0,7.0548,325900.0,NEAR BAY\n-122.19,37.84,18.0,1617.0,210.0,533.0,194.0,11.6017,392600.0,NEAR BAY\n-122.2,37.84,35.0,2865.0,460.0,1072.0,443.0,7.4882,319300.0,NEAR BAY\n-
122.21,37.83,34.0,5065.0,788.0,1627.0,766.0,6.8976,333300.0,NEAR BAY\n-122.19,37.83,28.0,1326.0,184.0,463.0,190.0,8.2049,335200.0,NEAR BAY\n-122.2,37.83,26.0,1589.0,223.0,542.0,211.0,8.401,351200.0,NEAR BAY\n-
122.19,37.83,29.0,1791.0,271.0,661.0,269.0,6.8538,368900.0,NEAR BAY\n-122.19,37.82,32.0,1835.0,264.0,635.0,263.0,8.317,365900.0,NEAR BAY\n-122.2,37.82,37.0,1229.0,181.0,420.0,176.0,7.0175,366700.0,NEAR BAY\n-
122.2,37.82,39.0,3770.0,534.0,1265.0,500.0,6.3302,362800.0,NEAR BAY\n-122.18,37.81,30.0,292.0,38.0,126.0,52.0,6.3624,483300.0,NEAR BAY\n-122.21,37.82,52.0,2375.0,333.0,813.0,350.0,7.0549,331400.0,NEAR BAY\n-
122.2,37.81,45.0,2964.0,436.0,1067.0,426.0,6.7851,323500.0,NEAR BAY\n-122.21,37.8,50.0,2833.0,605.0,1260.0,552.0,2.8929,216700.0,NEAR BAY\n-122.21,37.8,38.0,2254.0,535.0,951.0,487.0,3.0812,233100.0,NEAR BAY\n-
122.21,37.81,52.0,1389.0,212.0,510.0,224.0,5.2402,296400.0,NEAR BAY\n-122.22,37.81,52.0,1971.0,335.0,765.0,308.0,6.5217,273700.0,NEAR BAY\n-122.22,37.8,52.0,2183.0,465.0,1129.0,460.0,3.2632,227700.0,NEAR BAY\n-
122.22,37.8,52.0,2286.0,464.0,1073.0,441.0,3.0298,199600.0,NEAR BAY\n-122.22,37.8,52.0,2721.0,541.0,1185.0,515.0,4.5428,239800.0,NEAR BAY\n-122.22,37.81,52.0,2024.0,339.0,756.0,340.0,4.072,270100.0,NEAR BAY\n-
```

We'll now begin working with Pandas. Pandas is the principle library for data management in python. It's primary mechanism of data storage is the dataframe, a two dimensional table, where each column represents a datatype, and each row a specific data element in the set.

To work with dataframes, we have to first read in the csv file and convert it to a dataframe using the code below.

```
[3] # We'll now import the holy grail of python datascience: Pandas!
    import pandas as pd
    housing = pd.read_csv('housing.csv')
```

```
[ ] housing.head() # show the first few elements of the dataframe
                   # typically this is the first thing you do
                   # to see how the dataframe looks like
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

A dataset may have different types of features

- real valued
- Discrete (integers)
- categorical (strings)
- Boolean

The two categorical features are essentialy the same as you can always map a categorical string/character to an integer.

In the dataset example, all our features are real valued floats, except ocean proximity which is categorical.

```
[ ] # to see a concise summary of data types, null values, and counts
    # use the info() method on the dataframe
    housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
[ ] # you can access individual columns similarly
    # to accessing elements in a python dict
    housing["ocean_proximity"].head() # added head() to avoid printing many columns..
```

```
0    NEAR BAY
1    NEAR BAY
2    NEAR BAY
3    NEAR BAY
4    NEAR BAY
Name: ocean_proximity, dtype: object
```

```
[ ] # to access a particular row we can use iloc
    housing.iloc[1]
```

```
longitude              -122.22
latitude                 37.86
housing_median_age        21.0
total_rooms             7099.0
total_bedrooms          1106.0
population              2401.0
households              1138.0
median_income           8.3014
median_house_value    358500.0
ocean_proximity       NEAR BAY
Name: 1, dtype: object
```

```
[ ] # one other function that might be useful is
    # value_counts(), which counts the number of occurences
    # for categorical features
    housing["ocean_proximity"].value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

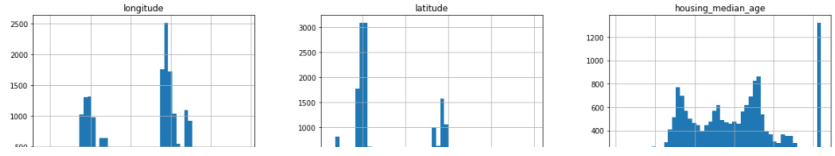
```
[ ] # The describe function compiles your typical statistics for each
    # column
    housing.describe()
```

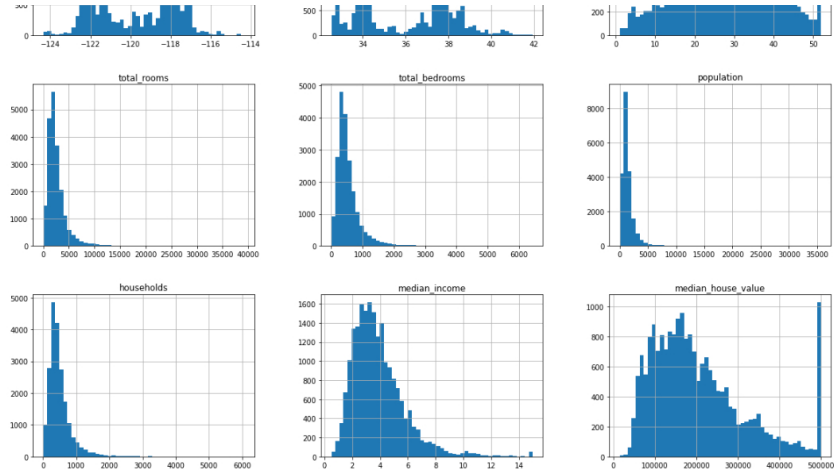
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section here

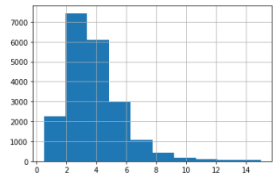
Let's start visualizing the dataset

```
[ ] # We can draw a histogram for each of the dataframes features
    # using the hist function
    housing.hist(bins=50, figsize=(20,15))
    # save_fig("attribute_histogram_plots")
    plt.show() # pandas internally uses matplotlib, and to display all the figures
               # the show() function must be called
```






```
[ ] # if you want to have a histogram on an individual feature:
    housing["median_income"].hist()
    plt.show()
```



We can convert a floating point feature to a categorical feature by binning or by defining a set of intervals.

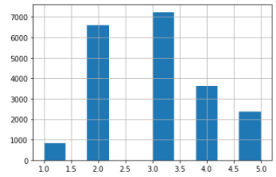
For example, to bin the households based on median_income we can use the pd.cut function

```
[ ] # assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
    housing["income_cat"] = pd.cut(housing["median_income"],
                                   bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                                   labels=[1, 2, 3, 4, 5])

    housing["income_cat"].value_counts()
```

```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

```
[ ] housing["income_cat"].hist()
```

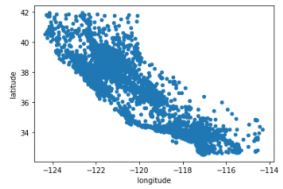
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb88abd0100>
```



Next let's visualize the household incomes based on latitude & longitude coordinates

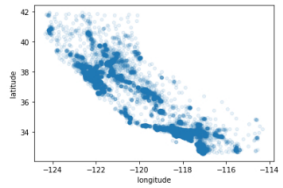
```
[ ] ## here's a not so interestting way plotting it
    housing.plot(kind="scatter", x="longitude", y="latitude")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb88cd1dd30>
```



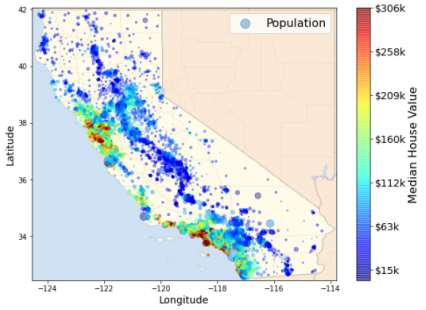

```
[ ] # we can make it look a bit nicer by using the alpha parameter,
    # it simply plots less dense areas lighter.
    housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb887b64bb0>
```




```
[ ] # A more interesting plot is to color code (heatmap) the dots
    # based on income. The code below achieves this

    # Please note: In order for this to work, ensure that you've loaded an image
    # of california (california.png) into this directory prior to running this

    import matplotlib.image as mpimg
    california_img=mpimg.imread('california.png')
    ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                          s=housing['population']/100, label="Population",
                          c="median_house_value", cmap=plt.get_cmap("jet"),
                          colorbar=False, alpha=0.4,
                         )
```

```
# overlay the california map on the plotted scatter plot
# note: plt.imshow still refers to the most recent figure
# that hasn't been plotted yet.
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on median_house_value feature
prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cb = plt.colorbar()
cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
cb.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
plt.show()
```



Not suprisingly, the most expensive houses are concentrated around the San Francisco/Los Angeles areas.

Up until now we have only visualized feature histograms and basic statistics.

When developing machine learning models the predictiveness of a feature for a particular target of intrest is what's important.

It may be that only a few features are useful for the target at hand, or features may need to be augmented by applying certain transfomrations.

None the less we can explore this using correlation matrices.

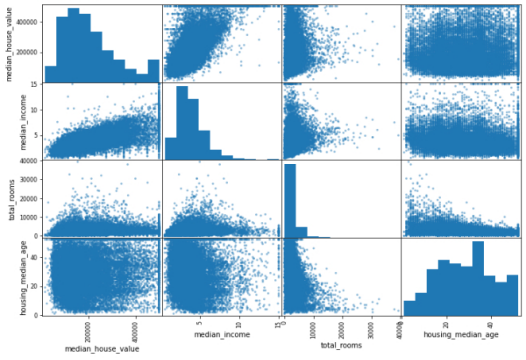
```
[ ] corr_matrix = housing.corr()
```

```
[ ] # for example if the target is "median_house_value", most correlated features can be sorted
    # which happens to be "median_income". This also intuitively makes sense.
    corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

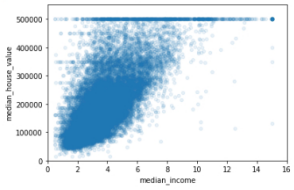
```
[ ] # the correlation matrix for different attributes/features can also be plotted
    # some features may show a positive correlation/negative correlation or
    # it may turn out to be completely random!
    from pandas.plotting import scatter_matrix
    attributes = ["median_house_value", "median_income", "total_rooms",
                  "housing_median_age"]
    scatter_matrix(housing[attributes], figsize=(12, 8))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fb88f536d90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb887d48a00>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb887d8ff10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb887fa4220>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fb88f693190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb88816b700>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb88816b970>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb888023370>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fb8882e4be0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb887f1c580>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb887d0ecd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb887ca4160>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fb887cac520>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb887da5910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb88dac1d00>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb887d85130>]],
      dtype=object)
```




```
[ ] # median income vs median house vlue plot plot 2 in the first row of top figure
    housing.plot(kind="scatter", x="median_income", y="median_house_value",
                 alpha=0.1)
    plt.axis([0, 16, 0, 550000])
```

```
(0.0, 16.0, 0.0, 550000.0)
```



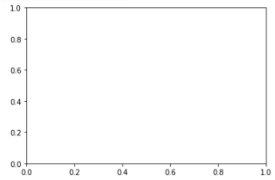

```
[ ] # obtain new correlations
    corr_matrix = housing.corr()
    corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

```
[ ] housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
                 alpha=0.2)
    plt.axis([0, 5, 0, 520000])
    plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

                                 8 frames

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'rooms_per_household'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364
   3365         if is_scalar(key) and isna(key) and not self.hasnans:

KeyError: 'rooms_per_household'
```

SEARCH STACK OVERFLOW



Preparing Dastaset for ML

Augmenting Features

New features can be created by combining different columns from our data set.

- rooms_per_household = total_rooms / households
- bedrooms_per_room = total_bedrooms / total_rooms
- etc.

```
[ ] housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
    housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
    housing["population_per_household"]=housing["population"]/housing["households"]
```

Dealing With Incomplete Data

```
[ ] # have you noticed when looking at the dataframe summary certain rows
    # contained null values? we can't just leave them as nulls and expect our
    # model to handle them for us...
    sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
    sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity	income_cat	rooms_per_household	bedrooms_per_room	population_per_household
290	-122.16	37.77	47.0	1256.0	NaN	570.0	218.0	4.3750	161900.0	NEAR BAY	3	5.761468	NaN	2.6146
341	-122.17	37.75	38.0	992.0	NaN	732.0	259.0	1.6196	85100.0	NEAR BAY	2	3.830116	NaN	2.8262
538	-122.28	37.78	29.0	5154.0	NaN	3741.0	1273.0	2.5762	173400.0	NEAR BAY	2	4.048704	NaN	2.9387
563	-122.24	37.75	45.0	891.0	NaN	384.0	146.0	4.9489	247100.0	NEAR BAY	4	6.102740	NaN	2.6301
696	-122.10	37.69	41.0	746.0	NaN	387.0	161.0	3.9063	178400.0	NEAR BAY	3	4.633540	NaN	2.4037

```
[ ] sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1: simply drop rows that have null values
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity	income_cat	rooms_per_household	bedrooms_per_room	population_per_household

```
[ ] sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop the complete feature
```

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	median_house_value	ocean_proximity	income_cat	rooms_per_household	bedrooms_per_room	population_per_household
290	-122.16	37.77	47.0	1256.0	570.0	218.0	4.3750	161900.0	NEAR BAY	3	5.761468	NaN	2.614679
341	-122.17	37.75	38.0	992.0	732.0	259.0	1.6196	85100.0	NEAR BAY	2	3.830116	NaN	2.826255
538	-122.28	37.78	29.0	5154.0	3741.0	1273.0	2.5762	173400.0	NEAR BAY	2	4.048704	NaN	2.938727
563	-122.24	37.75	45.0	891.0	384.0	146.0	4.9489	247100.0	NEAR BAY	4	6.102740	NaN	2.630137
696	-122.10	37.69	41.0	746.0	387.0	161.0	3.9063	178400.0	NEAR BAY	3	4.633540	NaN	2.403727

```
[ ] median = housing["total_bedrooms"].median()
    sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3: replace na values with median values
    sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity	income_cat	rooms_per_household	bedrooms_per_room	population_per_household
290	-122.16	37.77	47.0	1256.0	435.0	570.0	218.0	4.3750	161900.0	NEAR BAY	3	5.761468	NaN	2.6146
341	-122.17	37.75	38.0	992.0	435.0	732.0	259.0	1.6196	85100.0	NEAR BAY	2	3.830116	NaN	2.8262
538	-122.28	37.78	29.0	5154.0	435.0	3741.0	1273.0	2.5762	173400.0	NEAR BAY	2	4.048704	NaN	2.9387
563	-122.24	37.75	45.0	891.0	435.0	384.0	146.0	4.9489	247100.0	NEAR BAY	4	6.102740	NaN	2.6301
696	-122.10	37.69	41.0	746.0	435.0	387.0	161.0	3.9063	178400.0	NEAR BAY	3	4.633540	NaN	2.4037

Now that we've played around with this, lets finalize this approach by replacing the nulls in our final dataset

```
[ ] housing["total_bedrooms"].fillna(median, inplace=True)
```

Could you think of another plausible imputation for this dataset?

Dealing with Non-Numeric Data

So we're almost ready to feed our dataset into a machine learning model, but we're not quite there yet!

Generally speaking all models can only work with numeric data, which means that if you have Categorical data you want included in your model, you'll need to do a numeric conversion. We'll explore this more later, but for now we'll take one approach to converting our `ocean_proximity` field into a numeric one.

```
[ ] from sklearn.preprocessing import LabelEncoder

    # creating instance of labelencoder
    labelencoder = LabelEncoder()
    # Assigning numerical values and storing in another column
    housing['ocean_proximity'] = labelencoder.fit_transform(housing['ocean_proximity'])
    housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	3
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	3
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	3
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	3
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	3

Divide up the Dataset for Machine Learning

After having cleaned your dataset you're ready to train your machine learning model.

To do so you'll aim to divide your data into:

- train set
- test set

In some cases you might also have a validation set as well for tuning hyperparameters (don't worry if you're not familiar with this term yet..)

In supervised learning setting your train set and test set should contain (**feature**, **target**) tuples.

- **feature**: is the input to your model
- **target**: is the ground truth label
  - when target is categorical the task is a classification task
  - when target is floating point the task is a regression task

We will make use of **scikit-learn** python package for preprocessing.

Scikit learn is pretty well documented and if you get confused at any point simply look up the function/object!

```
[ ] from sklearn.model_selection import StratifiedShuffleSplit
    # let's first start by creating our train and test sets
    split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
    for train_index, test_index in split.split(housing, housing["income_cat"]):
        train_set = housing.loc[train_index]
        test_set = housing.loc[test_index]
```

```
[ ] housing_training = train_set.drop("median_house_value", axis=1) # drop labels for training set features
                                                                    # the input to the model should not contain the true label
    housing_labels = train_set["median_house_value"].copy()
```

```
[ ] housing_testing = test_set.drop("median_house_value", axis=1) # drop labels for training set features
                                                                  # the input to the model should not contain the true label
    housing__test_labels = test_set["median_house_value"].copy()
```

Select a model and train

Once we have prepared the dataset it's time to choose a model.

As our task is to predict the median_house_value (a floating value), regression is well suited for this.

```
[ ] from sklearn.linear_model import LinearRegression

    lin_reg = LinearRegression()
    lin_reg.fit(housing_training, housing_labels)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-d48e9837d7f2> in <module>
      2
      3 lin_reg = LinearRegression()
----> 4 lin_reg.fit(housing_training, housing_labels)
      5
      6

 4 frames
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py in _assert_all_finite(X, allow_nan, msg_dtype)
    112         ):
    113             type_err = "infinity" if allow_nan else "NaN, infinity"
--> 114             raise ValueError(
    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None else X.dtype

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

SEARCH STACK OVERFLOW

```
[ ] # let's try our model on a few testing instances
    data = housing_testing.iloc[:5]
    labels = housing__test_labels.iloc[:5]

    print("Predictions:", lin_reg.predict(data))
    print("Actual labels:", list(labels))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-35-0460b4868e4f> in <module>
      3 labels = housing__test_labels.iloc[:5]
      4
----> 5 print("Predictions:", lin_reg.predict(data))
      6 print("Actual labels:", list(labels))

 1 frames
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_base.py in _decision_function(self, X)
    344
    345         X = self._validate_data(X, accept_sparse=["csr", "csc", "coo"], reset=False)
--> 346         return safe_sparse_dot(X, self.coef_.T, dense_output=True) + self.intercept_
    347
    348     def predict(self, X):

AttributeError: 'LinearRegression' object has no attribute 'coef_'
```

SEARCH STACK OVERFLOW

We can evaluate our model using certain metrics, a fitting metric for regresison is the mean-squared-loss

$$L(\hat{Y}, Y) = \sum_{i}^{N} (\hat{y}_i - y_i)^2$$

where $\hat{y}$ is the predicted value, and y is the ground truth label.

```
[ ] from sklearn.metrics import mean_squared_error

    preds = lin_reg.predict(housing_testing)
    mse = mean_squared_error(housing__test_labels, preds)
    rmse = np.sqrt(mse)
    rmse
```

```
68330.90371034428
```

Is this a good result? What do you think an acceptable error rate is for this sort of problem?

TODO: Applying the end-end ML steps to a different dataset.

Ok now it's time to get to work! We will apply what we've learnt to another dataset (airbnb dataset). For this project we will attempt to **predict the airbnb rental price based on other features in our given dataset.**

Visualizing Data

Load the data + statistics

Let's do the following set of tasks to get us warmed up:

- load the dataset
- display the first few rows of the data
- drop the following columns: name, host_id, host_name, last_review, neighbourhood
- display a summary of the statistics of the loaded data

```
[4] airbnb = pd.read_csv("AB_NYC_2019.csv")
    airbnb.head()
```

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	avail
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	1	9	2018-10-19	0.21	6	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	1	45	2019-05-21	0.38	2	
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150	3	0	NaN	NaN	1	
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	1	270	2019-07-05	4.64	1	
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	10	9	2018-11-19	0.10	1	

```
[5] airbnb = airbnb.drop(["name", "host_id", "host_name", "last_review", "neighbourhood"], axis=1)
    airbnb.describe()
```

	id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365
count	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	38843.000000	48895.000000	48895.000000
mean	1.901714e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.373221	7.143982	112.781327
std	1.098311e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.680442	32.952519	131.622289
min	2.539000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.010000	1.000000	0.000000
25%	9.471945e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.190000	1.000000	0.000000
50%	1.967728e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	0.720000	1.000000	45.000000
75%	2.915218e+07	40.763115	-73.936275	175.000000	5.000000	24.000000	2.020000	2.000000	227.000000
max	3.648724e+07	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.500000	327.000000	365.000000

Some Basic Visualizations

Let's try another popular python graphics library: Plotly.

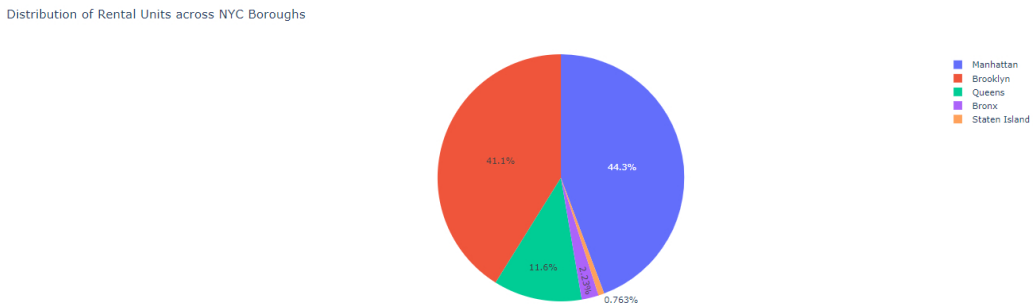
You can find documentation and all the examples you'll need here: Plotly Documentation

Let's start out by getting a better feel for the distribution of rentals in the market.

Generate a pie chart showing the distribution of rental units across NYC's 5 Buroughs (`neighbourhood_groups` in the dataset)

```
[6] import plotly.express as px
```

```
[8] groups = ["Manhattan", "Brooklyn", "Queens", "Bronx", "Staten Island"]
    pie_slices = airbnb["neighbourhood_group"].value_counts()
    fig = px.pie(pie_slices, values='neighbourhood_group', names=groups, title="Distribution of Rental Units across NYC Boroughs")
    fig.show()
```



Plot the total number_of_reviews per neighbourhood_group

We now want to see the total number of reviews left for each neighborhood group in the form of a Bar Chart (where the X-axis is the neighbourhood group and the Y-axis is a count of review.

This is a two step process:

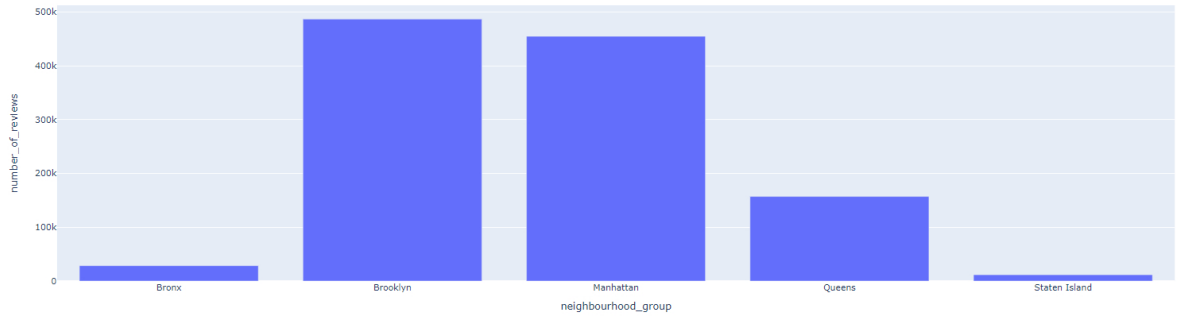
1. You'll have to sum up the reviews per neighbourhood group **(hint! try using the groupby function)**
2. Then use Plotly to generate the graph

```
[7] reviews = airbnb.groupby(['neighbourhood_group']).sum()
    reviews.head()
```

neighbourhood_group	id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365
Bronx	24803796188	44565.50071	-8.060805e+04	95459	4976	28371	1609.94	2437	180843
Brooklyn	367035792483	817931.96821	-1.486715e+06	2500600	121761	486574	21104.98	45925	2015070

Manhattan	406683958152	883012.01680	-1.602364e+06	4264527	185833	454569	21158.08	277073	2425586
Queens	123263814259	230784.85597	-4.185631e+05	563867	29358	156950	8879.05	23005	818464
Staten Island	8055857451	15147.61383	-2.764147e+04	42825	1802	11541	587.99	865	74480

```
[9] fig = px.bar(data_frame=reviews,y='number_of_reviews')

    fig.show()

    # reviews.sum().plot(kind='bar')
    # plt.ylabel("Number of Reviews")
    # plt.xlabel("Neighbourhood Group")
    # plt.title("Number of Reviews per Neighbourhood Group")
```

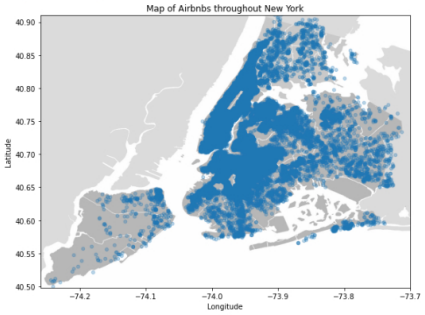


Plot a map of airbnbs throughout New York (if it gets too crowded take a subset of the data, and try to make it look nice if you can :)).

For reference you can use the Matplotlib code above to replicate this graph here.

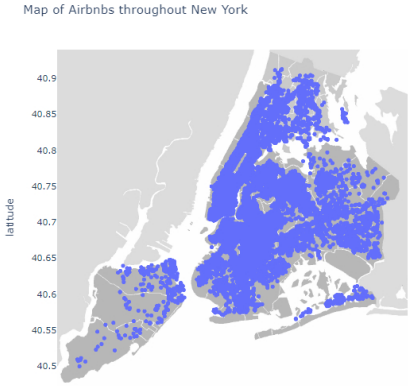
```
[10] import matplotlib.image as mpimg
     import os
     ny_img=mpimg.imread('nyc.png')

     airbnb.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7), alpha=0.3)

     plt.imshow(ny_img, extent=[-74.26, -73.7, 40.498, 40.91], alpha=0.5)
     plt.ylabel("Latitude")
     plt.xlabel("Longitude")
     plt.title("Map of Airbnbs throughout New York")
```

```
Text(0.5, 1.0, 'Map of Airbnbs throughout New York')
```



Now try to recreate this plot using Plotly's Scatterplot functionality. Note that the increased interactivity of the plot allows for some very cool functionality

```
[11] import plotly.graph_objects as go
     from PIL import Image

     nyc = Image.open("/content/nyc.png")
     fig = px.scatter(airbnb, x="longitude", y="latitude", title="Map of Airbnbs throughout New York")

     fig.add_layout_image(
            source=nyc,
            opacity=0.5,
            xref="paper",
            yref="paper",
            x=0,
            y=1,
            xanchor="left",
            yanchor="top",
            layer="below",
            sizing="stretch",
            sizex=1.0,
            sizey=1.0,
        )
     fig.update_layout(template="plotly_white", width=650, height=650,
                       xaxis_showgrid=False, yaxis_showgrid=False)
     fig.show()
```



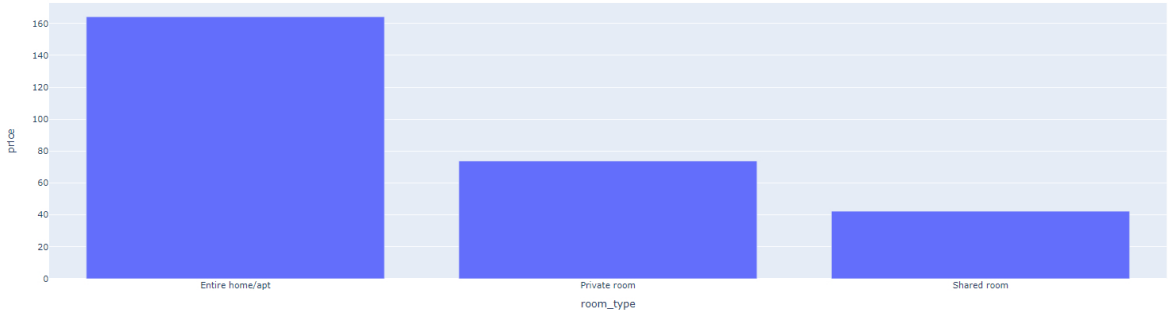
Use Plotly to plot the average price of room types in Brooklyn who have at least 10 Reviews.

Like with the previous example you'll have to do a little bit of data engineering before you actually generate the plot.

Generally I'd recommend the following series of steps:

1. Filter the data by neighborhood group and number of reviews to arrive at the subset of data relevant to this graph.
2. Groupby the room type
3. Take the mean of the price for each roomtype group
4. FINALLY (seriously!?!?) plot the result

```
[12] data = airbnb.where((airbnb['number_of_reviews'] >= 10) & (airbnb['neighbourhood_group'] == 'Brooklyn')).groupby(['room_type']).mean()

     data.head()
     fig = px.bar(data_frame=data, y='price')
     fig.show()
```




```
[ ]
```

Prepare the Data

Feature Engineering

Let's create a new binned feature, `price_cat` that will divide our dataset into quintiles (1-5) in terms of price level (you can choose the levels to assign)

Do a value count to check the distribution of values

```
[13] # assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
     airbnb["price_cat"] = pd.cut(airbnb["price"],
                                  bins=5,
                                  labels=[1, 2, 3, 4, 5])

     airbnb["price_cat"].value_counts()
```

```
1    48809
2       54
3       16
4        9
5        7
Name: price_cat, dtype: int64
```

Now engineer at least one new feature.

```
[14] airbnb["reviews_per_min_nights"] = airbnb["number_of_reviews"]/airbnb["minimum_nights"]
     airbnb.head()
```

	id	neighbourhood_group	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365	price_cat	reviews_per_min_nights
0	2539	Brooklyn	40.64749	-73.97237	Private room	149	1	9	0.21	6	365	1	9.0
1	2595	Manhattan	40.75362	-73.98377	Entire home/apt	225	1	45	0.38	2	355	1	45.0
2	3647	Manhattan	40.80902	-73.94190	Private room	150	3	0	NaN	1	365	1	0.0
3	3831	Brooklyn	40.68514	-73.95976	Entire home/apt	89	1	270	4.64	1	194	1	270.0
4	5022	Manhattan	40.79851	-73.94399	Entire home/apt	80	10	9	0.10	1	0	1	0.9

Data Imputation

Determine if there are any null-values and if there are impute them.

```
[15] sample_incomplete_rows = airbnb[airbnb.isnull().any(axis=1)].head()

     # remove last_review column
     # airbnb = airbnb.drop(["last_review"], axis=1)

     # set reviews_per_month as median
     median = airbnb["reviews_per_month"].median()
     airbnb["reviews_per_month"].fillna(median, inplace=True)

     # set host_name to Spider-Man
     # placeholder = "Spider-Man"
     # airbnb["host_name"].fillna(placeholder, inplace=True)

     # set name to COOL AIRBNB
     # placeholder2 = "COOL AIRBNB"
     # airbnb["name"].fillna(placeholder2, inplace=True)

     sample_incomplete_rows
```

	id	neighbourhood_group	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365	price_cat	reviews_per_min_nights
2	3647	Manhattan	40.80902	-73.94190	Private room	150	3	0	NaN	1	365	1	0.0
19	7750	Manhattan	40.79685	-73.94872	Entire home/apt	190	7	0	NaN	2	249	1	0.0
26	8700	Manhattan	40.86754	-73.92639	Private room	80	4	0	NaN	1	0	1	0.0
36	11452	Brooklyn	40.68876	-73.94312	Private room	35	60	0	NaN	1	365	1	0.0
38	11943	Brooklyn	40.63702	-73.96327	Private room	150	1	0	NaN	1	365	1	0.0

Numeric Conversions

Finally, review what features in your dataset are non-numeric and convert them.

```
[16] from sklearn.preprocessing import LabelEncoder

     #airbnb.head()

     # creating instance of labelencoder
     labelencoder = LabelEncoder()
     # Assigning numerical values and storing in another column
     # airbnb['name'] = labelencoder.fit_transform(airbnb['name'])
     # airbnb['neighbourhood'] = labelencoder.fit_transform(airbnb['neighbourhood'])
     airbnb['neighbourhood_group'] = labelencoder.fit_transform(airbnb['neighbourhood_group'])
     airbnb['room_type'] = labelencoder.fit_transform(airbnb['room_type'])
     airbnb.head()
```

	id	neighbourhood_group	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365	price_cat	reviews_per_min_nights
0	2539	1	40.64749	-73.97237	1	149	1	9	0.21	6	365	1	9.0
1	2595	2	40.75362	-73.98377	0	225	1	45	0.38	2	355	1	45.0
2	3647	2	40.80902	-73.94190	1	150	3	0	0.72	1	365	1	0.0
3	3831	1	40.68514	-73.95976	0	89	1	270	4.64	1	194	1	270.0
4	5022	2	40.79851	-73.94399	0	80	10	9	0.10	1	0	1	0.9

Prepare data for Machine Learning

Set aside 20% of the data as test test (80% train, 20% test).

Using our `StratifiedShuffleSplit` function example from above, let's split our data into a 80/20 Training/Testing split using `neighbourhood_group` to partition the dataset

```
[17] from sklearn.model_selection import StratifiedShuffleSplit
     # let's first start by creating our train and test sets
     split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
     for train_index, test_index in split.split(airbnb, airbnb["neighbourhood_group"]):
         train_set = airbnb.loc[train_index]
         test_set = airbnb.loc[test_index]

     print(train_set.shape)
     print(test_set.shape)
```

```
(39116, 13)
(9779, 13)
```

Finally, remove your labels `price` from your testing and training cohorts, and create separate label features.

```
[18] airbnb_training = train_set.drop("price", axis=1) # drop labels for training set features
                                                       # the input to the model should not contain the true label
     airbnb_labels = train_set["price"].copy()


     airbnb_testing = test_set.drop("price", axis=1) # drop labels for training set features
                                                     # the input to the model should not contain the true label
     airbnb_test_labels = test_set["price"].copy()
```

Fit a model of your choice

The task is to predict the price, you could refer to the housing example on how to train and evaluate your model using **MSE**. Provide both **test and train set MSE values**.

```
[19] from sklearn.linear_model import LinearRegression

     lin_reg = LinearRegression()
     lin_reg.fit(airbnb_training, airbnb_labels)

     data = airbnb_testing.iloc[:5]
     labels = airbnb_test_labels.iloc[:5]

     linearRegressionPrediction = lin_reg.predict(data)
     actualLabels = list(labels)

     print("Predictions:", linearRegressionPrediction)
     print("Actual labels:", actualLabels)
```

```
Predictions: [128.8387445  -43.01601957 201.18198088 201.87461609 204.77704443]
Actual labels: [120, 35, 200, 75, 150]
```

```
from sklearn.metrics import mean_squared_error

pred = lin_reg.predict(airbnb_training)
mse2 = mean_squared_error(airbnb_labels, pred)
print("Train")
print(mse2)

preds = lin_reg.predict(airbnb_testing)
mse = mean_squared_error(airbnb_test_labels, preds)
print("Test")
print(mse)
```

```
Train
15639.185915400443
Test
17054.008951878004
```