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.

In [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.

In [2]:

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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14428\1613494533.py in <module>
----> 1 from google.colab import files
      2 files.upload()

ModuleNotFoundError: No module named 'google'
```

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.

In [ ]:

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

In [ ]:

```
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
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342

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.

In [ ]:

```
# 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):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [ ]:

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

Out[ ]:

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

In [ ]:

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

Out[ ]:

```
longitude              -122.22
latitude                 37.86
housing_median_age          21
total_rooms               7099
total_bedrooms            1106
population                2401
households                1138
median_income           8.3014
median_house_value      358500
ocean_proximity       NEAR BAY
Name: 1, dtype: object
```

In [ ]:

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

Out[ ]:

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

In [ ]:

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

Out[ ]:

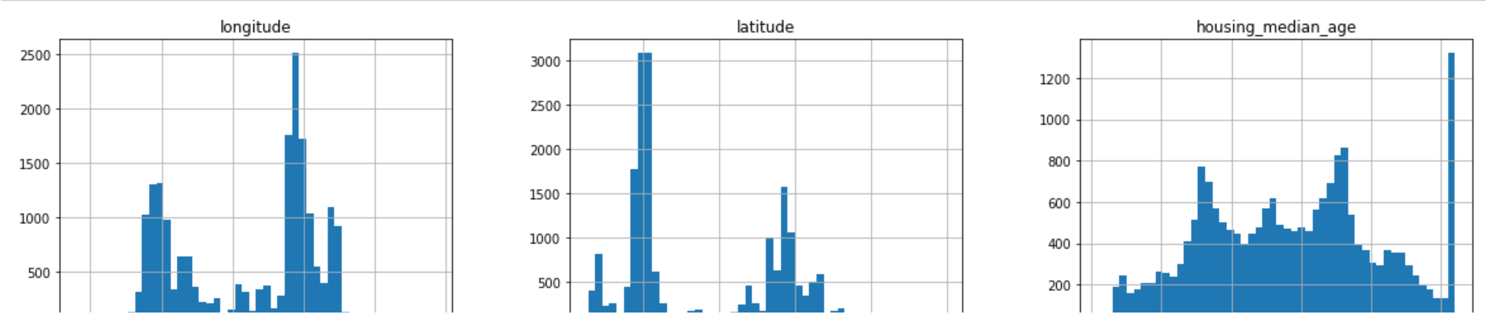
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.0000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.8706
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.8998
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.4999
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.5634
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.5348
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.7432
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.0001

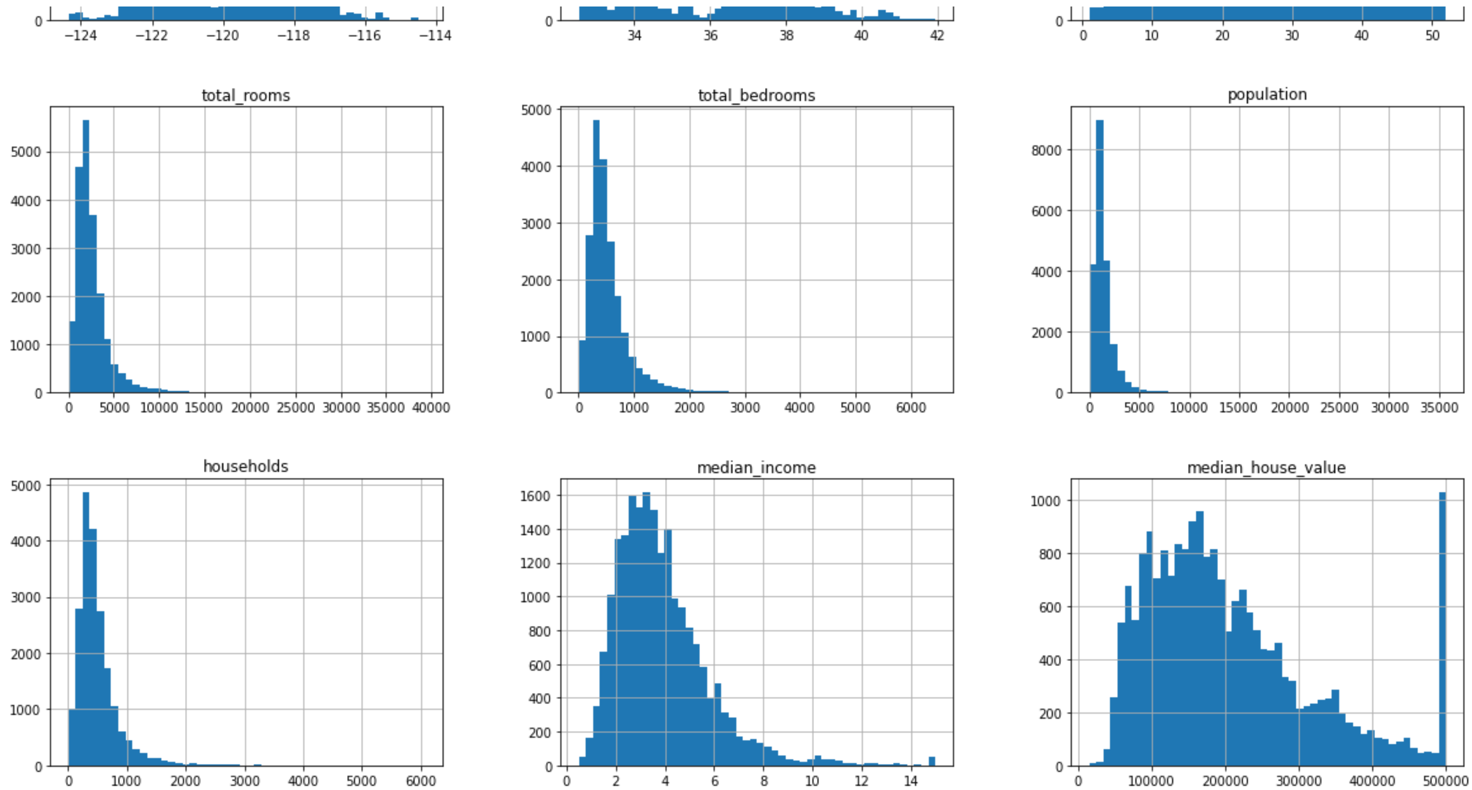
If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section [here](#)

Let's start visualizing the dataset

In [ ]:

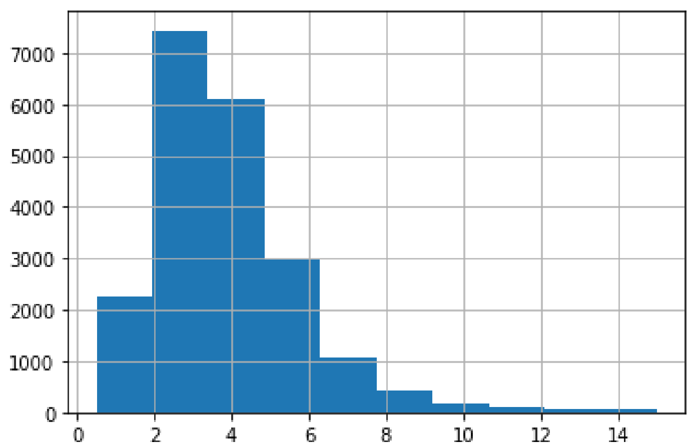
```
# We can draw a histogram for each of the dataframes features
# using the hist function
housing.hist(bins=50, figsize=(20,15))
# save_fig("attribute_histogram_plots")
plt.show()  # pandas internally uses matplotlib, and to display all the figures
            # the show() function must be called
```





In [ ]:

```
# 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

In [ ]:

```
# 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()
```

Out[ ]:

```
3    7236
2    6581
4    3639
5    2362
```

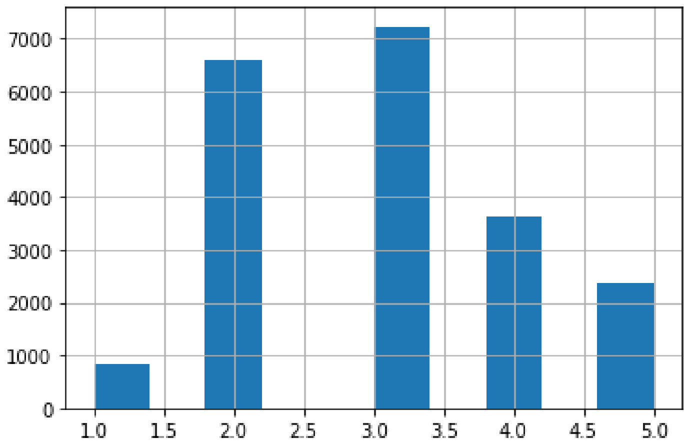
```
1      822
Name: income_cat, dtype: int64
```

In [ ]:

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

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbd2970b6a0>
```



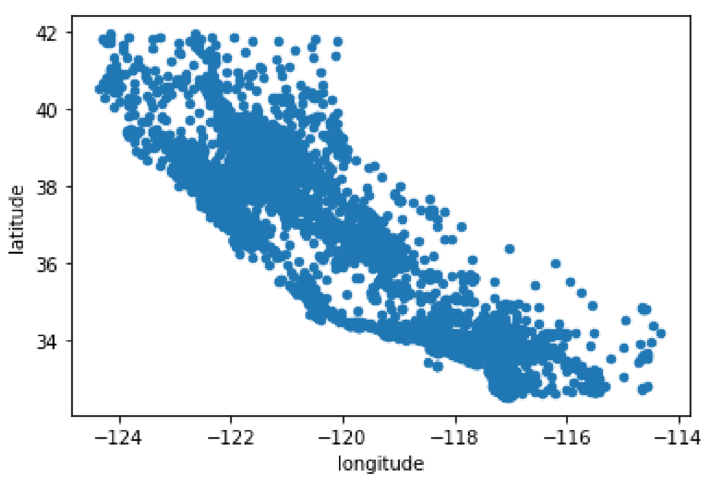
Next let's visualize the household incomes based on latitude & longitude coordinates

In [ ]:

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

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbd2955b220>
```

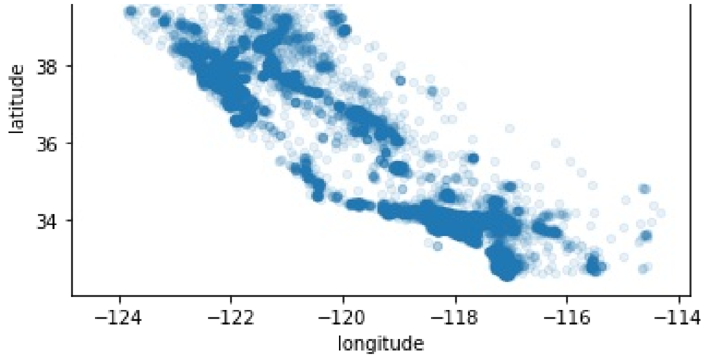


In [ ]:

```
# 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)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbd2952d3d0>
```



In [ ]:

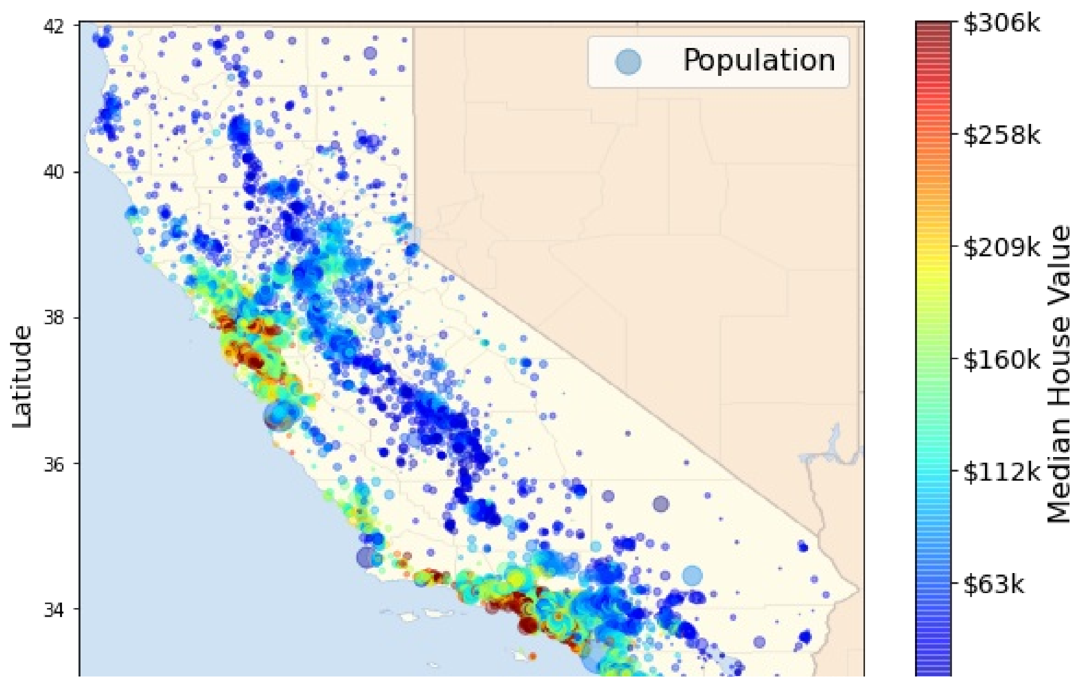
```
# A more interesting plot is to color code (heatmap) the dots
# based on income. The code below achieves this

# 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 califronia 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.

In [ ]:

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

In [ ]:

```
# 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)
```

Out[ ]:

```
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
```

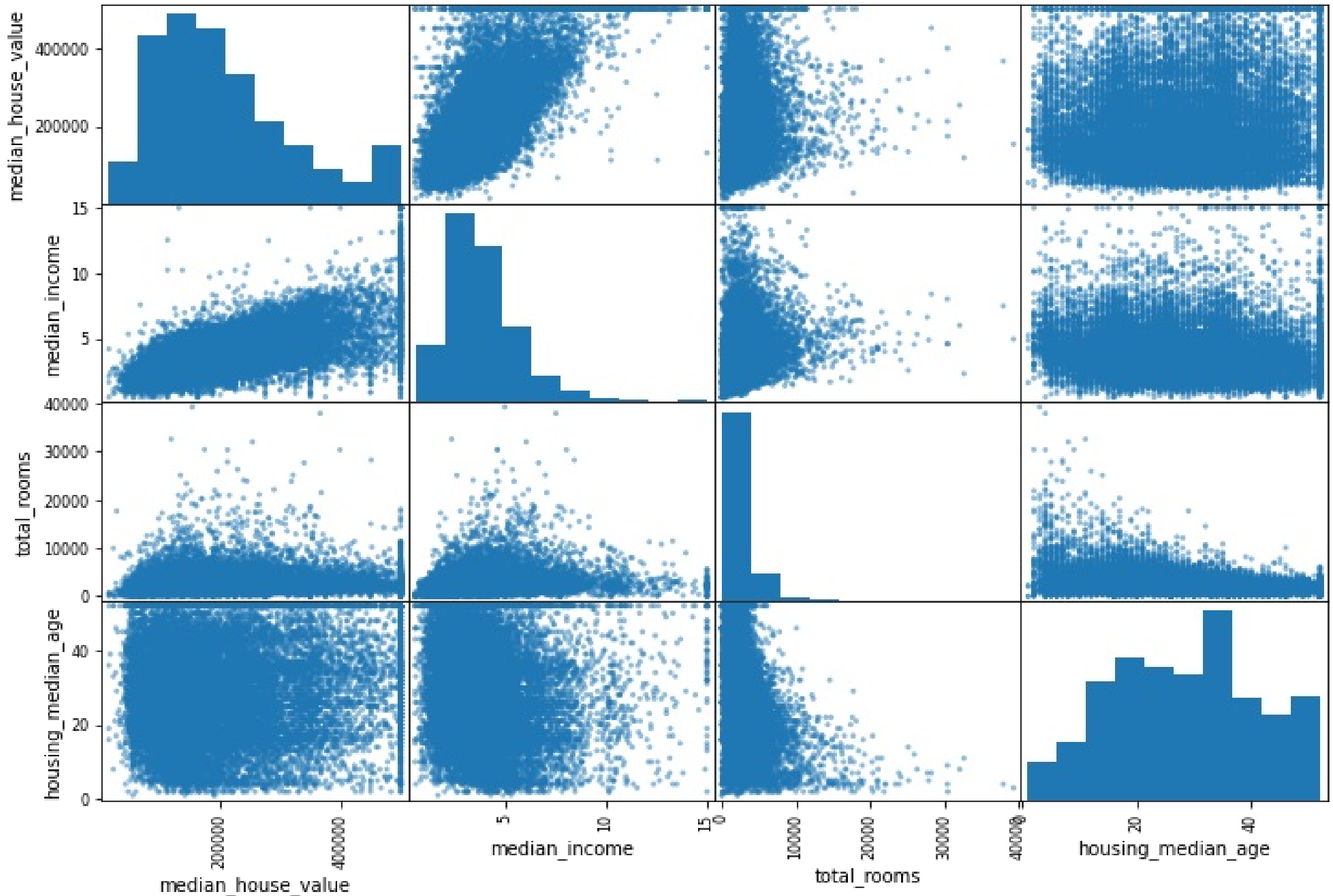
In [ ]:

```
# 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))
```

Out[ ]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fbd296e89a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29dd32b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29ec66d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29da5ac0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29d07850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29c9d220>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29c9d310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29ce1760>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29fdfeb0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29d5cf70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29e5f550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29ea9c70>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29e083d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd29c07af0>,
```

```
  <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd2a010250>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x7fbd2a141970>]],
dtype=object)
```

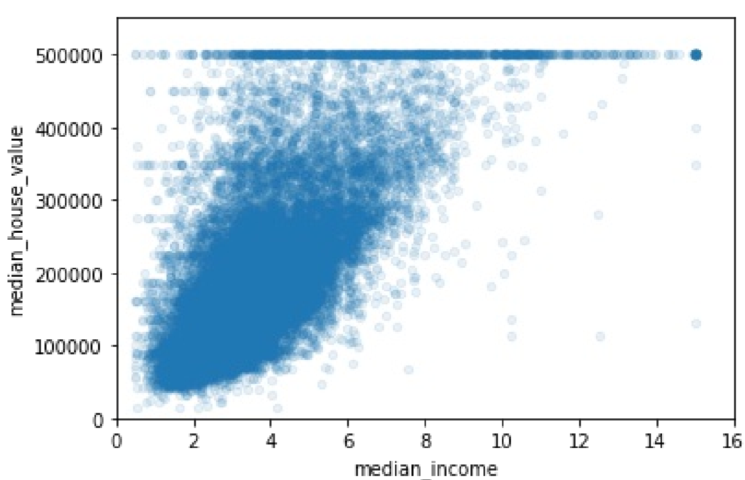


In [ ]:

```
# 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])
```

Out[ ]:

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



In [ ]:

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

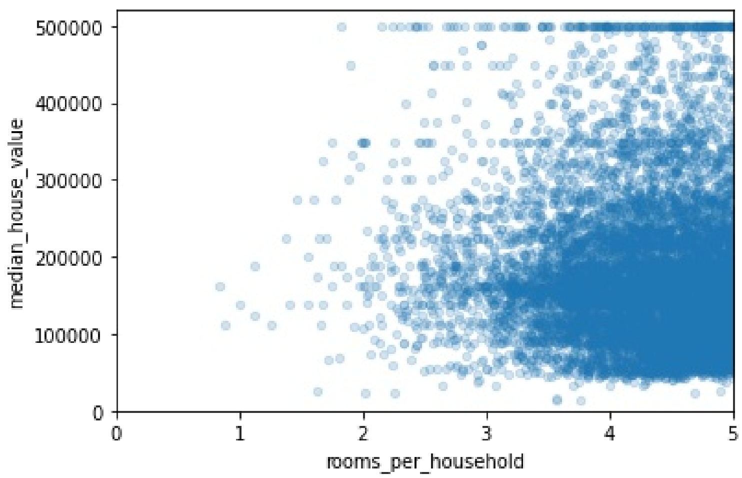
Out[ ]:

Out[ ]:

```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
total_bedrooms              0.049686
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.255880
Name: median_house_value, dtype: float64
```

In [ ]:

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



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.**

In [ ]:

```
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

In [ ]:

```
# 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
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_v

In [ ]:

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

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_v

In [ ]:

```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop the complete f
eature
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	median_house_value	ocean_p
290	-122.16	37.77	47.0	1256.0	570.0	218.0	4.3750	161900.0	NE
341	-122.17	37.75	38.0	992.0	732.0	259.0	1.6196	85100.0	NE
538	-122.28	37.78	29.0	5154.0	3741.0	1273.0	2.5762	173400.0	NE
563	-122.24	37.75	45.0	891.0	384.0	146.0	4.9489	247100.0	NE
696	-122.10	37.69	41.0	746.0	387.0	161.0	3.9063	178400.0	NE

In [ ]:

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

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_hous
290	-122.16	37.77	47.0	1256.0	435.0	570.0	218.0	4.3750	1
341	-122.17	37.75	38.0	992.0	435.0	732.0	259.0	1.6196	
538	-122.28	37.78	29.0	5154.0	435.0	3741.0	1273.0	2.5762	1
563	-122.24	37.75	45.0	891.0	435.0	384.0	146.0	4.9489	2
696	-122.10	37.69	41.0	746.0	435.0	387.0	161.0	3.9063	1

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

In [ ]:

```
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.

In [ ]:

```
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()
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342

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](https://scikit-learn.org) python package for preprocessing.

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

In [ ]:

```
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]
```

In [ ]:

```
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()
```

In [ ]:

```
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.

In [ ]:

```
from sklearn.linear_model import LinearRegression

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

```
Predictions: [412720.84851599 308250.79784537 236395.96466479 191878.53496752
 252722.41196865]
Actual labels: [500001.0, 162500.0, 204600.0, 159700.0, 184000.0]
```

In [ ]:

```
# 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))
```

```
Predictions: [412720.84851599 308250.79784537 236395.96466479 191878.53496752
 252722.41196865]
Actual labels: [500001.0, 162500.0, 204600.0, 159700.0, 184000.0]
```

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.

In [ ]:

```
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
```

Out[ ]:

```
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**

In [ ]:

```
airbnb = pd.read_csv('./datasets/airbnb/AB_NYC_2019.csv')
airbnb.head()
```

Out[ ]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	min
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150	
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	

In [ ]:

```
airbnb.drop(columns=["name", "host_id", "host_name", "last_review"])
airbnb.describe()
```

Out[ ]:

Out[ ]:

	id	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_r
count	4.889500e+04	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	38843.0
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.3
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.6
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.0
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.1
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	0.7
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.000000	5.000000	24.000000	2.0
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.5

Some Basic Visualizations

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

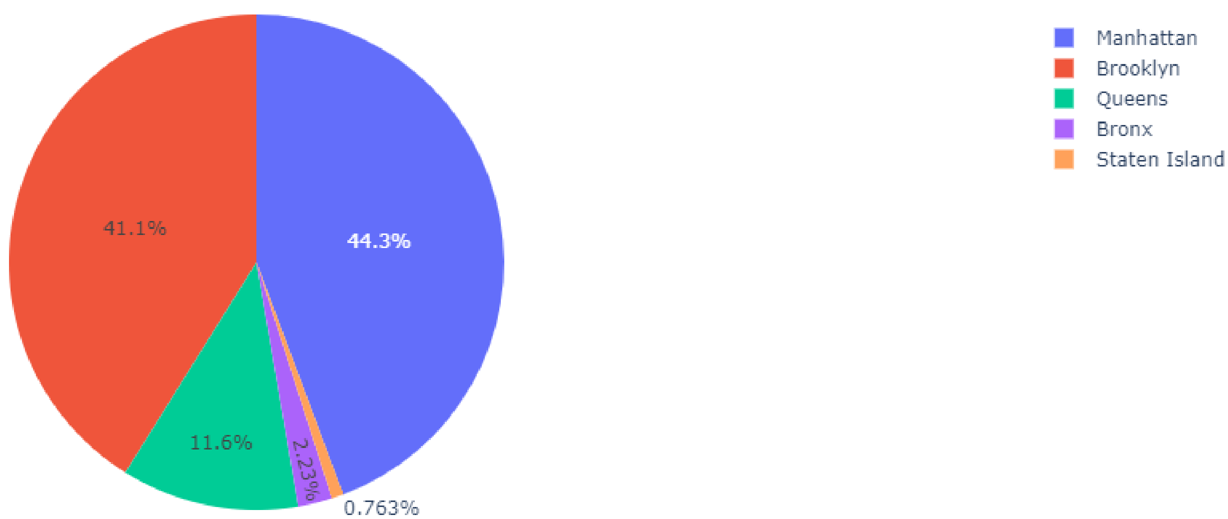
You can find documentation and all the examples you'll need here: [Plotly Documentation]

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)

In [ ]:

```
import plotly.express as px
fig = px.pie(airbnb, names="neighbourhood_group")
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:

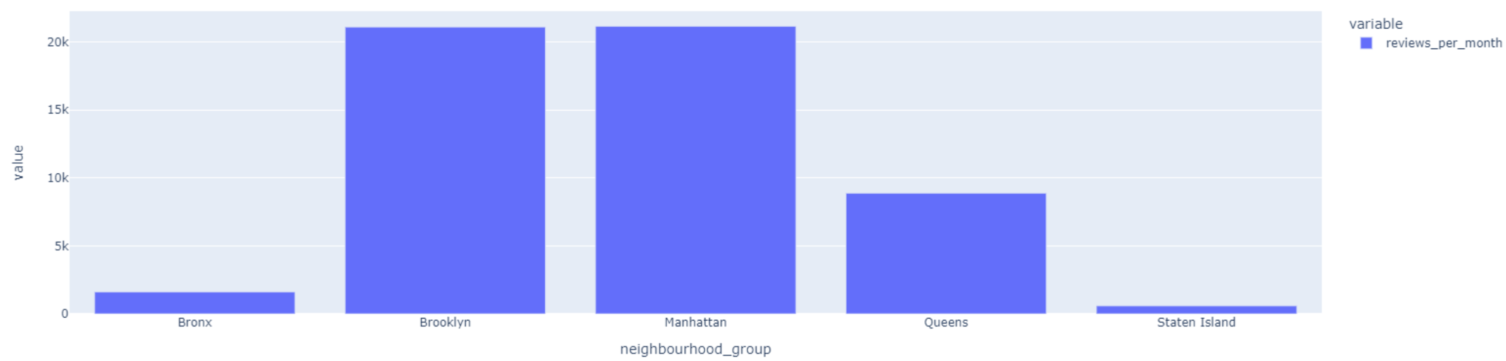
1. **You'll have to sum up the reviews per neighbourhood group (hint! try using the groupby function)**
2. **Then use Plotly to generate the graph**

In [ ]:

```
total = airbnb.groupby("neighbourhood_group")["reviews_per_month"].sum()
```

In [ ]:

```
px.bar(total)
```



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.

In [ ]:

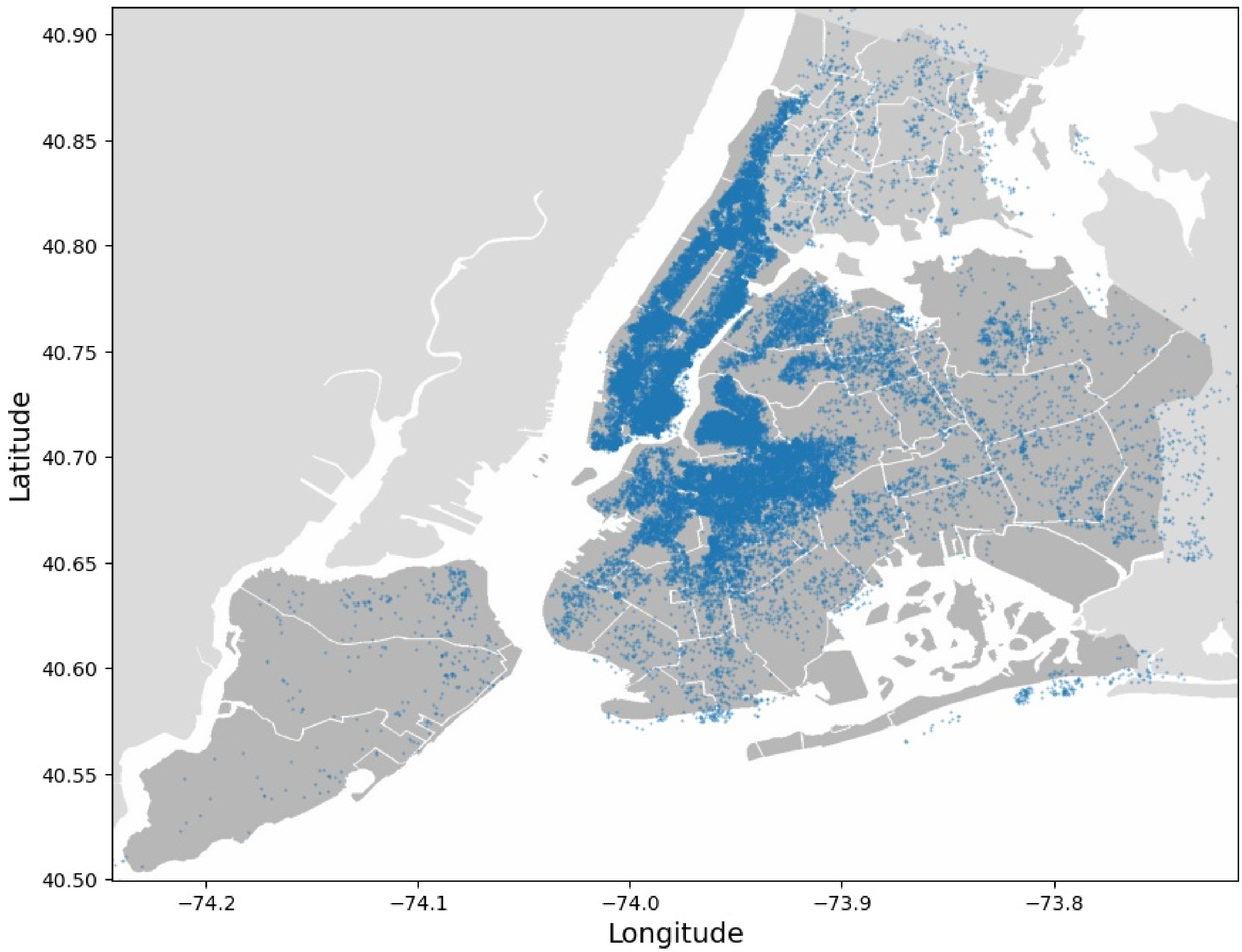
```
import matplotlib.image as mpimg
```

```
import matplotlib.image as mpimg
nyc_img=mpimg.imread('./images/nyc.png')
ax = airbnb.plot(kind="scatter", x="longitude", y="latitude", figsize=(15, 8), alpha=0.5, s
=0.25)

plt.imshow(nyc_img, extent=[-74.2444, -73.713, 40.4998, 40.9131], alpha=0.5)
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)
```

Out[ ]:

```
Text(0.5, 0, 'Longitude')
```



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

In [ ]:

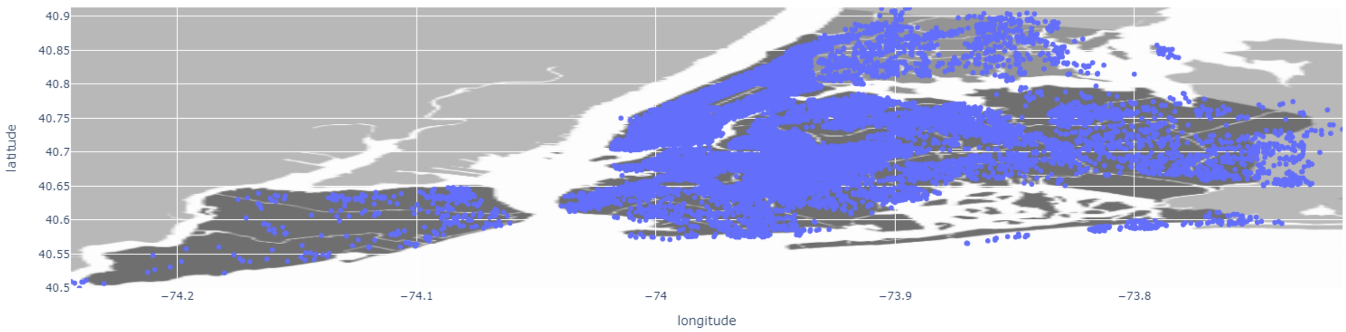
```
import base64
#set a local image as a background
image_filename = './images/nyc.png'
plotly_logo = base64.b64encode(open(image_filename, 'rb').read())

fig = px.scatter(airbnb, x="longitude", y="latitude")
fig.update_layout(
                images= [dict(
                    source='data:image/png;base64,{}'.format(plotly_logo.decode()),
                    xref="paper", yref="paper",
                    x=0, y=1,
```

```
                    sizex=1, sizey=1,
                    xanchor="left",
                    yanchor="top",
                    sizing="stretch",
                    layer="below")])

fig.update_xaxes(range=[-74.2444, -73.713])
fig.update_yaxes(range=[40.4998, 40.9131])

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**

In [ ]:

```
avg_prices_room_type = airbnb.where(airbnb["number_of_reviews"] > 10).groupby("room_type").
mean()

fig = px.bar(avg_prices_room_type, 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

In [ ]:

```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2)

airbnb["price_cat"] = pd.cut(airbnb["price"], bins=[-1., 80., 160., 240., 320., np.inf], la
bels=[1,2,3,4,5])

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

Out[ ]:

```
2    17878
1    17098
3     7396
4     3358
5     3165
Name: price_cat, dtype: int64
```

Now engineer at least one new feature.

In [ ]:

```
airbnb["max_stays"] = airbnb["availability_365"] / airbnb["minimum_nights"]
airbnb.head()
```

Out[ ]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	min
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150	
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	

Data Imputation

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

In [ ]:

```
airbnb["reviews_per_month"].fillna(value=0.0, inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48895 non-null  int32
 2   host_id                         48895 non-null  int64
 3   host_name                       48895 non-null  int32
 4   neighbourhood_group             48895 non-null  int32
 5   neighbourhood                   48895 non-null  int32
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  int32
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     48895 non-null  int32
 13  reviews_per_month               48895 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
 16  price_cat                       48895 non-null  category
 17  max_stays                       48895 non-null  float64
dtypes: category(1), float64(4), int32(6), int64(7)
memory usage: 5.3 MB
```

Numeric Conversions

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

In [ ]:

```python
from sklearn.preprocessing import LabelEncoder

airbnb.info()

labelencoder = LabelEncoder()

airbnb['name'] = labelencoder.fit_transform(airbnb['name'])
airbnb['host_name'] = labelencoder.fit_transform(airbnb['host_name'])
airbnb['neighbourhood_group'] = labelencoder.fit_transform(airbnb['neighbourhood_group'])
airbnb['neighbourhood'] = labelencoder.fit_transform(airbnb['neighbourhood'])
airbnb['room_type'] = labelencoder.fit_transform(airbnb['room_type'])
airbnb['last_review'] = labelencoder.fit_transform(airbnb['last_review'])
airbnb.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48895 non-null  int32
 2   host_id                         48895 non-null  int64
 3   host_name                       48895 non-null  int32
 4   neighbourhood_group             48895 non-null  int32
 5   neighbourhood                   48895 non-null  int32
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  int32
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     48895 non-null  int32
 13  reviews_per_month               48895 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
 16  price_cat                       48895 non-null  category
 17  max_stays                       48895 non-null  float64
dtypes: category(1), float64(4), int32(6), int64(7)
memory usage: 5.3 MB
```

Out[ ]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_night
0	2539	12328	2787	4989	1	108	40.64749	-73.97237	1	149	1
1	2595	37455	2845	4785	2	127	40.75362	-73.98377	0	225	1
2	3647	43543	4632	2909	2	94	40.80902	-73.94190	1	150	3
3	3831	14783	4869	6203	1	41	40.68514	-73.95976	0	89	1
4	5022	18693	7192	5923	2	61	40.79851	-73.94399	0	80	10

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

In [ ]:

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2)
for train_index, test_index in split.split(airbnb, airbnb["neighbourhood_group"]):
    train_set = airbnb.loc[train_index]
    test_set = airbnb.loc[test_index]
```

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

In [ ]:

```
airbnb_training = train_set.drop("price", axis=1)
airbnb_labels = train_set["price"].copy()

airbnb_testing = test_set.drop("price", axis=1)
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 .

In [ ]:

```
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]

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

```
Predictions: [ 28.31021694 122.39585103  56.83625629 112.66328421 235.24332501]
Actual labels: [50, 150, 37, 160, 222]
```

In [ ]:

```
from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(airbnb_testing)
mse = mean_squared_error(airbnb__test_labels, preds)
print ("Test MSE: ", mse)

preds = lin_reg.predict(airbnb_training)
mse = mean_squared_error(airbnb_labels, preds)
print ("Train MSE: ", mse)
```

```
Test MSE:  37479.49398875076
Train MSE:  41088.72329200572
```