

CSM148_Project_1_W24_JACOB_SAYONO

January 22, 2024

0.1 24W-COM SCI-M148 Project 1

Name: Jacob Sayono

UID: 505368811

0.1.1 Submission Guidelines (Due: Jan 29 before the class)

1. Please fill in your name and UID above.
2. Please submit a **PDF printout** of your Jupyter Notebook to **Gradescope**. If you have any trouble accessing Gradescope, please let a TA know ASAP.
3. When submitting to Gradescope, you will be taken to a page that asks you to assign questions and pages. As the PDF can get long, please make sure to assign pages to corresponding questions to ensure the readers know where to look.

0.2 Introduction

Welcome to **CS148 - Introduction to Data Science!** 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

0.3 Working with Real Data

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

0.4 Setup

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

```
[ ]: 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
```

0.5 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, flexible 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()
```

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.

```
[ ]: # 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  \
0      -122.23      37.88              41.0          880.0           129.0
1      -122.22      37.86              21.0         7099.0          1106.0
2      -122.24      37.85              52.0         1467.0           190.0
3      -122.25      37.85              52.0         1274.0           235.0
4      -122.25      37.85              52.0         1627.0           280.0

      population  households  median_income  median_house_value  ocean_proximity
0           322.0        126.0         8.3252          452600.0        NEAR BAY
1          2401.0       1138.0         8.3014          358500.0        NEAR BAY
2           496.0        177.0         7.2574          352100.0        NEAR BAY
3           558.0        219.0         5.6431          341300.0        NEAR BAY
4           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 essentially 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 occurrences
# for categorical features
housing["ocean_proximity"].value_counts()

```

```

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

```

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

```
[ ]:
count    longitude    latitude    housing_median_age    total_rooms    \
mean      -119.569704    35.631861      28.639486    2635.763081
std         2.003532     2.135952     12.585558    2181.615252
min       -124.350000    32.540000      1.000000      2.000000
25%       -121.800000    33.930000     18.000000    1447.750000
50%       -118.490000    34.260000     29.000000    2127.000000
75%       -118.010000    37.710000     37.000000    3148.000000
max        -114.310000    41.950000     52.000000   39320.000000

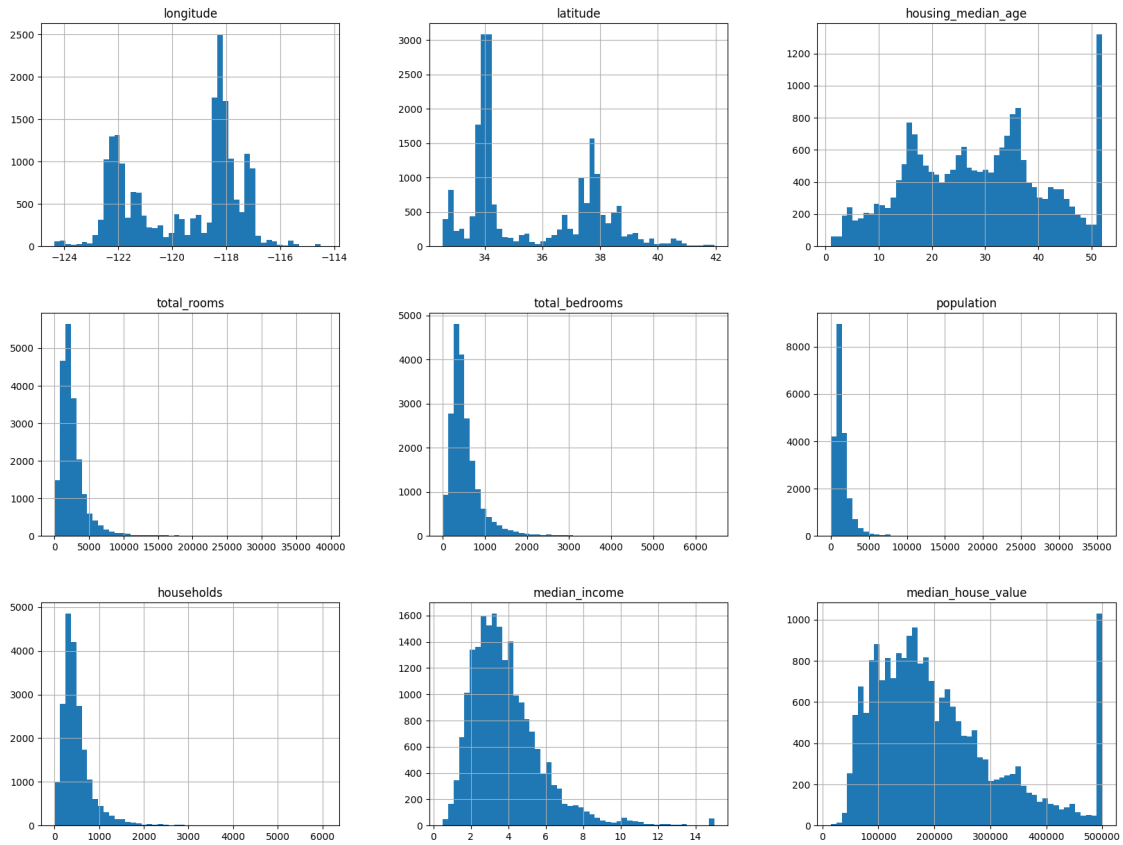
count    total_bedrooms    population    households    median_income    \
mean         537.870553    1425.476744    499.539680      3.870671
std         421.385070    1132.462122    382.329753      1.899822
min           1.000000      3.000000      1.000000      0.499900
25%         296.000000     787.000000     280.000000      2.563400
50%         435.000000    1166.000000     409.000000      3.534800
75%         647.000000    1725.000000     605.000000      4.743250
max        6445.000000   35682.000000    6082.000000     15.000100

count    median_house_value
mean     206855.816909
std     115395.615874
min       14999.000000
25%      119600.000000
50%      179700.000000
75%      264725.000000
max      500001.000000
```

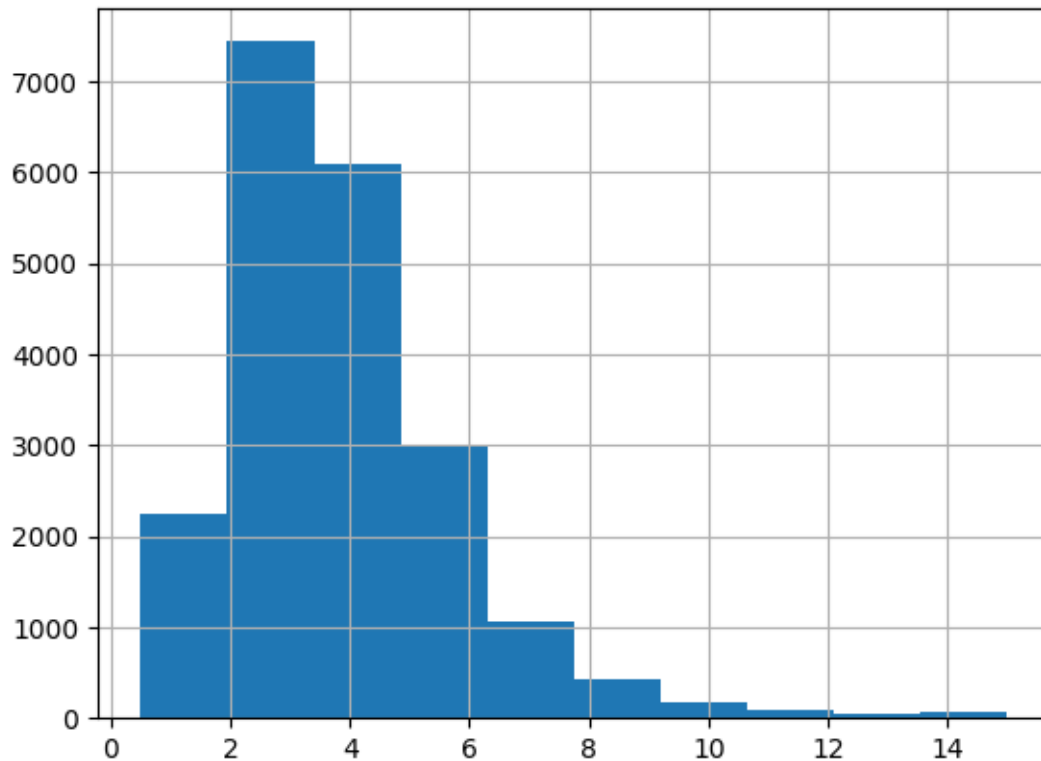
If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section [here](#)

0.6 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

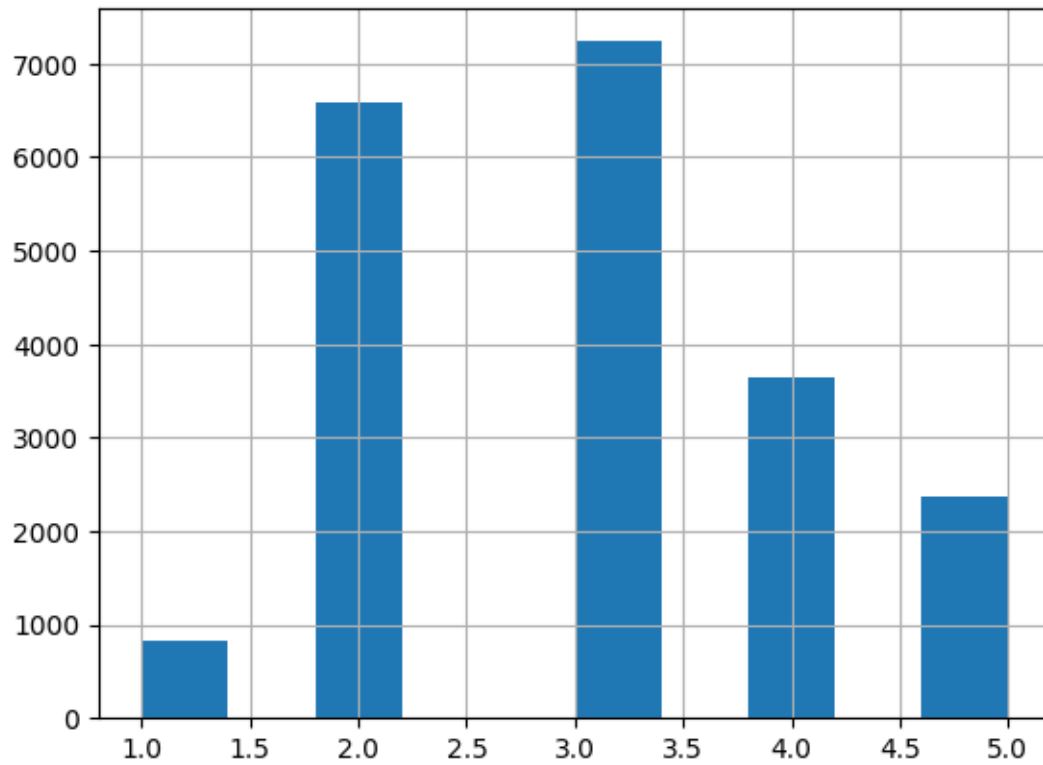
```
[ ]: # assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
housing["income_cat"] = pd.cut(housing["median_income"],
                                bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                                labels=[1, 2, 3, 4, 5])

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

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

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

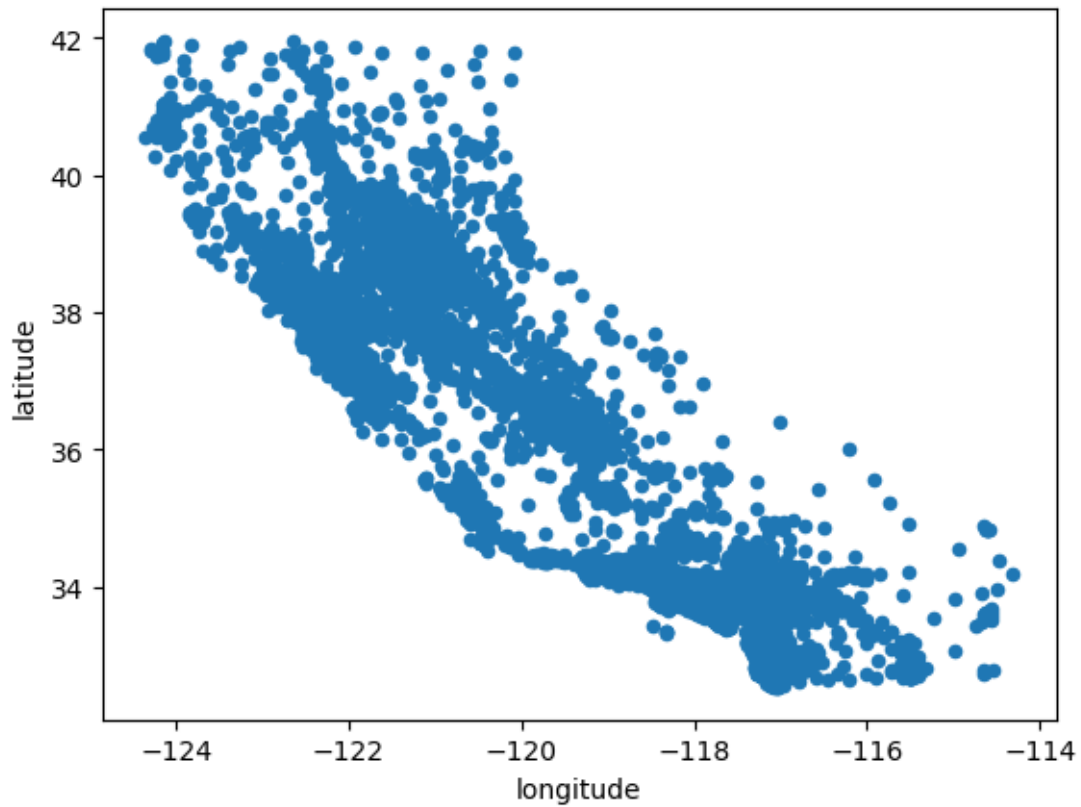
```
[ ]: <AxesSubplot: >
```



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

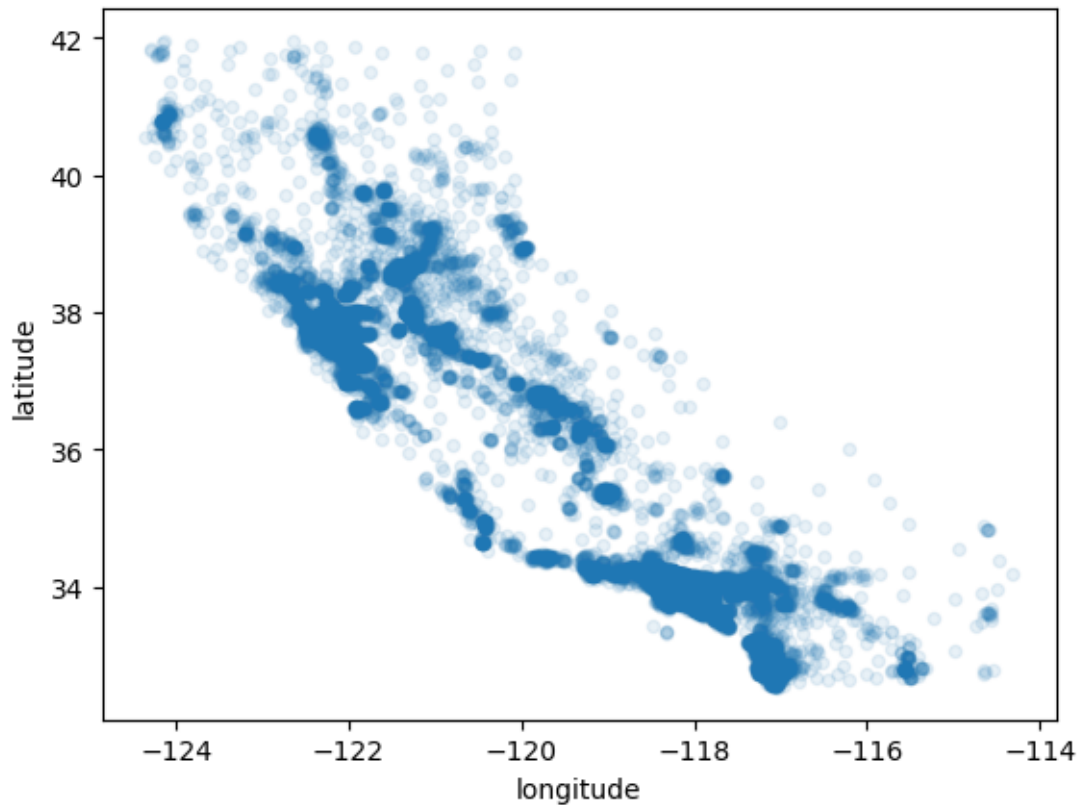
```
[ ]: ## here's a not so interesting way plotting it  
housing.plot(kind="scatter", x="longitude", y="latitude")
```

```
[ ]: <AxesSubplot: xlabel='longitude', ylabel='latitude'>
```

```
[ ]: # 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)
```

```
[ ]: <AxesSubplot: xlabel='longitude', ylabel='latitude'>
```



```
[ ]: # 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

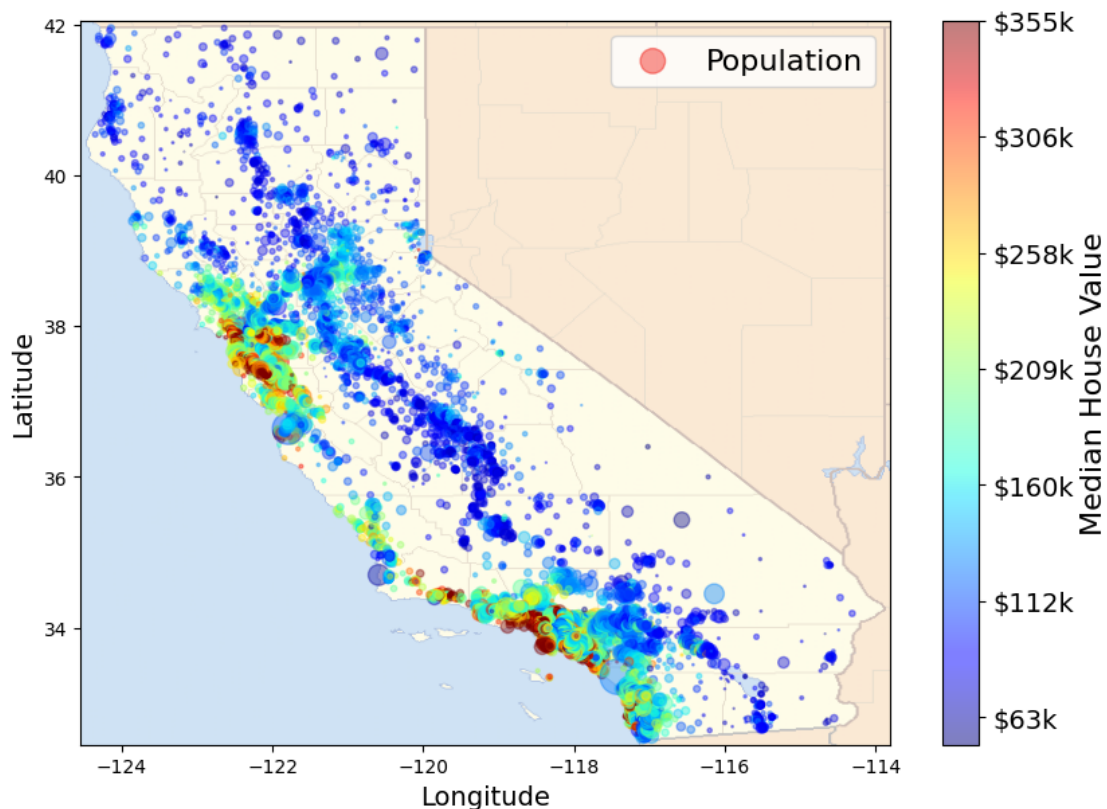
import matplotlib.image as mpimg
california_img=mpimg.imread('california.png')
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                  s=housing['population']/100, label="Population",
                  c="median_house_value", cmap=plt.get_cmap("jet"),
       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

/tmp/ipykernel_19578/2129115766.py:26: UserWarning:

FixedFormatter should only be used together with FixedLocator



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 transformations.

None the less we can explore this using correlation matrices.

```
[ ]: corr_matrix = housing.corr(numeric_only=True)
```

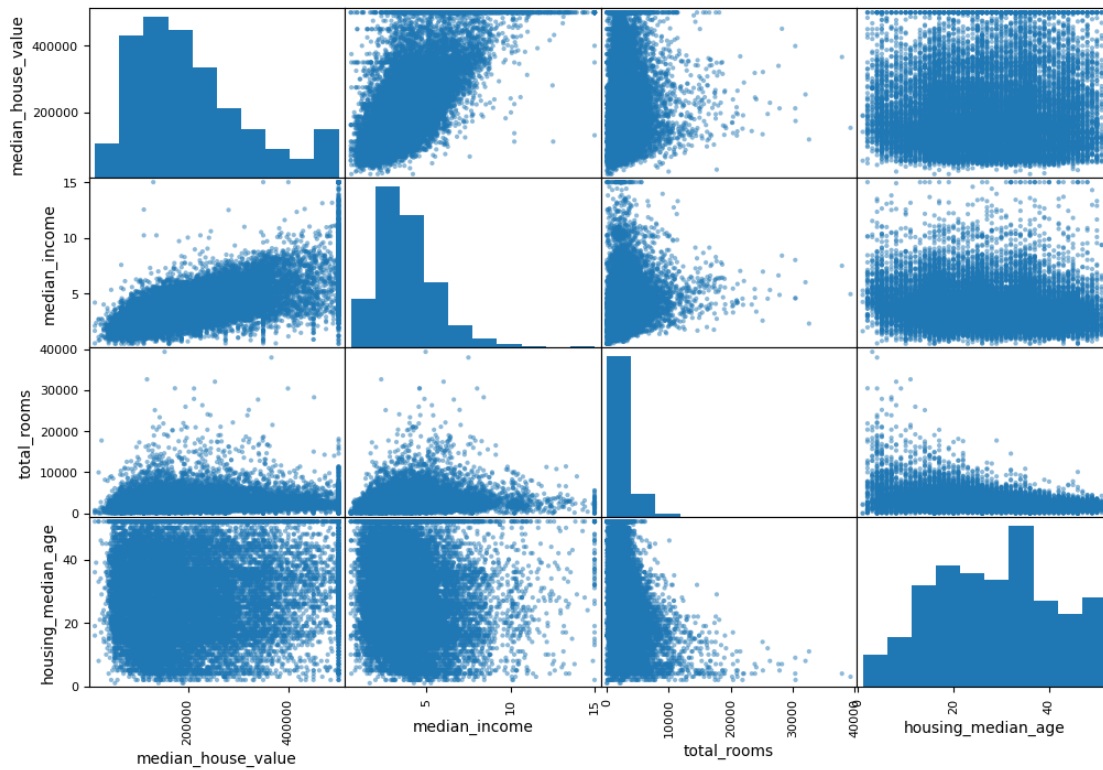
```
[ ]: # for example if the target is "median_house_value", most correlated features
      ↪ can be sorted
      # which happens to be "median_income". This also intuitively makes sense.
      corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
[ ]: 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))
```

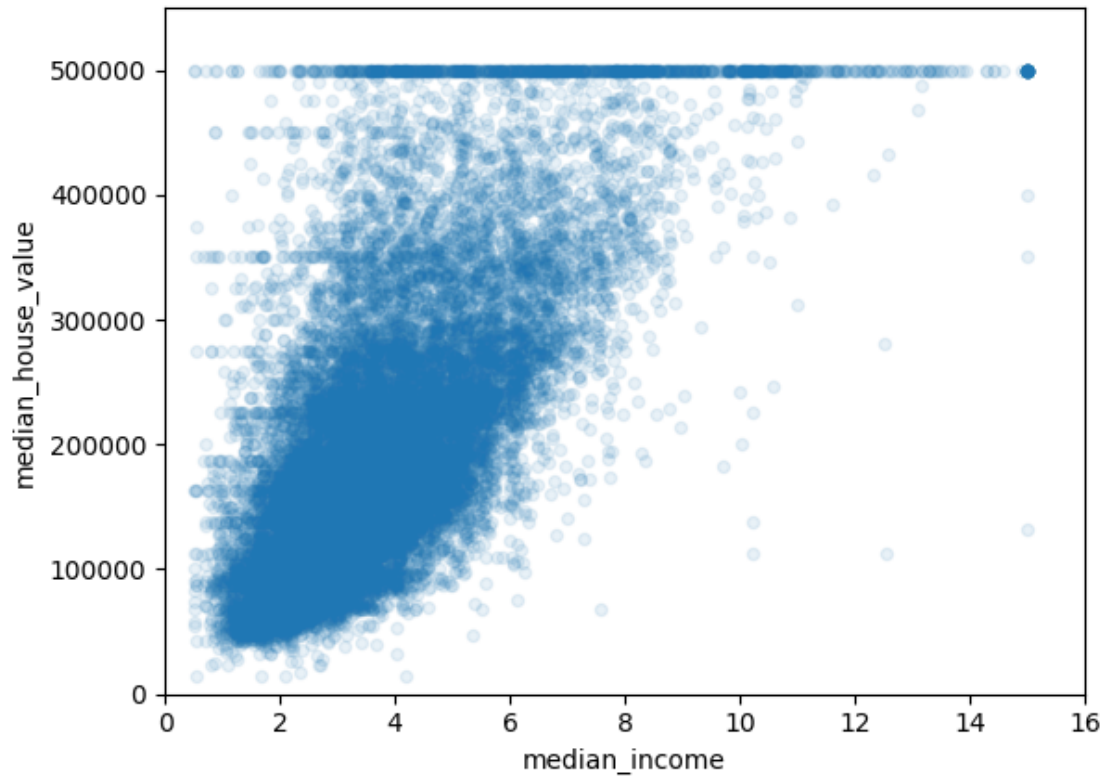
```
[ ]: array([[<AxesSubplot: xlabel='median_house_value', ylabel='median_house_value'>,
              <AxesSubplot: xlabel='median_income', ylabel='median_house_value'>,
              <AxesSubplot: xlabel='total_rooms', ylabel='median_house_value'>,
              <AxesSubplot: xlabel='housing_median_age',
              ylabel='median_house_value'>],
            [<AxesSubplot: xlabel='median_house_value', ylabel='median_income'>,
              <AxesSubplot: xlabel='median_income', ylabel='median_income'>,
              <AxesSubplot: xlabel='total_rooms', ylabel='median_income'>,
              <AxesSubplot: xlabel='housing_median_age', ylabel='median_income'>],
            [<AxesSubplot: xlabel='median_house_value', ylabel='total_rooms'>,
              <AxesSubplot: xlabel='median_income', ylabel='total_rooms'>,
              <AxesSubplot: xlabel='total_rooms', ylabel='total_rooms'>,
              <AxesSubplot: xlabel='housing_median_age', ylabel='total_rooms'>],
            [<AxesSubplot: xlabel='median_house_value', ylabel='housing_median_age'>,
              <AxesSubplot: xlabel='median_income', ylabel='housing_median_age'>,
              <AxesSubplot: xlabel='total_rooms', ylabel='housing_median_age'>,
              <AxesSubplot: xlabel='housing_median_age',
```

```
ylabel='housing_median_age'>]],
      dtype=object)
```



```
[ ]: # median income vs median house value 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(numeric_only=True)
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
```

0.7 Preparing Dastaset for ML

0.7.1 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  \
290      -122.16    37.77             47.0         1256.0             NaN
341      -122.17    37.75             38.0           992.0             NaN
538      -122.28    37.78             29.0        5154.0             NaN
563      -122.24    37.75             45.0           891.0             NaN
696      -122.10    37.69             41.0           746.0             NaN

      population  households  median_income  median_house_value  \
290          570.0        218.0         4.3750         161900.0
341          732.0        259.0         1.6196          85100.0
538         3741.0       1273.0         2.5762        173400.0
563          384.0        146.0         4.9489        247100.0
696          387.0        161.0         3.9063        178400.0

      ocean_proximity  income_cat
290          NEAR BAY           3
341          NEAR BAY           2
538          NEAR BAY           2
563          NEAR BAY           4
696          NEAR BAY           3
```

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

```
[ ]: Empty DataFrame
Columns: [longitude, latitude, housing_median_age, total_rooms, total_bedrooms,
population, households, median_income, median_house_value, ocean_proximity,
income_cat]
Index: []
```

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

```
[ ]:      longitude  latitude  housing_median_age  total_rooms  population  \
290      -122.16    37.77             47.0         1256.0          570.0
341      -122.17    37.75             38.0           992.0          732.0
538      -122.28    37.78             29.0        5154.0        3741.0
563      -122.24    37.75             45.0           891.0          384.0
```

696	-122.10	37.69	41.0	746.0	387.0
-----	---------	-------	------	-------	-------

	households	median_income	median_house_value	ocean_proximity	income_cat
290	218.0	4.3750	161900.0	NEAR BAY	3
341	259.0	1.6196	85100.0	NEAR BAY	2
538	1273.0	2.5762	173400.0	NEAR BAY	2
563	146.0	4.9489	247100.0	NEAR BAY	4
696	161.0	3.9063	178400.0	NEAR BAY	3

```
[ ]: median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 1
↳3: replace na values with median values
sample_incomplete_rows
```

```
[ ]: longitude latitude housing_median_age total_rooms total_bedrooms \
290 -122.16 37.77 47.0 1256.0 435.0
341 -122.17 37.75 38.0 992.0 435.0
538 -122.28 37.78 29.0 5154.0 435.0
563 -122.24 37.75 45.0 891.0 435.0
696 -122.10 37.69 41.0 746.0 435.0
```

	population	households	median_income	median_house_value	\
290	570.0	218.0	4.3750	161900.0	
341	732.0	259.0	1.6196	85100.0	
538	3741.0	1273.0	2.5762	173400.0	
563	384.0	146.0	4.9489	247100.0	
696	387.0	161.0	3.9063	178400.0	

	ocean_proximity	income_cat
290	NEAR BAY	3
341	NEAR BAY	2
538	NEAR BAY	2
563	NEAR BAY	4
696	NEAR BAY	3

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?

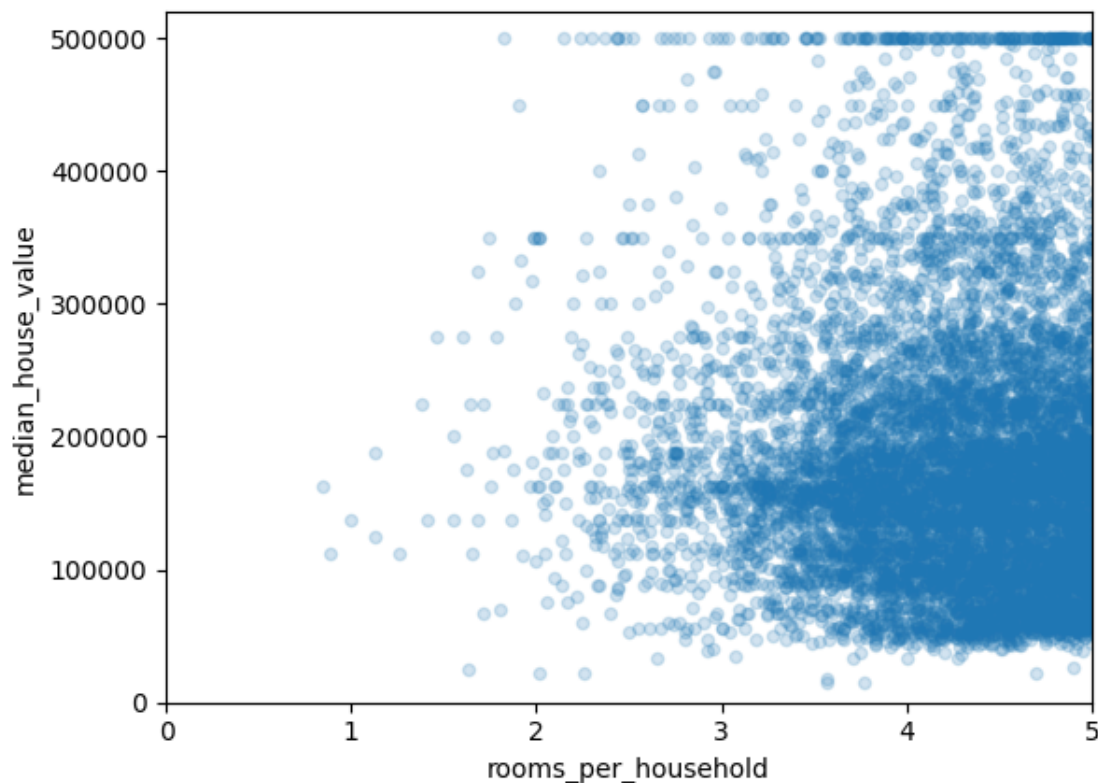
0.7.2 Augmenting Features

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

- $\text{rooms_per_household} = \text{total_rooms} / \text{households}$
- $\text{bedrooms_per_room} = \text{total_bedrooms} / \text{total_rooms}$
- etc.


```
[ ]: housing["rooms_per_household"] = housing["total_rooms"]/(housing["households"]_
    ↪+ 1e-6)
housing["bedrooms_per_room"] = housing["total_bedrooms"]/
    ↪(housing["total_rooms"] + 1e-6)
housing["population_per_household"]=housing["population"]/
    ↪(housing["households"] + 1e-6)

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



0.7.3 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  \
0      -122.23    37.88           41.0           880.0           129.0
1      -122.22    37.86           21.0          7099.0          1106.0
2      -122.24    37.85           52.0          1467.0           190.0
3      -122.25    37.85           52.0          1274.0           235.0
4      -122.25    37.85           52.0          1627.0           280.0

      population  households  median_income  median_house_value  ocean_proximity  \
0           322.0        126.0         8.3252         452600.0              3
1          2401.0       1138.0         8.3014        358500.0              3
2           496.0        177.0         7.2574        352100.0              3
3           558.0        219.0         5.6431        341300.0              3
4           565.0        259.0         3.8462        342200.0              3

      income_cat  rooms_per_household  bedrooms_per_room  population_per_household
0              5              6.984127              0.146591              2.555556
1              5              6.238137              0.155797              2.109842
2              5              8.288136              0.129516              2.802260
3              4              5.817352              0.184458              2.547945
4              3              6.281853              0.172096              2.181467
```

0.7.4 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()
```

0.7.5 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)
```

```
[ ]: LinearRegression()
```

```
[ ]: # 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: [418197.21048505 305620.51781477 232253.02900545 188754.57142336
251166.41766859]
Actual labels: [500001.0, 162500.0, 204600.0, 159700.0, 184000.0]
```

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

$$L(\hat{Y}, Y) = \frac{1}{N} \sum_i^N (\hat{y}_i - y_i)^2$$

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
```

```
[ ]: 67694.08184344396
```

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

1 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.**

2 Visualizing Data

2.0.1 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

```
[ ]: import pandas as pd
airbnb = pd.read_csv('AB_NYC_2019.csv') # we load the pandas dataframe
airbnb.head()
```

```
[ ]:      id      name  host_id \
0  2539  Clean & quiet apt home by the park    2787
1  2595           Skylit Midtown Castle    2845
2  3647  THE VILLAGE OF HARLEM...NEW YORK !    4632
3  3831           Cozy Entire Floor of Brownstone    4869
4  5022  Entire Apt: Spacious Studio/Loft by central park    7192

      host_name  neighbourhood_group  neighbourhood  latitude  longitude \
0         John         Brooklyn    Kensington  40.64749  -73.97237
1      Jennifer         Manhattan    Midtown  40.75362  -73.98377
2     Elisabeth         Manhattan    Harlem  40.80902  -73.94190
3  LisaRoxanne         Brooklyn  Clinton Hill  40.68514  -73.95976
4         Laura         Manhattan    East Harlem  40.79851  -73.94399
```

	room_type	price	minimum_nights	number_of_reviews	last_review	\
0	Private room	149	1	9	2018-10-19	
1	Entire home/apt	225	1	45	2019-05-21	
2	Private room	150	3	0	NaN	
3	Entire home/apt	89	1	270	2019-07-05	
4	Entire home/apt	80	10	9	2018-11-19	

	reviews_per_month	calculated_host_listings_count	availability_365
0	0.21	6	365
1	0.38	2	355
2	NaN	1	365
3	4.64	1	194
4	0.10	1	0

```
[ ]: columns_to_drop = ['name', 'host_id', 'host_name', 'last_review',
↪ 'neighbourhood']
airbnb_drop = airbnb.drop(columns=columns_to_drop)
```

```
[ ]: airbnb_drop.describe()
```

```
[ ]:
```

	id	latitude	longitude	price	minimum_nights	\
count	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	
mean	1.901714e+07	40.728949	-73.952170	152.720687	7.029962	
std	1.098311e+07	0.054530	0.046157	240.154170	20.510550	
min	2.539000e+03	40.499790	-74.244420	0.000000	1.000000	
25%	9.471945e+06	40.690100	-73.983070	69.000000	1.000000	
50%	1.967728e+07	40.723070	-73.955680	106.000000	3.000000	
75%	2.915218e+07	40.763115	-73.936275	175.000000	5.000000	
max	3.648724e+07	40.913060	-73.712990	10000.000000	1250.000000	

	number_of_reviews	reviews_per_month	calculated_host_listings_count	\
count	48895.000000	38843.000000	48895.000000	
mean	23.274466	1.373221	7.143982	
std	44.550582	1.680442	32.952519	
min	0.000000	0.010000	1.000000	
25%	1.000000	0.190000	1.000000	
50%	5.000000	0.720000	1.000000	
75%	24.000000	2.020000	2.000000	
max	629.000000	58.500000	327.000000	

	availability_365
count	48895.000000
mean	112.781327
std	131.622289
min	0.000000
25%	0.000000

```

50%          45.000000
75%          227.000000
max           365.000000

```

```
[ ]: airbnb_drop.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 11 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     48895 non-null  int64
1   neighbourhood_group                   48895 non-null  object
2   latitude                             48895 non-null  float64
3   longitude                             48895 non-null  float64
4   room_type                             48895 non-null  object
5   price                                 48895 non-null  int64
6   minimum_nights                       48895 non-null  int64
7   number_of_reviews                    48895 non-null  int64
8   reviews_per_month                    38843 non-null  float64
9   calculated_host_listings_count       48895 non-null  int64
10  availability_365                      48895 non-null  int64
dtypes: float64(3), int64(6), object(2)
memory usage: 4.1+ MB

```

2.0.2 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 Boroughs (neighbourhood_groups in the dataset)

```
[ ]: import plotly.express as px
fig = px.pie(airbnb, names='neighbourhood_group', 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

```
[ ]: neighborhood = airbnb.groupby('neighbourhood_group')['number_of_reviews'].sum().
    ↪reset_index()
neighborhood.head()
```

```
[ ]:  neighbourhood_group  number_of_reviews
0          Bronx          28371
1       Brooklyn      486574
2       Manhattan      454569
3         Queens      156950
4   Staten Island       11541
```

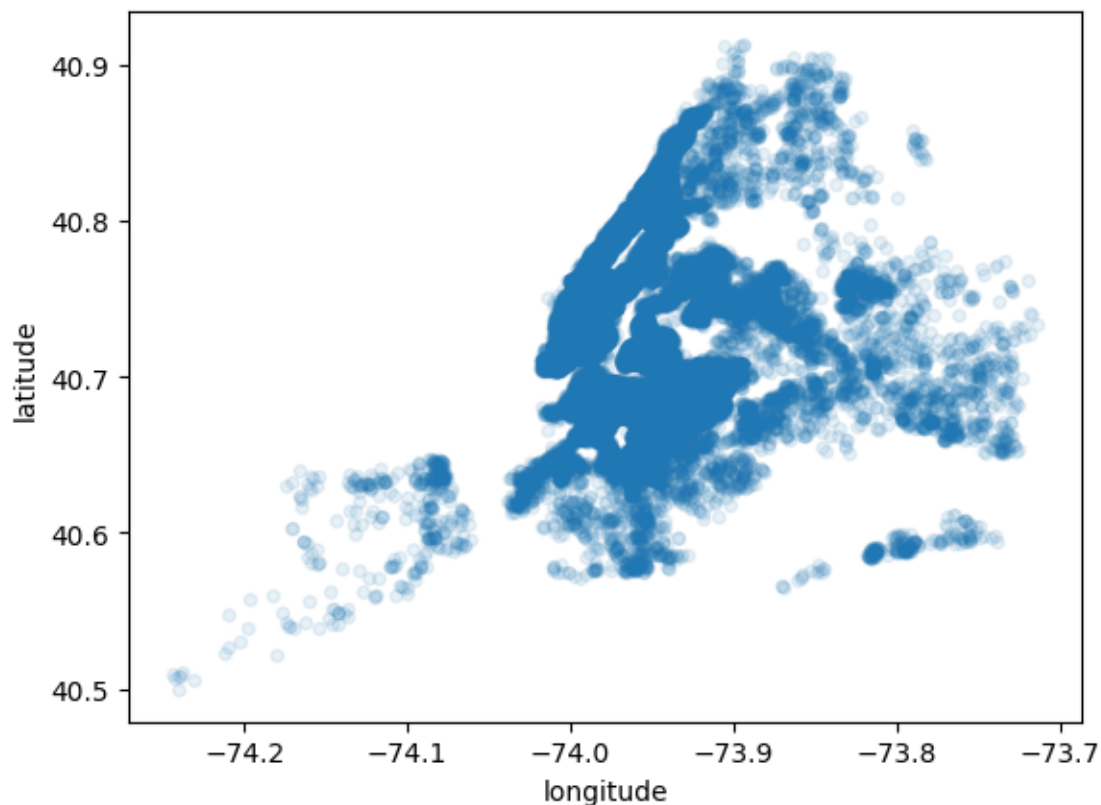
```
[ ]: fig = px.bar(neighborhood, x='neighbourhood_group', y='number_of_reviews',
    ↪title='Total Number of Reviews per Neighbourhood Group')
fig.show()
```

2.0.3 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.

```
[ ]: airbnb.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

```
[ ]: <AxesSubplot: xlabel='longitude', ylabel='latitude'>
```



```
[ ]: miniairbnb = airbnb[airbnb['number_of_reviews'] > 10].sample(frac=0.5)
```

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

# load an image of New York
import matplotlib.image as mpimg
nyc_img=mpimg.imread('nyc.png', -1)

# 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.

# WRITE YOUR CODE HERE #
ax = miniairbnb.plot(kind="scatter", x="longitude", y="latitude",
    figsize=(11,7),
    s=miniairbnb['number_of_reviews']/10, label="Number of
    Reviews",
    c="number_of_reviews", cmap=plt.get_cmap("jet"),
    colorbar=False, alpha=0.4)

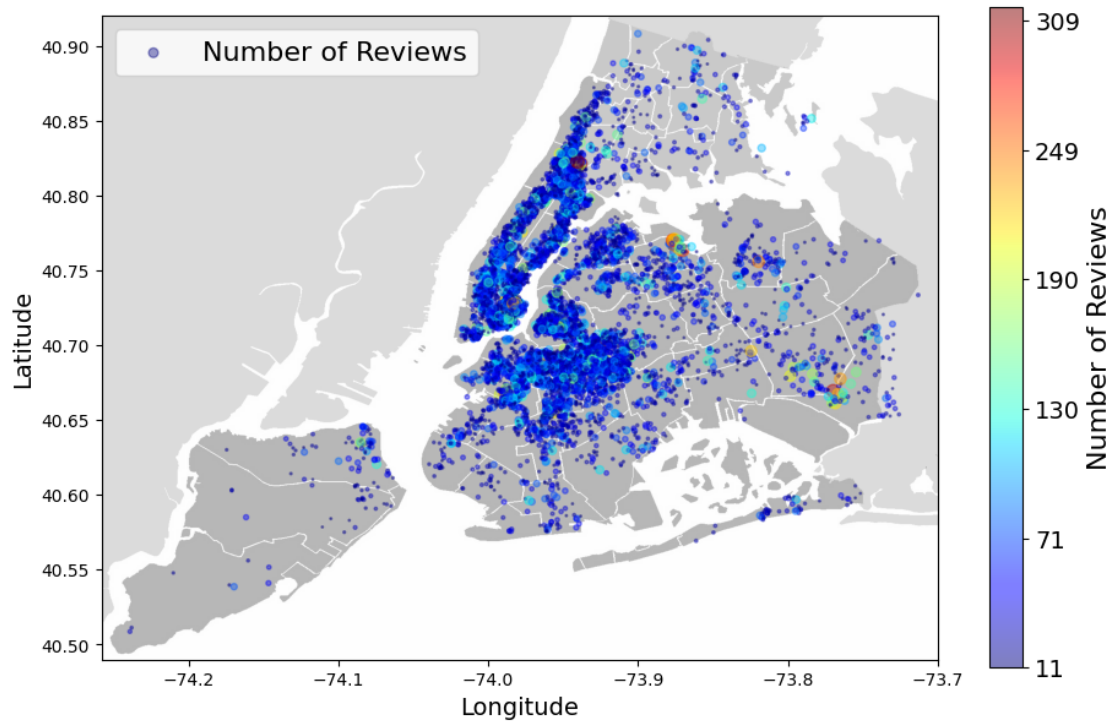
plt.imshow(nyc_img, extent=[-74.258, -73.7, 40.49, 40.92], alpha=0.5,
    cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on median_house_value feature
prices = miniairbnb["number_of_reviews"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cb = plt.colorbar()
cb.ax.set_yticklabels(["%d"%(round(v)) for v in tick_values], fontsize=14)
cb.set_label('Number of Reviews', fontsize=16)

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

/tmp/ipykernel_19578/3231214392.py:27: UserWarning:

FixedFormatter should only be used together with FixedLocator



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

```
[ ]: fig = px.scatter_mapbox(miniairbnb, lat="latitude", lon="longitude",
                             color="number_of_reviews", size="number_of_reviews",
                             color_continuous_scale="Jet", size_max=15, zoom=10,
                             mapbox_style="open-street-map")

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

# WRITE YOUR CODE HERE #
fig.update_layout(
    mapbox_style="white-bg",
    mapbox_layers=[
        {
            "below": 'traces',
            "sourcetype": "image",
            "source": f"data:image/png;base64,{plotly_logo}",
            "coordinates": [[-74.258, 40.49], [-73.7, 40.49], [-73.7, 40.92],
↪ [-74.258, 40.92]]
        }
    ]
)
```

```

    ]
)

fig.add_layout_image(
    dict(
        source=f"data:image/png;base64,{plotly_logo}",
        layer="below"
    )
)

fig.show()

```

2.0.4 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

```

[ ]: # WRITE YOUR CODE HERE #
brooklyn_data = airbnb[(airbnb['neighbourhood_group'] == 'Brooklyn') &
    ↪(airbnb['number_of_reviews'] >= 10)]
avg_price = brooklyn_data.groupby('room_type')['price'].mean().reset_index()

```

```

[ ]: # WRITE YOUR CODE HERE #
fig = px.bar(avg_price, x='room_type', y='price', title='Average Price of Room_
    ↪Types in Brooklyn')
fig.show()

```

3 Prepare the Data

```

[ ]: airbnb_drop.head()

```

```

[ ]:
   id neighbourhood_group  latitude  longitude  room_type  price \
0  2539                Brooklyn  40.64749   -73.97237  Private room    149
1  2595                Manhattan  40.75362   -73.98377  Entire home/apt    225
2  3647                Manhattan  40.80902   -73.94190  Private room    150
3  3831                Brooklyn  40.68514   -73.95976  Entire home/apt     89
4  5022                Manhattan  40.79851   -73.94399  Entire home/apt     80

   minimum_nights  number_of_reviews  reviews_per_month \
0                1                  9                0.21
1                1                 45                0.38

```

2	3	0	NaN
3	1	270	4.64
4	10	9	0.10

	calculated_host_listings_count	availability_365
0	6	365
1	2	355
2	1	365
3	1	194
4	1	0

3.0.1 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

```
[ ]: # assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
airbnb_drop['price_cat'] = pd.qcut(airbnb_drop['price'], q=5, labels=False)

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

```
[ ]: price_cat
3    10809
0    10063
1     9835
2     9804
4     8384
Name: count, dtype: int64
```

3.0.2 Data Imputation

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

```
[ ]: airbnb_drop.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 12 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     48895 non-null  int64
1   neighbourhood_group                   48895 non-null  object
2   latitude                             48895 non-null  float64
3   longitude                             48895 non-null  float64
4   room_type                             48895 non-null  object
5   price                                 48895 non-null  int64
```

```

6   minimum_nights                48895 non-null  int64
7   number_of_reviews             48895 non-null  int64
8   reviews_per_month            38843 non-null  float64
9   calculated_host_listings_count 48895 non-null  int64
10  availability_365              48895 non-null  int64
11  price_cat                    48895 non-null  int64
dtypes: float64(3), int64(7), object(2)
memory usage: 4.5+ MB

```

```

[ ]: # WRITE YOUR CODE HERE #
airbnb_drop.isnull().sum()
airbnb_drop['reviews_per_month'].fillna(0, inplace=True)

```

3.0.3 Numeric Conversions

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

```

[ ]: # WRITE YOUR CODE HERE #
airbnb_drop['neighbourhood_group'] = pd.
    ↳factorize(airbnb_drop['neighbourhood_group'])[0]
airbnb_drop['room_type'] = pd.factorize(airbnb_drop['room_type'])[0]

```

4 Prepare Data for Machine Learning

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

```

[ ]: from sklearn.model_selection import StratifiedShuffleSplit
    # let's first start by creating our train and test sets

    # WRITE YOUR CODE HERE #
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(airbnb_drop,
    ↳airbnb_drop["price_cat"]):
    train_set = airbnb_drop.loc[train_index]
    test_set = airbnb_drop.loc[test_index]

```

```

[ ]: test_set.info()

```

```

<class 'pandas.core.frame.DataFrame'>
Index: 9779 entries, 34229 to 20813
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id                    9779 non-null   int64
1   neighbourhood_group    9779 non-null   int64
2   latitude              9779 non-null   float64

```

```

3  longitude          9779 non-null  float64
4  room_type          9779 non-null  int64
5  price              9779 non-null  int64
6  minimum_nights     9779 non-null  int64
7  number_of_reviews  9779 non-null  int64
8  reviews_per_month  9779 non-null  float64
9  calculated_host_listings_count  9779 non-null  int64
10 availability_365    9779 non-null  int64
11 price_cat          9779 non-null  int64
dtypes: float64(3), int64(9)
memory usage: 993.2 KB

```

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

```

[ ]: # WRITE YOUR CODE HERE #
train_labels = train_set['price'].copy()
test_labels = test_set['price'].copy()
train_set = train_set.drop(['price', 'price_cat'], axis=1)
test_set = test_set.drop(['price', 'price_cat'], axis=1)

```

```

[ ]: train_set.head()

```

```

[ ]:
      id  neighbourhood_group  latitude  longitude  room_type  \
40334  31283904              1  40.72846  -73.98457          1
12438   9578325              0  40.67924  -73.98718          1
35502  28181243              0  40.66891  -73.93495          1
6553   4750578              0  40.68589  -73.95759          1
19465  15529937              0  40.60983  -73.95887          0

      minimum_nights  number_of_reviews  reviews_per_month  \
40334              1                10              1.67
12438              1               120              2.73
35502              3                 2              0.24
6553              1                 0              0.00
19465              2                26              0.98

      calculated_host_listings_count  availability_365
40334                             1                332
12438                             2                275
35502                             1                362
6553                              1                 0
19465                             3                101

```

5 Fit a linear regression model

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**.

```
[ ]: # WRITE YOUR CODE HERE #  
from sklearn.linear_model import LinearRegression  
from sklearn.metrics import mean_squared_error  
  
lin_reg = LinearRegression()  
lin_reg.fit(train_set, train_labels)  
  
train_predictions = lin_reg.predict(train_set)  
test_predictions = lin_reg.predict(test_set)  
  
train_mse = mean_squared_error(train_labels, train_predictions)  
test_mse = mean_squared_error(test_labels, test_predictions)  
  
print(f"Train MSE: {train_mse}\n Test MSE: {test_mse}")
```

```
Train MSE: 57270.60756968636  
Test MSE: 39332.40162778953
```