Project1-Alice Lu(605840292)

April 25, 2023

1 Introduction

Welcome to **M148- Data Science Fundamentals!** This course is designed to equip you with the tools and experiences necessary to start you off on a life-long exploration of datascience. We do not assume a prerequisite knowledge or experience in order to take the course.

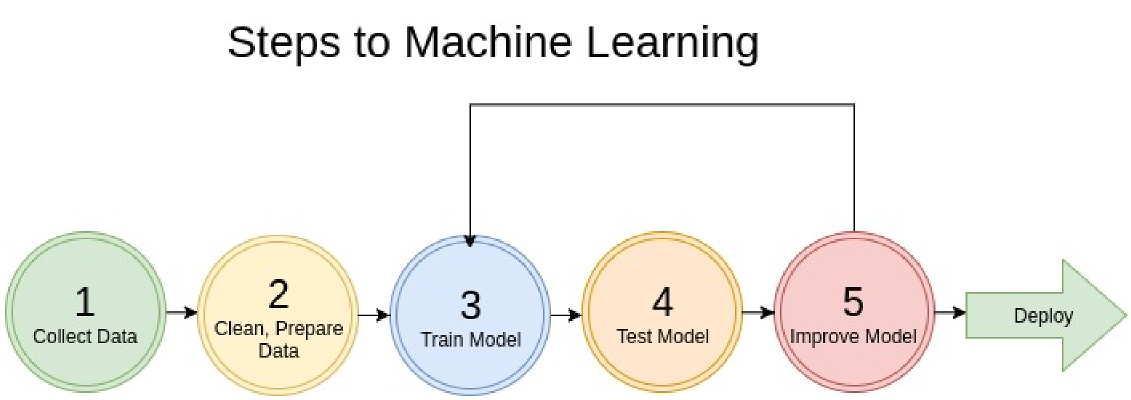
For this first project we will introduce you to the end-to-end process of doing a datascience project. Our goals for this project are to:

1. Familiarize you with the development environment for doing datascience
2. Get you comfortable with the python coding required to do datascience
3. Provide you with an sample end-to-end project to help you visualize the steps needed to complete a project on your own
4. Ask you to recreate a similar project on a separate dataset

In this project you will work through an example project end to end. Many of the concepts you will encounter will be unclear to you. That is OK! The course is designed to teach you these concepts in further detail. For now our focus is simply on having you replicate the code successfully and seeing a project through from start to finish.

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 model and train
5. Does it meet the requirements? Fine tune the model



1.1 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

1.2 Submission Instructions

Project is due April 26th at 12:00 pm noon. To submit the project, please save the notebook as a pdf file and submit the assignment via Gradescope. In addition, Make sure that all figures are legible and sufficiently large.

2 Example Datascience Exercise

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

2.1 Setup

Before getting started, it is always good to check the versions of important packages. Knowing the version number makes it easier to lookup correct documenation.

To run this project, you will need the following packages installed with at least the minimial version number provided: - Python Version >= 3.9 - Scitkit-learn >= 1.0.2 - Numpy >= 1.18.5 - Scipy >= 1.1.0 - Pandas >= 1.4.0 - Matplotlib >= 3.3.2

The following code imports these packages and checks their version number. If any assertion error occurs, you may not have the correct version installed.

Important: If installed using a package manager like Anaconda or pip, these dependencies should be resolved. Please follow the python setup guide provided during discussion of week 1.

[2]:
```
#Import and Version Test
#Python version test
import sys
assert sys.version_info >= (3, 9) # python>=3.9

#Machine learning library
import sklearn
assert sklearn.__version__ >= "1.0.2" # sklearn >= 1.0.2

#numerical packages in python
import numpy as np
assert np.__version__ >= "1.18.5" # numpy >= 1.18.5

#Another numerical package, unused directly but is implicitly used in sklearn
#Check the version just in case
```

```python
import scipy as scp
assert scp.__version__ >= "1.1.0" # scipy >= 1.1.0

#Package for data manipulation and analysis
import pandas as pd
assert pd.__version__ >= "1.4.0" # pandas >= 1.4.0

#matplotlib magic for inline figures
import matplotlib # plotting library
assert matplotlib.__version__ >= "3.3.2" # matplotlib >= 3.3.2
%matplotlib inline
```

[3]:
```python
import os
import tarfile
import urllib
DATASET_PATH = os.path.join("datasets", "housing")
```

[4]:
```python
#Other setup with necessary plotting

#Instead of using matplotlib direclty, we will use their nice pyplot interface␣
↪defined as plt
import matplotlib.pyplot as plt

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

# Plotting Utilities

# Where to save the figures
ROOT_DIR = "."
IMAGES_PATH = os.path.join(ROOT_DIR, "images")
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_name, tight_layout=True, fig_extension="png", resolution=300):
    '''
        plt.savefig wrapper. refer to
        https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.savefig.html
    '''
    path = os.path.join(IMAGES_PATH, fig_name + "." + fig_extension)
    print("Saving figure", fig_name)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

2.2 Step 1. Getting the data

2.2.1 Intro to Data Exploration Using Pandas

In this section we will load the dataset, do some cleaning, 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

[5]:
```
import pandas as pd

def load_housing_data(housing_path):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

First, we load the dataset into pandas Dataframe which you can think about as an array/table. The Dataframe has a lot of useful functionality which we will use throughout the class.

[6]:
```
housing = load_housing_data(DATASET_PATH) # we load the pandas dataframe
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
```

[6]:
```
   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)

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.

[7]:
```
# 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
```

[8]:
```python
# you can access individual columns similarly
# to accessing elements in a python dict
print(housing["ocean_proximity"].head()) # added head() to avoid printing many␣
↪columns.

#Additionally, columns can be accessed as attirbutes of the dataframe object
#This method is convenient to access data but should be used with care since␣
↪you can't overwrite
#built in functions like housing.min()
print(housing.ocean_proximity.head())
```

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

[9]:
```python
# to access a particular row we can use iloc
housing.iloc[1]
```

[9]:
```
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
```

[10]:
```python
# one other function that might be useful is
# value_counts(), which counts the number of occurences
# for categorical features
housing["ocean_proximity"].value_counts()
```

[10]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

[11]:
```python
# The describe function compiles your typical statistics for each␣
↪non-categorical column
housing.describe()
```

[11]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
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

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
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

       median_house_value
count        20640.000000
```

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

We can also perform groupings based on categorical values and analyze each group.

[12]:
```python
housing_group = housing.groupby('ocean_proximity')
#Has the mean for every column grouped by ocean proximity
housing_mean = housing_group.mean()
housing_mean
```

[12]:
```
                  longitude   latitude  housing_median_age  total_rooms  \
ocean_proximity
<1H OCEAN       -118.847766  34.560577           29.279225  2628.343586
INLAND          -119.732990  36.731829           24.271867  2717.742787
ISLAND          -118.354000  33.358000           42.400000  1574.600000
NEAR BAY        -122.260694  37.801057           37.730131  2493.589520
NEAR OCEAN      -119.332555  34.738439           29.347254  2583.700903

                 total_bedrooms   population  households  median_income  \
ocean_proximity
<1H OCEAN            546.539185  1520.290499  517.744965       4.230682
INLAND               533.881619  1391.046252  477.447565       3.208996
ISLAND               420.400000   668.000000  276.600000       2.744420
NEAR BAY             514.182819  1230.317467  488.616157       4.172885
NEAR OCEAN           538.615677  1354.008653  501.244545       4.005785

                 median_house_value
ocean_proximity
<1H OCEAN             240084.285464
INLAND                124805.392001
ISLAND                380440.000000
NEAR BAY              259212.311790
NEAR OCEAN            249433.977427
```

[13]:
```python
#We can also get the subset of data associated with that group

housing_inland = housing_group.get_group("INLAND")
housing_inland
```

[13]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
954      -121.92     37.64                46.0       1280.0           209.0
957      -121.90     37.66                18.0       7397.0          1137.0
965      -121.88     37.68                23.0       2234.0           270.0
```

```
967      -121.88    37.67                 16.0       4070.0           624.0
968      -121.88    37.67                 25.0       2244.0           301.0
…            …        …                 …          …            …
20635    -121.09    39.48                 25.0       1665.0           374.0
20636    -121.21    39.49                 18.0        697.0           150.0
20637    -121.22    39.43                 17.0       2254.0           485.0
20638    -121.32    39.43                 18.0       1860.0           409.0
20639    -121.24    39.37                 16.0       2785.0           616.0

       population  households  median_income  median_house_value  \
954         512.0       208.0         5.1406            315600.0
957        3126.0      1115.0         6.4994            323000.0
965         854.0       286.0         7.3330            337200.0
967        1543.0       577.0         6.5214            311500.0
968         937.0       324.0         6.4524            296900.0
…             …          …             …                 …
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

      ocean_proximity
954            INLAND
957            INLAND
965            INLAND
967            INLAND
968            INLAND
…                 …
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[6551 rows x 10 columns]
```

[14]:
```
#We can thus performs operations on each group separately
housing_inland.describe()
```

[14]:
```
        longitude     latitude  housing_median_age   total_rooms  \
count  6551.00000  6551.000000         6551.000000   6551.000000
mean   -119.73299    36.731829           24.271867   2717.742787
std       1.90095     2.116073           12.018020   2385.831111
min    -123.73000    32.640000            1.000000      2.000000
25%    -121.35000    34.180000           15.000000   1404.000000
50%    -120.00000    36.970000           23.000000   2131.000000
```

```
75%    -117.84000    38.550000            33.000000   3216.000000
max    -114.31000    41.950000            52.000000  39320.000000

       total_bedrooms    population   households  median_income  \
count     6496.000000   6551.000000  6551.000000    6551.000000
mean       533.881619   1391.046252   477.447565       3.208996
std        446.117778   1168.670126   392.252095       1.437465
min          2.000000      5.000000     2.000000       0.499900
25%        282.000000    722.000000   254.000000       2.188950
50%        423.000000   1124.000000   385.000000       2.987700
75%        636.000000   1687.000000   578.000000       3.961500
max       6210.000000  16305.000000  5358.000000      15.000100

       median_house_value
count         6551.000000
mean        124805.392001
std          70007.908494
min          14999.000000
25%          77500.000000
50%         108500.000000
75%         148950.000000
max         500001.000000
```

Grouping is a powerful technique within pandas and a recommend reading the user guide to understand it better here

In addition to grouping, we can also filter out the data based on our desired criteria.

[15]:
```
housing_expensive= housing[(housing["median_house_value"] > 50000)]
housing_expensive.head()
```

[15]:
```
   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
```

[16]:
```
#We can combine multiple criteria
housing_expensive_small= housing[(housing["median_house_value"] > 50000)&
 ↪(housing["population"] < 1000)]
```

```
housing_expensive_small.head()
```

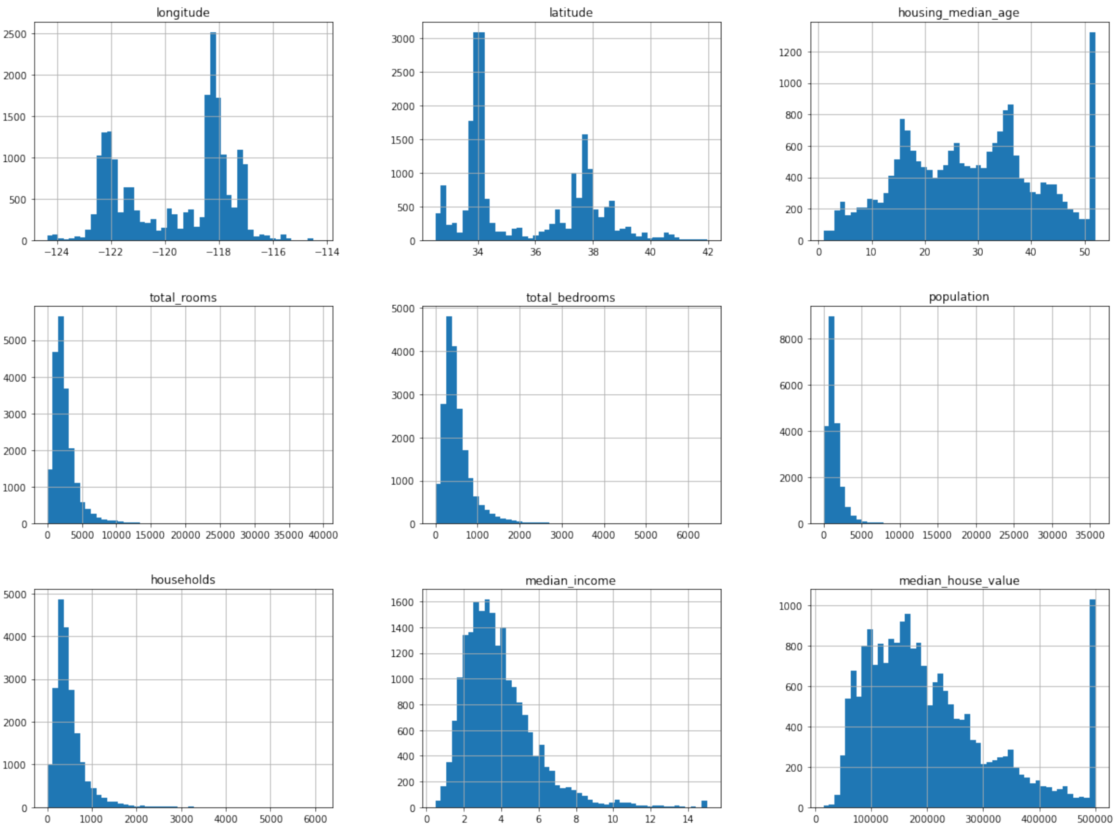
[16]:
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0
5    -122.25     37.85                52.0        919.0           213.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        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
5       413.0       193.0         4.0368            269700.0        NEAR BAY
```

If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section of pandas here and for a full look at all the functionaltiy that pandas offers you can check out the user guide of pandas here

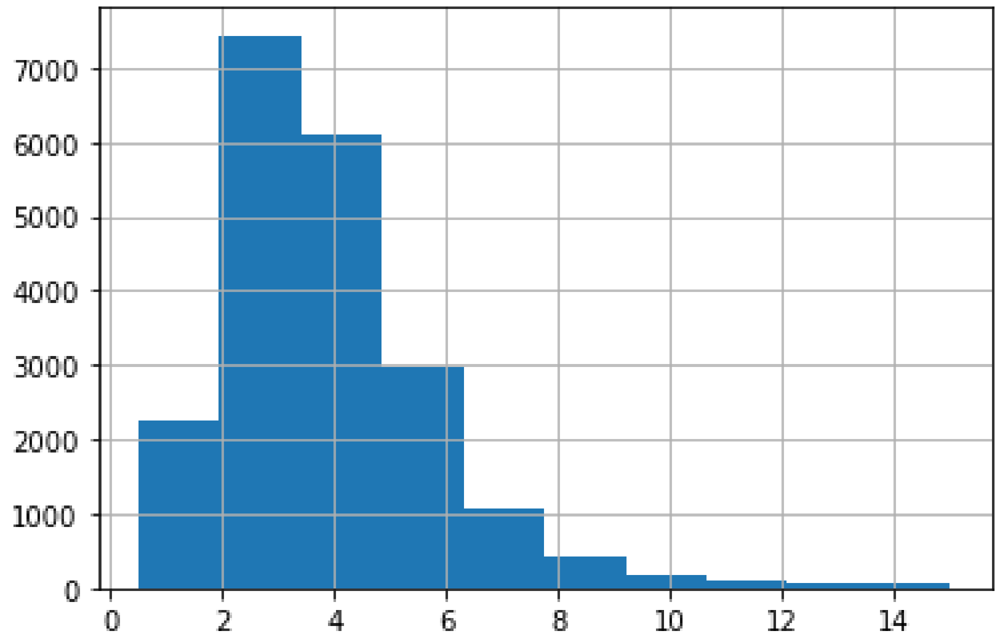
2.3 Step 2. Visualizing the data

2.3.1 Let's start visualizing the dataset

[17]:
```
# We can draw a histogram for each of the dataframes features
# using the built-in hist function of Dataframe
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
```

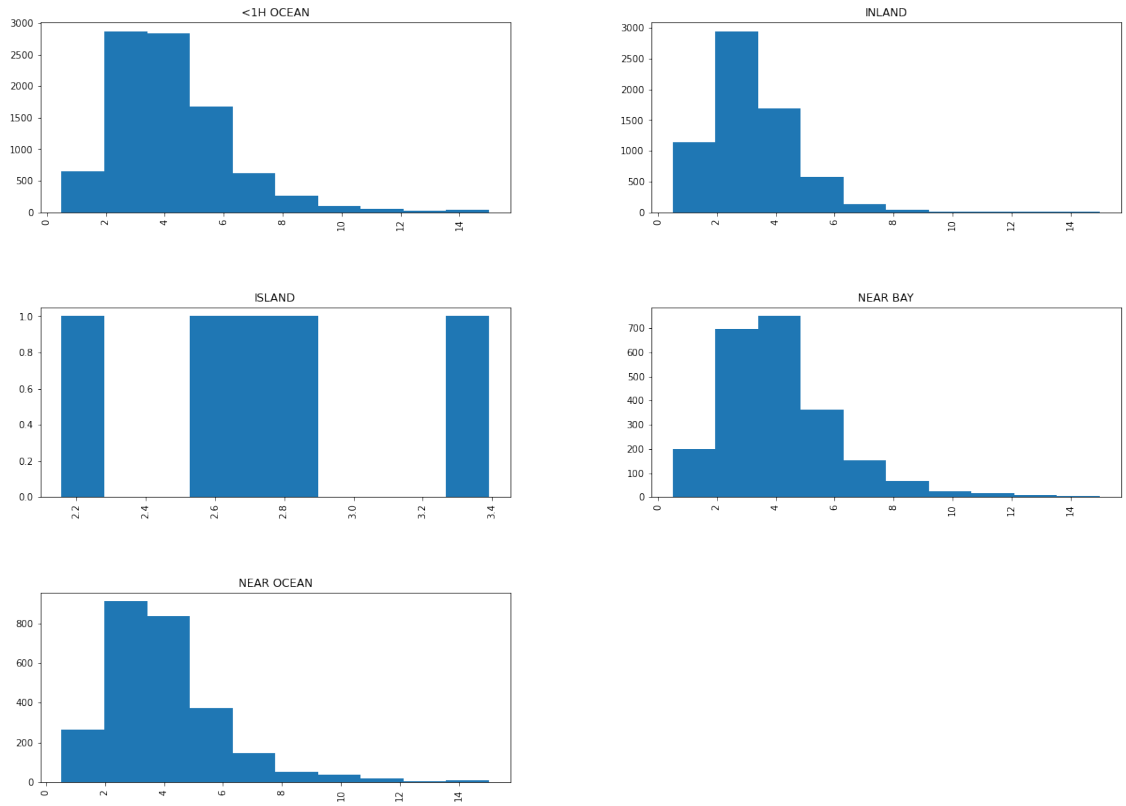


[18]:
```
# if you want to have a histogram on an individual feature:
housing["median_income"].hist()
plt.show()
```



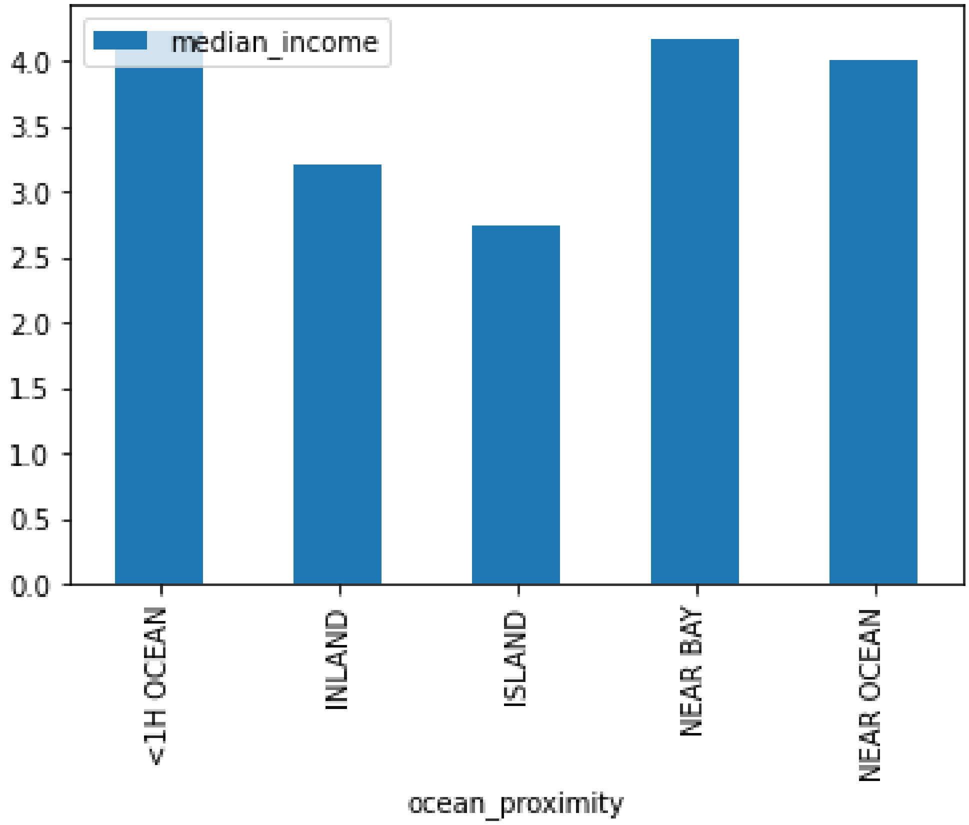
[19]:

```
#You can even plot histograms by specifying the groupings using by
housing["median_income"].hist(by= housing["ocean_proximity"],figsize=(20,15))
plt.show()
```



[20]:
```python
#We can also plot statistics of each groupings
housing_group_mean = housing.groupby("ocean_proximity").mean()

housing_group_mean.plot.bar(y ="median_income")
```

[20]:
```
<AxesSubplot:xlabel='ocean_proximity'>
```



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

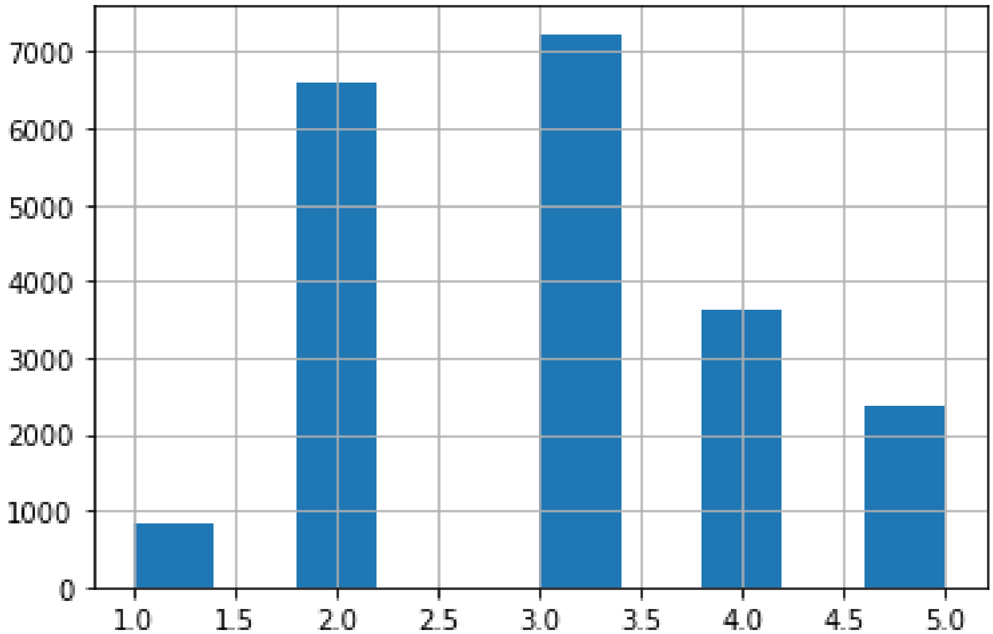
For example, to bin the households based on median_income we can use the pd.cut function. Note that we use np.inf to represent infinity which is internally handeled. Thus, the last bin is $(6, \infty)$.

[21]:
```
# 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()
```

[21]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

[22]:
```
housing["income_cat"].hist()
```

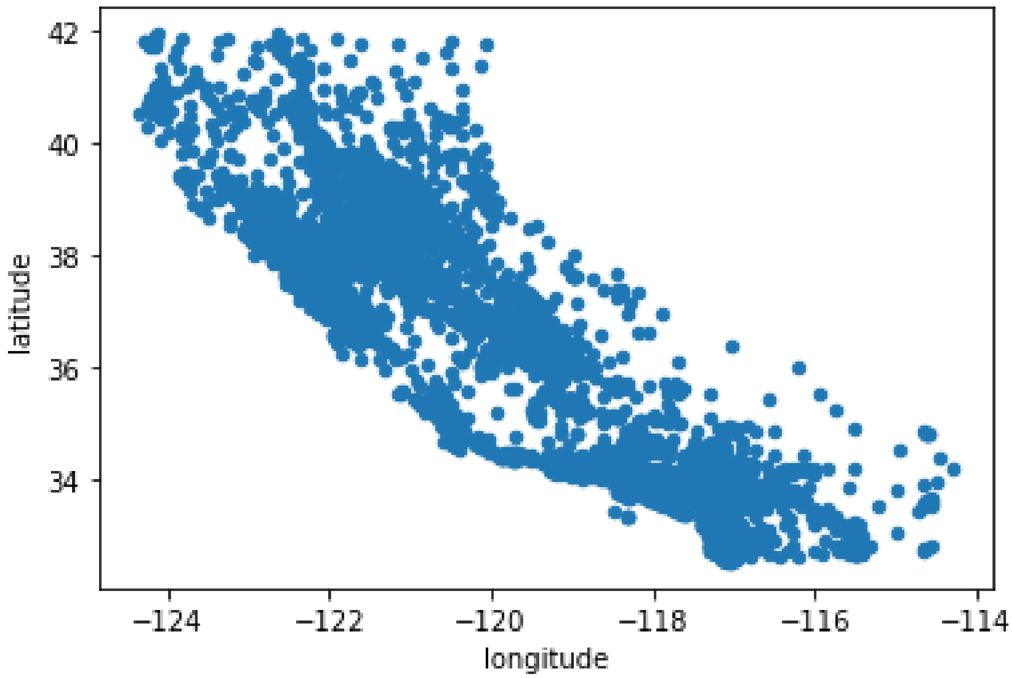
[22]: <AxesSubplot:>



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

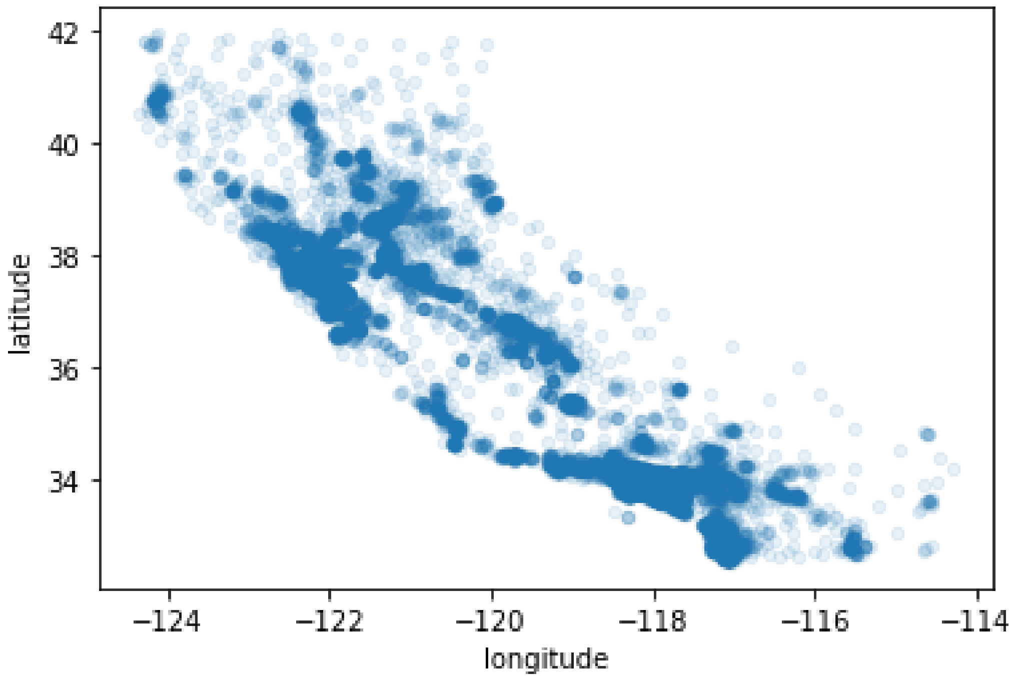
[23]:
```
## here's a not so interesting way of plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
#save_fig("bad_visualization_plot")
```

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



[24]:
```
# 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)
#save_fig("better_visualization_plot")
```

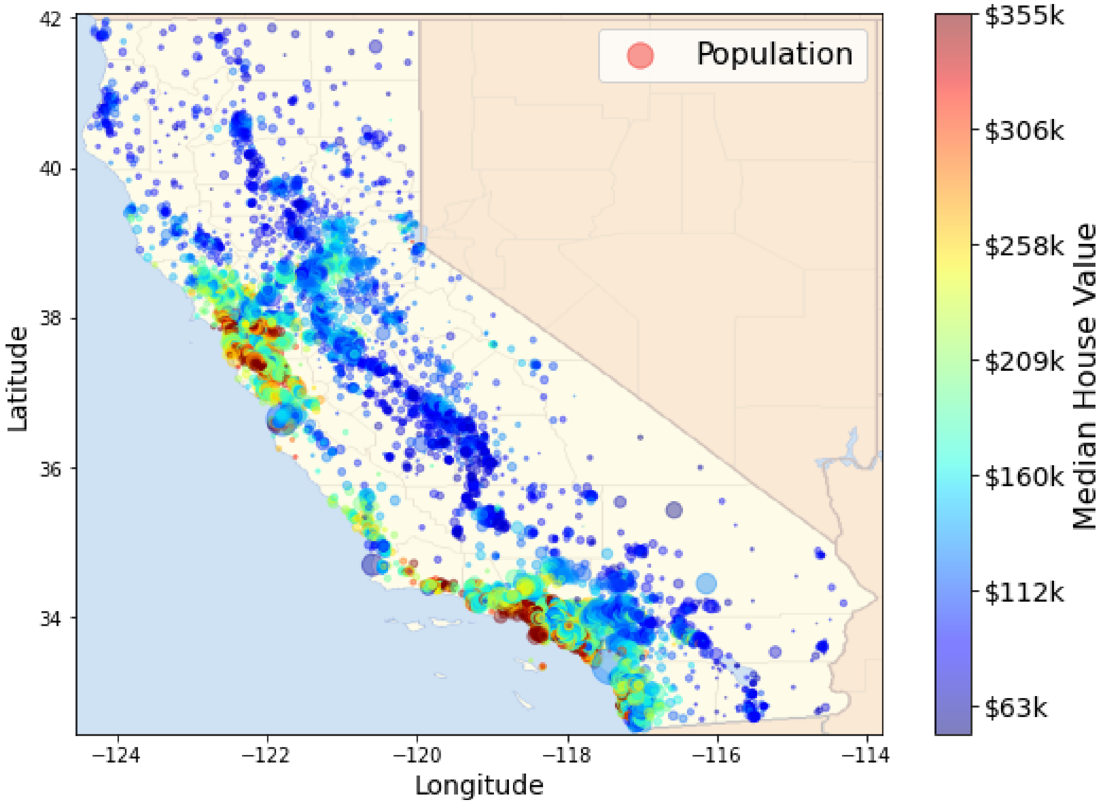
[24]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



[25]:
```
# A more interesting plot is to color code (heatmap) the dots
# based on income. The code below achieves this

# load an image of california
images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename = "california.png"

import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
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)
#save_fig("california_housing_prices_plot")
plt.show()
```

```
/var/folders/98/vwplyq_x2ddf8y2lpl0r0h7w0000gn/T/ipykernel_25187/1369257286.py:2
8: UserWarning: FixedFormatter should only be used together with FixedLocator
  cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values],
fontsize=14)
```



Not suprisingly, we can see that 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 interest 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.

Nonetheless we can explore this using correlation matrices. Each row and column of the correlation matrix represents a non-categorical feature in our dataset and each element specifies the correlation between the row and column features. Correlation is a measure of how the change in one feature affects the other feature. For example, a positive correlation means that as one feature gets larger, then the other feature will also generally get larger. Note that a feature is always fully correlated to itself which is why the diagonal of the correlation matrix is just all 1s.

[26]:
```
corr_matrix = housing.corr()
corr_matrix
```

[26]:
```
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924664           -0.108197     0.044568
latitude            -0.924664  1.000000            0.011173    -0.036100
housing_median_age  -0.108197  0.011173            1.000000    -0.361262
total_rooms          0.044568 -0.036100           -0.361262     1.000000
total_bedrooms       0.069608 -0.066983           -0.320451     0.930380
population           0.099773 -0.108785           -0.296244     0.857126
households           0.055310 -0.071035           -0.302916     0.918484
median_income       -0.015176 -0.079809           -0.119034     0.198050
median_house_value  -0.045967 -0.144160            0.105623     0.134153

                    total_bedrooms  population  households  median_income  \
longitude                 0.069608    0.099773    0.055310      -0.015176
latitude                 -0.066983   -0.108785   -0.071035      -0.079809
housing_median_age       -0.320451   -0.296244   -0.302916      -0.119034
total_rooms               0.930380    0.857126    0.918484       0.198050
total_bedrooms            1.000000    0.877747    0.979728      -0.007723
population                0.877747    1.000000    0.907222       0.004834
households                0.979728    0.907222    1.000000       0.013033
median_income            -0.007723    0.004834    0.013033       1.000000
median_house_value        0.049686   -0.024650    0.065843       0.688075

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

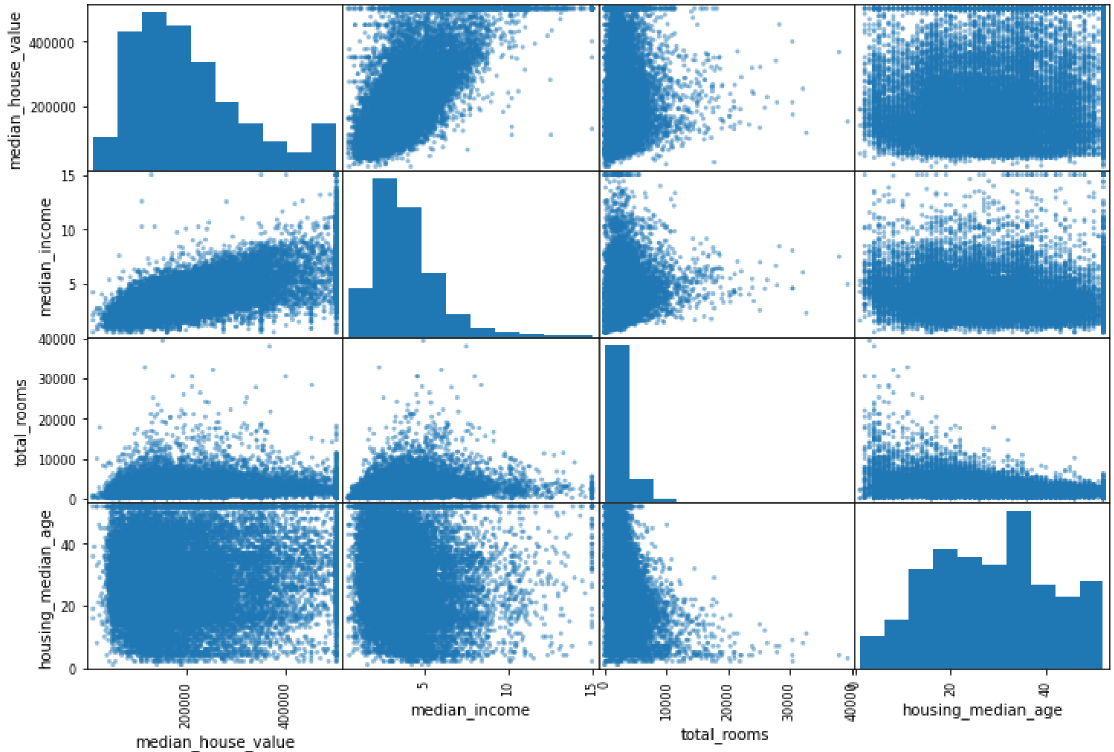
[27]:
```
# for example if the target is "median_house_value", most correlated features␣
↪can be sorted
```

```
# which happens to be "median_income". This also intuitively makes sense.
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[27]:
```
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
```

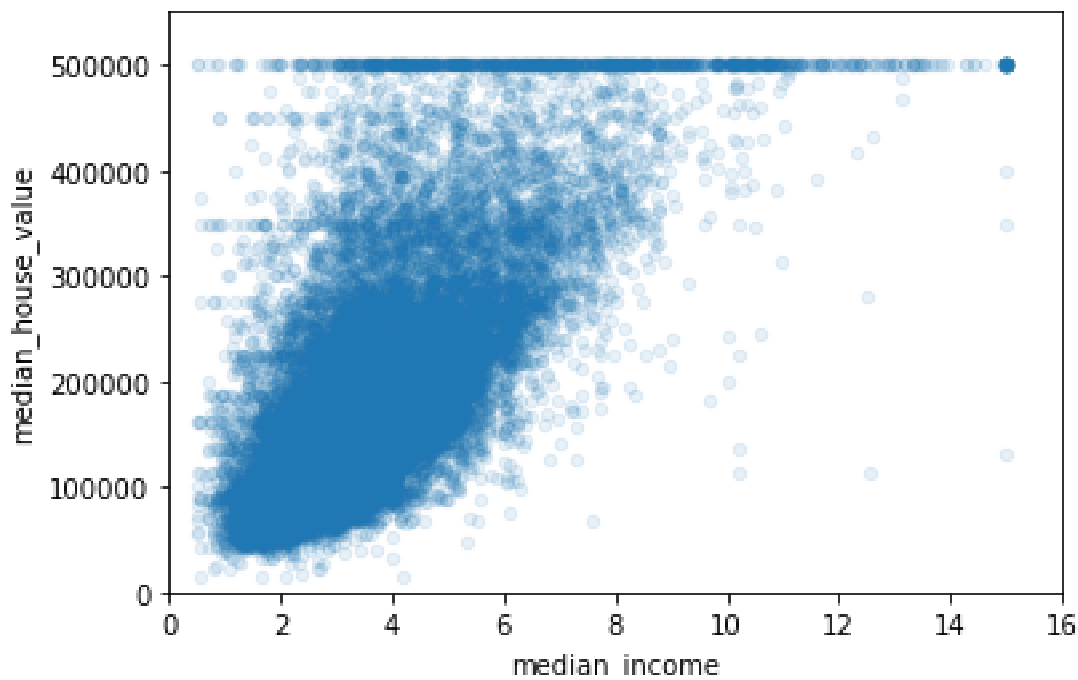
[28]:
```
# We can plot a scatter matrix for different attributes/features
# to see how some features may show a positive correlation/negative correlation␣
 ↪or
# it may turn out to be completely random!
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
#save_fig("scatter_matrix_plot")
```

[28]:
```
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)
```



[29]:
```
# median income vs median house value 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])
#save_fig("income_vs_house_value_scatterplot")
```

[29]:
```
(0.0, 16.0, 0.0, 550000.0)
```



2.3.2 Augmenting Features: Simple Example

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.

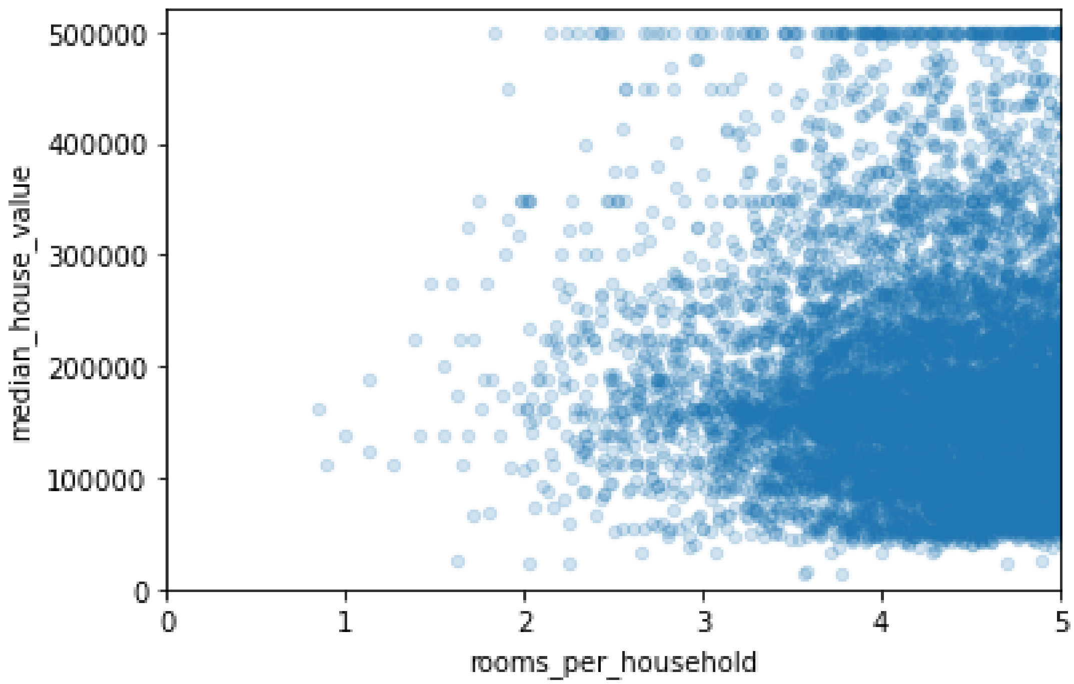
[30]:
```
#A new column in the dataframe can be made the same away you add a new element␣
↪to a dict
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

[31]:
```
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[31]:
```
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
```

[32]:
```python
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



[33]:
```python
housing.describe()
```

[33]:

	longitude	latitude	housing_median_age	total_rooms	\
count	20640.000000	20640.000000	20640.000000	20640.000000	
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	

```
       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
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

       median_house_value  rooms_per_household  bedrooms_per_room  \
count        20640.000000         20640.000000       20433.000000
mean        206855.816909             5.429000           0.213039
std         115395.615874             2.474173           0.057983
min          14999.000000             0.846154           0.100000
25%         119600.000000             4.440716           0.175427
50%         179700.000000             5.229129           0.203162
75%         264725.000000             6.052381           0.239821
max         500001.000000           141.909091           1.000000

       population_per_household
count              20640.000000
mean                   3.070655
std                   10.386050
min                    0.692308
25%                    2.429741
50%                    2.818116
75%                    3.282261
max                 1243.333333
```

2.3.3 Augmenting Features: Advanced Example

In addition to augmenting the data using these simple operations, we can also do some advanced augmentation by bringing information from another dataset.

In this case, we are going to find the distance between the houses and the 10 biggest cities in California during 1990. Intuitively, the location of major cities can strongly impact the value of a home. Thus, our new feature will be the distance of the home to the closest big city among the 10 biggest cities.

To perform this feature extraction, we will use the provided dataset "city_data.csv". We will also employ some helper functions and use the pd.apply function to do the augmentation.

[34]:
```python
#Loads the city data
def load_city_data(housing_path):
    csv_path = os.path.join(housing_path, "city_data.csv")
    return pd.read_csv(csv_path)
```

```
city_data = load_city_data(DATASET_PATH)
city_data
```

[34]:

	City	Latitude	Longitude	Pop_1990
0	Anaheim	33.835292	-117.914503	266406
1	Fresno	36.746842	-119.772586	354202
2	Long Beach	33.768322	-118.195617	429433
3	Los Angeles	34.052233	-118.243686	3485398
4	Oakland	37.804364	-122.271114	372242
5	Sacramento	38.581572	-121.494400	369365
6	San Diego	32.715328	-117.157256	1110549
7	San Francisco	37.774931	-122.419417	723959
8	San Jose	37.339386	-121.894956	782248
9	Santa Ana	33.745572	-117.867833	293742

[35]:

```
#For ease of use, we will convert city_data into a python dict
#where the key is the city name and the value is the coordinates
city_dict = {}
for dat in city_data.iterrows(): #iterates through the rows of the dataframe
    row = dat[1]
    city_dict[row["City"]] = (row["Latitude"],row["Longitude"])

print(city_dict)
```

```
{'Anaheim': (33.835292, -117.914503), 'Fresno': (36.746842, -119.772586), 'Long
Beach': (33.768322, -118.195617), 'Los Angeles': (34.052233, -118.243686),
'Oakland': (37.804364, -122.271114), 'Sacramento': (38.581572, -121.4944), 'San
Diego': (32.715328, -117.157256), 'San Francisco': (37.774931, -122.419417),
'San Jose': (37.339386, -121.894956), 'Santa Ana': (33.745572, -117.867833)}
```

[36]:

```
#Helper functions

#This function is used to calculate the distance between two points on a␣
 ↪latitude and longitude grid.
#You don't need to understand the math but know that it takes into account the␣
 ↪curverature of the earth
#to make an accurate distance measurement.
#While we could have used the geopy package to do this for us, this way we␣
 ↪don't have to install it.
def distance_func(loc_a,loc_b):
    """
    Calculates the haversine distance between coordinates
    on the latitude and longitude grid.
    Distance is in km.
    """
    lat1,lon1 = loc_a
    lat2,lon2 = loc_b
```

```
    r = 6371
    phi1 = np.radians(lat1)
    phi2 = np.radians(lat2)
    delta_phi = np.radians(lat2 - lat1)
    delta_lambda = np.radians(lon2 - lon1)
    a = np.sin(delta_phi / 2)**2 + np.cos(phi1) * np.cos(phi2) *   np.
 ↪sin(delta_lambda / 2)**2
    res = r * (2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a)))
    return np.round(res, 2)


#Calculates closest point to the location given in kilometers
def closest_point(location, location_dict):
    """ take a tuple of latitude and longitude and
        compare to a dictionary of locations where
        key = location name and value = (lat, long)
        returns tuple of (closest_location , distance)
        distance is in kilometers"""
    closest_location = None
    for city in location_dict.keys():
        distance = distance_func(location, location_dict[city])
        if closest_location is None:
            closest_location = (city, distance)
        elif distance < closest_location[1]:
            closest_location = (city, distance)
    return closest_location

#Example
closest_point((37.774931,-120.419417), city_dict)
```

[36]: ('Fresno', 127.85)

[37]:
```
#Now we apply the closest_point function to every data point in housing
#Axis = 1 specifies that apply will send each row one by one into the␣
 ↪designated function
#We use the lambda function to catch the row and then disperse its arguments␣
 ↪into closest_point
housing['close_city'] = housing.apply(lambda x:␣
 ↪closest_point((x['latitude'],x['longitude']),city_dict), axis = 1)

#Since closest point outputed a tuple of names and distance, we have to split␣
 ↪it up.
housing['close_city_name'] = [x[0] for x in housing['close_city'].values]
housing['close_city_dist'] = [x[1] for x in housing['close_city'].values]

#Drop the redundant column
housing = housing.drop('close_city', axis=1)
```

[38]:
```python
#Now, let us look at our new features
housing.head()
```

[38]:
```
   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

  income_cat  rooms_per_household  bedrooms_per_room  \
0          5             6.984127           0.146591
1          5             6.238137           0.155797
2          5             8.288136           0.129516
3          4             5.817352           0.184458
4          3             6.281853           0.172096

   population_per_household close_city_name  close_city_dist
0                  2.555556         Oakland             9.15
1                  2.109842         Oakland             7.64
2                  2.802260         Oakland             5.76
3                  2.547945         Oakland             5.40
4                  2.181467         Oakland             5.40
```

[39]:
```python
#We can also look at the new statistics
housing.describe()
```

[39]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
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

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
```

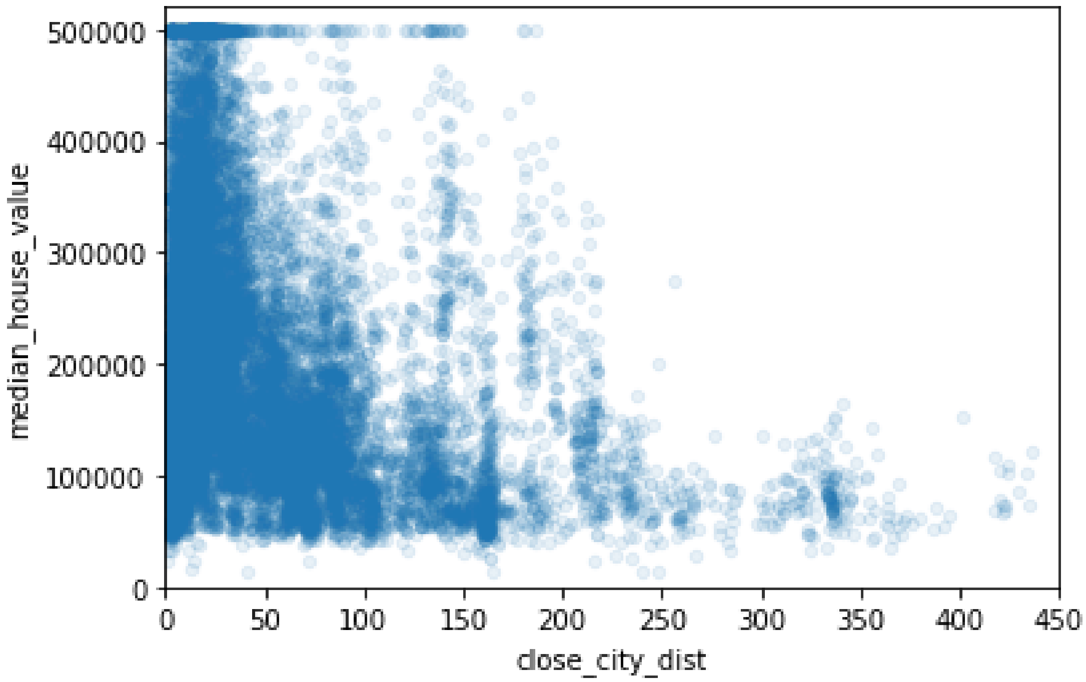
```
std         421.385070   1132.462122    382.329753        1.899822
min           1.000000      3.000000      1.000000        0.499900
25%         296.000000    787.000000    280.000000        2.563400
50%         435.000000   1166.000000    409.000000        3.534800
75%         647.000000   1725.000000    605.000000        4.743250
max        6445.000000  35682.000000   6082.000000       15.000100

       median_house_value  rooms_per_household  bedrooms_per_room  \
count        20640.000000         20640.000000       20433.000000
mean        206855.816909             5.429000           0.213039
std         115395.615874             2.474173           0.057983
min          14999.000000             0.846154           0.100000
25%         119600.000000             4.440716           0.175427
50%         179700.000000             5.229129           0.203162
75%         264725.000000             6.052381           0.239821
max         500001.000000           141.909091           1.000000

       population_per_household  close_city_dist
count              20640.000000     20640.000000
mean                   3.070655        44.045134
std                   10.386050        55.807465
min                    0.692308         0.420000
25%                    2.429741        10.510000
50%                    2.818116        20.930000
75%                    3.282261        58.365000
max                 1243.333333       436.610000
```

Now, let us see if the new feature provides some information about housing prices by looking at the correlation.

[40]:
```python
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[40]:
```
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
close_city_dist            -0.307777
Name: median_house_value, dtype: float64
```

[41]:
```
housing.plot(kind="scatter", x="close_city_dist", y="median_house_value",
             alpha=0.1)
plt.axis([0, 450, 0, 520000])
plt.show()
```



Observation: From the correlation, we can see a negative correlation implying that the farther a house is from a big city, the less it costs. From the plot, we can confirm the negative correlation. We can also note that most houses are within 250 km of the big cities which can indicate that everything past 250 is an outlier or should be treated differently like farm land.

2.4 Step 3. Preprocess the data for your machine learning algorithm

Once we've visualized the data, and have a certain understanding of how the data looks like. It's time to clean!

Most of your time will be spent on this step, although the datasets used in this project are relatively nice and clean… in the real world it could get real dirty.

After having cleaned your dataset you're aiming for: - 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 here!

2.4.1 Dealing With Incomplete Data

[42]:
```
# 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 so we'll have to devise a method for dealing with␣
↪them...
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

[42]:
```
     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  rooms_per_household  bedrooms_per_room  \
290        NEAR BAY          3             5.761468                NaN
341        NEAR BAY          2             3.830116                NaN
538        NEAR BAY          2             4.048704                NaN
563        NEAR BAY          4             6.102740                NaN
696        NEAR BAY          3             4.633540                NaN

     population_per_household close_city_name  close_city_dist
290                  2.614679         Oakland            10.49
341                  2.826255         Oakland            10.75
538                  2.938727         Oakland             2.82
563                  2.630137         Oakland             6.63
696                  2.403727         Oakland            19.70
```

[43]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1: simply␣
↪drop rows that have null values
```

[43]:
```
Empty DataFrame
Columns: [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,
```

```
close_city_name, close_city_dist]
Index: []
```

[44]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop␣
 ↪the complete feature
```

[44]:
```
     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

     rooms_per_household  bedrooms_per_room  population_per_household  \
290             5.761468                NaN                  2.614679
341             3.830116                NaN                  2.826255
538             4.048704                NaN                  2.938727
563             6.102740                NaN                  2.630137
696             4.633540                NaN                  2.403727

    close_city_name  close_city_dist
290         Oakland            10.49
341         Oakland            10.75
538         Oakland             2.82
563         Oakland             6.63
696         Oakland            19.70
```

[45]:
```
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
```

[45]:
```
     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  rooms_per_household  bedrooms_per_room  \
290        NEAR BAY          3             5.761468                NaN
341        NEAR BAY          2             3.830116                NaN
538        NEAR BAY          2             4.048704                NaN
563        NEAR BAY          4             6.102740                NaN
696        NEAR BAY          3             4.633540                NaN

     population_per_household close_city_name  close_city_dist
290                  2.614679         Oakland            10.49
341                  2.826255         Oakland            10.75
538                  2.938727         Oakland             2.82
563                  2.630137         Oakland             6.63
696                  2.403727         Oakland            19.70
```

The option where we replace the null values with a new number is known as imputation.

Could you think of another plausible imputation for this dataset instead of using the median? (Not graded)

Option 1: replace by other statistics: **mean or mode**

Option 2: find complete row with similar features and replace their "total_bedrooms" value.

2.4.2 Using Scikit-learn transformers to preprocess data

We have shown some operations that we want to perform on the dataset. While it is possible to manually perform it all yourselves, it is much easier to offload some of the work to the many fantastic machine learning packages. One such example is scikit-learn where we will demonstrate the use of a transformer to handle some of the work.

Consider a situation where we want to normalize the data for each feature. This involves calculating the mean μ and standard deviation σ for that feature and applying $\frac{z-\mu}{\sigma}$ where z is the feature value. We will show how to perform this using StandardScalar.

[46]:
```python
from sklearn.preprocessing import StandardScaler

#Extract two real valued columns
housing_sub = housing[["housing_median_age","total_rooms"]]

scaler = StandardScaler() #initiate class
#Calling .fit lets scaler calculate the mean and standard deviation, i.e.␣
↪trains the standardizer
scaler.fit(housing_sub)
print("Mean: ",scaler.mean_)
```

```
print("Std: ",scaler.scale_)

#To perform the standardization, use the .transform function
housing_std= scaler.transform(housing_sub)
print("Transfrom output")
print(housing_std)

#As a shorthand, the function .fit_transform performs both operations
housing_std_2= scaler.fit_transform(housing_sub)
print("Fit Transfrom output")
print(housing_std_2)
```

```
Mean:  [  28.63948643 2635.7630814 ]
Std:  [  12.58525273 2181.56240174]
Transfrom output
[[ 0.98214266 -0.8048191 ]
 [-0.60701891  2.0458901 ]
 [ 1.85618152 -0.53574589]
 …
 [-0.92485123 -0.17499526]
 [-0.84539315 -0.35559977]
 [-1.00430931  0.06840827]]
Fit Transfrom output
[[ 0.98214266 -0.8048191 ]
 [-0.60701891  2.0458901 ]
 [ 1.85618152 -0.53574589]
 …
 [-0.92485123 -0.17499526]
 [-0.84539315 -0.35559977]
 [-1.00430931  0.06840827]]
```

2.4.3 Prepare Data using a pipeline

Now, we will show how we can use scikit learn to create a pipeline that performs all the data preparation in one clean function call. For simplicity, we will not perform the closest city feature extraction in this pipeline.

It is very useful to combine several steps into one to make the process much simpler to understand and easy to alter.

[47]:
```
housing = load_housing_data(DATASET_PATH) # Load the dataset

housing_features = housing.drop("median_house_value", axis=1) # drop labels for␣
 ↪training set features
                                                              # the input to the model␣
 ↪should not contain the true label
housing_target = housing["median_house_value"].copy()
```

[48]:
```
housing_features.head()
```

[48]:
```
   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 ocean_proximity
0       322.0       126.0         8.3252        NEAR BAY
1      2401.0      1138.0         8.3014        NEAR BAY
2       496.0       177.0         7.2574        NEAR BAY
3       558.0       219.0         5.6431        NEAR BAY
4       565.0       259.0         3.8462        NEAR BAY
```

[49]:
```
# This cell implements the complete pipeline for preparing the data
# using sklearns TransformerMixins
# Earlier we mentioned different types of features: categorical, and floats.
# In the case of floats we might want to convert them to categories.
# On the other hand categories in which are not already represented as integers␣
↪must be mapped to integers before
# feeding to the model.

# Additionally, categorical values could either be represented as one-hot␣
↪vectors or simple as normalized/unnormalized integers.
# Here we encode them using one hot vectors.

# DO NOT WORRY IF YOU DO NOT UNDERSTAND ALL THE STEPS OF THIS PIPELINE.␣
↪CONCEPTS LIKE NORMALIZATION,
# ONE-HOT ENCODING ETC. WILL ALL BE COVERED IN DISCUSSION

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin
, ↪

######Processing Real Valued Features
# column indices
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class AugmentFeatures(BaseEstimator, TransformerMixin):
```

```
    '''
    implements the previous features we had defined
    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"]
    '''
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        #Note that we do not use the pandas indexing anymore
        #This is due to sklearn transforming the dataframe into a numpy array␣
 ↪during the processing
        #Thus, depending on where AugmentFeatures is in the pipeline, a␣
 ↪different input type can be expected
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

#Example of using AugmentFeatures
housing_features_num = housing_features.drop("ocean_proximity", axis=1) #␣
 ↪remove the categorical features
attr_adder = AugmentFeatures(add_bedrooms_per_room=False) #Create transformer␣
 ↪object
housing_extra_attribs = attr_adder.transform(housing_features_num.values)␣
 ↪#housing_num.values extracts the numpy array of the datafram

print("Example of Augment Features Transformer")
print(housing_extra_attribs[0])


#Pipiline for real valued features
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")), #Imputes using median
        ('attribs_adder', AugmentFeatures(add_bedrooms_per_room=True)), #
        ('std_scaler', StandardScaler()),
    ])
```

```
#Example
#Output is a numpy array
housing_features_num_tr = num_pipeline.fit_transform(housing_features_num)
print("Example Output of Pipeline for numerical output")
print(housing_features_num_tr[0])
```

```
Example of Augment Features Transformer
[-122.23         37.88         41.          880.          129.
  322.          126.            8.3252        6.98412698    2.55555556]
Example Output of Pipeline for numerical output
[-1.32783522  1.05254828  0.98214266 -0.8048191  -0.97247648 -0.9744286
 -0.97703285  2.34476576  0.62855945 -0.04959654 -1.02998783]
```

[50]:

```
#Full Pipeline

#Splits names into numerical and categorical features
numerical_features = list(housing_features_num)
categorical_features = ["ocean_proximity"]

#Applies different transformations on numerical columns vs categorial columns
full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])


#Example of full pipeline
#Output is a numpy array
housing_prepared = full_pipeline.fit_transform(housing_features)
print("Example Output of full Pipeline")
print(housing_prepared[0])
```

```
Example Output of full Pipeline
[-1.32783522  1.05254828  0.98214266 -0.8048191  -0.97247648 -0.9744286
 -0.97703285  2.34476576  0.62855945 -0.04959654 -1.02998783  0.
  0.          0.          1.          0.        ]
```

Now, we have a pipeline that easily processes the input data into our desired form.

2.4.4 Splitting our dataset

First we need to carve out our dataset into a training and testing cohort. To do this we'll use train_test_split, a very elementary tool that arbitrarily splits the data into training and testing cohorts.

Note that we first perform the train test split on the data before it was processed in the pipeline and then separatelyprocess the train and test data. This is done to avoid injecting information into the test data from the train data such filling in missing values in the test data with knowledge of the train data.

[51]:
```
from sklearn.model_selection import train_test_split
data_target = housing['median_house_value']
train, test, target, target_test = train_test_split(housing_features,␣
↪data_target, test_size=0.3, random_state=0)

train = full_pipeline.fit_transform(train)
test = full_pipeline.transform(test)
```

2.4.5 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.

[52]:
```
from sklearn.linear_model import LinearRegression

#Instantiate a linear regresion class
lin_reg = LinearRegression()
#Train the class using the .fit function
lin_reg.fit(train, target)

# let's try the full preprocessing pipeline on a few training instances
data = test
labels = target_test

#Uses predict to get the predicted target values
print("Predictions:", lin_reg.predict(data)[:5])
print("Actual labels:", list(labels)[:5])
```

```
Predictions: [211700.27512595 283365.83148515 179320.30143123  92739.78279873
 295847.53922669]
Actual labels: [136900.0, 241300.0, 200700.0, 72500.0, 460000.0]
```

[53]:
```
from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(test)
mse = mean_squared_error(target_test, preds)
rmse = np.sqrt(mse)
rmse
```

[53]:
```
69599.33569383297
```

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

We will apply what we've learnt to another dataset (NYC airbnb dataset from 2019). We will predict airbnb price based on other features.

Note: You do not have to use only one cell when programming your code and can do it over multiple cells.

3.1 [50 pts] Visualizing Data

3.1.1 [10 pts] Load the data + statistics

- Load the dataset: airbnb/AB_NYC_2019.csv and display the first 5 few rows of the data

[54]:
```
def load_airbnb_data(airbnb_path):
    csv_path = os.path.join(airbnb_path, "AB_NYC_2019.csv")
    return pd.read_csv(csv_path)
```

[55]:
```
DATASET_PATH_AIRBNB = os.path.join("datasets", "airbnb")
airbnb = load_airbnb_data(DATASET_PATH_AIRBNB)
airbnb.head(5)
```

[55]:
```
     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
```

As we can see, there has different types of features and missing values in the dataset.

- Pull up info on the data type for each of the data fields. Will any of these be problematic feeding into your model (you may need to do a little research on this)? Discuss:

[56]:
```
airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48879 non-null  object
 2   host_id                         48895 non-null  int64
 3   host_name                       48874 non-null  object
 4   neighbourhood_group             48895 non-null  object
 5   neighbourhood                   48895 non-null  object
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     38843 non-null  object
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

1. incompatible data formats: if model cannot handle categorical data, then it will run well.
2. Bias on model model may develop a bias towards one type of data leading to poor performance.
3. Overfitting or underfitting model may not properly learn the patterns in the data due to the presence of different data types

[There are different types of features such as real valued (latitude, longitude…), discrete (id, host_id…), categorical (name, host_name…). We should drop a few features that should not helpful such as id and host_id and see if we can convert data type from discrete to real valued.]

- Drop the following columns: name, id, host_id, host_name, last_review, neighbourhood, and reviews_per_month and display first 5 rows

[57]:
```
drop_columns = ["name",
                "id",
                "host_id",
                "host_name",
```

```
                "last_review",
                "neighbourhood",
                "reviews_per_month"]
airbnb = airbnb.drop(drop_columns, axis=1)
airbnb.head()
```

[57]:

```
  neighbourhood_group  latitude  longitude        room_type  price  \
0            Brooklyn  40.64749  -73.97237     Private room    149
1           Manhattan  40.75362  -73.98377  Entire home/apt    225
2           Manhattan  40.80902  -73.94190     Private room    150
3            Brooklyn  40.68514  -73.95976  Entire home/apt     89
4           Manhattan  40.79851  -73.94399  Entire home/apt     80

   minimum_nights  number_of_reviews  calculated_host_listings_count  \
0               1                  9                               6
1               1                 45                               2
2               3                  0                               1
3               1                270                               1
4              10                  9                               1

   availability_365
0               365
1               355
2               365
3               194
4                 0
```

- Display a summary of the statistics of the loaded data using .describe

[58]:

```
airbnb.describe()
```

[58]:

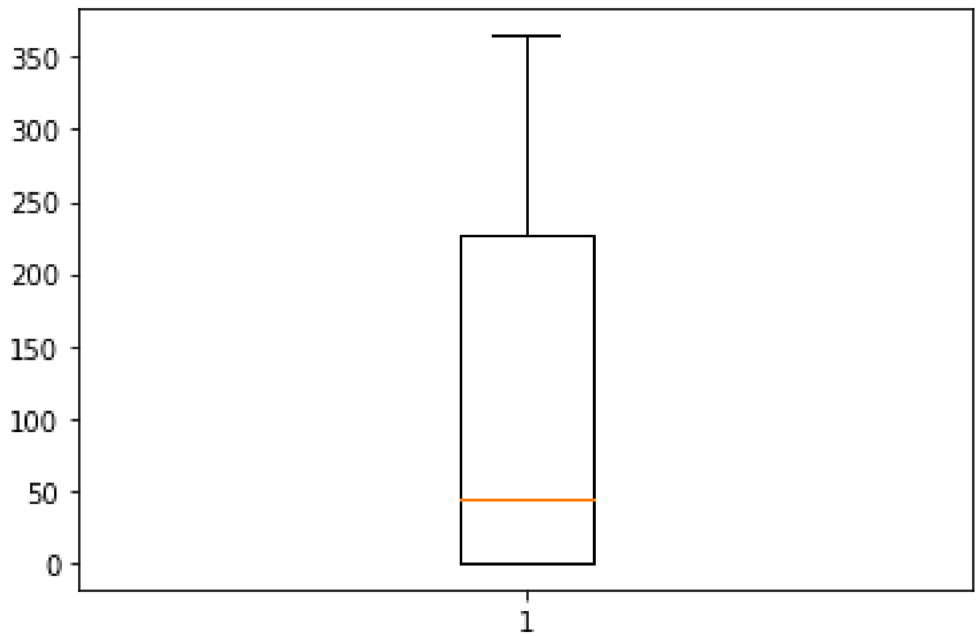
```
           latitude     longitude         price  minimum_nights  \
count  48895.000000  48895.000000  48895.000000    48895.000000
mean      40.728949    -73.952170    152.720687        7.029962
std        0.054530      0.046157    240.154170       20.510550
min       40.499790    -74.244420      0.000000        1.000000
25%       40.690100    -73.983070     69.000000        1.000000
50%       40.723070    -73.955680    106.000000        3.000000
75%       40.763115    -73.936275    175.000000        5.000000
max       40.913060    -73.712990  10000.000000     1250.000000

       number_of_reviews  calculated_host_listings_count  availability_365
count       48895.000000                    48895.000000      48895.000000
mean           23.274466                        7.143982        112.781327
std            44.550582                       32.952519        131.622289
min             0.000000                        1.000000          0.000000
25%             1.000000                        1.000000          0.000000
50%             5.000000                        1.000000         45.000000
```

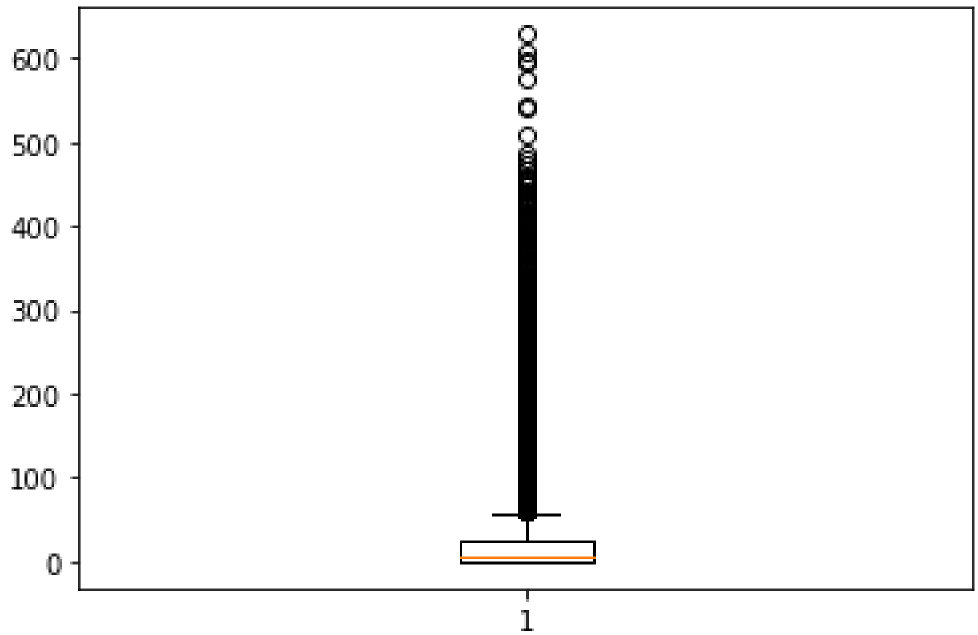
75%	24.000000	2.000000	227.000000
max	629.000000	327.000000	365.000000

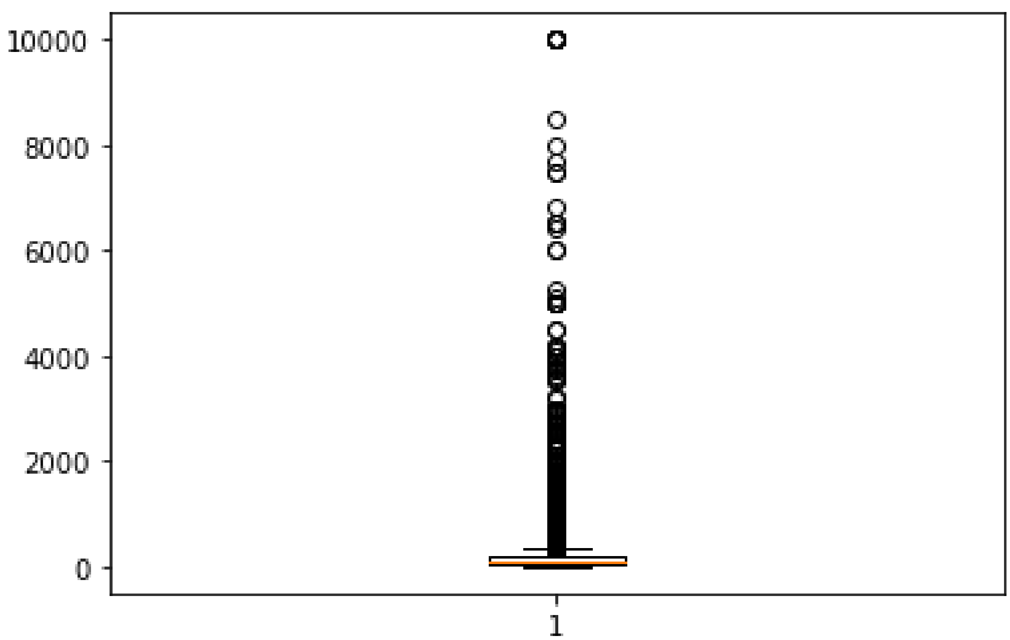
3.1.2 [10 pts] Plot boxplots for the following 3 features: availability_365, number_of_reviews, price

You may use either pandas or matplotlib to plot the boxplot

[59]:
```
boxplot_features=["availability_365",
                  "number_of_reviews",
                  "price"]
for i in range(len(boxplot_features)):
    plt.boxplot(airbnb[boxplot_features[i]])
    plt.show()
```







- What do you observe from the boxplot about the features? Anything suprising?
availability box plot: mean availability is around 50 days, and Q1 is around 0, Q3 around 225 days. Maximum available day is more than 350days.

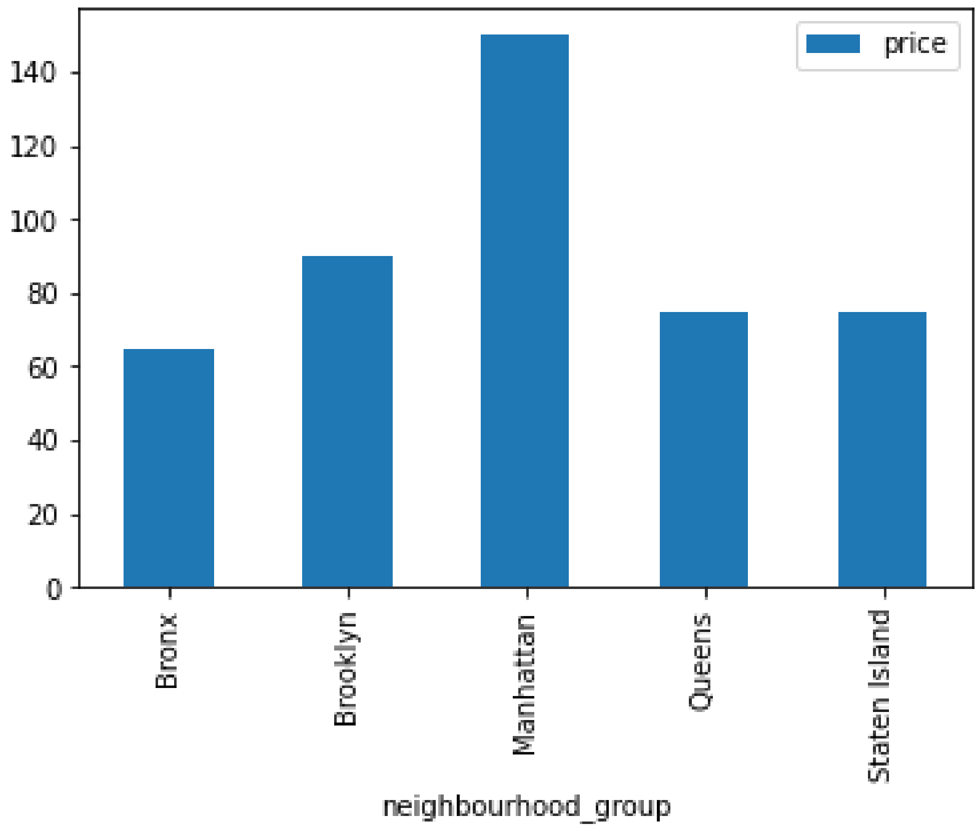
number of reviews box plot: median number of rewivew is around 25, and Q1 is around 0, Q3 around 50. It's right-skewed. But there's a lot of outliers.

price box plot: median price is very low which uner 2000 a lot. It's also right-skewed distribution with a lot of outliers.

3.1.3 [10 pts] Plot median price of a listing per neighbourhood_group using a bar plot

[60]:
```
median_price = airbnb.groupby(['neighbourhood_group']).median()
median_price.plot.bar(y='price')
```

[60]: <AxesSubplot:xlabel='neighbourhood_group'>



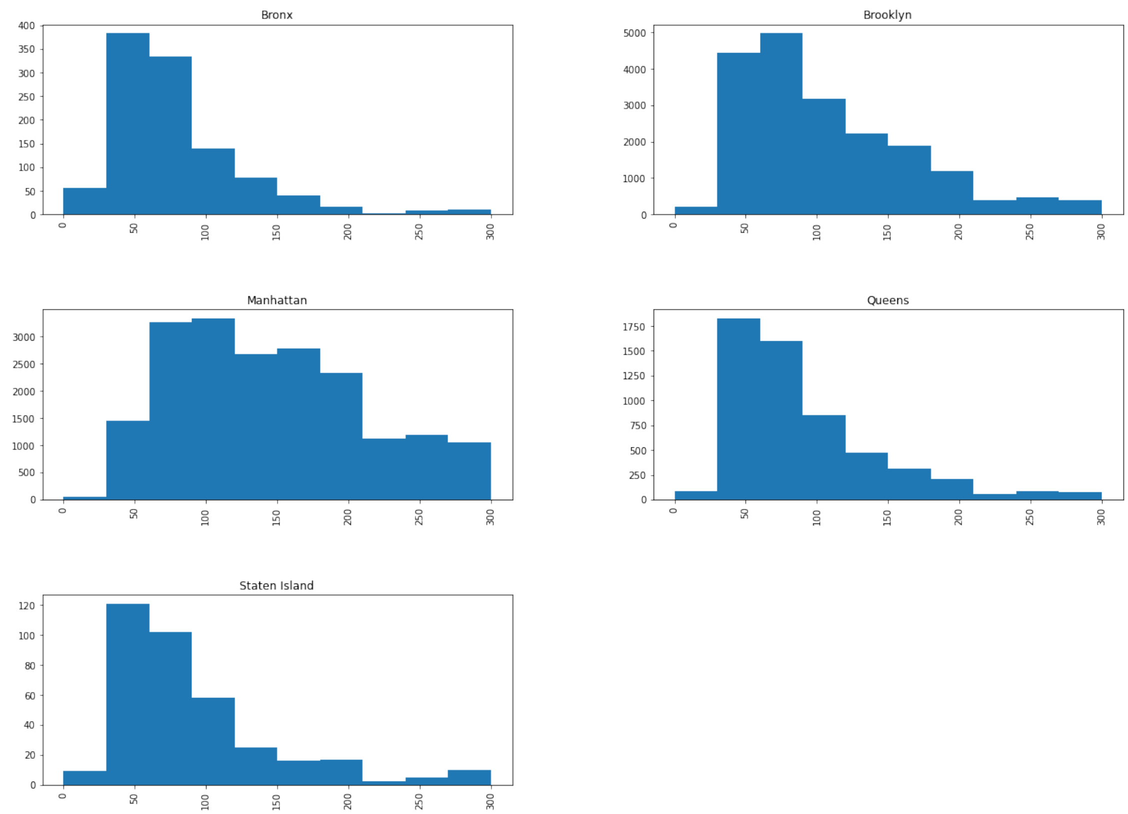
- Describe what you expected to see with these features and what you actually observed
[Expected: price of airbnb will be different based on location and neighbourhood]

[Actually: Median price of airbnb are effected a lot by different neighbourhood_group. Airbnb at Manhattan is much expensive than other places]

- So we can see different neighborhoods have dramatically different pricepoints, but how does the price breakdown by range. To see let's do a histogram of price by

neighborhood to get a better sense of the distribution. To prevent outliers from affecting the histogram, use the input *range = [0,300]* in the histogram function which will upperbound the max price to 300 and ignore the outliers.

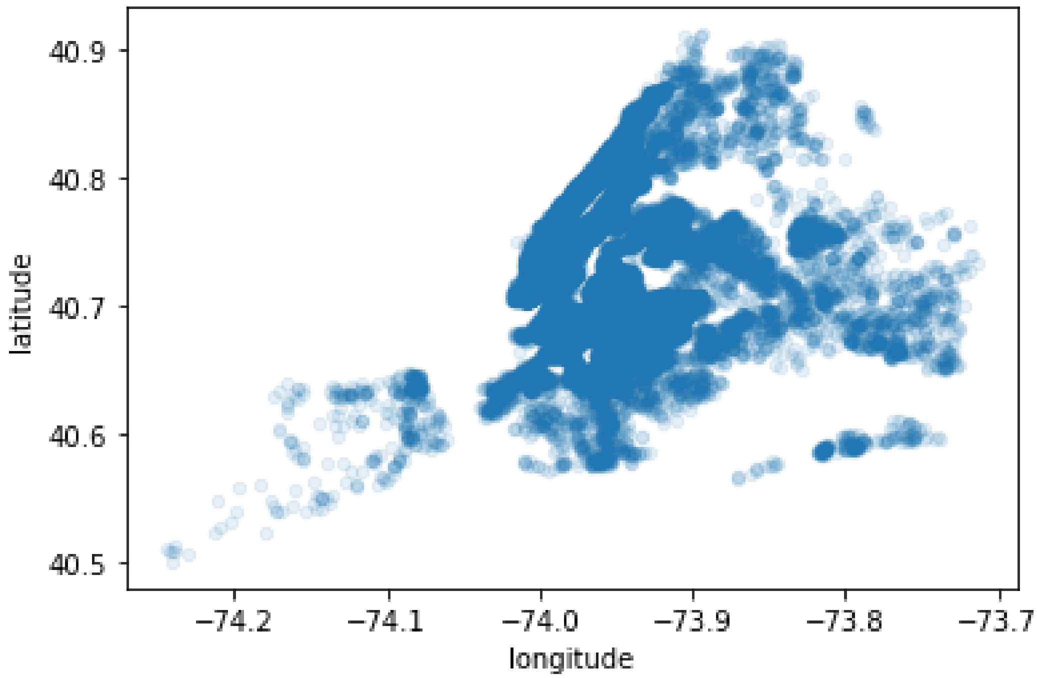
[61]:
```
#Your code
airbnb['price'].hist(by=airbnb['neighbourhood_group'], range = [0,300],figsize=(20,15))
plt.show()
```



3.1.4 [5 pts] Plot a map of airbnbs throughout New York. You do not need to overlay a map.

[62]:
```
airbnb.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

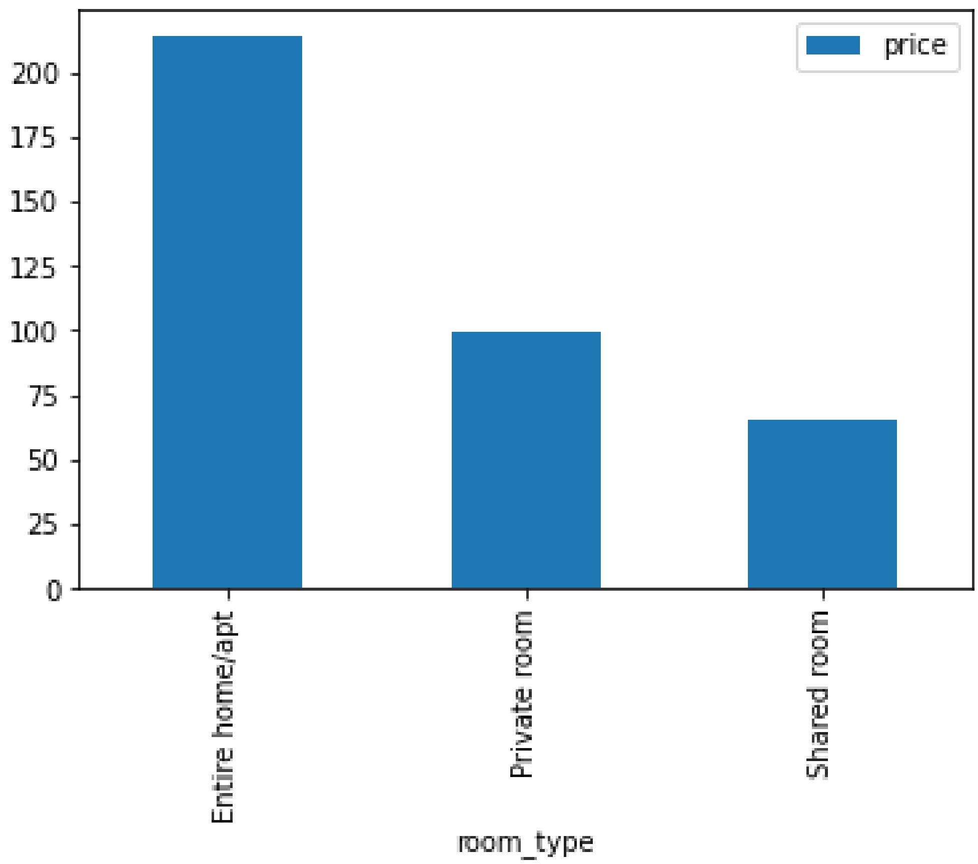
[62]:
```
<AxesSubplot:xlabel='longitude', ylabel='latitude'>
```



3.1.5 [10 pts] Plot median price of room types who have availability greater than 180 days and neighbourhood_group is Manhattan

[63]:
```
Manhattan_180 = airbnb[(airbnb["availability_365"] > 180)&␣
  ↪(airbnb["neighbourhood_group"] == 'Manhattan')]
Manhattan_180_median = Manhattan_180.groupby('room_type').median()
Manhattan_180_median.plot.bar(y='price')
```

[63]: <AxesSubplot:xlabel='room_type'>



3.1.6 [5 pts] Find features that correlate with price

Using the correlation matrix: - which features have positive correlation with the price? - which features have negative correlation with the price?

[64]:
```
corr_matrix_airbnb = airbnb.corr()
corr_matrix_airbnb
```

[64]:
```
                                latitude  longitude     price  minimum_nights  \
latitude                        1.000000   0.084788  0.033939        0.024869
longitude                       0.084788   1.000000 -0.150019       -0.062747
price                           0.033939  -0.150019  1.000000        0.042799
minimum_nights                  0.024869  -0.062747  0.042799        1.000000
number_of_reviews              -0.015389   0.059094 -0.047954       -0.080116
calculated_host_listings_count  0.019517  -0.114713  0.057472        0.127960
availability_365               -0.010983   0.082731  0.081829        0.144303

                                number_of_reviews  \
latitude                                -0.015389
longitude                                0.059094
```

```
price                                 -0.047954
minimum_nights                        -0.080116
number_of_reviews                      1.000000
calculated_host_listings_count        -0.072376
availability_365                       0.172028

                                calculated_host_listings_count  \
latitude                                              0.019517
longitude                                            -0.114713
price                                                 0.057472
minimum_nights                                        0.127960
number_of_reviews                                    -0.072376
calculated_host_listings_count                        1.000000
availability_365                                      0.225701

                                availability_365
latitude                               -0.010983
longitude                               0.082731
price                                   0.081829
minimum_nights                          0.144303
number_of_reviews                       0.172028
calculated_host_listings_count          0.225701
availability_365                        1.000000
```

[65]:

```
corr_matrix_airbnb["price"].sort_values(ascending=False)
```

[65]:

```
price                             1.000000
availability_365                  0.081829
calculated_host_listings_count    0.057472
minimum_nights                    0.042799
latitude                          0.033939
number_of_reviews                -0.047954
longitude                        -0.150019
Name: price, dtype: float64
```

[Positive Correlation: availability_365, calculated_host_listings_count, minimum_nights, latitude]

[Negative Correlation: number_of_reviews, longitude]

- Plot the full Scatter Matrix to see the correlation between prices and the other features

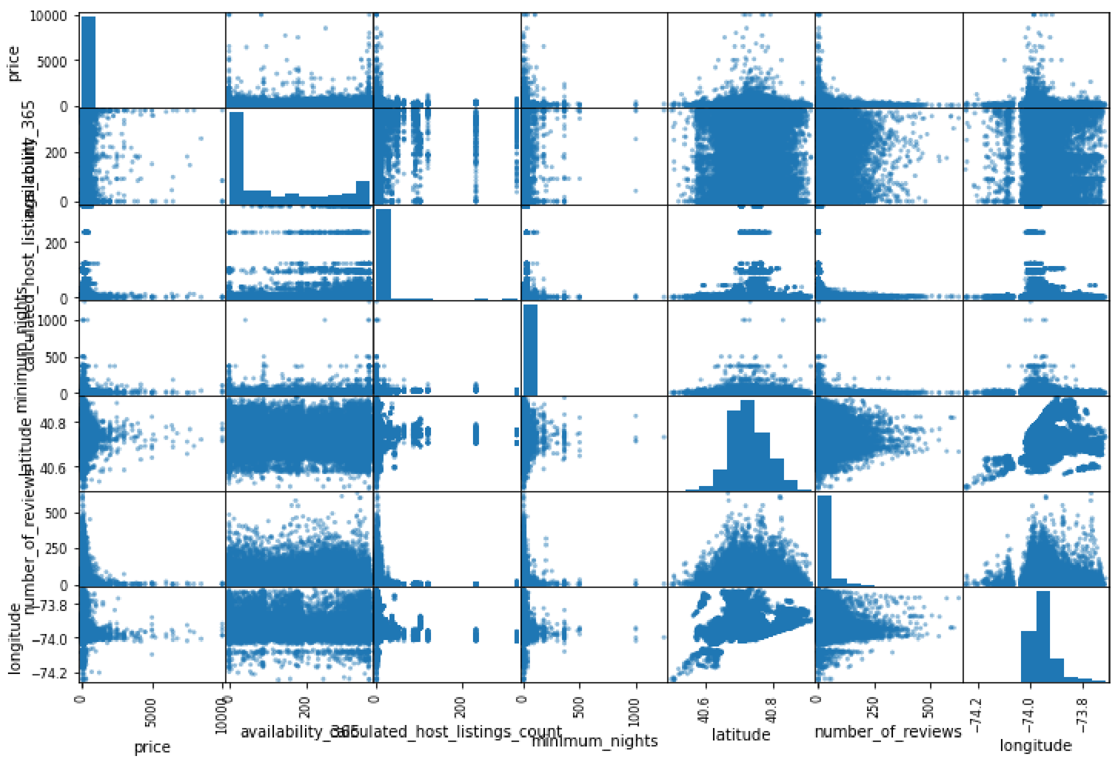
[66]:

```
from pandas.plotting import scatter_matrix
attributes = ["price",
              "availability_365",
              "calculated_host_listings_count",
              "minimum_nights",
             "latitude",
```

```
            "number_of_reviews",
             "longitude"]
scatter_matrix(airbnb[attributes], figsize=(12, 8))
```

[66]:
```
array([[<AxesSubplot:xlabel='price', ylabel='price'>,
        <AxesSubplot:xlabel='availability_365', ylabel='price'>,
        <AxesSubplot:xlabel='calculated_host_listings_count', ylabel='price'>,
        <AxesSubplot:xlabel='minimum_nights', ylabel='price'>,
        <AxesSubplot:xlabel='latitude', ylabel='price'>,
        <AxesSubplot:xlabel='number_of_reviews', ylabel='price'>,
        <AxesSubplot:xlabel='longitude', ylabel='price'>],
       [<AxesSubplot:xlabel='price', ylabel='availability_365'>,
        <AxesSubplot:xlabel='availability_365', ylabel='availability_365'>,
        <AxesSubplot:xlabel='calculated_host_listings_count',
ylabel='availability_365'>,
        <AxesSubplot:xlabel='minimum_nights', ylabel='availability_365'>,
        <AxesSubplot:xlabel='latitude', ylabel='availability_365'>,
        <AxesSubplot:xlabel='number_of_reviews', ylabel='availability_365'>,
        <AxesSubplot:xlabel='longitude', ylabel='availability_365'>],
       [<AxesSubplot:xlabel='price', ylabel='calculated_host_listings_count'>,
        <AxesSubplot:xlabel='availability_365',
ylabel='calculated_host_listings_count'>,
        <AxesSubplot:xlabel='calculated_host_listings_count',
ylabel='calculated_host_listings_count'>,
        <AxesSubplot:xlabel='minimum_nights',
ylabel='calculated_host_listings_count'>,
        <AxesSubplot:xlabel='latitude',
ylabel='calculated_host_listings_count'>,
        <AxesSubplot:xlabel='number_of_reviews',
ylabel='calculated_host_listings_count'>,
        <AxesSubplot:xlabel='longitude',
ylabel='calculated_host_listings_count'>],
       [<AxesSubplot:xlabel='price', ylabel='minimum_nights'>,
        <AxesSubplot:xlabel='availability_365', ylabel='minimum_nights'>,
        <AxesSubplot:xlabel='calculated_host_listings_count',
ylabel='minimum_nights'>,
        <AxesSubplot:xlabel='minimum_nights', ylabel='minimum_nights'>,
        <AxesSubplot:xlabel='latitude', ylabel='minimum_nights'>,
        <AxesSubplot:xlabel='number_of_reviews', ylabel='minimum_nights'>,
        <AxesSubplot:xlabel='longitude', ylabel='minimum_nights'>],
       [<AxesSubplot:xlabel='price', ylabel='latitude'>,
        <AxesSubplot:xlabel='availability_365', ylabel='latitude'>,
        <AxesSubplot:xlabel='calculated_host_listings_count',
ylabel='latitude'>,
        <AxesSubplot:xlabel='minimum_nights', ylabel='latitude'>,
        <AxesSubplot:xlabel='latitude', ylabel='latitude'>,
        <AxesSubplot:xlabel='number_of_reviews', ylabel='latitude'>,
```

```
 <AxesSubplot:xlabel='longitude', ylabel='latitude'>],
[<AxesSubplot:xlabel='price', ylabel='number_of_reviews'>,
 <AxesSubplot:xlabel='availability_365', ylabel='number_of_reviews'>,
 <AxesSubplot:xlabel='calculated_host_listings_count',
ylabel='number_of_reviews'>,
 <AxesSubplot:xlabel='minimum_nights', ylabel='number_of_reviews'>,
 <AxesSubplot:xlabel='latitude', ylabel='number_of_reviews'>,
 <AxesSubplot:xlabel='number_of_reviews', ylabel='number_of_reviews'>,
 <AxesSubplot:xlabel='longitude', ylabel='number_of_reviews'>],
[<AxesSubplot:xlabel='price', ylabel='longitude'>,
 <AxesSubplot:xlabel='availability_365', ylabel='longitude'>,
 <AxesSubplot:xlabel='calculated_host_listings_count',
ylabel='longitude'>,
 <AxesSubplot:xlabel='minimum_nights', ylabel='longitude'>,
 <AxesSubplot:xlabel='latitude', ylabel='longitude'>,
 <AxesSubplot:xlabel='number_of_reviews', ylabel='longitude'>,
 <AxesSubplot:xlabel='longitude', ylabel='longitude'>]],
dtype=object)
```



3.2 [30 pts] Prepare the Data

3.2.1 [5 pts] Partition the data into the features and the target data. The target data is price. Then partition the feature data into categorical and numerical features.

[80]:
```
target = airbnb['price']
features = airbnb.drop('price', axis=1)
categorical_features = ['neighbourhood_group',
                        'room_type']
categorical_data = airbnb[categorical_features]
numerical_features = ['latitude',
                      'longitude',
                      'minimum_nights',
                      'number_of_reviews',
                      'calculated_host_listings_count',
                      'availability_365']
numerical_data = airbnb[numerical_features]
```

3.2.2 [10 pts] Create a scikit learn Transformer that augments the numerical data with the following two features

- Max_yearly_bookings = availability_365 / minimum_nights
- Distance from airbnb to the NYC JFK Airport
  - Latitude: 40.641766 , Longitude: -73.780968

Make sure to append these new features in this order.

You may use the previously defined distance_func for the distance calculation.

Note that this Transformer will be applied after imputation so we do not have to worry about Nulls in the data.

[70]:
```
loc_JFK=(40.641766,-73.780968)
class AugmentFeaturesAirbnb(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    airbnb["Max_yearly_bookings"] = airbnb['availability_365']/
↪airbnb['minimum_nights']
    airbnb['Distance_to_JFK'] = distance_func(location of airbnb, loc_JFK)
    '''
    def __init__(self):
        self.availabel_ix = 5
        self.min_nights_ix = 2
        self.latitude_ix = 0
        self.longtitude_ix = 1
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        Max_yearly_bookings = X[:, self.availabel_ix] / X[:, self.min_nights_ix]
```

```
        Distance_to_JFK = np.apply_along_axis(lambda x: distance_func((x[self.
↪latitude_ix],
                                                                        x[self.
↪longtitude_ix]), loc_JFK),1,X)
        return np.c_[X, Max_yearly_bookings, Distance_to_JFK]
```

-Test your new agumentation class by applying it to the numerical data you created. Print out the first 3 rows of the resultant data. Do not worry about missing data since none of the features we used involved nulls.

[81]:
```
adder_airbnb = AugmentFeaturesAirbnb()
extra_airbnb = adder_airbnb.transform(numerical_data.values)
print(extra_airbnb[0:3])
```

```
[[ 40.64749    -73.97237      1.            9.            6.
  365.         365.          16.16       ]
 [ 40.75362    -73.98377      1.           45.            2.
  355.         355.          21.14       ]
 [ 40.80902    -73.9419       3.            0.            1.
  365.         121.66666667  23.02       ]]
```

3.2.3 [10 pts] Create a sklearn pipeline that performs the following operations of the feature data

Now, we will create a full pipeline that processes the data before creating the model.

For the numerical data, perfrom the following operations in order: - Use a SimpleImputer that imputes using the median value - Use the custom feature augmentation made in the previous part - Use StandardScaler to standardize the mean and standard deviation

For categorical features, perform the following: - Perform one hot encoding on all the remaining categorical features: {neighbourhood_group, room_type}

After making the pipeline, perform the transform operation on the feature data and print out the first 3 rows.

[82]:
```
from sklearn.preprocessing import StandardScaler

#Pipiline for real valued features
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeaturesAirbnb()),
        ('std_scaler', StandardScaler()),
    ])

#Full Pipeline

#Applies different transformations on numerical columns vs categorial columns
full_pipeline = ColumnTransformer([
```

```
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(categories='auto', sparse=False),␣
 ↪categorical_features),
    ])


#Example of full pipeline
#Output is a numpy array
airbnb_prepared = full_pipeline.fit_transform(features)
print("Example Output of full Pipeline")
print(airbnb_prepared[0:3,])
```

```
Example Output of full Pipeline
[[-1.4938492  -0.43765209 -0.29399621 -0.32041358 -0.03471643  1.91625031
   3.59673033 -0.49694202  0.          1.          0.          0.
   0.          0.          1.          0.        ]
 [ 0.45243602 -0.68463915 -0.29399621  0.48766493 -0.15610444  1.84027456
   3.48197973  0.65100123  0.          0.          1.          0.
   0.          1.          0.          0.        ]
 [ 1.46839948  0.22249666 -0.19648442 -0.52243321 -0.18645145  1.91625031
   0.80446575  1.08436134  0.          0.          1.          0.
   0.          0.          1.          0.        ]]
```

3.2.4 [5 pts] Set aside 20% of the data as test test (80% train, 20% test). Apply previously created pipeline to the train and test data separately as shown in the introduction example.

[99]:
```
from sklearn.model_selection import train_test_split

target = airbnb['price']
features = airbnb.drop(['price'], axis=1)

train_X, test_X, train_Y, test_Y = train_test_split(features, target,␣
 ↪test_size=0.2, random_state=0)

train_X = full_pipeline.fit_transform(train_X)
test_X = full_pipeline.transform(test_X)
```

3.3 [20 pts] 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 the mean squared error (MSE). Provide both test and train set MSE values.

[107]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(train_X, train_Y)
print("predictions of test is", lin_reg.predict(test_X)[:5],
```

```
    "\nactual value of test is", list(test_Y[:5]))
```

```
predictions of test is [260.13023711 266.26127786 172.2907673  113.20779642
166.37660682]
actual value of test is [225, 649, 300, 26, 125]
```

[89]:
```
from sklearn.metrics import mean_squared_error
preds_train = lin_reg.predict(train_X)
mse_train=mean_squared_error(train_Y, preds_train)

preds_test = lin_reg.predict(test_X)
mse_test = mean_squared_error(test_Y, preds_test)
print("train MSE",mse_train, "\ntest MSE", mse_test)
```

```
train MSE 52635.8112639203
test MSE 48605.52201240954
```

We can see test MSE smaller than train MSE which means it could be a good model and may not have overfitting problems.