Project1-Main

April 13, 2021

0.1 Introduction

Welcome to **CS188 - 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.

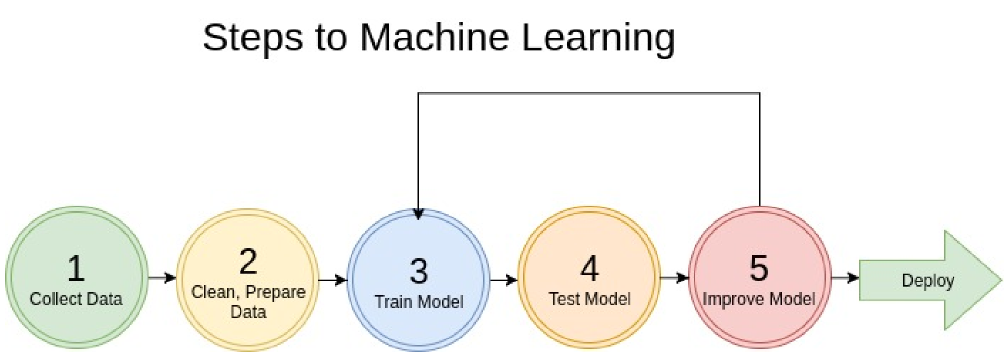
For this first project we will introduce you to the end-to-end process of doing a datascience project. Our goals for this project are to:

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



steps

0.2 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

0.3 Submission Instructions

When you have completed this assignment please save the notebook as a PDF file and submit the assignment via Gradescope

1 Example Datascience Exercise

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

1.1 Setup

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

import numpy as np #numerical package in python
import os
%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

# 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
```

```
        Args:
            fig_name (str): name of the figrue
            tight_layout (bool): adjust subplot to fit in the figure area
            fig_extension (str): file format to save the figure in
            resolution (int): figure resolution
    '''
    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]:
```
import os
import tarfile
import urllib
DATASET_PATH = os.path.join("datasets", "housing")
```

1.2 Step 1. Getting the data

1.2.1 Intro to Data Exploration Using Pandas

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

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

[3]:
```
import pandas as pd

def load_housing_data(housing_path):
    '''
        loads housing.csv dataset stored

        Args:
            housing_path (str): path to folder containing housing datased

        Returns:
            pd.DataFrame
    '''
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

[4]:
```
pd.DataFrame
```

[4]:
```
pandas.core.frame.DataFrame
```

[5]:
```
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
```

[5]:

```
   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.

[6]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

[7]:

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

[7]:
```
0    NEAR BAY
1    NEAR BAY
2    NEAR BAY
3    NEAR BAY
4    NEAR BAY
Name: ocean_proximity, dtype: object
```

[8]:
```python
# to access a particular row we can use iloc
housing.iloc[1]
```

[8]:
```
longitude               -122.22
latitude                  37.86
housing_median_age           21
total_rooms                7099
total_bedrooms             1106
population                 2401
households                 1138
median_income            8.3014
median_house_value       358500
ocean_proximity        NEAR BAY
Name: 1, dtype: object
```

[9]:
```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()
```

[9]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

[10]:
```python
# The describe function compiles your typical statistics for each
# column
housing.describe()
```

[10]:
```
          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
```

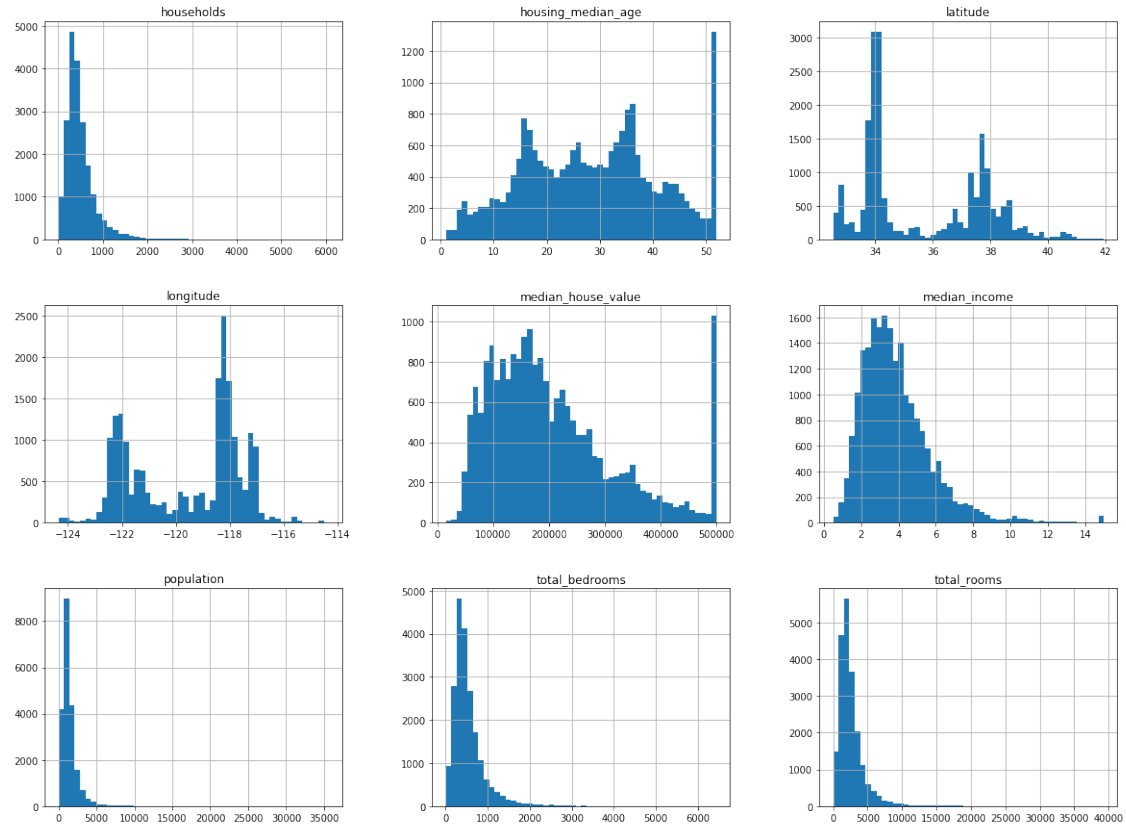
```
mean      537.870553   1425.476744    499.539680       3.870671
std       421.385070   1132.462122    382.329753       1.899822
min         1.000000      3.000000      1.000000       0.499900
25%       296.000000    787.000000    280.000000       2.563400
50%       435.000000   1166.000000    409.000000       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
```

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

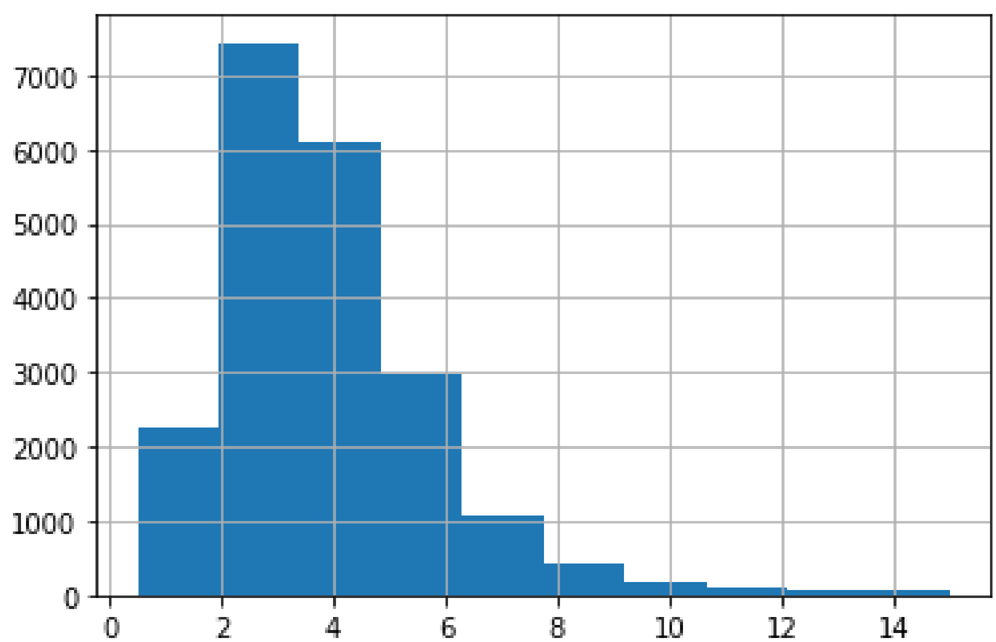
1.3 Step 2. Visualizing the data

1.3.1 Let's start visualizing the dataset

[11]:
```
# 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
```



[12]:
```
# 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.

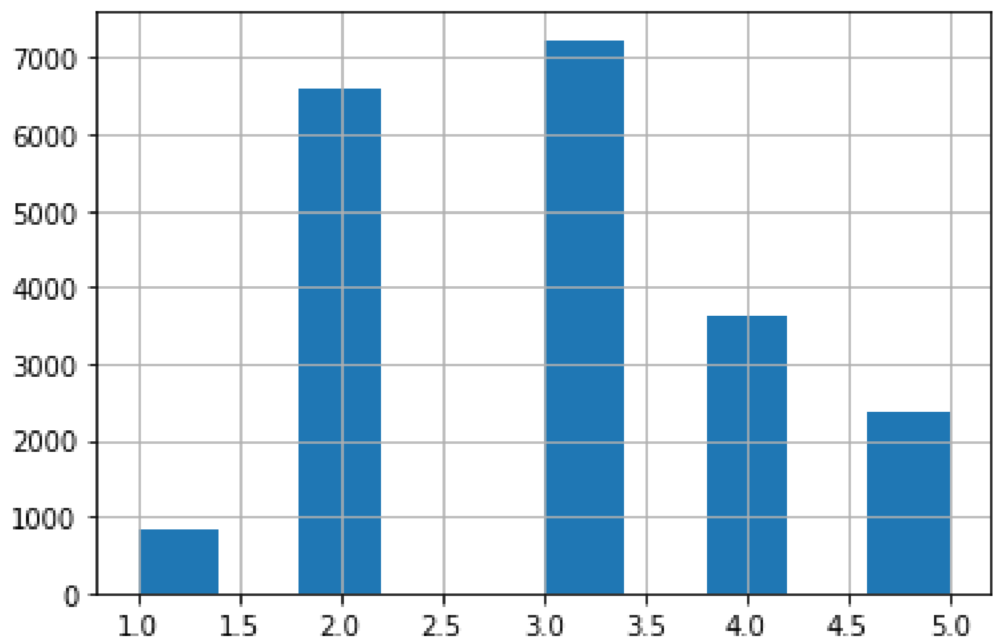
For example, to bin the households based on median_income we can use the pd.cut function

[13]:
```
# 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()
```

[13]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

[14]:
```
housing["income_cat"].hist()
```

[14]: <matplotlib.axes._subplots.AxesSubplot at 0xa11751f98>

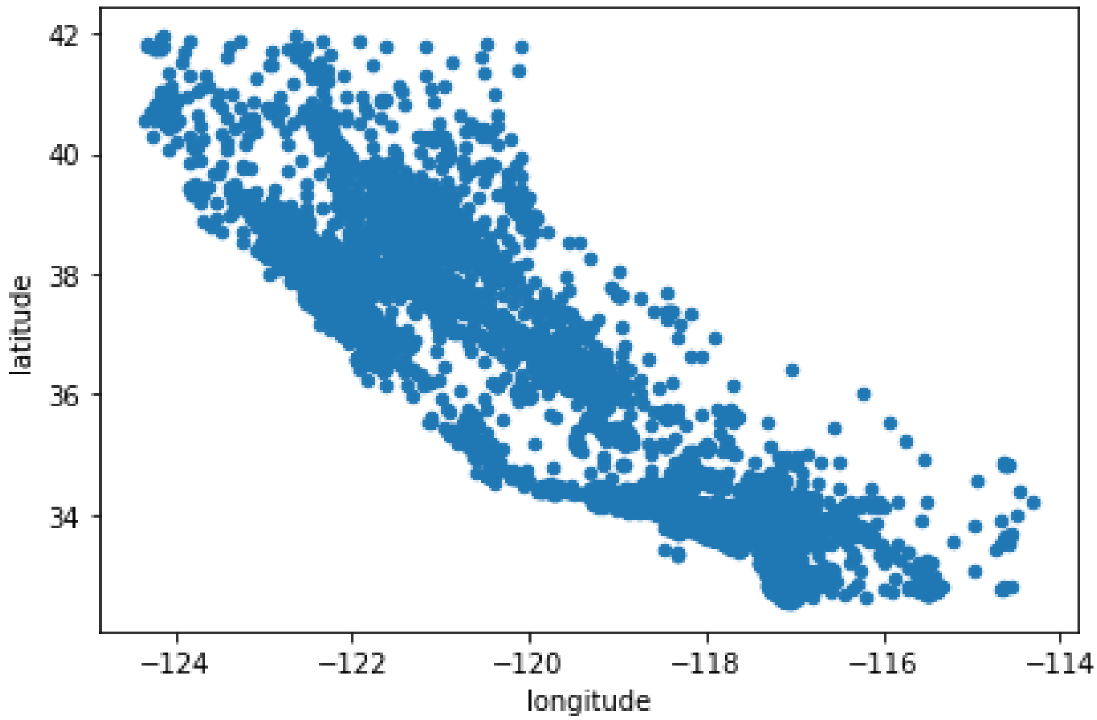


Next let's visualize the household incomes based on latitude & longitude coordinates

[15]:
```
## here's a not so interestting way of plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
```

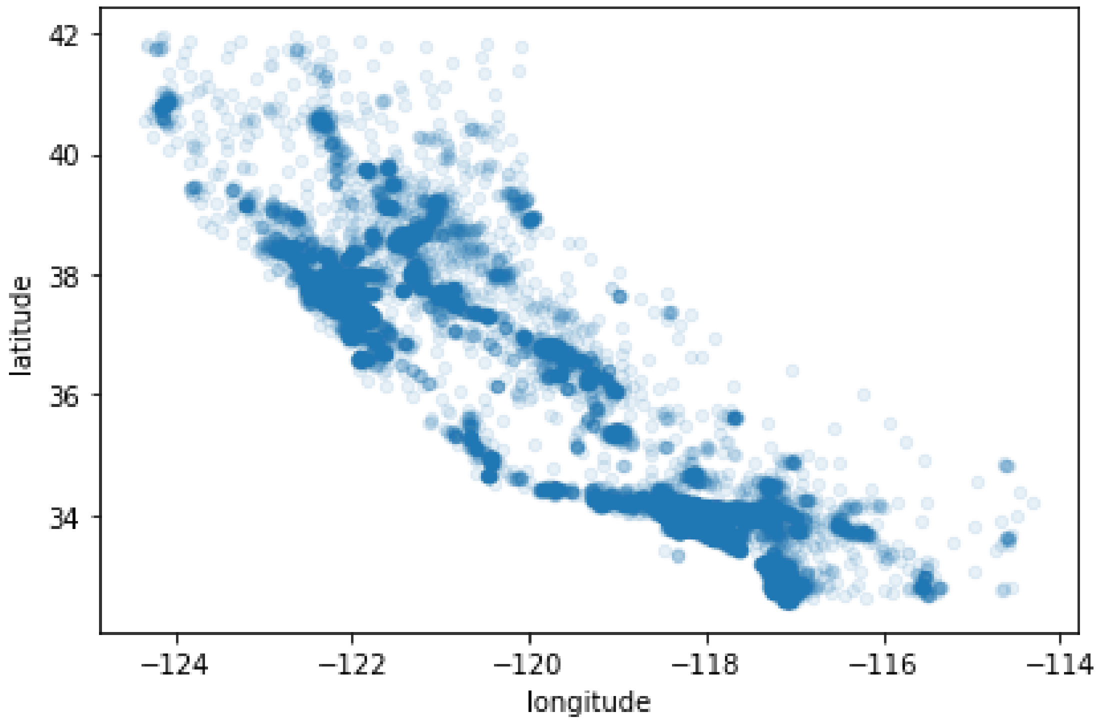
```
save_fig("bad_visualization_plot")
```

```
Saving figure bad_visualization_plot
```



[16]:
```
# 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")
```

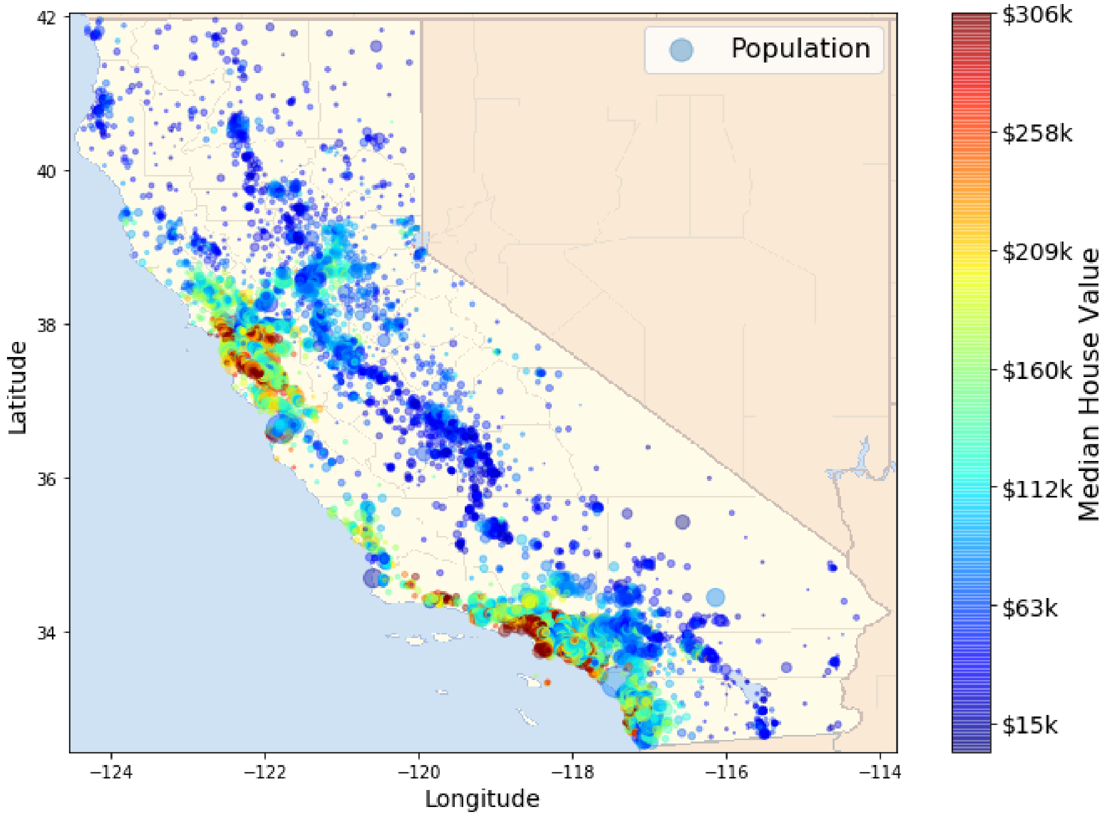
```
Saving figure better_visualization_plot
```



[17]:
```
# 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()
```

```
Saving figure california_housing_prices_plot
```



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 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. If you need to brush up on correlation take a look here.

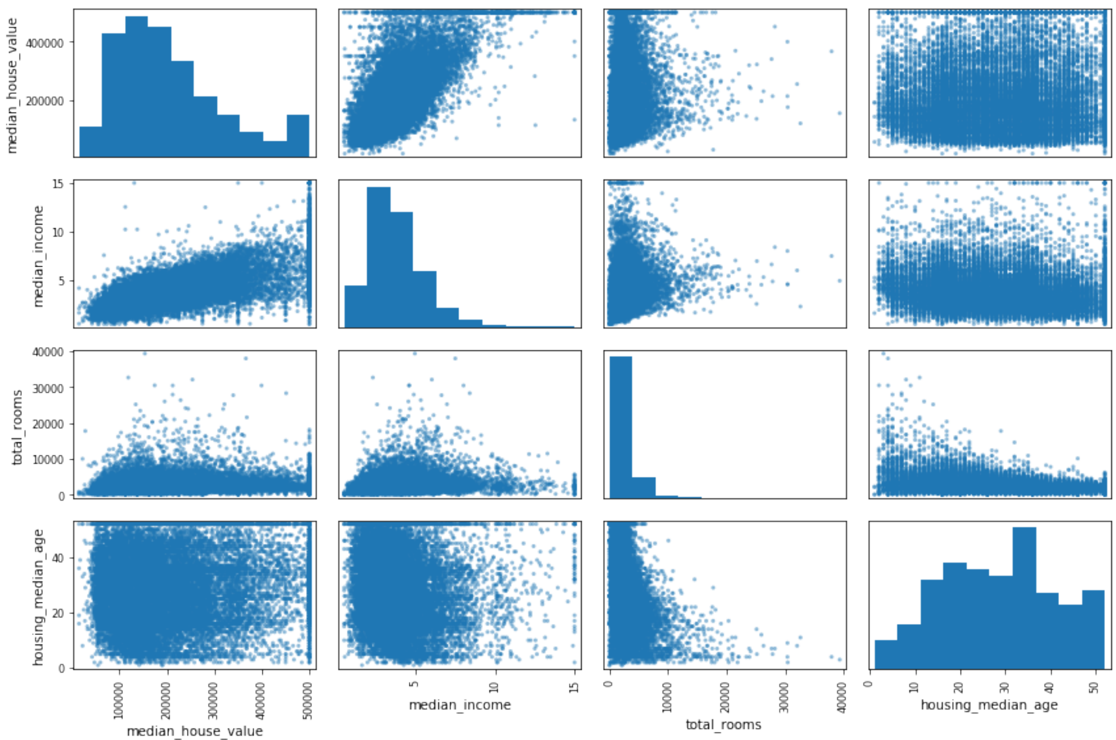
[18]:
```
corr_matrix = housing.corr() # compute the correlation matrix
```

[19]:
```
# 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)
```

[19]:
```
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
```

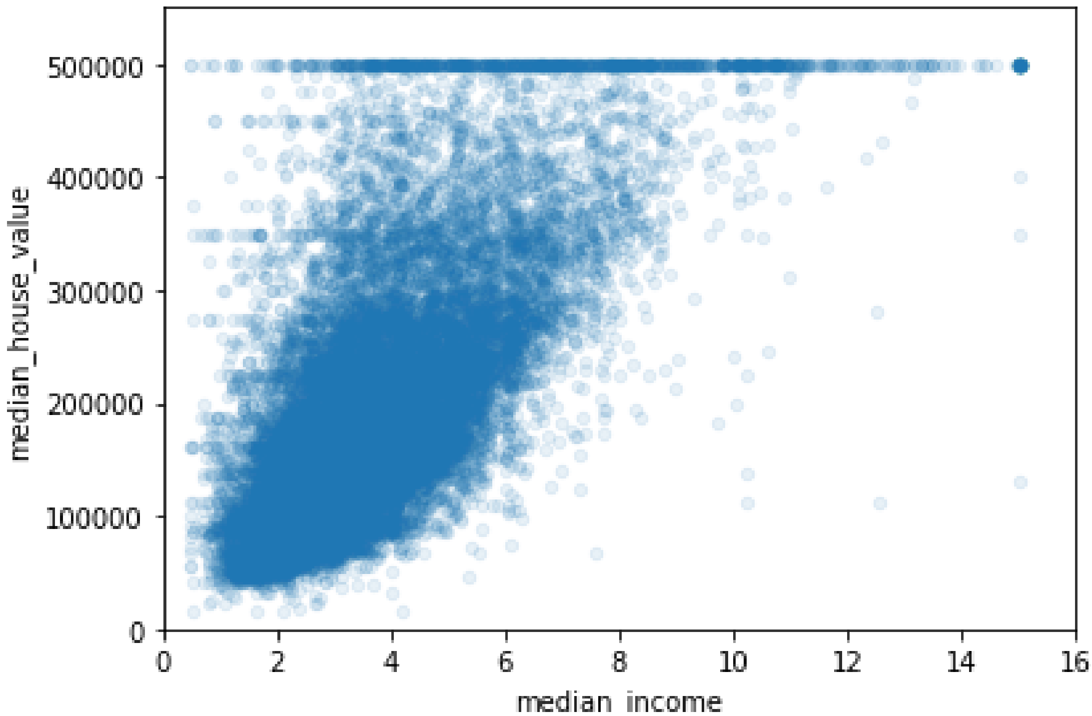
[20]:
```
# the correlation matrix for different attributes/features can also be plotted
# some features may show a positive correlation/negative correlation or
# it may turn out to be completely random!
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```

```
Saving figure scatter_matrix_plot
```



[21]:
```
# median income vs median house vlue plot plot 2 in the first row of top figure
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
save_fig("income_vs_house_value_scatterplot")
```

```
Saving figure income_vs_house_value_scatterplot
```



1.3.2 Augmenting Features

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

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

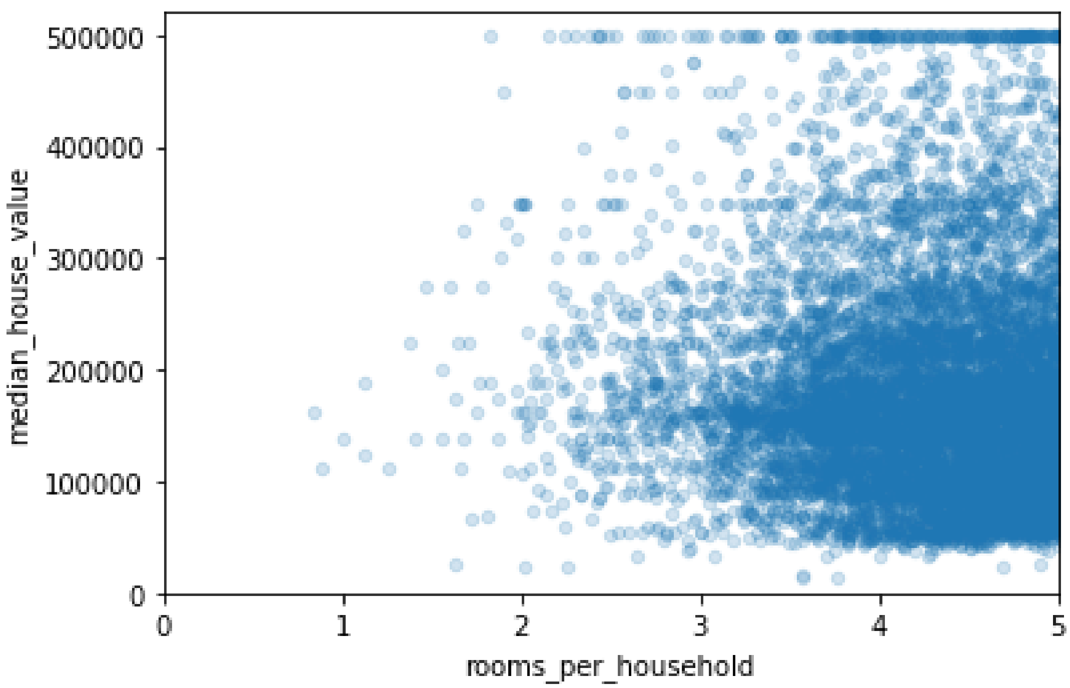
[22]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

[23]:
```
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

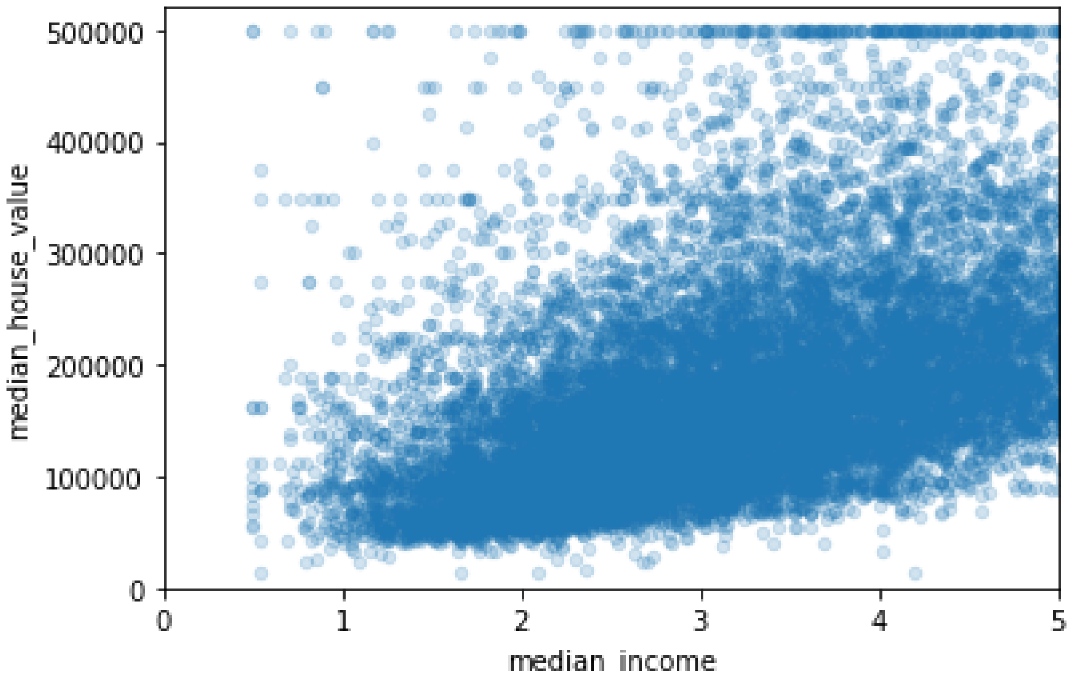
[23]:
```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
total_bedrooms              0.049686
population_per_household   -0.023737
population                 -0.024650
```

```
longitude                     -0.045967
latitude                      -0.144160
bedrooms_per_room             -0.255880
Name: median_house_value, dtype: float64
```

[24]:
```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



[25]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



[26]:
```
housing.describe()
```

[26]:
```
          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
```

1.4 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!

1.4.1 Dealing With Incomplete Data

[27]:
```
# 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
```

[27]:
```
     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
290                  2.614679
341                  2.826255
538                  2.938727
563                  2.630137
696                  2.403727
```

[28]:
```
sample_incomplete_rows.dropna(subset=["total_bedroms"])    # option 1: simply␣
↪drop rows that have null values
```

[28]:
```
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]
Index: []
```

[29]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop␣
↪the complete feature
```

[29]:
```
     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
```

[30]:
```
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
```

[30]:
```
     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
290                  2.614679
341                  2.826255
538                  2.938727
563                  2.630137
696                  2.403727
```

Could you think of another plausible imputation for this dataset? (Not graded)

1.4.2 Prepare Data

Recall we are trying to predict the median house value, our features will contain longitude, latitude, housing_median_age… and our target will be median_house_value

[31]:
```
```

```
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_labels = housing["median_house_value"].copy()
```

[32]:
```
housing_features.head()
```

[32]:
```
   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 income_cat  \
0       322.0       126.0         8.3252        NEAR BAY          5
1      2401.0      1138.0         8.3014        NEAR BAY          5
2       496.0       177.0         7.2574        NEAR BAY          5
3       558.0       219.0         5.6431        NEAR BAY          4
4       565.0       259.0         3.8462        NEAR BAY          3

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

[33]:
```
# 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


imputer = SimpleImputer(strategy="median") # use median imputation for missing␣
 ↪values
housing_num = housing_features.drop("ocean_proximity", axis=1) # remove the␣
 ↪categorical feature
# column index
rooms_idx, bedrooms_idx, population_idx, households_idx = 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):
        rooms_per_household = X[:, rooms_idx] / X[:, households_idx]
        population_per_household = X[:, population_idx] / X[:, households_idx]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_idx] / X[:, rooms_idx]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = AugmentFeatures(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values) # generate new␣
 ↪features

# this will be are numirical pipeline
```

```
# 1. impute, 2. augment the feature set 3. normalize using StandardScaler()
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)

numerical_features = list(housing_num)
categorical_features = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

housing_prepared = full_pipeline.fit_transform(housing_features)
```

1.4.3 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.

[34]:
```
from sklearn.model_selection import train_test_split
data_target = housing['median_house_value']
train, test, target, target_test = train_test_split(housing_prepared,␣
 ↪data_target, test_size=0.3, random_state=0)
```

1.4.4 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.

[35]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(train, target)

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

print("Predictions:", lin_reg.predict(data)[:5])
print("Actual labels:", list(labels)[:5])
```

```
Predictions: [207828.06448011 281099.80175494 176021.36890539  93643.46744928
 304674.47047758]
Actual labels: [136900.0, 241300.0, 200700.0, 72500.0, 460000.0]
```

[36]:
```python
from sklearn.metrics import mean_squared_error

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

[36]:
```
67879.86844243006
```

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

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

3 [35 pts] Visualizing Data

3.0.1 [5 pts] Load the data + statistics

- load the dataset
- display the first few rows of the data

[37]:
```python
DATASET_PATH = os.path.join("datasets", "airbnb")
```

[38]:
```python
def load_airbnb_data(data_path):
    csv_path = os.path.join(data_path, "AB_NYC_2019.csv")
    return pd.read_csv(csv_path)
```

[39]:
```python
df_main = load_airbnb_data(DATASET_PATH)
df_main.head(10)
```

[39]:
```
     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
5  5099         Large Cozy 1 BR Apartment In Midtown East     7322
6  5121                                   BlissArtsSpace!     7356
7  5178                  Large Furnished Room Near B'way      8967
8  5203                   Cozy Clean Guest Room - Family Apt     7490
9  5238                Cute & Cozy Lower East Side 1 bdrm     7549

     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
5       Chris          Manhattan         Murray Hill  40.74767  -73.97500
6       Garon           Brooklyn  Bedford-Stuyvesant  40.68688  -73.95596
7    Shunichi          Manhattan      Hell's Kitchen  40.76489  -73.98493
8   MaryEllen          Manhattan     Upper West Side  40.80178  -73.96723
9         Ben          Manhattan           Chinatown  40.71344  -73.99037

         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
5  Entire home/apt    200               3                 74  2019-06-22
6     Private room     60              45                 49  2017-10-05
7     Private room     79               2                430  2019-06-24
8     Private room     79               2                118  2017-07-21
9  Entire home/apt    150               1                160  2019-06-09

   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
5               0.59                               1               129
6               0.40                               1                 0
7               3.47                               1               220
8               0.99                               1                 0
9               1.33                               4               188
```

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

[40]:
```
df_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
id                                  48895 non-null int64
name                                48879 non-null object
host_id                             48895 non-null int64
host_name                           48874 non-null object
neighbourhood_group                 48895 non-null object
neighbourhood                       48895 non-null object
```

```
latitude                          48895 non-null float64
longitude                         48895 non-null float64
room_type                         48895 non-null object
price                             48895 non-null int64
minimum_nights                    48895 non-null int64
number_of_reviews                 48895 non-null int64
last_review                       38843 non-null object
reviews_per_month                 38843 non-null float64
calculated_host_listings_count    48895 non-null int64
availability_365                  48895 non-null int64
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

[Response here]

- drop the following columns: name, host_id, host_name, and last_review
- display a summary of the statistics of the loaded data

[41]:
```
df_main.drop(['name', 'host_id', 'host_name', 'last_review'], axis=1,␣
↪inplace=True)
```

[42]:
```
df_main.describe()
```

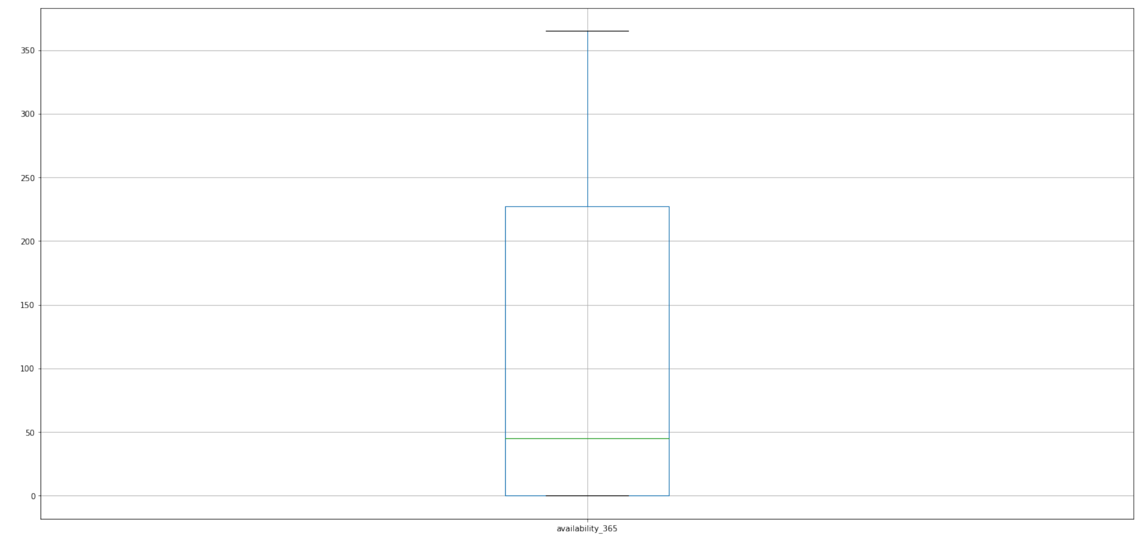
[42]:
```
                 id      latitude     longitude         price  minimum_nights  \
count  4.889500e+04  48895.000000  48895.000000  48895.000000    48895.000000
mean   1.901714e+07     40.728949    -73.952170    152.720687        7.029962
std    1.098311e+07      0.054530      0.046157    240.154170       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
```

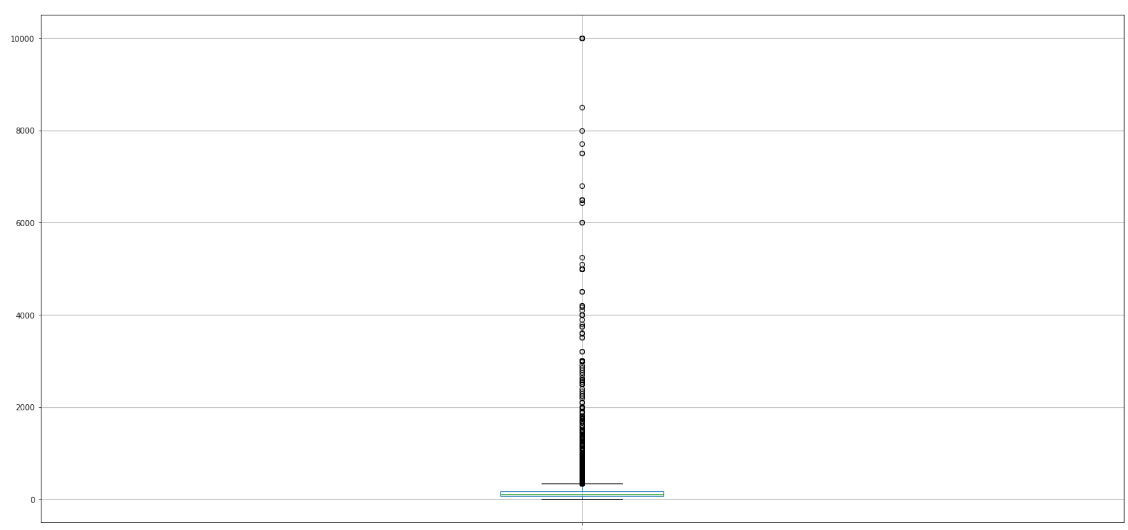
```
50%              45.000000
75%             227.000000
max             365.000000
```

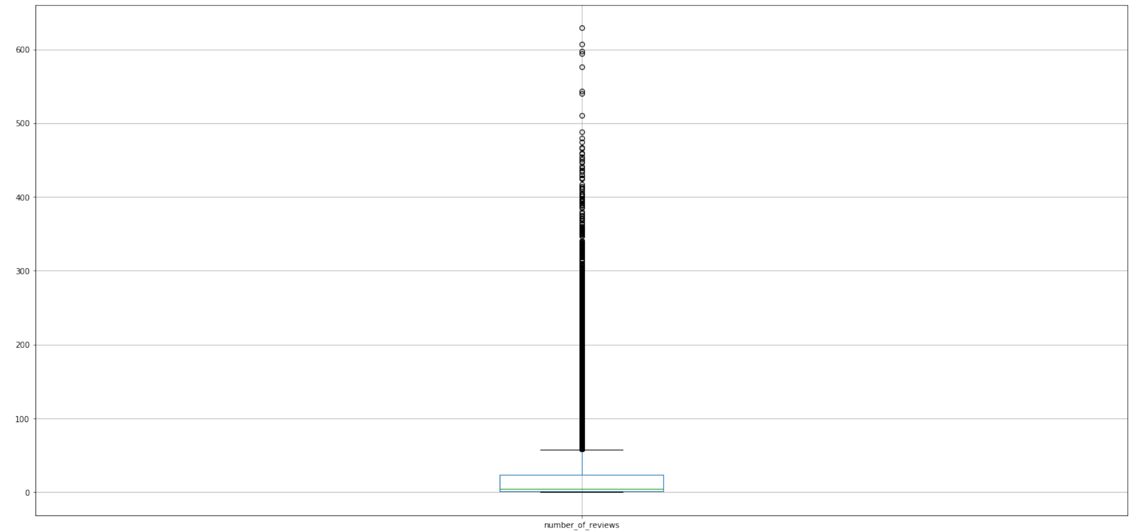
3.0.2 [5 pts] Boxplot 3 features of your choice

- plot boxplots for 3 features of your choice

[43]:
```python
columns_chosen = ['availability_365', 'price', 'number_of_reviews']
for column in columns_chosen:
    plt.figure(figsize=(25,12))
    df_main.boxplot([column])
```







- describe what you expected to see with these features and what you actually observed

[Response here]

My Response: Expected a normal distribution of the price and number of reviews but both have a high number of outliers, probably because the dataset is significantly skewed. Availability is more evenly distributed.

High variability in price with long tail values, review numbers much more compact, however availability has a wider variance.

3.0.3 [10 pts] Plot average price of a listing per neighbourhood_group

[44]:
```
df_main['neighbourhood_group'].value_counts()
```

[44]:
```
Manhattan        21661
Brooklyn         20104
Queens            5666
Bronx             1091
Staten Island      373
Name: neighbourhood_group, dtype: int64
```

[45]:
```
df_main.groupby('neighbourhood_group')['price'].mean().plot(kind='bar')
plt.title('Average Price per Neighbourhood Group')
plt.xlabel('Neighbourhood Group')
plt.ylabel('Price')
```

[45]:
```
Text(0, 0.5, 'Price')
```

- describe what you expected to see with these features and what you actually observed

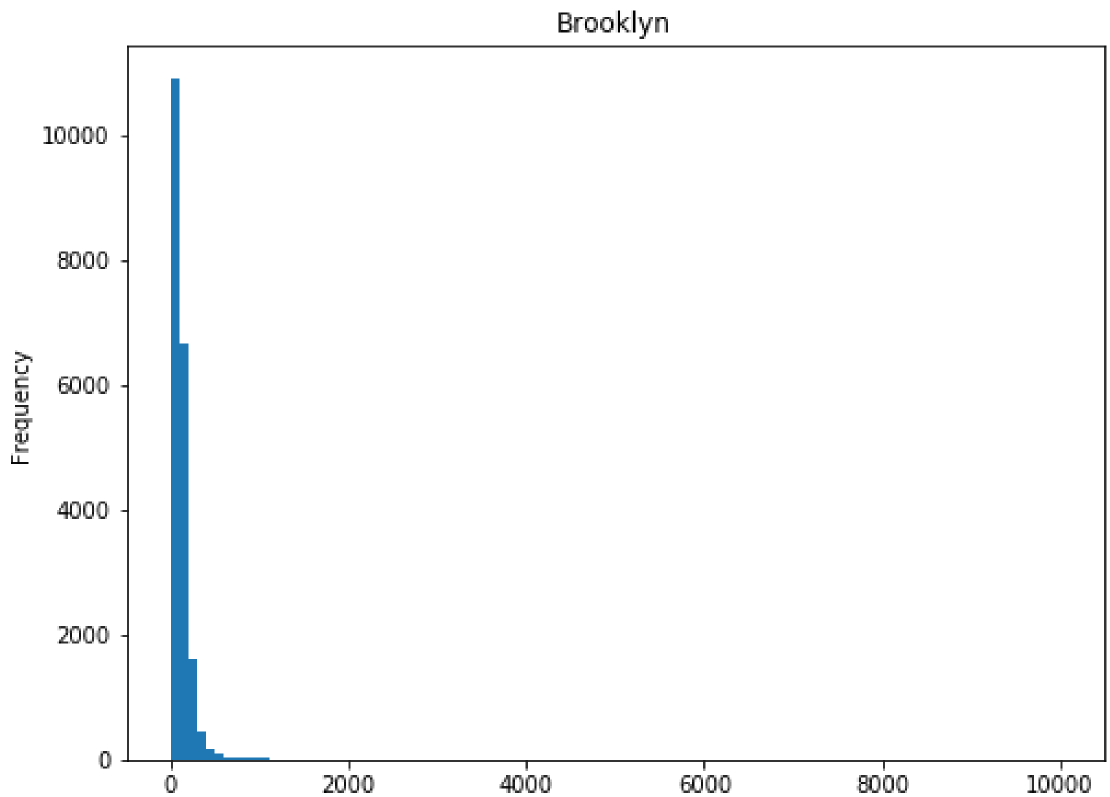
[Response here]
My Response: Manhattan has a higher average price than the other four, which are comparable to each other.

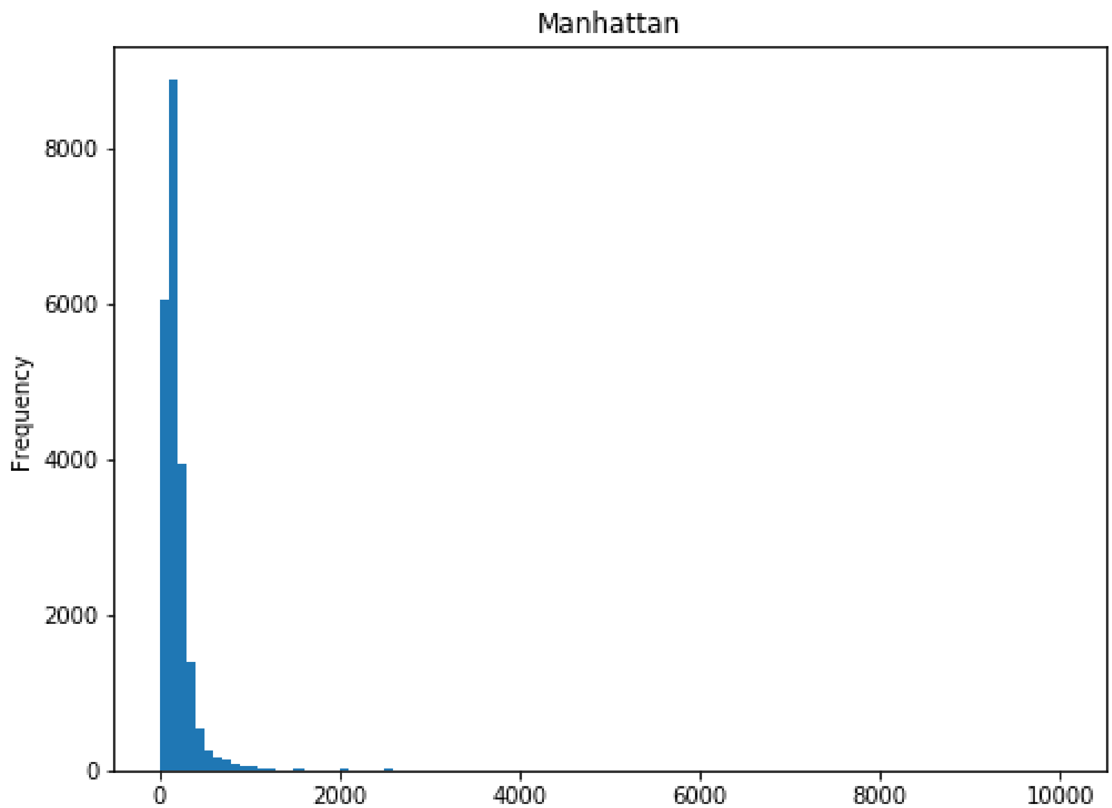
- So we can see different neighborhoods have dramatically different pricepoints, but how does the price breakdown by range. To see let's do a histogram of price by neighborhood to get a better sense of the distribution.

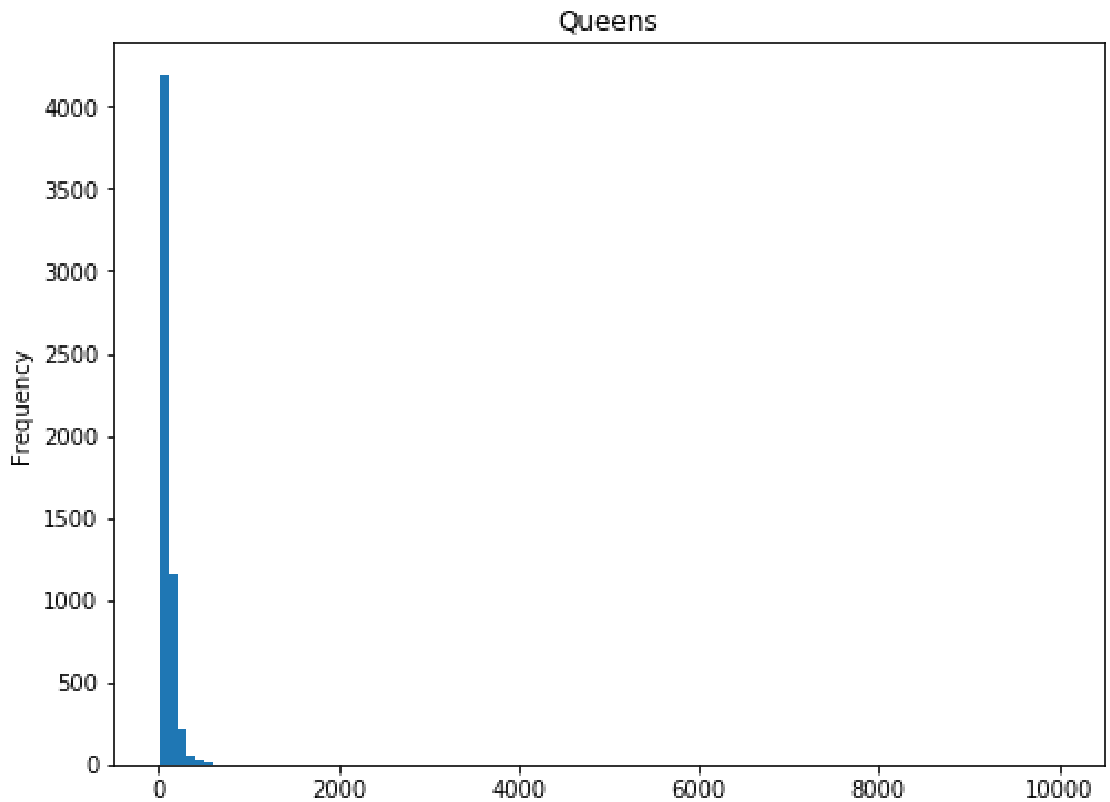
[46]:
```
unique_values = df_main['neighbourhood_group'].unique()
unique_values
```

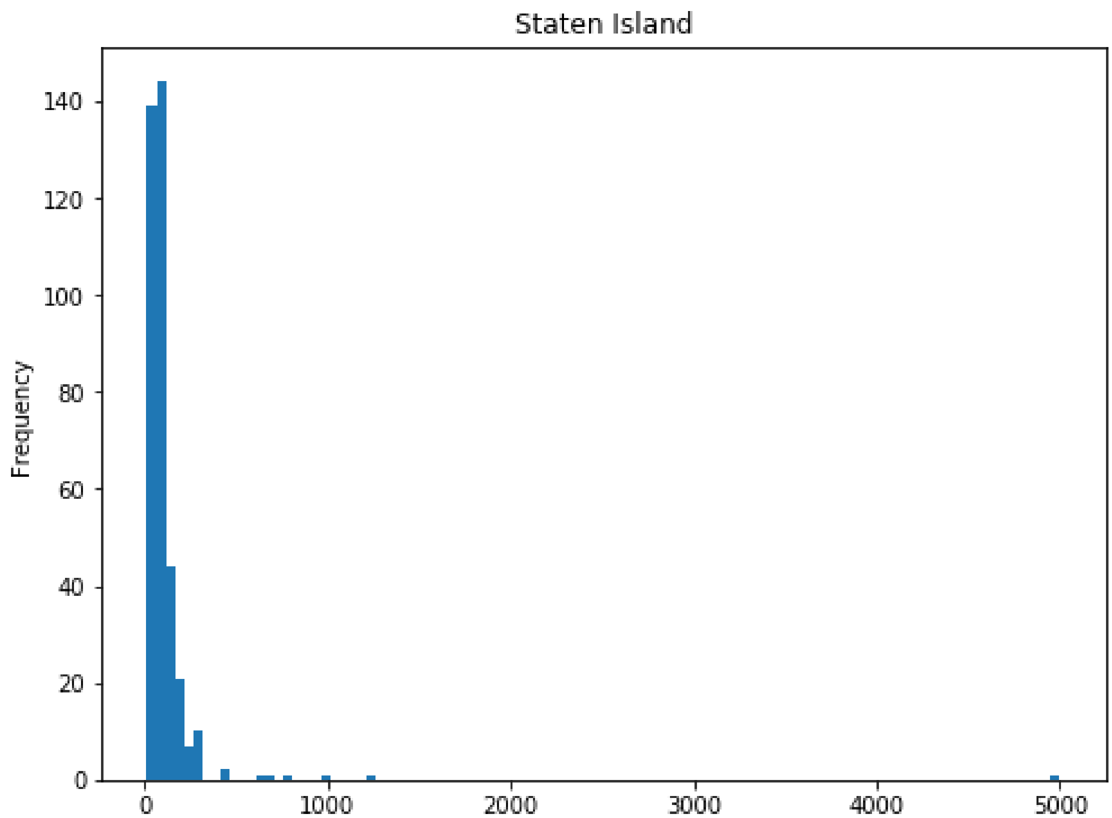
[46]:
```
array(['Brooklyn', 'Manhattan', 'Queens', 'Staten Island', 'Bronx'],
      dtype=object)
```

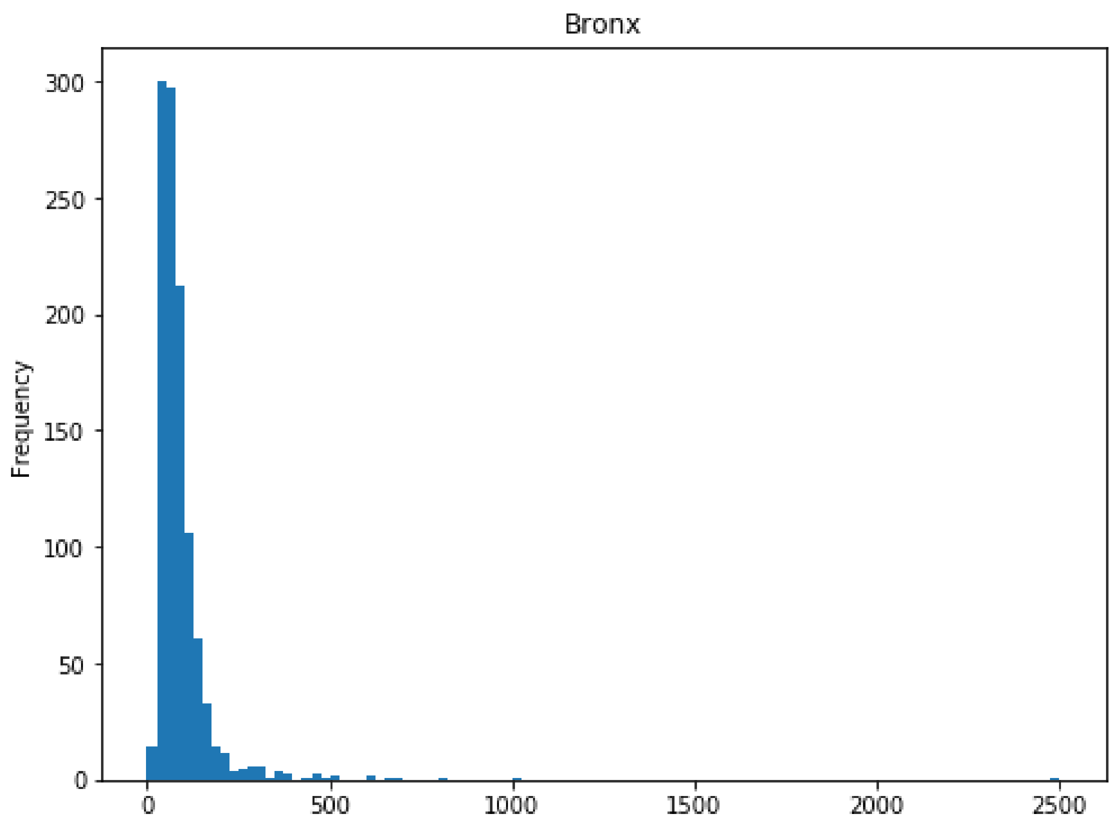
[47]:
```
for n in unique_values:
    temp_df = df_main[df_main.neighbourhood_group == n]
    plt.figure(figsize=(8,6))
    temp_df['price'].plot.hist(bins=100)
    plt.title(n)
```


Brooklyn
Frequency
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000


Manhattan
Frequency
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000


Queens
Frequency
4000
3500
3000
2500
2000
1500
1000
500
0
0
2000
4000
6000
8000
10000

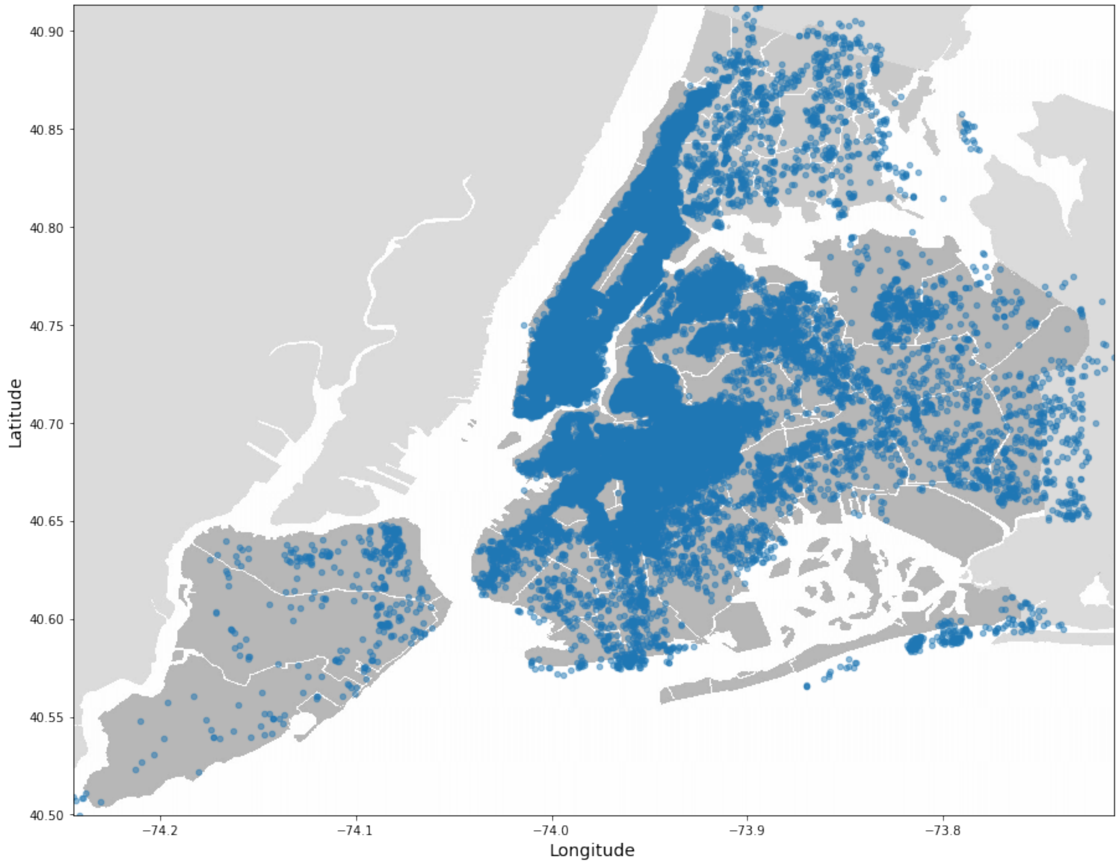

Staten Island
Frequency
0
20
40
60
80
100
120
140
0
1000
2000
3000
4000
5000



3.0.4 [5 pts] Plot 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 :)).

[48]:
```
images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename = "newyork.png"

import matplotlib.image as mpimg
newyork_img=mpimg.imread(os.path.join(images_path, filename))
ax = df_main.plot(kind="scatter", x="longitude", y="latitude", figsize=(25,12),
                       cmap=plt.get_cmap("jet"),
                       colorbar=False, alpha=0.5,
                      )
# overlay the new york map on the plotted scatter plot

plt.imshow(newyork_img, extent=[-74.244420, -73.712990, 40.499790, 40.913060],
 ↪alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)
```

```
plt.show()
```



3.0.5 [5 pts] Plot average price of room types who have availability greater than 180 days and neighbourhood_group is Manhattan

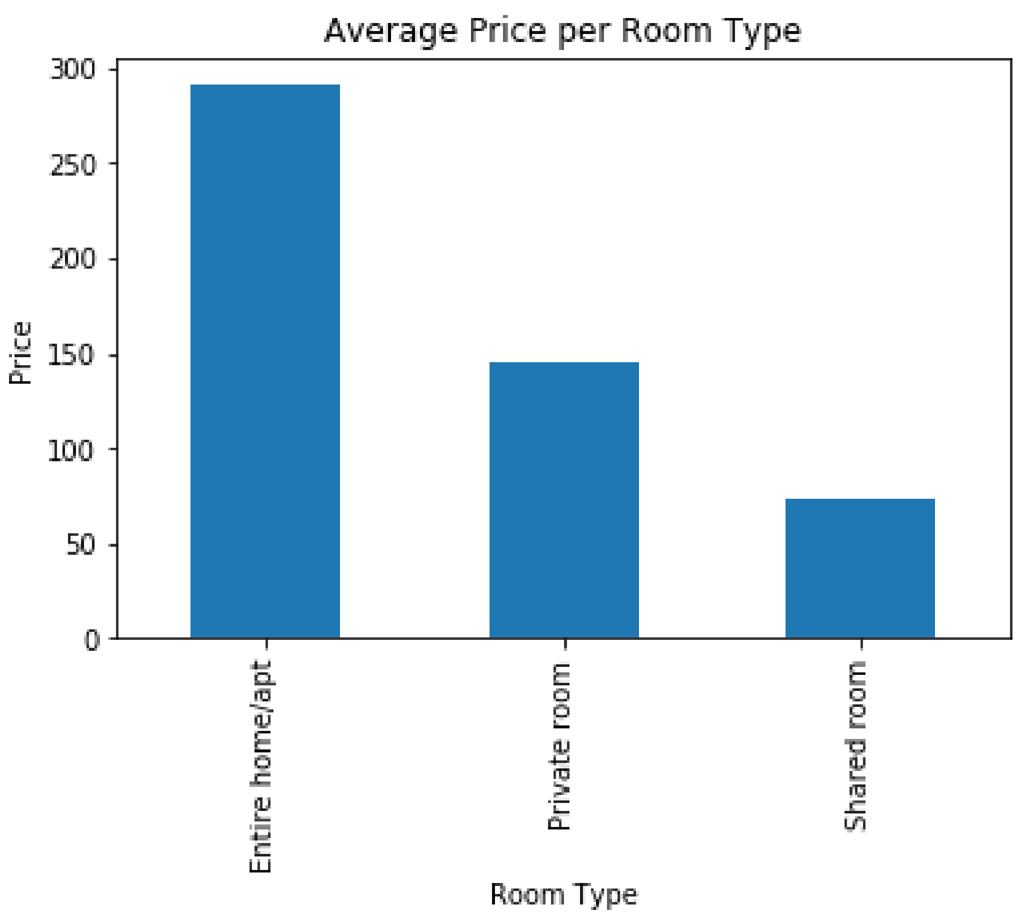
[49]:
```
temp_df = df_main[(df_main['availability_365'] > 180) &␣
 ↪(df_main['neighbourhood_group'] == 'Manhattan')]
temp_df.head()
```

[49]:
```
      id neighbourhood_group    neighbourhood  latitude  longitude  \
1   2595           Manhattan          Midtown  40.75362  -73.98377
2   3647           Manhattan           Harlem  40.80902  -73.94190
7   5178           Manhattan   Hell's Kitchen  40.76489  -73.98493
9   5238           Manhattan        Chinatown  40.71344  -73.99037
13  6021           Manhattan  Upper West Side  40.79826  -73.96113

          room_type  price  minimum_nights  number_of_reviews  \
1   Entire home/apt    225               1                 45
2      Private room    150               3                  0
```

```
7       Private room     79               2                430
9    Entire home/apt    150               1                160
13      Private room     85               2                113

    reviews_per_month  calculated_host_listings_count  availability_365
1                0.38                               2               355
2                 NaN                               1               365
7                3.47                               1               220
9                1.33                               4               188
13               0.91                               1               333
```

[50]:
```
temp_df.groupby('room_type')['price'].mean().plot(kind='bar')
plt.title('Average Price per Room Type')
plt.xlabel('Room Type')
plt.ylabel('Price')
```

[50]:
```
Text(0, 0.5, 'Price')
```

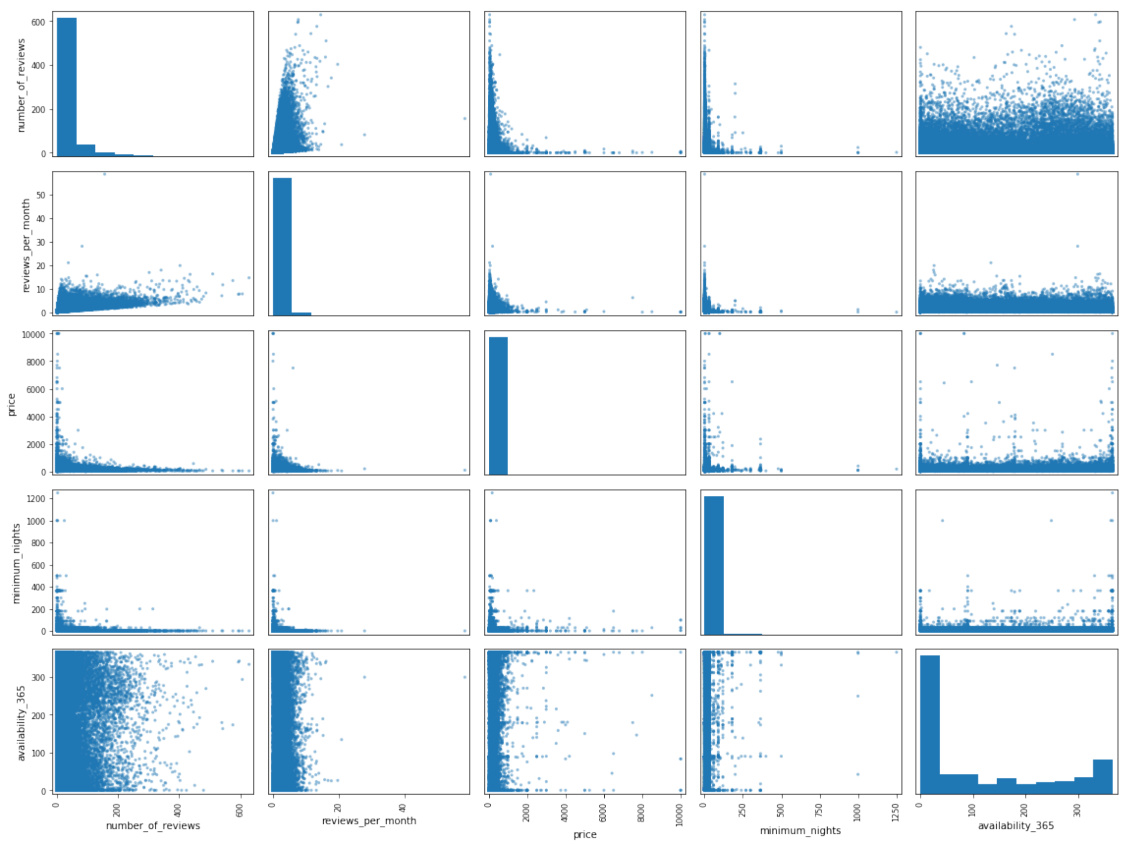


3.0.6 [5 pts] Plot correlation matrix

- which features have positive correlation?
- which features have negative correlation?

[51]:
```
corr_matrix = df_main.corr()
attributes = ["number_of_reviews", "reviews_per_month", "price",␣
 ↪"minimum_nights", "availability_365"]
scatter_matrix(df_main[attributes], figsize=(16, 12))
save_fig("scatter_matrix_plot2")
```

```
Saving figure scatter_matrix_plot2
```



[52]:
```
corr_matrix
```

[52]:

	id	latitude	longitude	price \
id	1.000000	-0.003125	0.090908	0.010619
latitude	-0.003125	1.000000	0.084788	0.033939
longitude	0.090908	0.084788	1.000000	-0.150019
price	0.010619	0.033939	-0.150019	1.000000
minimum_nights	-0.013224	0.024869	-0.062747	0.042799
number_of_reviews	-0.319760	-0.015389	0.059094	-0.047954

```
reviews_per_month                0.291828 -0.010142   0.145948 -0.030608
calculated_host_listings_count   0.133272  0.019517  -0.114713  0.057472
availability_365                 0.085468 -0.010983   0.082731  0.081829

                                minimum_nights  number_of_reviews  \
id                                   -0.013224          -0.319760
latitude                              0.024869          -0.015389
longitude                            -0.062747           0.059094
price                                 0.042799          -0.047954
minimum_nights                        1.000000          -0.080116
number_of_reviews                    -0.080116           1.000000
reviews_per_month                    -0.121702           0.549868
calculated_host_listings_count        0.127960          -0.072376
availability_365                      0.144303           0.172028

                                reviews_per_month  \
id                                       0.291828
latitude                                -0.010142
longitude                                0.145948
price                                   -0.030608
minimum_nights                          -0.121702
number_of_reviews                        0.549868
reviews_per_month                        1.000000
calculated_host_listings_count          -0.009421
availability_365                         0.185791

                                calculated_host_listings_count  \
id                                                    0.133272
latitude                                              0.019517
longitude                                            -0.114713
price                                                 0.057472
minimum_nights                                        0.127960
number_of_reviews                                    -0.072376
reviews_per_month                                    -0.009421
calculated_host_listings_count                        1.000000
availability_365                                      0.225701

                                availability_365
id                                      0.085468
latitude                               -0.010983
longitude                               0.082731
price                                   0.081829
minimum_nights                          0.144303
number_of_reviews                       0.172028
reviews_per_month                       0.185791
calculated_host_listings_count          0.225701
availability_365                        1.000000
```

[Response here]

In general, there does not seem to be a particularly strong correlation (positive or negative) between any pairing of these two features the exception of a (relatively) strong positive corrrelation reviews_per_month and number_of_reviews, which is expected. minimum_nights and price seem to have a slight positive correlation and reviews_per_month and minimum_nights seem to have a slight negative correlation but it doesn't seem apparent for any other pairings. A log transformation for some features (such as minimum_nights and price) may result a more clear correlation between pairings.

4 [30 pts] Prepare the Data

4.0.1 [5 pts] Augment the dataframe with two other features which you think would be useful

[53]:
```
df_main.columns
```

[53]:
```
Index(['id', 'neighbourhood_group', 'neighbourhood', 'latitude', 'longitude',
       'room_type', 'price', 'minimum_nights', 'number_of_reviews',
       'reviews_per_month', 'calculated_host_listings_count',
       'availability_365'],
      dtype='object')
```

[54]:
```
#I'm assuming that the price is given per day here, couldn't find further␣
 ↪information on it
df_main['min_price'] = df_main['price']*df_main['minimum_nights']
df_main['price_per_listings'] = df_main['price']/
 ↪df_main['calculated_host_listings_count']
```

[55]:
```
df_main.head(5)
```

[55]:
```
     id neighbourhood_group neighbourhood  latitude  longitude  \
0  2539            Brooklyn    Kensington  40.64749  -73.97237
1  2595           Manhattan       Midtown  40.75362  -73.98377
2  3647           Manhattan        Harlem  40.80902  -73.94190
3  3831            Brooklyn  Clinton Hill  40.68514  -73.95976
4  5022           Manhattan   East Harlem  40.79851  -73.94399

         room_type  price  minimum_nights  number_of_reviews  \
0     Private room    149               1                  9
1  Entire home/apt    225               1                 45
2     Private room    150               3                  0
3  Entire home/apt     89               1                270
4  Entire home/apt     80              10                  9

   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
```

```
   min_price  price_per_listings
0        149           24.833333
1        225          112.500000
2        450          150.000000
3         89           89.000000
4        800           80.000000
```

4.0.2 [5 pts] Impute any missing feature with a method of your choice, and briefly discuss why you chose this imputation method

[56]:
```
df_main.isnull().sum()
```

[56]:
```
id                                    0
neighbourhood_group                   0
neighbourhood                         0
latitude                              0
longitude                             0
room_type                             0
price                                 0
minimum_nights                        0
number_of_reviews                     0
reviews_per_month                 10052
calculated_host_listings_count        0
availability_365                      0
min_price                             0
price_per_listings                    0
dtype: int64
```

[57]:
```
df_main['reviews_per_month'].mean()
```

[57]:
```
1.3732214298586884
```

[58]:
```
df_main['reviews_per_month'].std()
```

[58]:
```
1.6804419952744627
```

[62]:
```
df_main_final = df_main.copy()
```

[63]:
```
df_main_final['reviews_per_month'].fillna(df_main['reviews_per_month'].
 ↪median(), inplace=True)
```

[64]:
```
df_main.head()
```

[64]:
```
     id neighbourhood_group neighbourhood  latitude  longitude  \
0  2539            Brooklyn    Kensington  40.64749  -73.97237
1  2595           Manhattan       Midtown  40.75362  -73.98377
2  3647           Manhattan        Harlem  40.80902  -73.94190
3  3831            Brooklyn  Clinton Hill  40.68514  -73.95976
4  5022           Manhattan   East Harlem  40.79851  -73.94399
```

```
         room_type  price  minimum_nights  number_of_reviews  \
0     Private room    149               1                  9
1  Entire home/apt    225               1                 45
2     Private room    150               3                  0
3  Entire home/apt     89               1                270
4  Entire home/apt     80              10                  9

   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

   min_price  price_per_listings
0        149           24.833333
1        225          112.500000
2        450          150.000000
3         89           89.000000
4        800           80.000000
```

[65]:
```
df_main_final.head()
```

[65]:
```
     id neighbourhood_group neighbourhood  latitude  longitude  \
0  2539            Brooklyn    Kensington  40.64749  -73.97237
1  2595           Manhattan       Midtown  40.75362  -73.98377
2  3647           Manhattan        Harlem  40.80902  -73.94190
3  3831            Brooklyn   Clinton Hill  40.68514  -73.95976
4  5022           Manhattan   East Harlem  40.79851  -73.94399

         room_type  price  minimum_nights  number_of_reviews  \
0     Private room    149               1                  9
1  Entire home/apt    225               1                 45
2     Private room    150               3                  0
3  Entire home/apt     89               1                270
4  Entire home/apt     80              10                  9

   reviews_per_month  calculated_host_listings_count  availability_365  \
0               0.21                               6               365
1               0.38                               2               355
2               0.72                               1               365
3               4.64                               1               194
4               0.10                               1                 0

   min_price  price_per_listings
0        149           24.833333
1        225          112.500000
2        450          150.000000
```

```
3          89             89.000000
4         800             80.000000
```

We can fill all null values of reviews per month with the median. This is because the standard deviation exceeds the mean which indicates the mean is not very representaive of the data. Thus, we replace it with the median; which should not affect the overall composition of the data.

4.0.3 [15 pts] Code complete data pipeline using sklearn mixins

[66]:
```
df_main_final.columns
```

[66]:
```
Index(['id', 'neighbourhood_group', 'neighbourhood', 'latitude', 'longitude',
       'room_type', 'price', 'minimum_nights', 'number_of_reviews',
       'reviews_per_month', 'calculated_host_listings_count',
       'availability_365', 'min_price', 'price_per_listings'],
      dtype='object')
```

[71]:
```
final_data = df_main_final.drop(["neighbourhood", "neighbourhood_group",␣
 ↪"room_type", "price"], axis=1)

numerical_features = list(final_data)
categorical_features = ["neighbourhood", "neighbourhood_group", "room_type"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

final_data_X = df_main_final.drop(columns=["price"])
final_data_Y = df_main_final["price"]
data_prepared = full_pipeline.fit_transform(final_data_X)
data_prepared
```

[71]:
```
<48895x242 sparse matrix of type '<class 'numpy.float64'>'
        with 782320 stored elements in Compressed Sparse Row format>
```

4.0.4 [5 pts] Set aside 20% of the data as test test (80% train, 20% test).

[72]:
```
X_train, X_test, Y_train, Y_test = train_test_split(final_data_X, final_data_Y,␣
 ↪test_size=0.2, random_state=42)
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)
```

```
X_train shape: (39116, 13)
Y_train shape: (39116,)
X_test shape: (9779, 13)
Y_test shape: (9779,)
```

5 [15 pts] Fit a model of your choice

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

[73]:
```
X_train_final = X_train.drop(["neighbourhood", "neighbourhood_group",␣
 ↪"room_type"], axis=1)
X_test_final = X_test.drop(["neighbourhood", "neighbourhood_group",␣
 ↪"room_type"], axis=1)

linreg = LinearRegression()
linreg.fit(X_train_final, Y_train)

preds = linreg.predict(X_train_final)
mse1 = mean_squared_error(Y_train, preds)
print ("Train MSE: ", mse1)
preds = linreg.predict(X_test_final)
mse2 = mean_squared_error(Y_test, preds)
print ("Test MSE: ", mse2)
```

```
Train MSE:  11201.842819463272
Test MSE:  11096.177490017448
```