Intro-End2End

January 22, 2020

0.1 Introduction

Welcome to **CS188 - Data Science Fundamentals!** We plan on having you go through some grueling training so you can start crunching data out there… in today's day and age "data is the new oil" or perhaps "snake oil" nonetheless, there's a lot of it, each with different purity (so pure that perhaps you could feed off it for a life time) or dirty which then at that point you can either decide to dump it or try to weed out something useful (that's where they need you…)

In this project you will work through an example project end to end.

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

`MLprocess.jpg`

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

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

0.3 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

# 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
```

[2]:
```
# Where to save the figures
ROOT_DIR = "."
IMAGES_PATH = os.path.join(ROOT_DIR, "images")
os.makedirs(IMAGES_PATH, exist_ok=True)

# TODO: how does it know what figure to save? Most recent figure?
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)
```

0.4 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):
```

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

[4]:
```
DATASET_PATH = os.path.join("datasets", "housing")
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
```

[4]:
```
   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.

[5]:
```
# 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
```

[6]:
```python
# you can access individual columns similarly
# to accessing elements in a python dict
housing["ocean_proximity"].head() # added head() to avoid printing many columns..
```

[6]:
```
0    NEAR BAY
1    NEAR BAY
2    NEAR BAY
3    NEAR BAY
4    NEAR BAY
Name: ocean_proximity, dtype: object
```

[7]:
```python
# to access a particular row we can use iloc
housing.iloc[1]
```

[7]:
```
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
```

[8]:
```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()
```

[8]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

[9]:
```python
# The describe function compiles your typical statistics for each
# column
housing.describe()
```

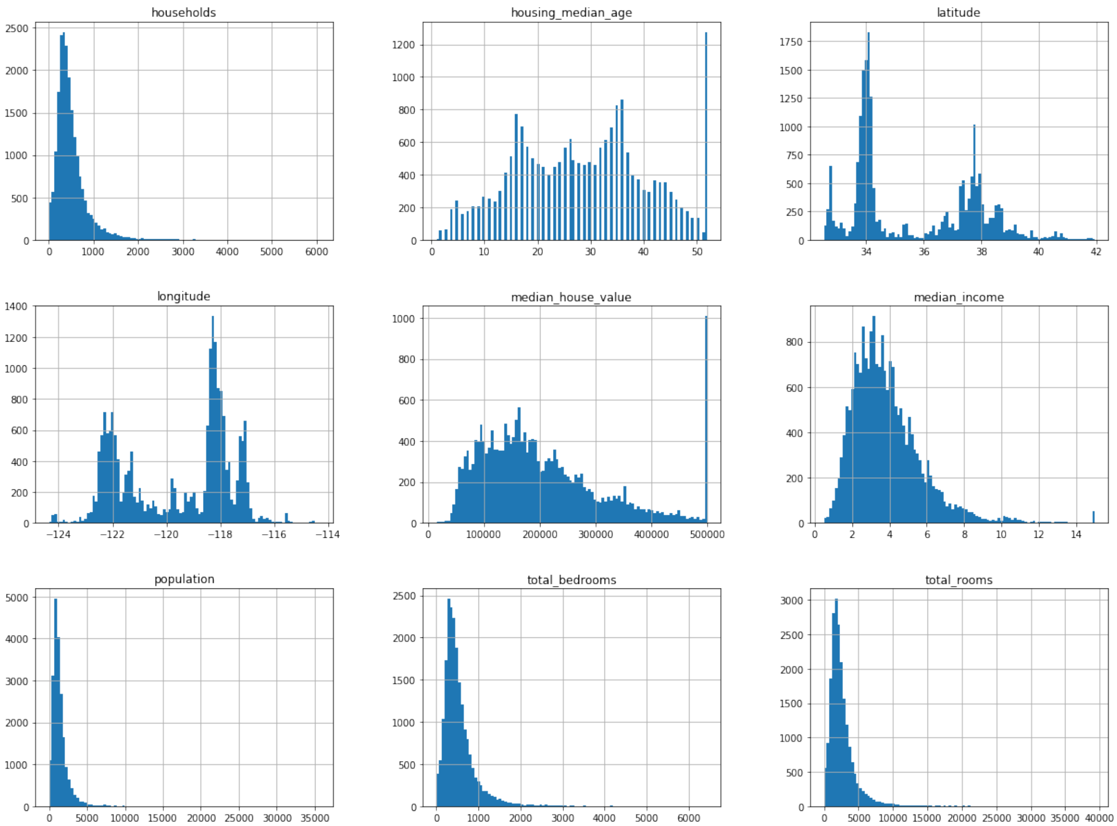
[9]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
```

```
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
```

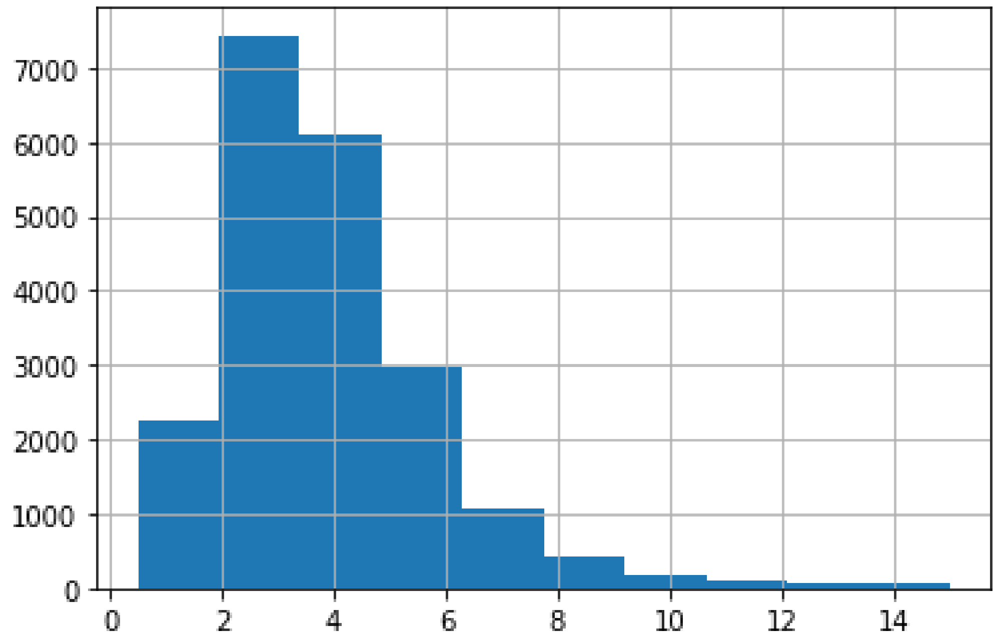
If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section here

0.5 Let's start visualizing the dataset

[10]:
```
# We can draw a histogram for each of the dataframes features
# using the hist function
# bins refers to number of intervals you split it into
housing.hist(bins=100, 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
```



[11]:
```
# 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

[12]:
```
# 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()
```

[12]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

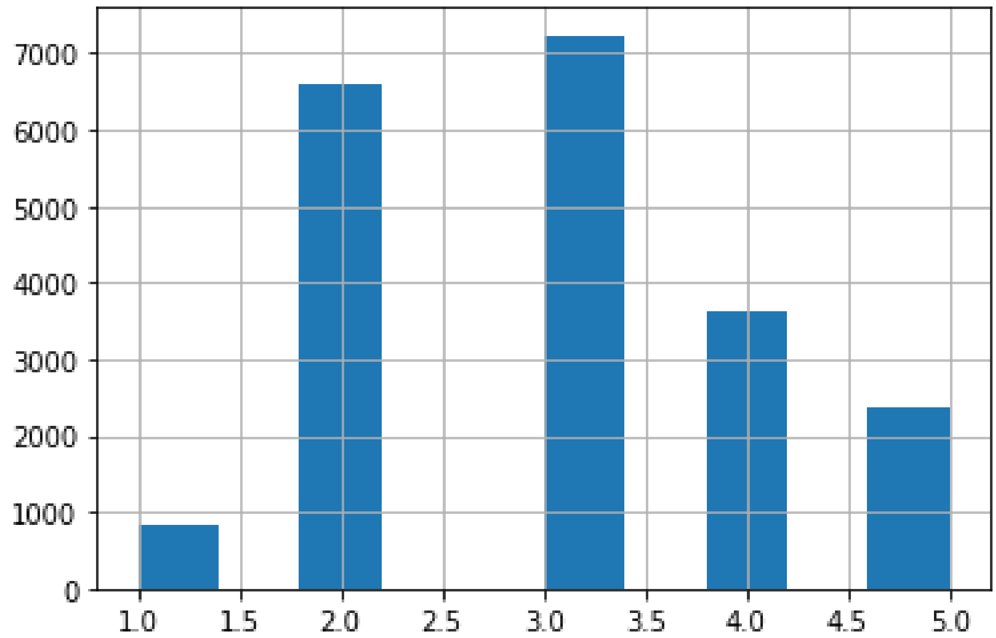
[13]:
```
housing["income_cat"]
```

[13]:
```
0         5
1         5
2         5
3         4
4         3
```

```
          ..
20635    2
20636    2
20637    2
20638    2
20639    2
Name: income_cat, Length: 20640, dtype: category
Categories (5, int64): [1 < 2 < 3 < 4 < 5]
```

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

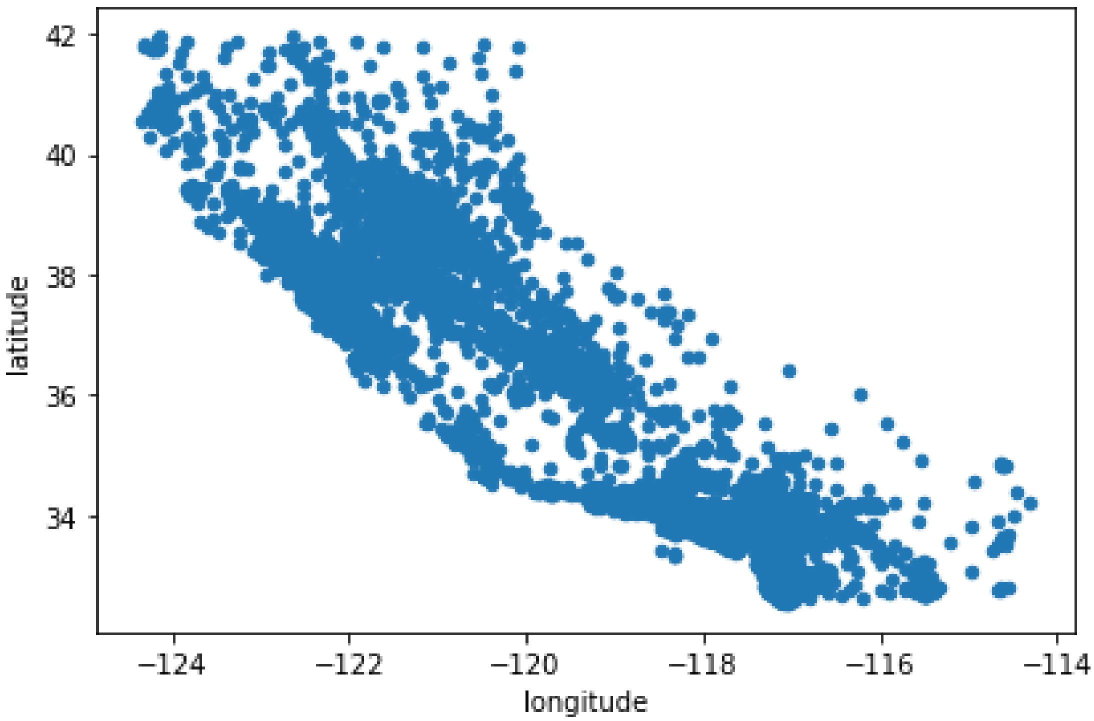
[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xa1ca50710>
```



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

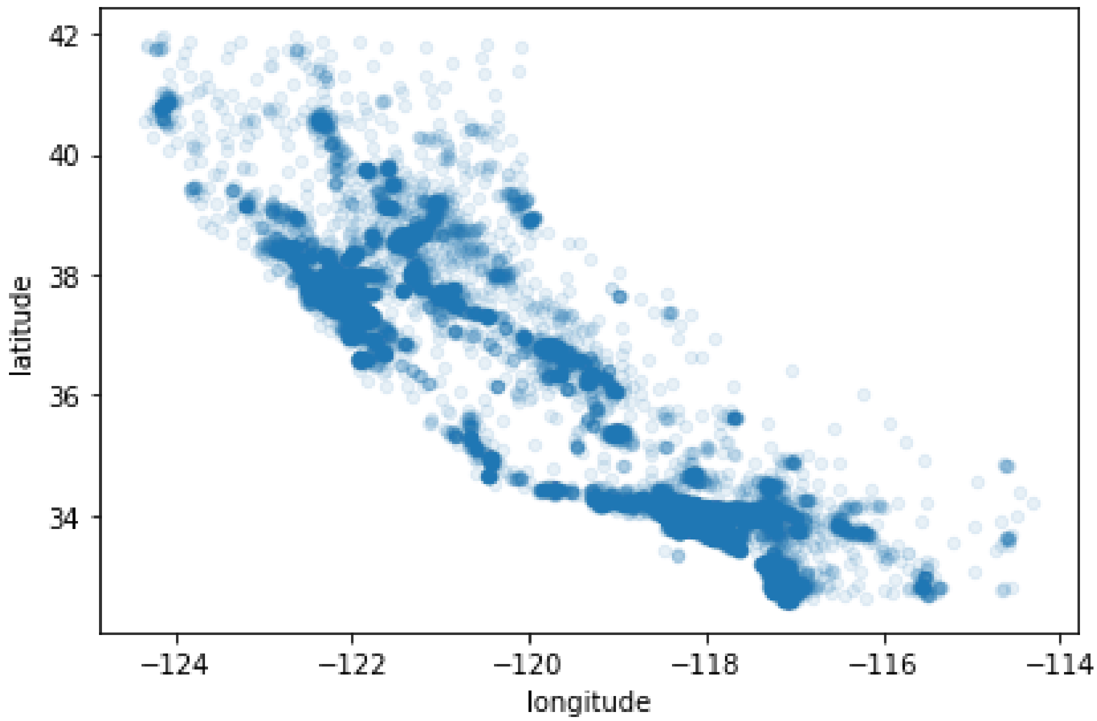
[15]:
```
## here's a not so interestting way plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

```
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")
```

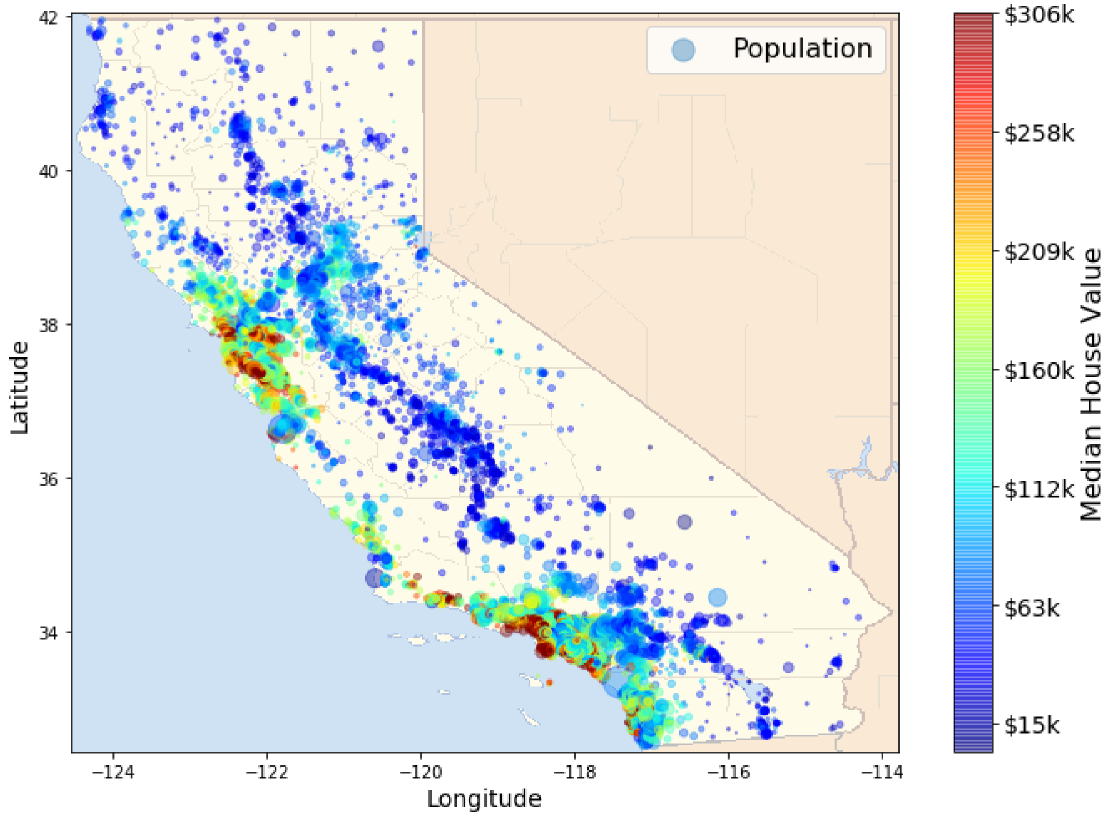
```
Saving figure better_visualization_plot
```



[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, the most expensive houses are concentrated around the San Francisco/Los Angeles areas.

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

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

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

None the less we can explore this using correlation matrices.

[18]:
```
corr_matrix = housing.corr()
```

[19]:
```
corr_matrix
```

[19]:
```
                    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
```

[20]:
```
# 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)
```

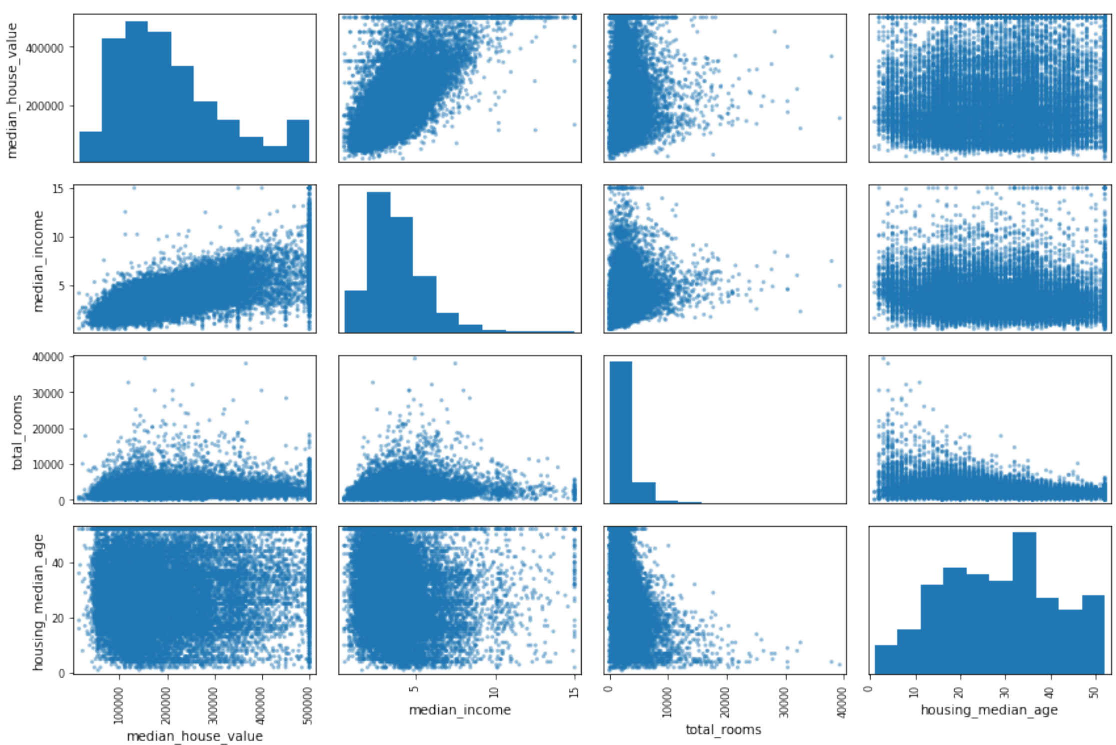
[20]:
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
```

```
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
```

[21]:

```
# 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
```

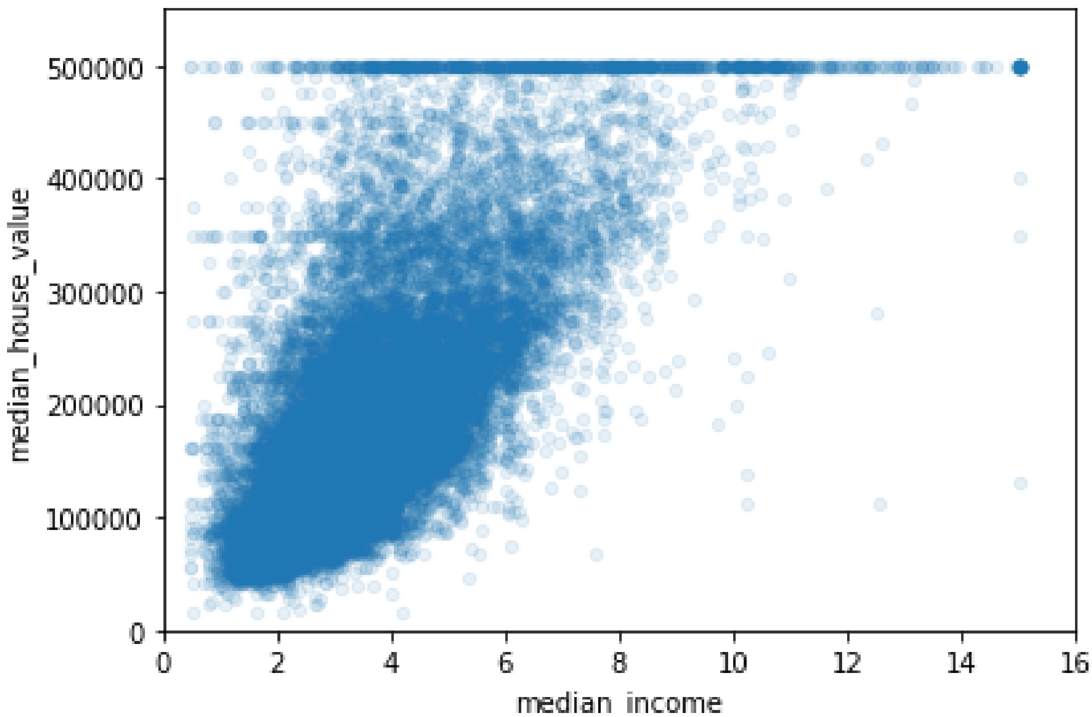


[22]:

```
# 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
```



0.5.1 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.

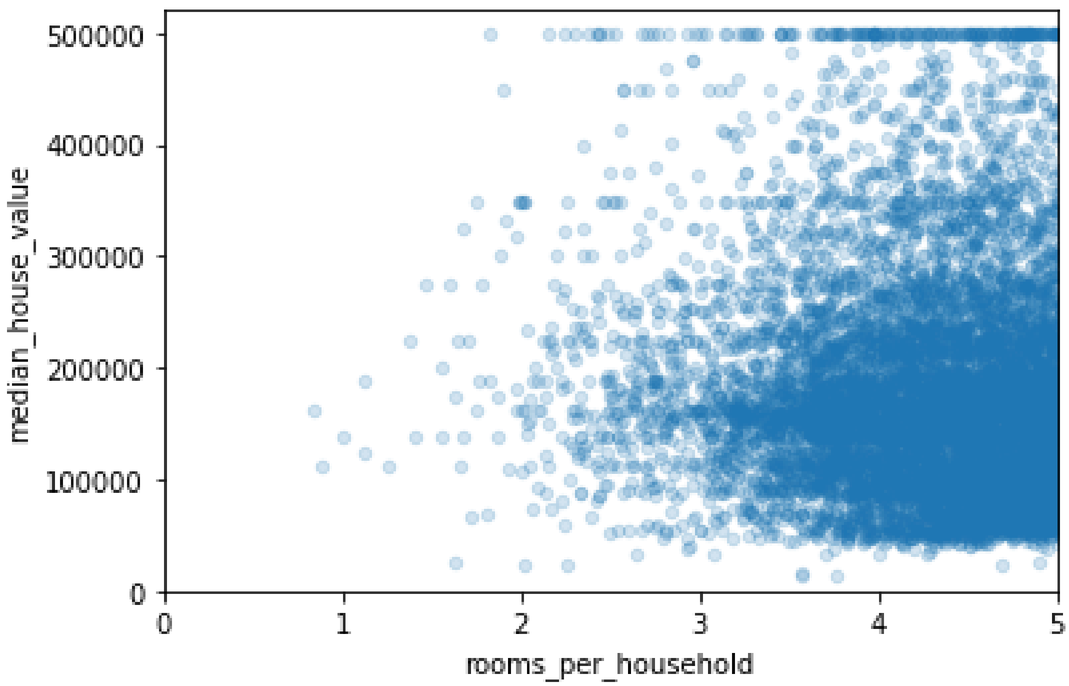
[23]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

[24]:
```
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[24]:
```
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
```

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



0.6 Preparing Dastaset for ML

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… 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!

[26]:
```
from sklearn.model_selection import StratifiedShuffleSplit
# let's first start by creating our train and test sets
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    train_set = housing.loc[train_index]
    test_set = housing.loc[test_index]
```

[27]:
```
housing = train_set.drop("median_house_value", axis=1) # drop labels for␣
 ↪training set features
                                                       # the input to the model␣
 ↪should not contain the true label
housing_labels = train_set["median_house_value"].copy()
```

[28]:
```
housing
```

[28]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
17606    -121.89     37.29                38.0       1568.0           351.0
18632    -121.93     37.05                14.0        679.0           108.0
14650    -117.20     32.77                31.0       1952.0           471.0
3230     -119.61     36.31                25.0       1847.0           371.0
3555     -118.59     34.23                17.0       6592.0          1525.0
...          ...       ...                 ...          ...             ...
6563     -118.13     34.20                46.0       1271.0           236.0
12053    -117.56     33.88                40.0       1196.0           294.0
13908    -116.40     34.09                 9.0       4855.0           872.0
11159    -118.01     33.82                31.0       1960.0           380.0
15775    -122.45     37.77                52.0       3095.0           682.0

       population  households  median_income ocean_proximity income_cat  \
17606       710.0       339.0         2.7042       <1H OCEAN          2
18632       306.0       113.0         6.4214       <1H OCEAN          5
14650       936.0       462.0         2.8621      NEAR OCEAN          2
3230       1460.0       353.0         1.8839          INLAND          2
3555       4459.0      1463.0         3.0347       <1H OCEAN          3
...           ...         ...            ...             ...        ...
6563        573.0       210.0         4.9312          INLAND          4
12053      1052.0       258.0         2.0682          INLAND          2
13908      2098.0       765.0         3.2723          INLAND          3
```

```
11159      1356.0      356.0         4.0625       <1H OCEAN          3
15775      1269.0      639.0         3.5750        NEAR BAY          3

       rooms_per_household  bedrooms_per_room  population_per_household
17606             4.625369           0.223852                  2.094395
18632             6.008850           0.159057                  2.707965
14650             4.225108           0.241291                  2.025974
3230              5.232295           0.200866                  4.135977
3555              4.505810           0.231341                  3.047847
...                    ...                ...                       ...
6563              6.052381           0.185681                  2.728571
12053             4.635659           0.245819                  4.077519
13908             6.346405           0.179609                  2.742484
11159             5.505618           0.193878                  3.808989
15775             4.843505           0.220355                  1.985915

[16512 rows x 13 columns]
```

0.6.1 Dealing With Incomplete Data

[29]:
```
# have you noticed when looking at the dataframe summary certain rows
# contained null values? we can't just leave them as nulls and expect our
# model to handle them for us...
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

[29]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
4629     -118.30     34.07                18.0       3759.0             NaN
6068     -117.86     34.01                16.0       4632.0             NaN
17923    -121.97     37.35                30.0       1955.0             NaN
13656    -117.30     34.05                 6.0       2155.0             NaN
19252    -122.79     38.48                 7.0       6837.0             NaN

       population  households  median_income ocean_proximity income_cat  \
4629       3296.0      1462.0         2.2708       <1H OCEAN          2
6068       3038.0       727.0         5.1762       <1H OCEAN          4
17923       999.0       386.0         4.6328       <1H OCEAN          4
13656      1039.0       391.0         1.6675          INLAND          2
19252      3468.0      1405.0         3.1662       <1H OCEAN          3

       rooms_per_household  bedrooms_per_room  population_per_household
4629              2.571135                NaN                  2.254446
6068              6.371389                NaN                  4.178817
17923             5.064767                NaN                  2.588083
13656             5.511509                NaN                  2.657289
19252             4.866192                NaN                  2.468327
```

[30]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1: simply
↪drop rows that have null values
```

[30]:
```
Empty DataFrame
Columns: [longitude, latitude, housing_median_age, total_rooms, total_bedrooms,
population, households, median_income, ocean_proximity, income_cat,
rooms_per_household, bedrooms_per_room, population_per_household]
Index: []
```

[31]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop the
↪complete feature
```

[31]:
```
       longitude  latitude  housing_median_age  total_rooms  population  \
4629     -118.30     34.07                18.0       3759.0      3296.0
6068     -117.86     34.01                16.0       4632.0      3038.0
17923    -121.97     37.35                30.0       1955.0       999.0
13656    -117.30     34.05                 6.0       2155.0      1039.0
19252    -122.79     38.48                 7.0       6837.0      3468.0

       households  median_income ocean_proximity income_cat  \
4629       1462.0         2.2708       <1H OCEAN          2
6068        727.0         5.1762       <1H OCEAN          4
17923       386.0         4.6328       <1H OCEAN          4
13656       391.0         1.6675          INLAND          2
19252      1405.0         3.1662       <1H OCEAN          3

       rooms_per_household  bedrooms_per_room  population_per_household
4629              2.571135                NaN                  2.254446
6068              6.371389                NaN                  4.178817
17923             5.064767                NaN                  2.588083
13656             5.511509                NaN                  2.657289
19252             4.866192                NaN                  2.468327
```

[32]:
```
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
```

[32]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
4629     -118.30     34.07                18.0       3759.0           433.0
6068     -117.86     34.01                16.0       4632.0           433.0
17923    -121.97     37.35                30.0       1955.0           433.0
13656    -117.30     34.05                 6.0       2155.0           433.0
19252    -122.79     38.48                 7.0       6837.0           433.0

       population  households  median_income ocean_proximity income_cat  \
4629       3296.0      1462.0         2.2708       <1H OCEAN          2
```

```
6068        3038.0       727.0          5.1762       <1H OCEAN           4
17923        999.0       386.0          4.6328       <1H OCEAN           4
13656       1039.0       391.0          1.6675          INLAND           2
19252       3468.0      1405.0          3.1662       <1H OCEAN           3

       rooms_per_household  bedrooms_per_room  population_per_household
4629              2.571135                NaN                  2.254446
6068              6.371389                NaN                  4.178817
17923             5.064767                NaN                  2.588083
13656             5.511509                NaN                  2.657289
19252             4.866192                NaN                  2.468327
```

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

0.6.2 Prepare Data

[33]:
```python
# 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.

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.drop("ocean_proximity", axis=1) # remove the categorical␣
↪feature
# column index
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):
        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]

attr_adder = AugmentFeatures(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)

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)
```

0.6.3 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.

[34]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

# let's try the full preprocessing pipeline on a few training instances
data = test_set.iloc[:5]
labels = housing_labels.iloc[:5]
data_prepared = full_pipeline.transform(data)

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

```
Predictions: [425717.48517515 267643.98033218 227366.19892733 199614.48287493
 161425.25185885]
Actual labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

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

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

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

[35]:
```
from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(housing_prepared)
mse = mean_squared_error(housing_labels, preds)
rmse = np.sqrt(mse)
rmse
```

[35]: 67784.32202861732

1 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.

2 [25 pts] Visualizing Data

2.0.1 [5 pts] Load the data + statistics

- load the dataset

- display the first few rows of the data
- drop the following columns: name, host_id, host_name, last_review
- display a summary of the statistics of the loaded data
- plot histograms for 3 features of your choice

[36]:
```
def load_airbnb_data():
    csv_path = os.path.join("datasets", "airbnb", "AB_NYC_2019.csv")
    return pd.read_csv(csv_path)
```

[37]:
```
# load the dataset
airbnb = load_airbnb_data()
# display the first few rows of the data
airbnb.head()
```

[37]:
```
     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
```

[38]:
```
# drop the following columns: name, host_id, host_name, last_review
# also drop id because we don't want this for our model
airbnb = airbnb.drop(columns=["id", "name", "host_id", "host_name",␣
 ↪"last_review"])
# display a summary of the statistics of the loaded data
```

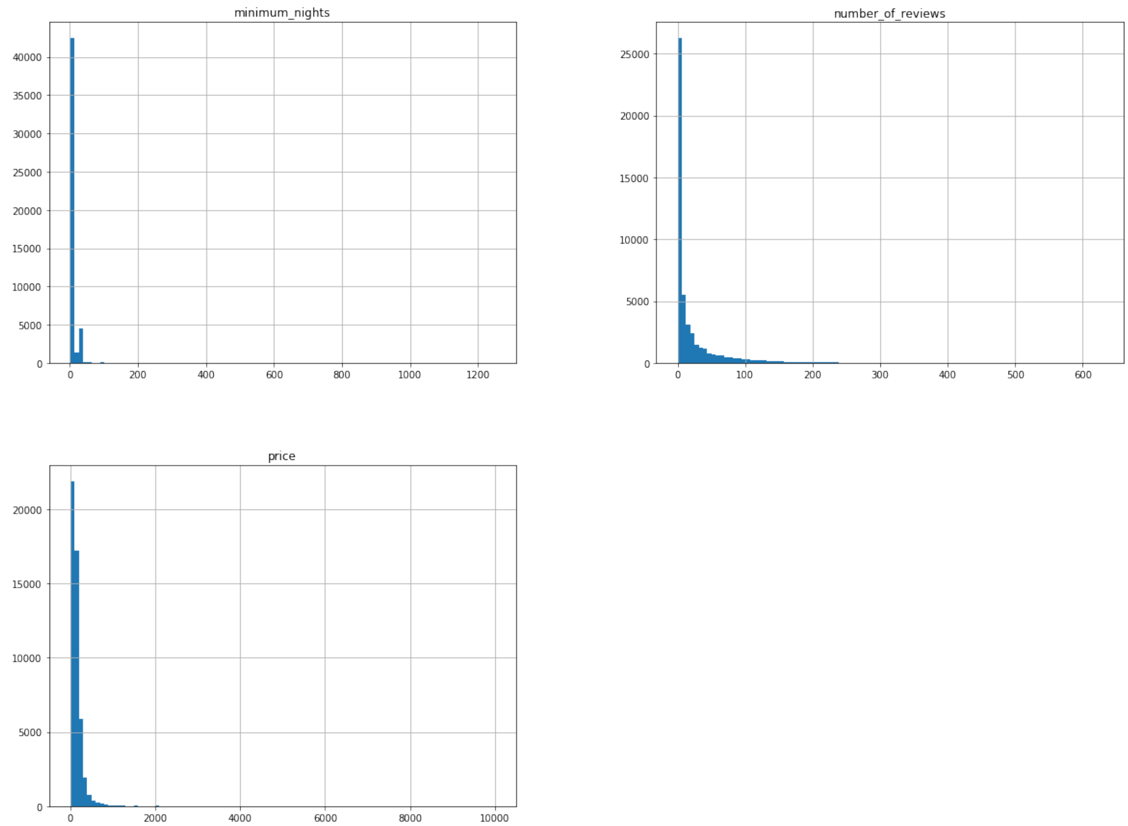
```
airbnb.describe()
```

[38]:

```
           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  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
```

[39]:

```
# plot histograms for 3 features of your choice
airbnb.hist(bins=100, column=["price", "number_of_reviews", "minimum_nights"],␣
 ↪figsize=(20, 15))
plt.show()
```

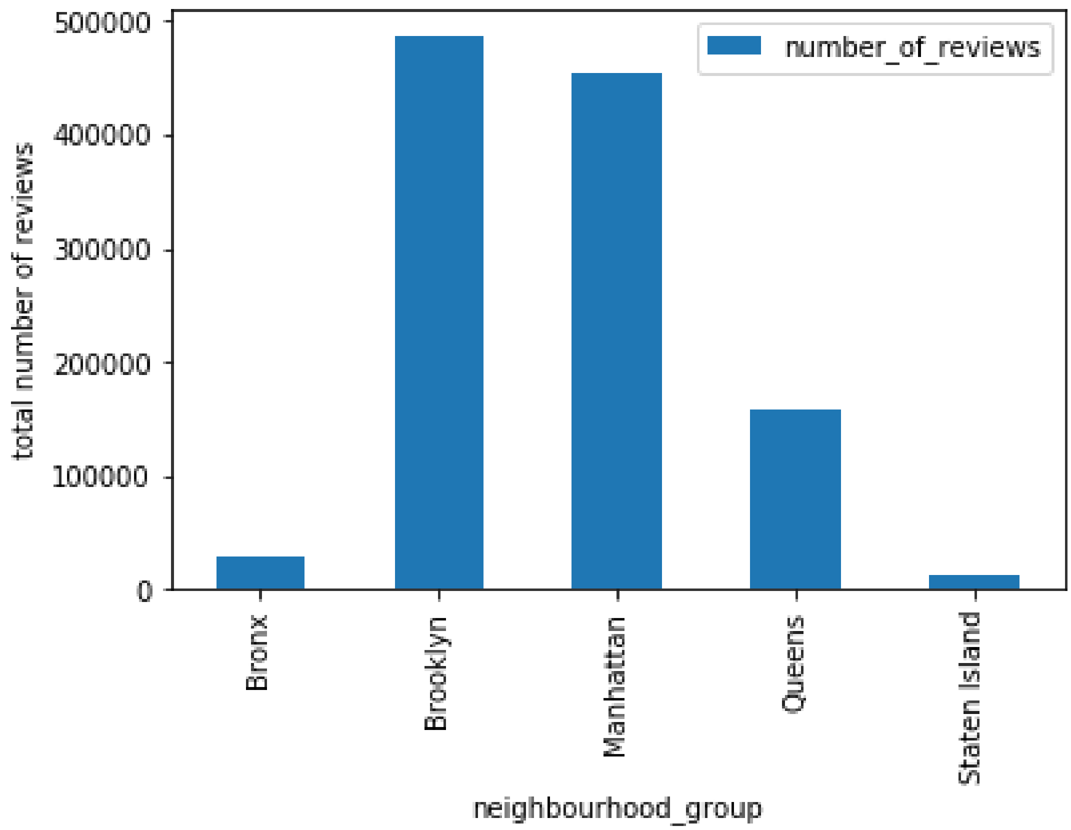


2.0.2 [5 pts] Plot total number_of_reviews per neighbourhood_group

[40]:
```
# use group by to get sum per neighbourhood_group
data = airbnb[["number_of_reviews", "neighbourhood_group"]].
 ↪groupby(['neighbourhood_group']).sum()
data
```

[40]:
```
                     number_of_reviews
neighbourhood_group
Bronx                            28371
Brooklyn                        486574
Manhattan                       454569
Queens                          156950
Staten Island                    11541
```

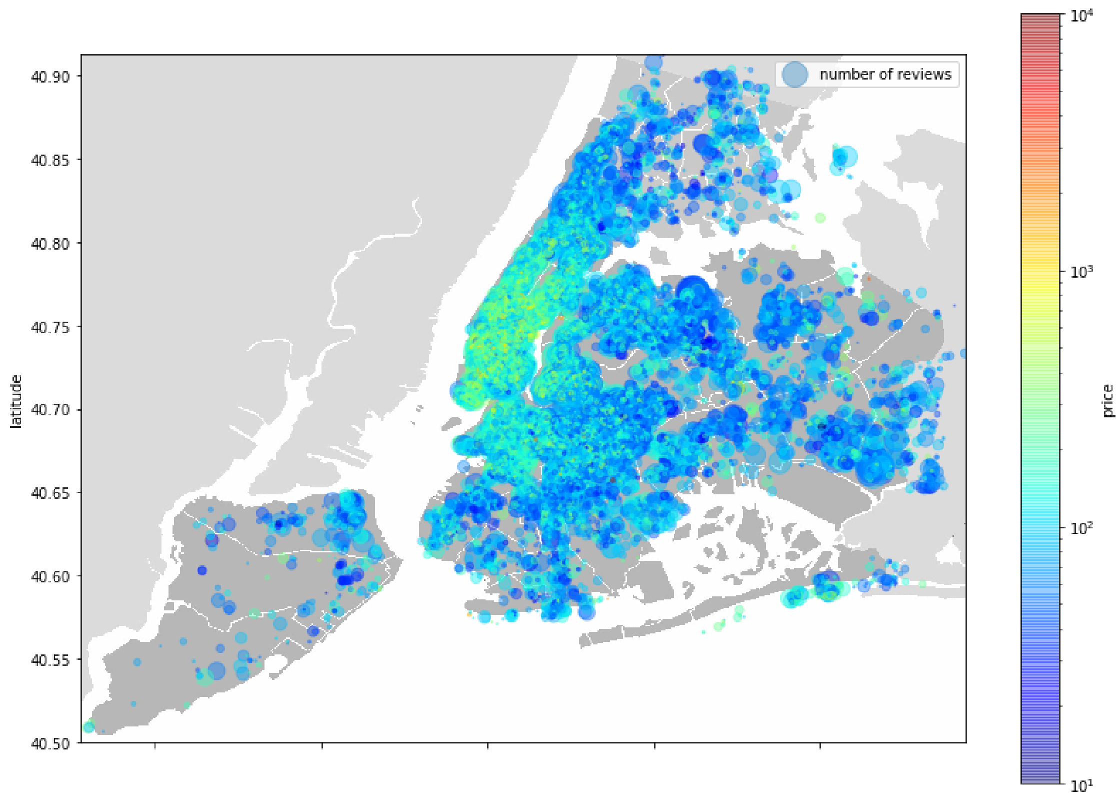
[41]:
```
ax = data.plot(kind='bar')
ax.set_ylabel("total number of reviews")
plt.show()
```



2.0.3 [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 :)).

[42]:
```
# compute where to place the axises
latitude_max, longitude_max = airbnb[["latitude", "longitude"]].max()
latitude_min, longitude_min = airbnb[["latitude", "longitude"]].min()

filename = "new_york_city.png"
new_york_city_img=mpimg.imread(os.path.join(images_path, filename))

# plot the airbnb data
airbnb.plot(
    kind='scatter',
    x="longitude",
    y="latitude",
    figsize = (14,10),
    label="number of reviews",
    s=airbnb["number_of_reviews"], # use number of reviews for size
    c="price", # use price for color
    norm=matplotlib.colors.LogNorm(), # use lognormal for coloring
```

```
    cmap=plt.get_cmap('jet'),
    alpha=0.4
)

# plot the image
plt.imshow(
    new_york_city_img,
    extent=[longitude_min, longitude_max, latitude_min, latitude_max],
    alpha=0.5,
    cmap=plt.get_cmap("jet")
)

plt.show()
```

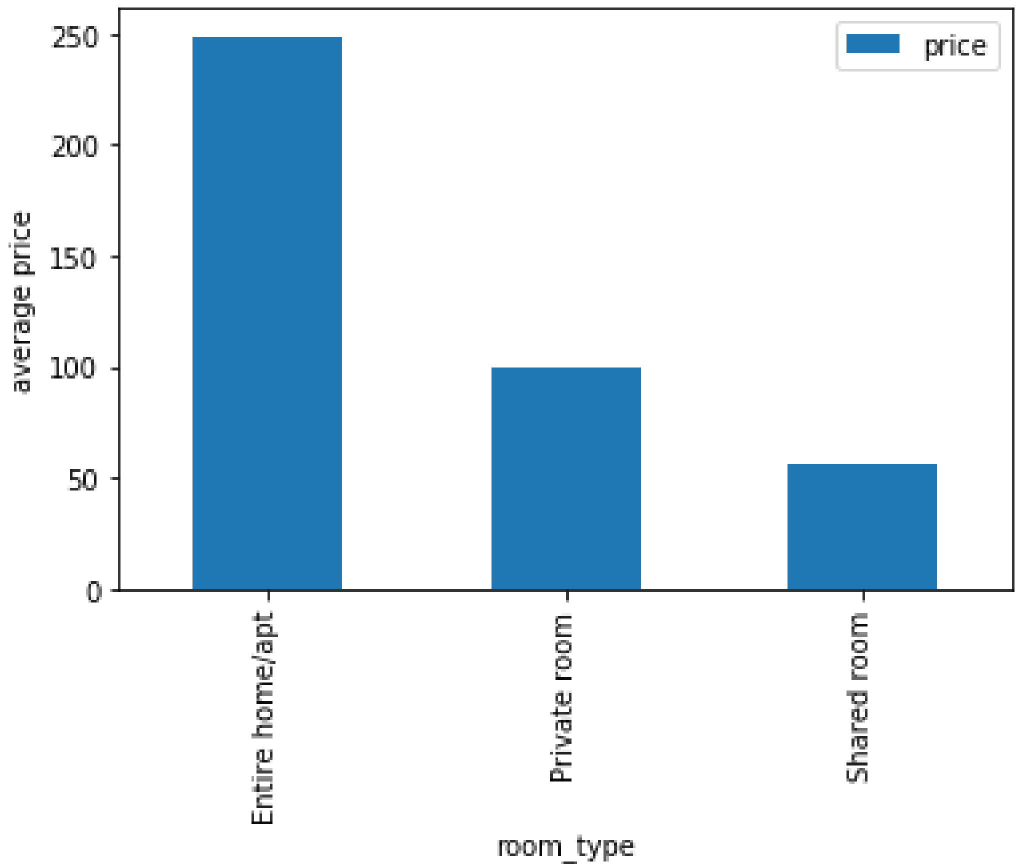


2.0.4 [5 pts] Plot average price of room types who have availability greater than 180 days.

[43]:
```
data = airbnb[["price", "room_type"]].loc[airbnb["availability_365"] > 180].
 ↪groupby(["room_type"]).mean()
data
```

[43]:
```
                             price
room_type
Entire home/apt  248.870817
Private room     100.028192
Shared room       56.941909
```

[44]:
```
ax = data.plot(kind='bar')
ax.set_ylabel("average price")
plt.show()
```



2.0.5 [5 pts] Plot correlation matrix

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

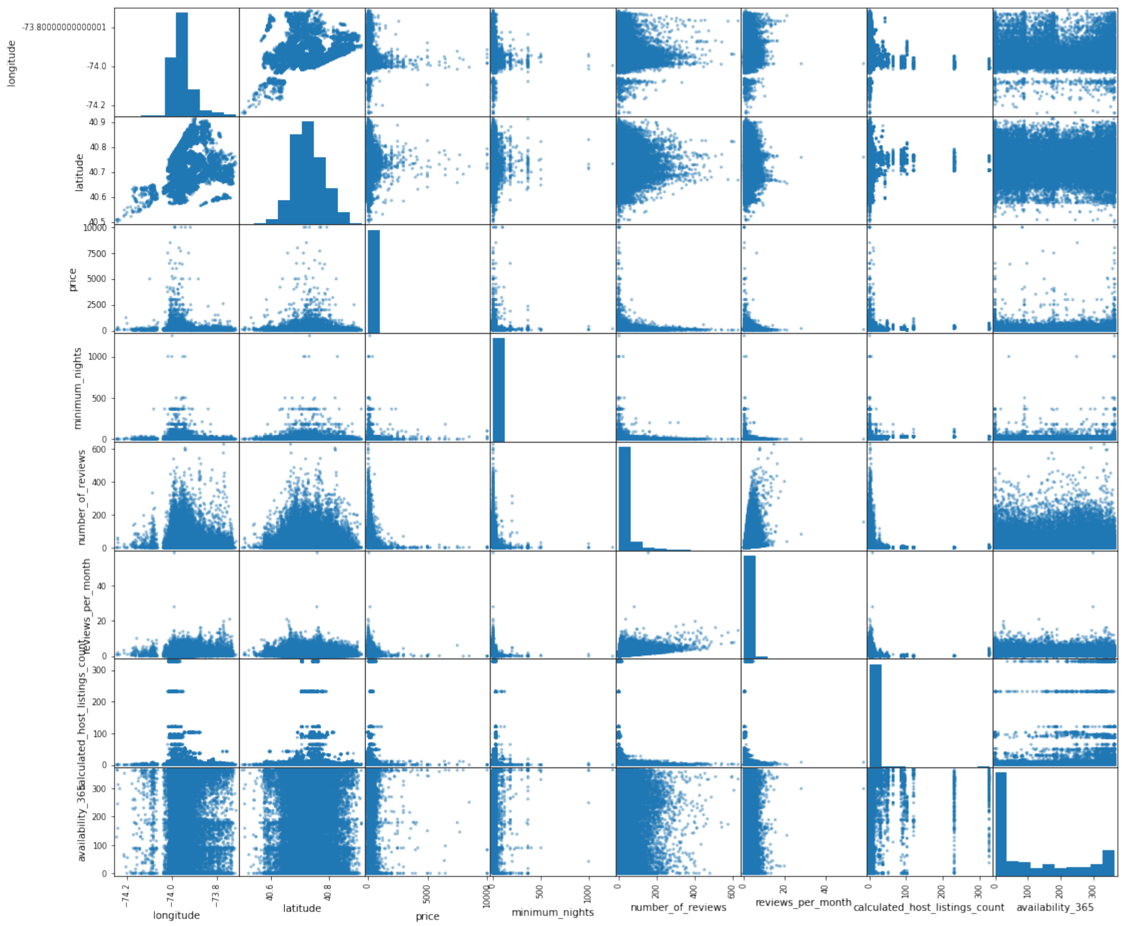
[45]:
```
airbnb.head()
```

[45]:
```
  neighbourhood_group neighbourhood  latitude  longitude        room_type  \
0            Brooklyn    Kensington  40.64749  -73.97237     Private room
```

```
1          Manhattan       Midtown  40.75362  -73.98377  Entire home/apt
2          Manhattan        Harlem  40.80902  -73.94190     Private room
3           Brooklyn  Clinton Hill  40.68514  -73.95976  Entire home/apt
4          Manhattan   East Harlem  40.79851  -73.94399  Entire home/apt

   price  minimum_nights  number_of_reviews  reviews_per_month  \
0    149               1                  9               0.21
1    225               1                 45               0.38
2    150               3                  0                NaN
3     89               1                270               4.64
4     80              10                  9               0.10

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

[46]:

```python
# select all numerical attributes
attributes = [
    "longitude",
    "latitude",
    "price",
    "minimum_nights",
    "number_of_reviews",
    "reviews_per_month",
    "calculated_host_listings_count",
    "availability_365"
]
scatter_matrix(airbnb[attributes], figsize=(18, 16))
plt.show()
```



- which features have positive correlation?
  - longitude and latitude have positive correlation. number_of_reviews and reviews_per_month have positive correlation as well.
- which features have negative correlation?
  - It doesn't seem like we have features with negative correlation.

3 [25 pts] Prepare the Data

3.0.1 [5 pts] Set aside 20% of the data as test test (80% train, 20% test).

[47]:
```
# StratifiedShuffleSplit does not work because there isn't enough data to
# split it in such a way it still maintains the distriubtion of the label
# in both train set and test set.
from sklearn.model_selection import train_test_split
airbnb_X = airbnb.drop("price", axis=1)
airbnb_y = airbnb["price"]
X_train, X_test, y_train, y_test = train_test_split(airbnb_X, airbnb_y,␣
↪test_size=0.2)
```

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

[48]:
```
  neighbourhood_group neighbourhood  latitude  longitude        room_type  \
0            Brooklyn    Kensington  40.64749  -73.97237     Private room
1           Manhattan       Midtown  40.75362  -73.98377  Entire home/apt
2           Manhattan        Harlem  40.80902  -73.94190     Private room
3            Brooklyn  Clinton Hill  40.68514  -73.95976  Entire home/apt
4           Manhattan   East Harlem  40.79851  -73.94399  Entire home/apt

   minimum_nights  number_of_reviews  reviews_per_month  \
0               1                  9               0.21
1               1                 45               0.38
2               3                  0                NaN
3               1                270               4.64
4              10                  9               0.10

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

[49]:
```
airbnb_y
```

[49]:
```
0        149
1        225
2        150
3         89
4         80
        ...
48890     70
48891     40
48892    115
48893     55
48894     90
Name: price, Length: 48895, dtype: int64
```

3.0.2 [5 pts] Augment the dataframe with two other features which you think would be useful

[50]:
```
# max_number_of_group_per_year indicates how many group of guests a host could␣
↪have in a year
airbnb_X["max_number_of_group_per_year"] = airbnb_X["availability_365"]/
↪airbnb_X["minimum_nights"]
# number of month indicates how long the post has been on the listings
```

```
airbnb_X["number_of_month"] = airbnb_X["number_of_reviews"]/
↪airbnb_X["reviews_per_month"]
```

[51]:
```
airbnb_X.head()
```

[51]:
```
  neighbourhood_group neighbourhood  latitude  longitude        room_type  \
0            Brooklyn    Kensington  40.64749  -73.97237     Private room
1           Manhattan       Midtown  40.75362  -73.98377  Entire home/apt
2           Manhattan        Harlem  40.80902  -73.94190     Private room
3            Brooklyn  Clinton Hill  40.68514  -73.95976  Entire home/apt
4           Manhattan   East Harlem  40.79851  -73.94399  Entire home/apt

   minimum_nights  number_of_reviews  reviews_per_month  \
0               1                  9               0.21
1               1                 45               0.38
2               3                  0                NaN
3               1                270               4.64
4              10                  9               0.10

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

   max_number_of_group_per_year  number_of_month
0                    365.000000        42.857143
1                    355.000000       118.421053
2                    121.666667              NaN
3                    194.000000        58.189655
4                      0.000000        90.000000
```

3.0.3 [5 pts] Impute any missing feature with a method of your choice, and briefly discuss why you chose this imputation method

[52]:
```
airbnb_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 12 columns):
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
minimum_nights                    48895 non-null int64
```

```
number_of_reviews                  48895 non-null int64
reviews_per_month                  38843 non-null float64
calculated_host_listings_count     48895 non-null int64
availability_365                   48895 non-null int64
max_number_of_group_per_year       48895 non-null float64
number_of_month                    38843 non-null float64
dtypes: float64(5), int64(4), object(3)
memory usage: 4.5+ MB
```

We see there are missing values for reviews_per_month and number_of_month.

reviews_per_month is missing probably means there is no reviews. So it makes sense to impute reviews_per_month with 0.

number_of_month is missing because reviews_per_month is missing (number_of_month an augumented feature calculated using reviews_per_month). If there is no review, then probably the post was listed not long time ago. So it makes sense to impute number_of_month with 0 as well.

[53]:
```python
sample_incomplete_rows = airbnb_X[airbnb_X.isnull().any(axis=1)].head()
sample_incomplete_rows
```

[53]:
```
   neighbourhood_group       neighbourhood  latitude  longitude  \
2            Manhattan              Harlem  40.80902  -73.94190
19           Manhattan         East Harlem  40.79685  -73.94872
26           Manhattan              Inwood  40.86754  -73.92639
36            Brooklyn  Bedford-Stuyvesant  40.68876  -73.94312
38            Brooklyn            Flatbush  40.63702  -73.96327

          room_type  minimum_nights  number_of_reviews  reviews_per_month  \
2      Private room               3                  0                NaN
19  Entire home/apt               7                  0                NaN
26     Private room               4                  0                NaN
36     Private room              60                  0                NaN
38     Private room               1                  0                NaN

    calculated_host_listings_count  availability_365  \
2                                1               365
19                               2               249
26                               1                 0
36                               1               365
38                               1               365

    max_number_of_group_per_year  number_of_month
2                     121.666667              NaN
19                     35.571429              NaN
26                      0.000000              NaN
36                      6.083333              NaN
38                    365.000000              NaN
```

[54]:
```python
# only reviews_per_month and number_of_month are missing values
# and we want to impute both of them with 0
sample_incomplete_rows.fillna(0, inplace=True)
sample_incomplete_rows
```

[54]:
```
   neighbourhood_group       neighbourhood  latitude  longitude  \
2            Manhattan              Harlem  40.80902  -73.94190
19           Manhattan         East Harlem  40.79685  -73.94872
26           Manhattan              Inwood  40.86754  -73.92639
36            Brooklyn  Bedford-Stuyvesant  40.68876  -73.94312
38            Brooklyn            Flatbush  40.63702  -73.96327

          room_type  minimum_nights  number_of_reviews  reviews_per_month  \
2      Private room               3                  0                0.0
19  Entire home/apt               7                  0                0.0
26     Private room               4                  0                0.0
36     Private room              60                  0                0.0
38     Private room               1                  0                0.0

    calculated_host_listings_count  availability_365  \
2                                1               365
19                               2               249
26                               1                 0
36                               1               365
38                               1               365

    max_number_of_group_per_year  number_of_month
2                     121.666667              0.0
19                     35.571429              0.0
26                      0.000000              0.0
36                      6.083333              0.0
38                    365.000000              0.0
```

3.0.4 [10 pts] Code complete data pipeline using sklearn mixins

[55]:
```python
# load airbnb data
airbnb = load_airbnb_data()

# drop features we don't want
airbnb = airbnb.drop(columns=["id", "name", "host_id", "host_name",␣
 ↪"last_review"])

# get features
airbnb_X = airbnb.drop(columns=["price"])

# get label
```

```
airbnb_y = airbnb["price"]

# remove the categorical feature
airbnb_num = airbnb_X.drop(columns=["neighbourhood_group", "neighbourhood",␣
↪"room_type"])

# list of numerical features
numerical_features = list(airbnb_num)

# list of categorical features
categorical_features = ["neighbourhood_group", "neighbourhood", "room_type"]

class AugmentFeatures(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        max_number_of_group_per_year = X["availability_365"] /␣
↪X["minimum_nights"]
        number_of_month = X["number_of_reviews"] / X["reviews_per_month"]
        return np.c_[X, max_number_of_group_per_year, number_of_month]

num_pipeline = Pipeline([
    ('attribs_adder', AugmentFeatures()),
    ('imputer', SimpleImputer(strategy="constant", fill_value=0)),
    ('std_scaler', StandardScaler()),
])

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

airbnb_prepared = full_pipeline.fit_transform(airbnb_X)
X_train, X_test, y_train, y_test = train_test_split(airbnb_prepared, airbnb_y,␣
↪test_size=0.2)
```

4 [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.

[56]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
```

[56]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

[57]:
```
preds = lin_reg.predict(X_test)
mse = mean_squared_error(y_test, preds)
print("test mse:", mse)
```

```
test mse: 40542.97025957861
```

[58]:
```
preds = lin_reg.predict(X_train)
mse = mean_squared_error(y_train, preds)
print("train mse:", mse)
```

```
train mse: 53294.8148603786
```

[ ]:
```

```