Intro-End2End

March 22, 2020

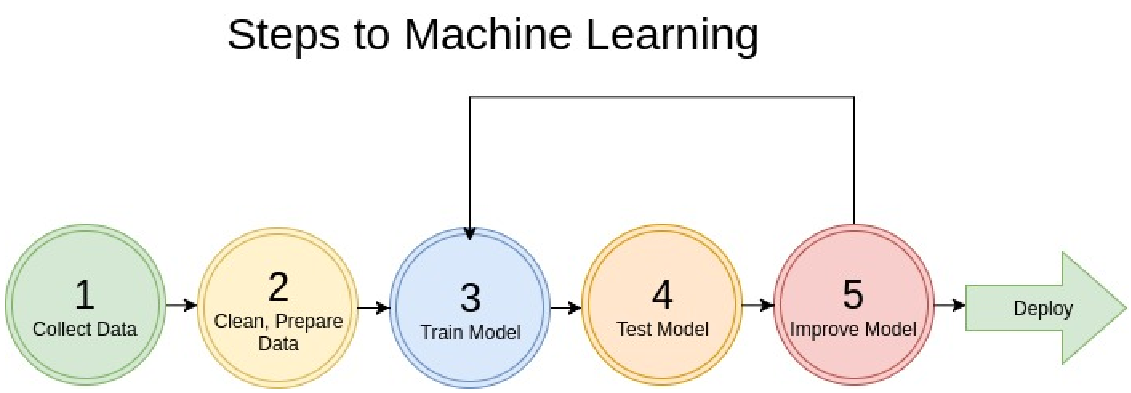
0.1 Introduction

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



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
%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
    '''
    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")
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
```

```
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

[6]:
```
# 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]:
```
# 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]:
```
# 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
```

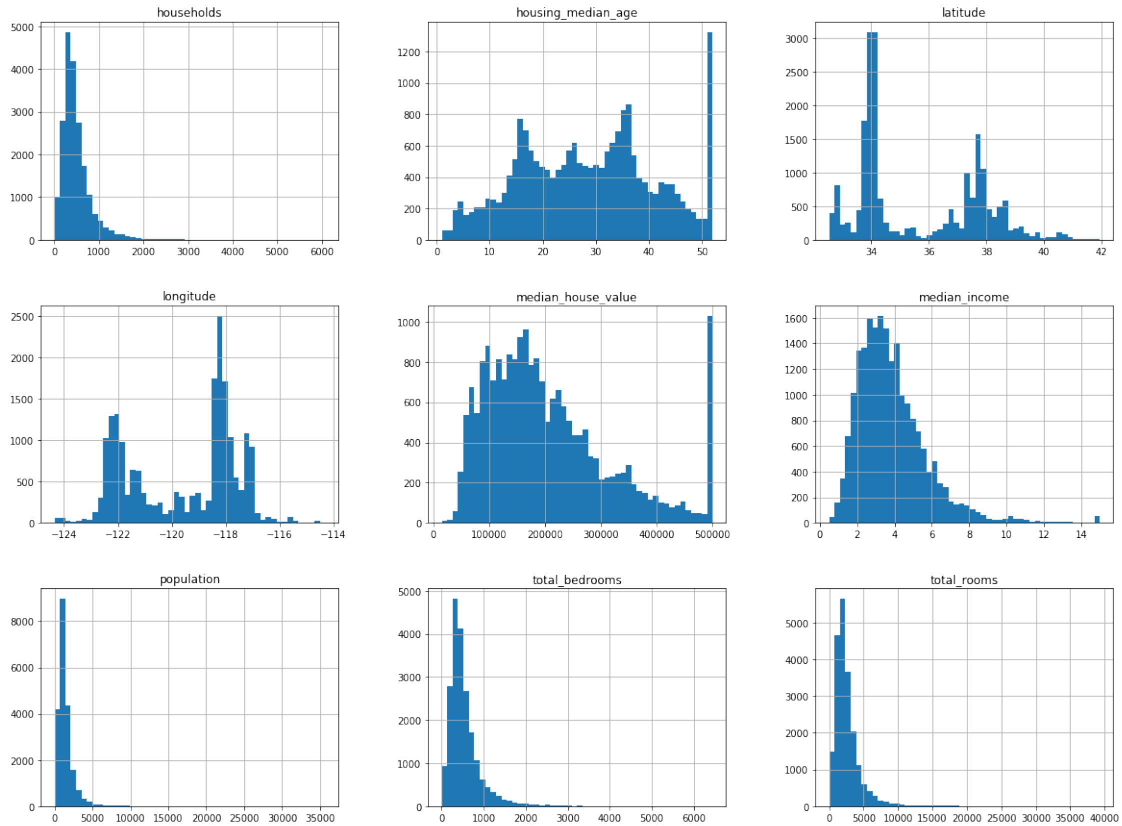
[9]:
```python
# The describe function compiles your typical statistics for each
# column
housing.describe()
```

[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
```

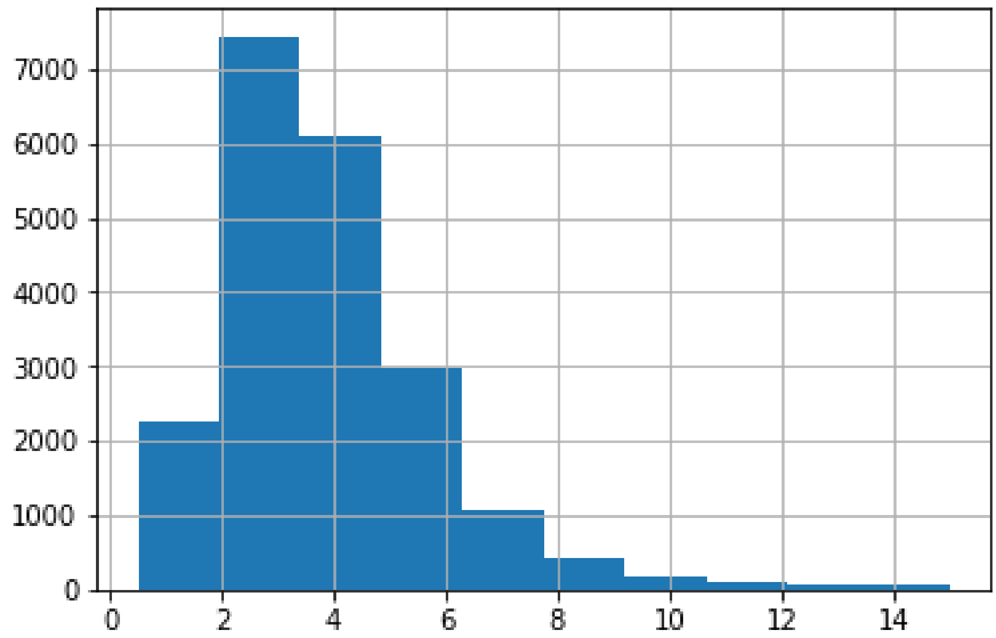
If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section here

0.5 Let's start visualizing the dataset

[10]:
```python
# 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
```



[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.

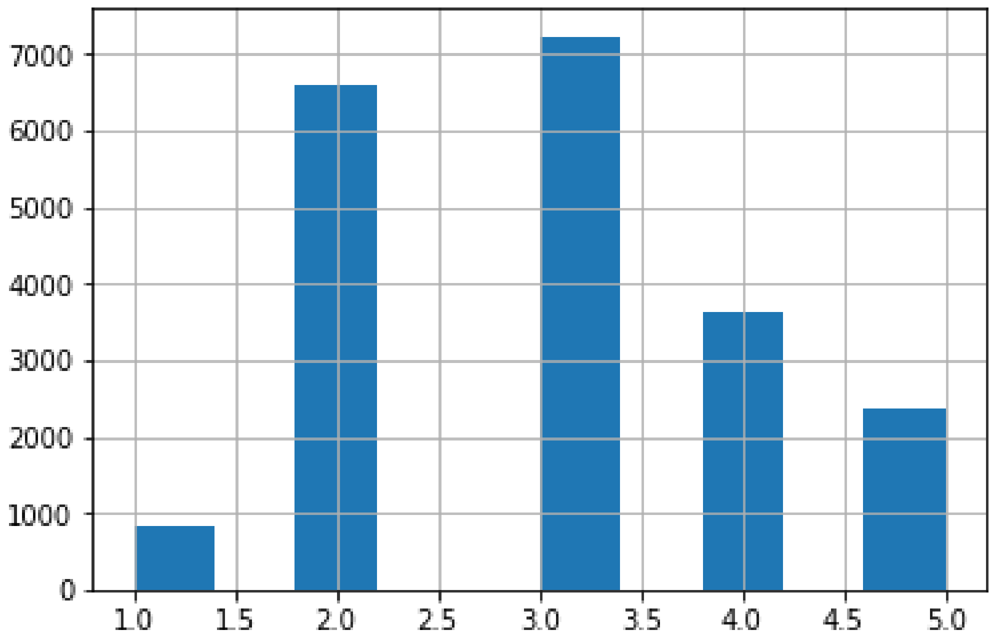
For example, to bin the households based on median_income we can use the pd.cut function

[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"].hist()
```

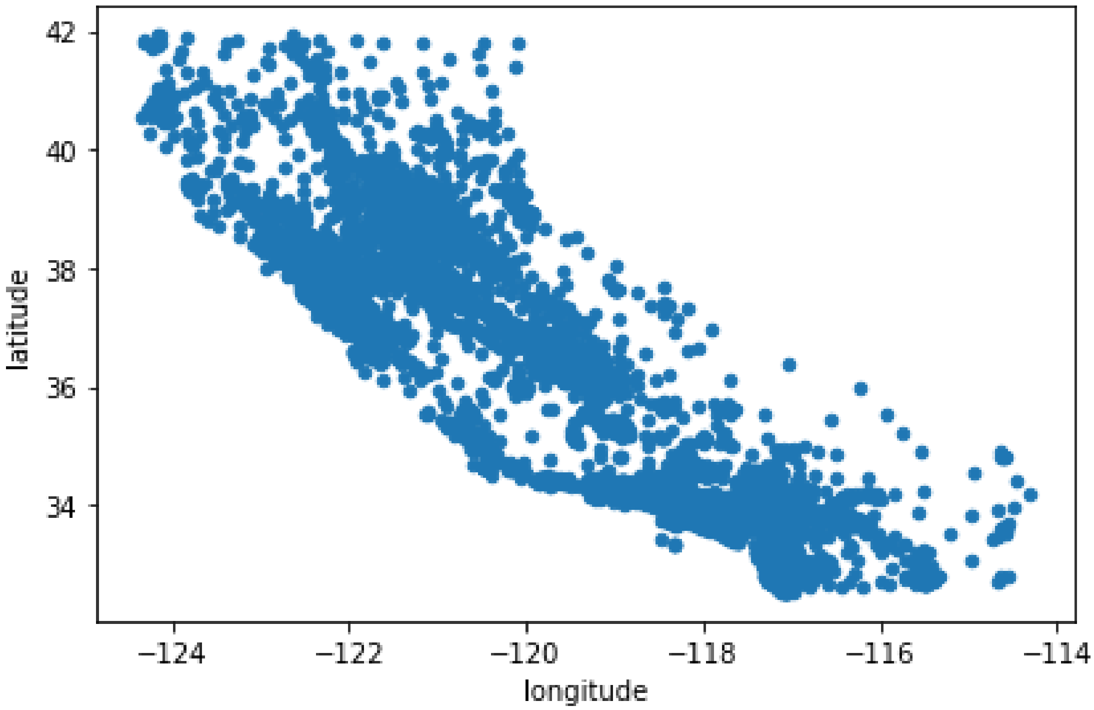
[13]: <matplotlib.axes._subplots.AxesSubplot at 0x12b769450>



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

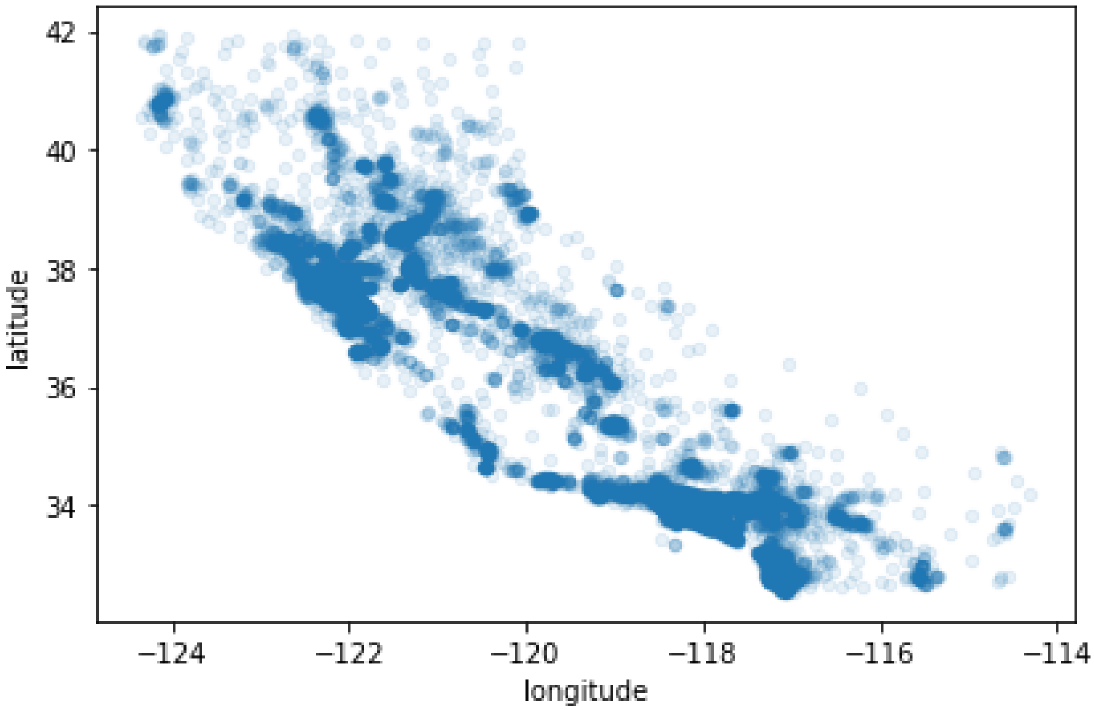
[14]:
```
## here's a not so interestting way plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

```
Saving figure bad_visualization_plot
```



[15]:
```python
# 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")
```

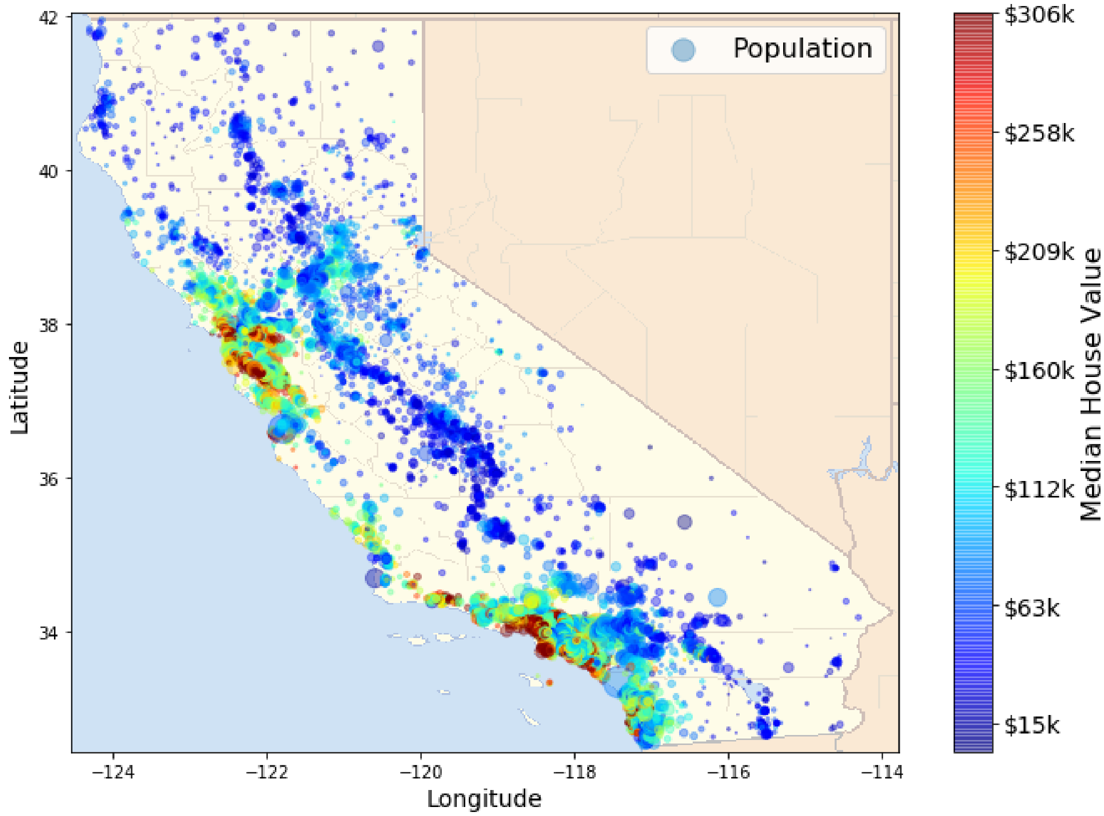
```
Saving figure better_visualization_plot
```



[16]:
```
# 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.

[17]:
```python
corr_matrix = housing.corr()
```

[18]:
```python
corr_matrix
```

[18]:
```
                    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
```

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

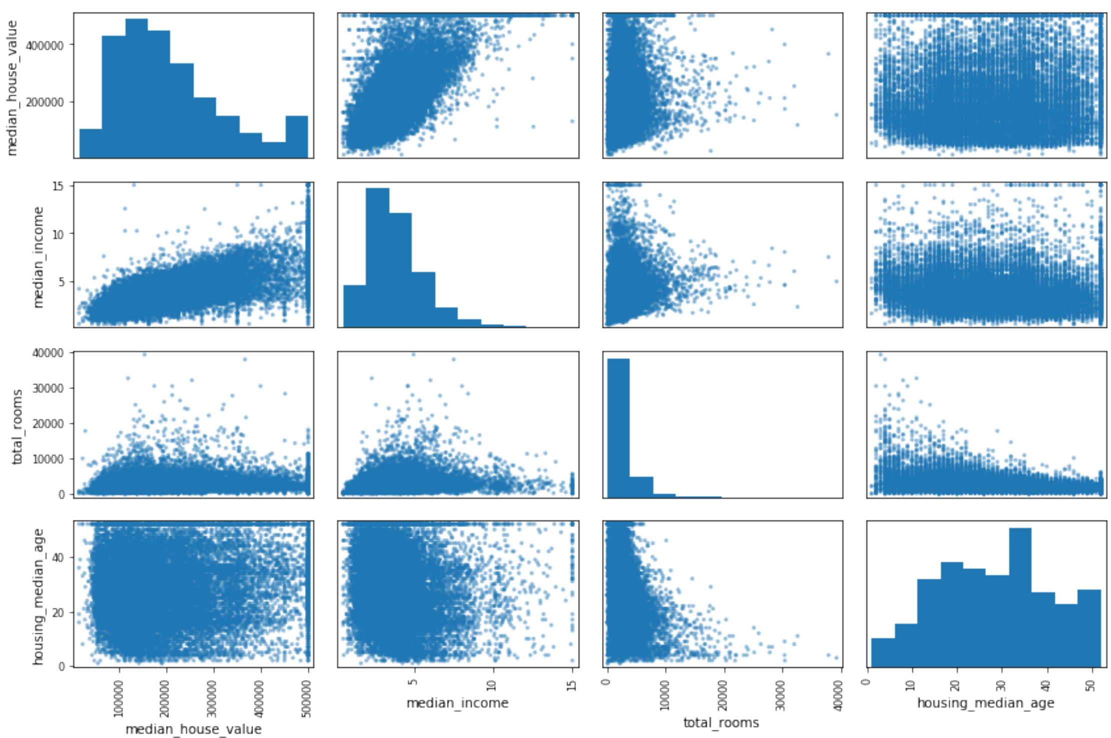
[19]:
```
median_house_value    1.000000
median_income         0.688075
```

```
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]:

```python
# 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
```

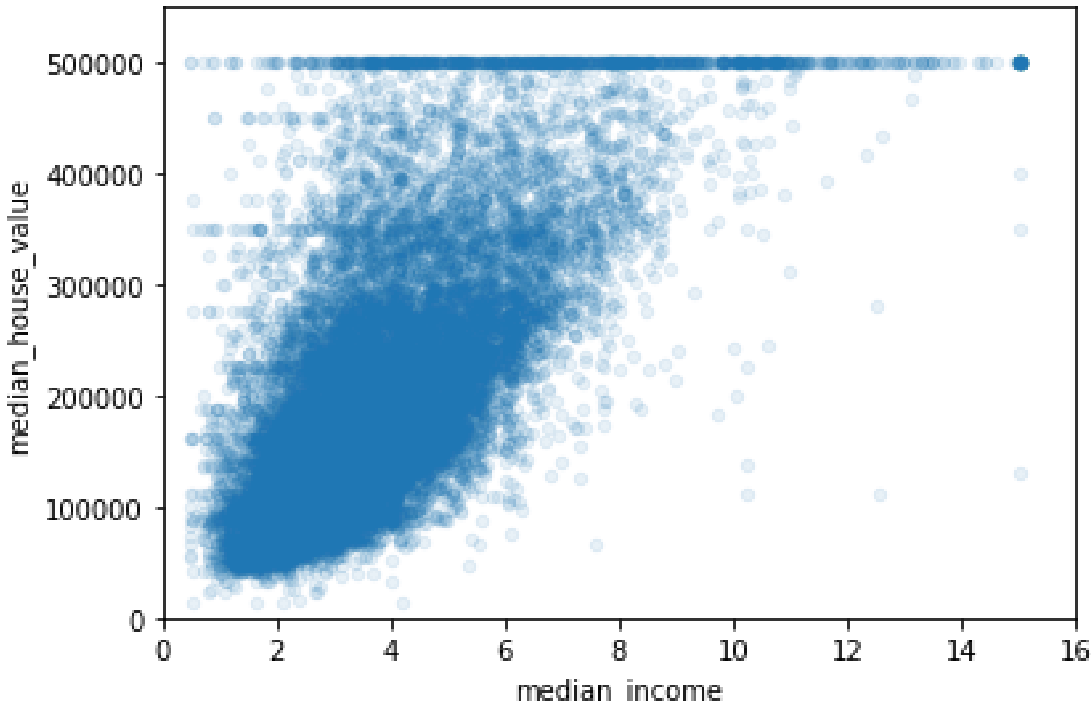


[21]:

```python
# 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.

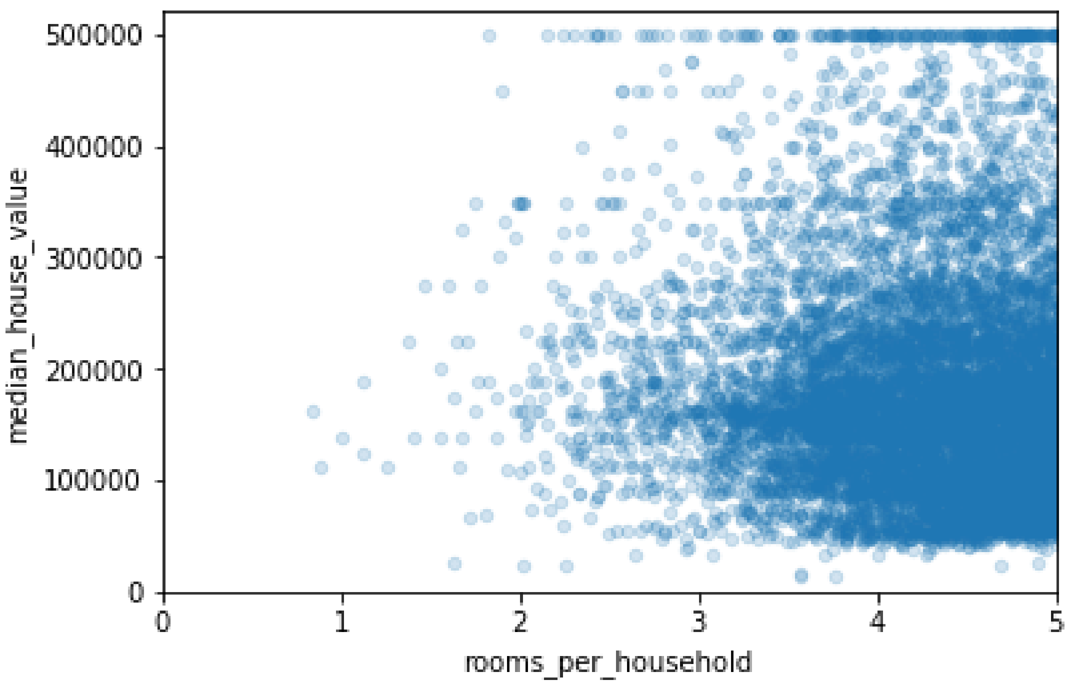
[22]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

[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.describe()
```

[25]:
```
          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
```

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]:
```python
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)
#print(type(split))
for train_index, test_index in split.split(housing, housing["income_cat"]):
    #print(train_index, test_index)
    train_set = housing.loc[train_index]
    test_set = housing.loc[test_index]
```

[27]:
```python
train_set.head()
```

[27]:
```
       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

       population  households  median_income  median_house_value  \
17606       710.0       339.0         2.7042            286600.0
18632       306.0       113.0         6.4214            340600.0
14650       936.0       462.0         2.8621            196900.0
3230       1460.0       353.0         1.8839             46300.0
3555       4459.0      1463.0         3.0347            254500.0

      ocean_proximity income_cat  rooms_per_household  bedrooms_per_room  \
17606       <1H OCEAN          2             4.625369           0.223852
18632       <1H OCEAN          5             6.008850           0.159057
14650      NEAR OCEAN          2             4.225108           0.241291
3230           INLAND          2             5.232295           0.200866
3555        <1H OCEAN          3             4.505810           0.231341

       population_per_household
17606                  2.094395
18632                  2.707965
14650                  2.025974
3230                   4.135977
3555                   3.047847
```

[28]:
```python
# Saves the modified training data in the housing variable
```

```python
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
# Keep track for later
housing_labels = train_set["median_house_value"].copy()
```

0.6.1 Dealing With Incomplete Data

[29]:
```python
# 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]:
```python
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]:
```python
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]:
```python
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"]
    '''
```

```python
    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)
```

[34]:
```python
housing_prepared[:3]
```

[34]:
```
array([[-1.15604281,  0.77194962,  0.74333089, -0.49323393, -0.44543821,
        -0.63621141, -0.42069842, -0.61493744, -0.95445595, -0.31205452,
         0.19380531, -0.08649871, -0.31205452, -0.08649871,  0.15531753,
         1.        ,  0.        ,  0.        ,  0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
        -0.99833135, -1.02222705,  1.33645936,  1.89030518,  0.21768338,
        -0.94074539, -0.03353391,  0.21768338, -0.03353391, -0.83628902,
         1.        ,  0.        ,  0.        ,  0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, -0.31365989, -0.15334458,
        -0.43363936, -0.0933178 , -0.5320456 , -0.95445595, -0.46531516,
         0.49916044, -0.09240499, -0.46531516, -0.09240499,  0.4222004 ,
```

```
0.         ,  0.         ,  0.         ,  0.         ,  1.         ]])
```

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.

[35]:
```
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]
```

```
/usr/local/lib/python3.7/site-
packages/sklearn/compose/_column_transformer.py:430: FutureWarning: Given
feature/column names or counts do not match the ones for the data given during
fit. This will fail from v0.24.
  FutureWarning)
```

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.

[36]:
```
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
```

[36]: 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

[37]:
```python
# Loading the dataset
airbnb_data = pd.read_csv('./datasets/airbnb/AB_NYC_2019.csv')

# Displaying the first few rows of the data
airbnb_data.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]:
```
airbnb_data.info()
```

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

[39]:
```
# Draw these columns inplace so it updates the current dataset itself
airbnb_data.drop(['name','host_id','host_name','last_review'], axis=1,␣
 ↪inplace=True)
airbnb_data.head()
```

[39]:
```
     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
```

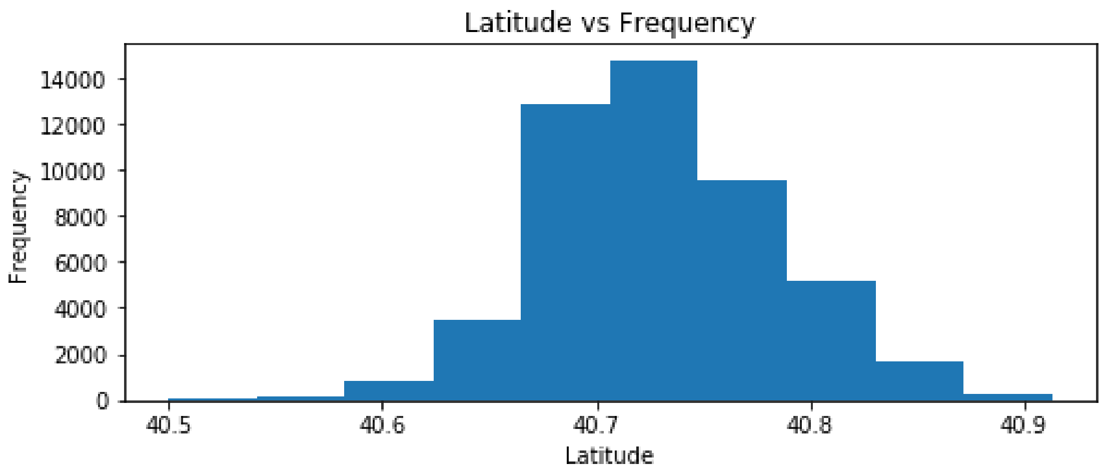
```
1            0.38                               2             355
2             NaN                               1             365
3            4.64                               1             194
4            0.10                               1               0
```

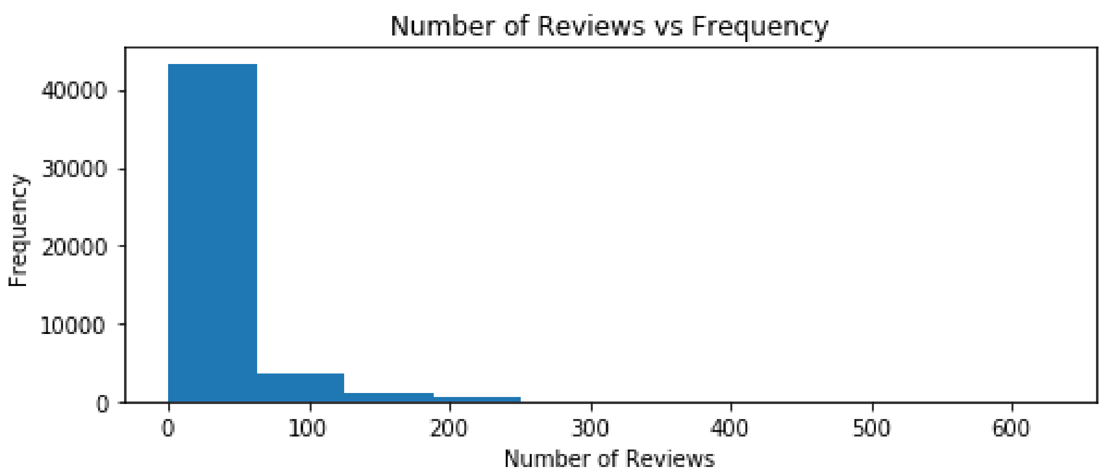
[40]:
```python
# Summary of the statistics of the loaded data:
airbnb_data.describe()
```

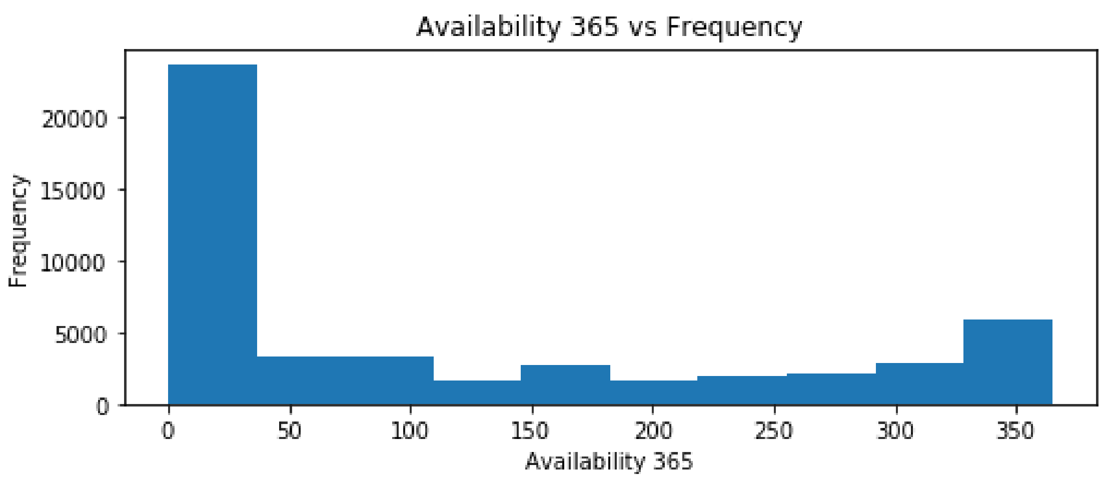
[40]:
```
                 id      latitude     longitude         price  minimum_nights  \
count  4.889500e+04  48895.000000  48895.000000  48895.000000    48895.000000
mean   1.901714e+07     40.728949    -73.952170    152.720687        7.029962
std    1.098311e+07      0.054530      0.046157    240.154170       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
```

[41]:
```python
# 3 graphs of features
fig, ax = plt.subplots(3)
fig.subplots_adjust(hspace=0.5)
fig.set_figheight(12)
fig.set_figwidth(8)

ax[0].hist(airbnb_data["latitude"])
ax[0].set_title('Latitude vs Frequency')
ax[0].set_xlabel('Latitude')
```

```
ax[0].set_ylabel('Frequency')

ax[1].hist(airbnb_data["number_of_reviews"])
ax[1].set_title('Number of Reviews vs Frequency')
ax[1].set_xlabel('Number of Reviews')
ax[1].set_ylabel('Frequency')

ax[2].hist(airbnb_data["availability_365"])
ax[2].set_title('Availability 365 vs Frequency')
ax[2].set_xlabel('Availability 365')
ax[2].set_ylabel('Frequency')

plt.show()
```


Latitude vs Frequency
Frequency
Latitude
14000
12000
10000
8000
6000
4000
2000
0
40.5
40.6
40.7
40.8
40.9

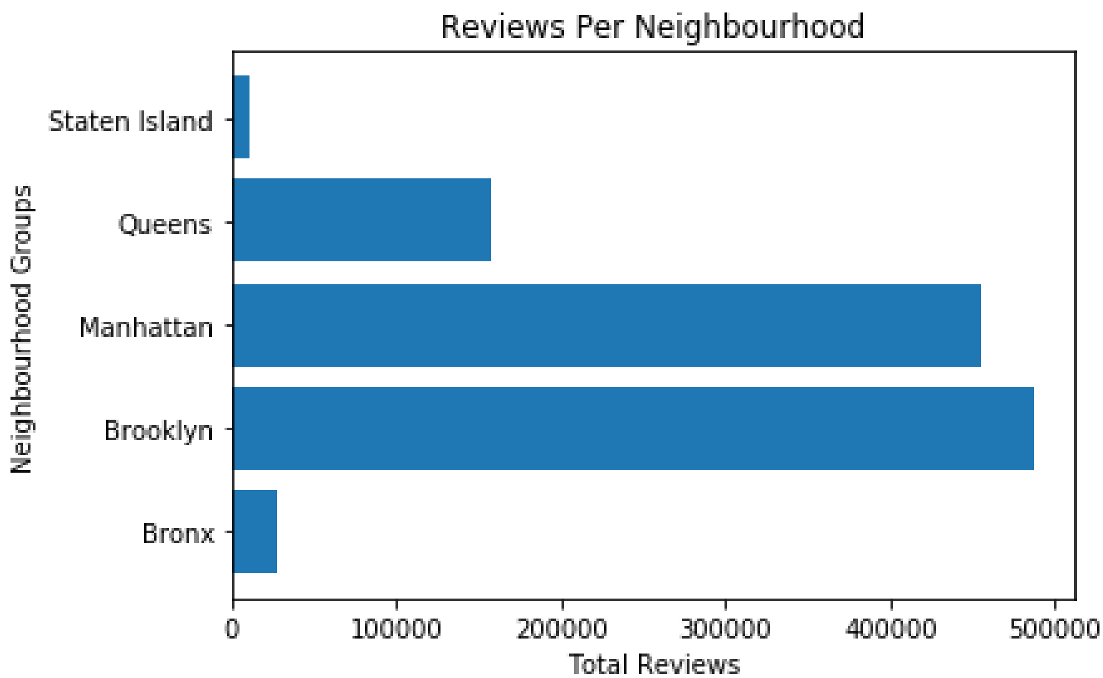

Number of Reviews vs Frequency
Frequency
Number of Reviews
40000
30000
20000
10000
0
0
100
200
300
400
500
600


Availability 365 vs Frequency
Frequency
Availability 365
20000
15000
10000
5000
0
0
50
100
150
200
250
300
350

2.0.2 [5 pts] Plot total number_of_reviews per neighbourhood_group

[42]:
```
data_subset = airbnb_data[['number_of_reviews','neighbourhood_group']]
data_subset.head()

# Use sum to calculate `total` number of reviews. Like a SQL group by.
reviews_per_neighbourhood_group = data_subset.groupby(['neighbourhood_group']).
 ↪sum().reset_index()
reviews_per_neighbourhood_group.columns =␣
 ↪['neighbourhood_group','total_reviews']

print(reviews_per_neighbourhood_group)

neighbourhood_group = reviews_per_neighbourhood_group['neighbourhood_group'].
 ↪tolist()
total_reviews = reviews_per_neighbourhood_group['total_reviews'].tolist()

fig, ax = plt.subplots()
ax.barh(neighbourhood_group, total_reviews)
ax.set_title('Reviews Per Neighbourhood')
ax.set_xlabel('Total Reviews')
ax.set_ylabel('Neighbourhood Groups')
plt.show()
```

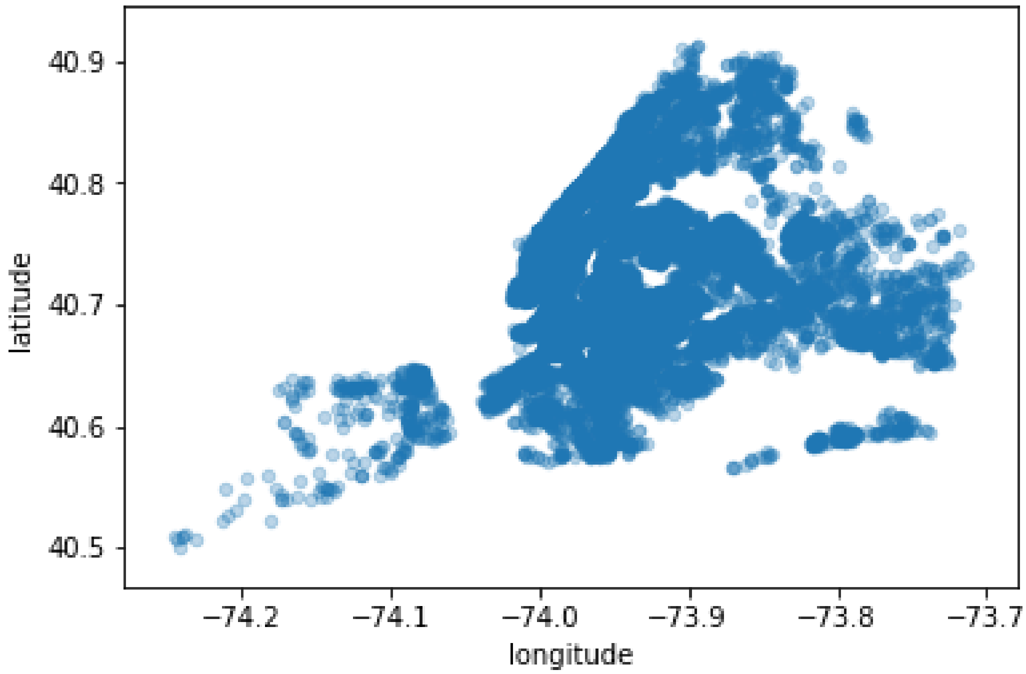
```
  neighbourhood_group  total_reviews
0               Bronx          28371
1            Brooklyn         486574
2           Manhattan         454569
3              Queens         156950
4       Staten Island          11541
```



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 :)).

[43]:
```
airbnb_data.plot(kind='scatter',x='longitude',y='latitude',alpha=0.3)
```

[43]: <matplotlib.axes._subplots.AxesSubplot at 0x13152b050>



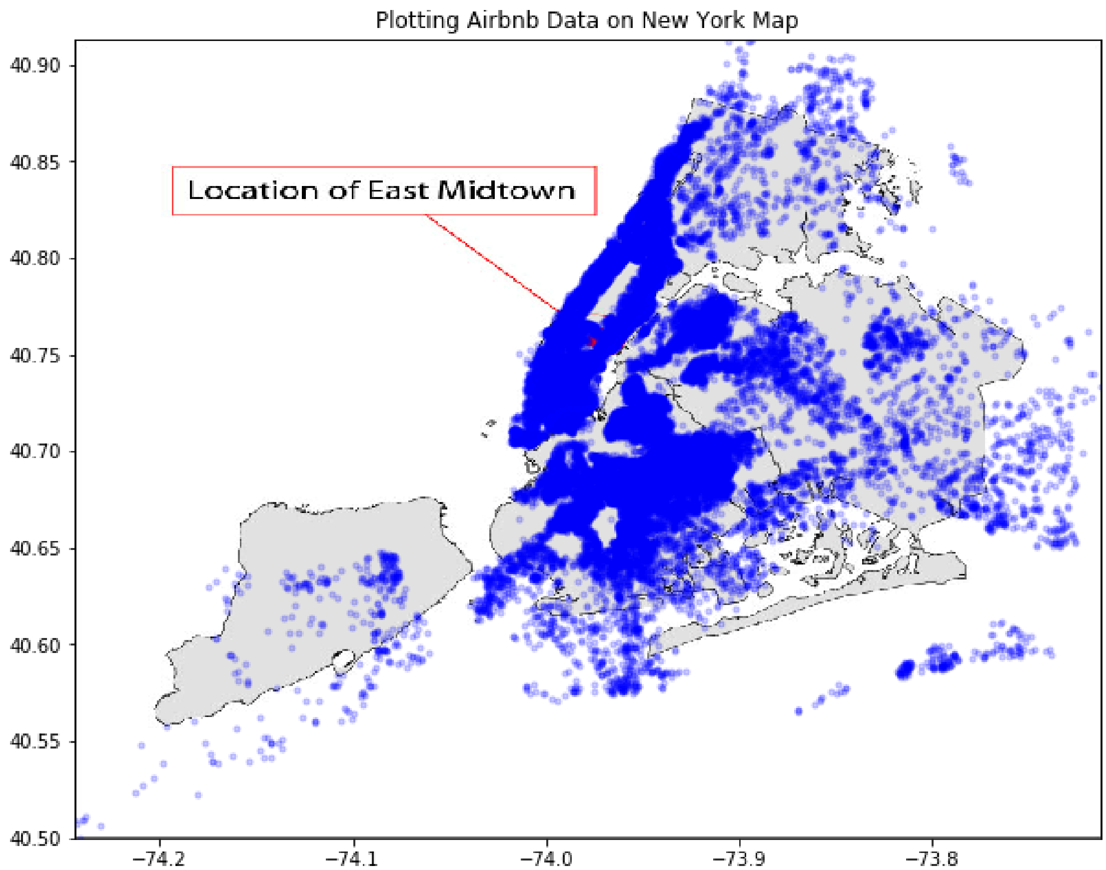
[44]:
```
# Finding coordinates for boundary
latitude_min = airbnb_data['latitude'].min()
latitude_max = airbnb_data['latitude'].max()
longitude_min = airbnb_data['longitude'].min()
longitude_max = airbnb_data['longitude'].max()
BBox = (longitude_min, longitude_max, latitude_min, latitude_max)
print(BBox)
```

```
(-74.24441999999999, -73.71299, 40.499790000000004, 40.913059999999994)
```

[45]:
```
# load an image of greater new york
images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename = "greater_new_york.jpg"

import matplotlib.image as mpimg
newyork_img=mpimg.imread(os.path.join(images_path, filename))

fig, ax = plt.subplots(figsize = (12,8))
ax.scatter(airbnb_data.longitude, airbnb_data.latitude, zorder=1, alpha=0.2,␣
 ↪c='b', s=10)
ax.set_title('Plotting Airbnb Data on New York Map')
ax.set_xlim(BBox[0],BBox[1])
ax.set_ylim(BBox[2],BBox[3])
```

```python
ax.imshow(newyork_img, zorder=0, extent = BBox)
```

[45]: <matplotlib.image.AxesImage at 0x1308950d0>

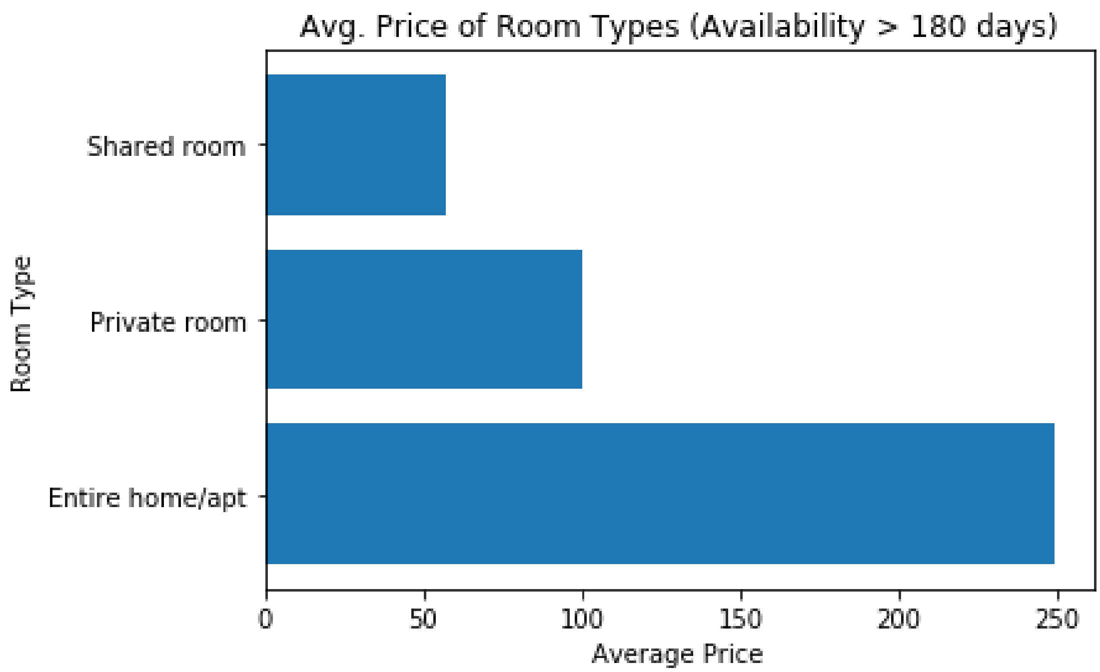


2.0.4 [5 pts] Plot average price of room types who have availability greater than 180 days.

[46]:
```python
availability = airbnb_data[airbnb_data['availability_365'] > 180]
availability = availability[['room_type','price']]
availability = availability.groupby(['room_type']).mean().reset_index()

print(availability)

room_types = availability['room_type'].tolist()
prices = availability['price'].tolist()

fig, ax = plt.subplots()
ax.barh(room_types, prices)
ax.set_title('Avg. Price of Room Types (Availability > 180 days)')
```

```
ax.set_xlabel('Average Price')
ax.set_ylabel('Room Type')
plt.show()
```

```
         room_type       price
0  Entire home/apt  248.870817
1     Private room  100.028192
2      Shared room   56.941909
```



2.0.5 [5 pts] Plot correlation matrix

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

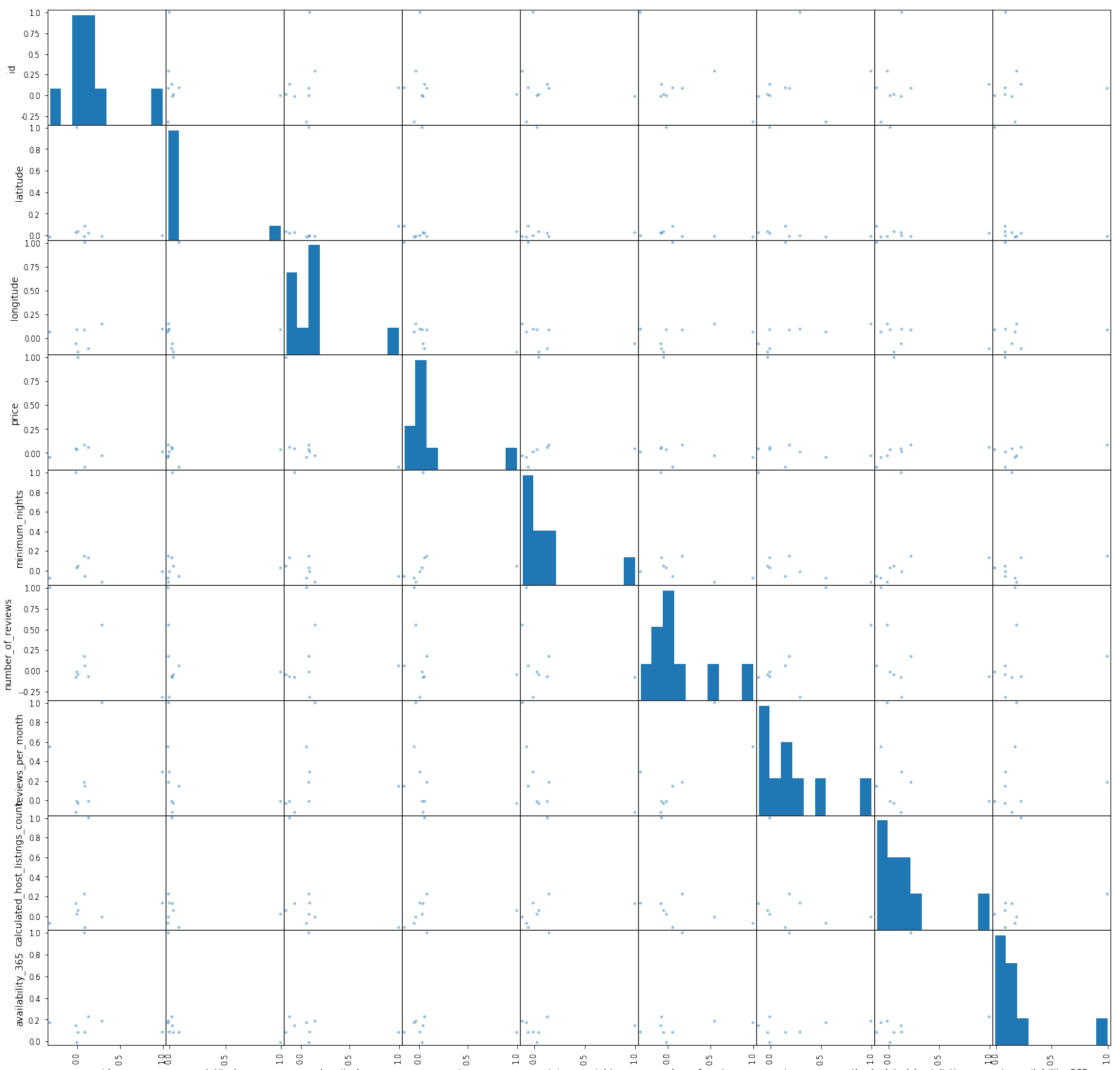
[47]:
```
airbnb_data.columns
```

[47]:
```
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')
```

[48]:
```
# No categorical, remove latitude and longitude
features=['price','minimum_nights','number_of_reviews','reviews_per_month',
          'calculated_host_listings_count','availability_365']
```

```python
scatter_matrix(airbnb_data[features], figsize=(12,12))
```

[48]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x130e8f910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1308b7950>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x130a32790>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x13092f890>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1309cf3d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12b16fa90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x12b435f10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12b5af8d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12b3c2550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12bc40210>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12bc03d50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12afea290>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x12b875190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12c7a04d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12b5c54d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12b76a610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12ed31b10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x130940e90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x12c8bab50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12fd3ced0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x13091fb90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12c7fcf10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12ec5bbd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12e98cf50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x12ece5c10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x13002af90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x130057c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12b911fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12be34c90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12bfc14d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x12c515cd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12c556510>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12c58bd10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12c5cc550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12c602d50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x12c642590>]],
      dtype=object)
```



[49]:
```
corr = airbnb_data.corr()
corr["price"].sort_values(ascending=False)
```

[49]:
```
price                             1.000000
availability_365                  0.081829
calculated_host_listings_count    0.057472
minimum_nights                    0.042799
latitude                          0.033939
id                                0.010619
reviews_per_month                -0.030608
number_of_reviews                -0.047954
longitude                        -0.150019
Name: price, dtype: float64
```

Based on the values seen above from the correlation of all input features with the target variable (price), the relationship can be split into two - namely positive correlation and negative correlation.

Positively Correlated Features:

- availability_365 0.081829
- calculated_host_listings_count 0.057472
- minimum_nights 0.042799
- latitude 0.033939
- id 0.010619

The most positively correlated feature (with price) in the input data is `availability_365`.

Negatively Correlated Features:

- reviews_per_month -0.030608
- number_of_reviews -0.047954
- longitude -0.150019

The most negatively correlated feature (with price) in the input data is `longitude`.

3 [25 pts] Prepare the Data

3.0.1 [5 pts] Augment the dataframe with two other features which you think would be useful

[50]:
```
airbnb_data['price_by_reviews'] = airbnb_data['price']/
↪airbnb_data['number_of_reviews']
airbnb_data['reviews_per_host_listings'] = airbnb_data['number_of_reviews']/
↪airbnb_data['calculated_host_listings_count']
```

[51]:
```
airbnb_data.head(5)
```

[51]:
```
     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

   price_by_reviews  reviews_per_host_listings
0         16.555556                        1.5
1          5.000000                       22.5
2               inf                        0.0
3          0.329630                      270.0
4          8.888889                        9.0
```

3.0.2 [5 pts] Impute any missing feature with a method of your choice, and briefly discuss why you chose this imputation method

[52]:
```python
airbnb_data[airbnb_data.isnull().any(axis=1)].head(3)
```

[52]:
```
      id neighbourhood_group neighbourhood  latitude  longitude  \
2   3647           Manhattan        Harlem  40.80902  -73.94190
19  7750           Manhattan   East Harlem  40.79685  -73.94872
26  8700           Manhattan        Inwood  40.86754  -73.92639

          room_type  price  minimum_nights  number_of_reviews  \
2      Private room    150               3                  0
19  Entire home/apt    190               7                  0
26     Private room     80               4                  0

    reviews_per_month  calculated_host_listings_count  availability_365  \
2                 NaN                               1               365
19                NaN                               2               249
26                NaN                               1                 0

    price_by_reviews  reviews_per_host_listings
2                inf                        0.0
19               inf                        0.0
26               inf                        0.0
```

[53]:
```python
airbnb_data.isna().head(3)
```

[53]:
```
      id  neighbourhood_group  neighbourhood  latitude  longitude  room_type  \
0  False                False          False     False      False      False
1  False                False          False     False      False      False
2  False                False          False     False      False      False

   price  minimum_nights  number_of_reviews  reviews_per_month  \
0  False           False              False              False
1  False           False              False              False
```

```
2  False           False              False               True

   calculated_host_listings_count  availability_365  price_by_reviews  \
0                           False             False             False
1                           False             False             False
2                           False             False             False

   reviews_per_host_listings
0                      False
1                      False
2                      False
```

[54]:
```
# Price by reviews is not a very logical feature, and additionally, theres too␣
↪many
# records with number_of_reviews = 0, and just dropping those rows would␣
↪greatly
# reduce the size of this dataset. Therefore, I decided to drop this entire␣
↪feature.
airbnb_data = airbnb_data.drop('price_by_reviews', axis=1)

# For reviews_per_month, I decided to find the median value and use it to␣
↪substitute
# all the NaN values in the 'reviews_per_month' column. Instead of losing rows␣
↪of
# data, I felt like this was more statistically significant and made more␣
↪sense, and a
# somewhat decent assumption to make, although it will skew the loss value␣
↪significantly.
reviews_per_month_median = airbnb_data['reviews_per_month'].median()
airbnb_data['reviews_per_month'].fillna(reviews_per_month_median, inplace=True)

# After the above imputations, if there were still any rows with NaNs (about␣
↪5-10 rows),
# I decided to drop them altogether since we had dealt with most of the NaNs in␣
↪a logical
# way prior to this.
airbnb_data.dropna(inplace=True)
```

3.0.3 [10 pts] Code complete data pipeline using sklearn mixins

[55]:
```
# Drop ID since it is a unique feature (and quite irrelevant)
# One hot-encoding vector will be massive = equal to number of unique IDs,␣
↪leading to a highly sparse matrix
airbnb_data = airbnb_data.drop('id', axis=1)
```

[56]:
```
print(len(list(airbnb_data['neighbourhood_group'].unique())))
# Primary source of feature expansion in model when this categorical feature is␣
↪one-hot encoded
print(len(list(airbnb_data['neighbourhood'].unique())))
print(len(list(airbnb_data['room_type'].unique())))
```

```
5
221
3
```

[57]:
```
# Create a copy of the output features to predict
labels = airbnb_data['price'].copy()
airbnb_data = airbnb_data.drop('price', axis=1)
```

[58]:
```
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

categorical_features = ['neighbourhood_group', 'neighbourhood', 'room_type']
airbnb_no_categorical = airbnb_data.drop(categorical_features, axis=1)

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('std_scaler', StandardScaler()),
    ])

numerical_features = list(airbnb_no_categorical)

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

airbnb_prepared = full_pipeline.fit_transform(airbnb_data)
```

[59]:
```
airbnb_prepared[:3]
```

[59]:
```
<3x237 sparse matrix of type '<class 'numpy.float64'>'
        with 33 stored elements in Compressed Sparse Row format>
```

Primary source for the expansion/creation of so many features comes from one-hot encoding, but particularly, from the `neighbourhood` feature, which has **220 unique values**, resulting in a really sparse one hot encoded matrix.

3.0.4 [5 pts] Set aside 20% of the data as test set (80% train, 20% test).

[60]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(airbnb_prepared, labels,␣
↪test_size=0.2, random_state=23)
```

[61]:
```
X_train.shape
```

[61]: (39116, 237)

[62]:
```
X_test.shape
```

[62]: (9779, 237)

[63]:
```
y_train.shape
```

[63]: (39116,)

[64]:
```
y_test.shape
```

[64]: (9779,)

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.

[65]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
```

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

[66]:
```
# Test set MSE
preds = lin_reg.predict(X_test)
mse = mean_squared_error(y_test, preds)
print("Test MSE:",mse)
```

```
Test MSE: 46378.827120361464
```

[67]:
```
# Train set MSE
preds = lin_reg.predict(X_train)
mse = mean_squared_error(y_train, preds)
print("Train MSE:",mse)
```

```
Train MSE: 52102.518239250894
```