CS 188 Project 1

January 21, 2020

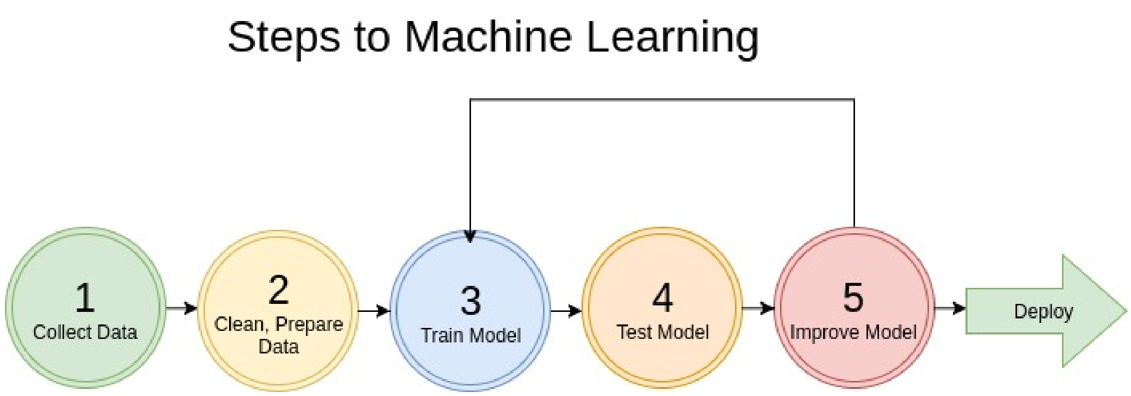
0.1 Introduction

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

[2]:
```
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)
```

[3]:
```
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

[4]:
```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)
```

[5]:
```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
```

[5]:
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

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

A dataset may have different types of features - real valued - Discrete (integers) - categorical (strings)

The two categorical features are essentialy the same as you can always map a categorical string/character to an integer.

In the dataset example, all our features are real valued floats, except ocean proximity which is categorical.

[6]:
```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):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
```

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

[7]:
```
# you can access individual columns similarly
# to accessing elements in a python dict
housing["ocean_proximity"].head() # added head() to avoid printing many columns.
↪.
```

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

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

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

[9]:
```
# one other function that might be useful is
# value_counts(), which counts the number of occurences
# for categorical features
housing["ocean_proximity"].value_counts()
```

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

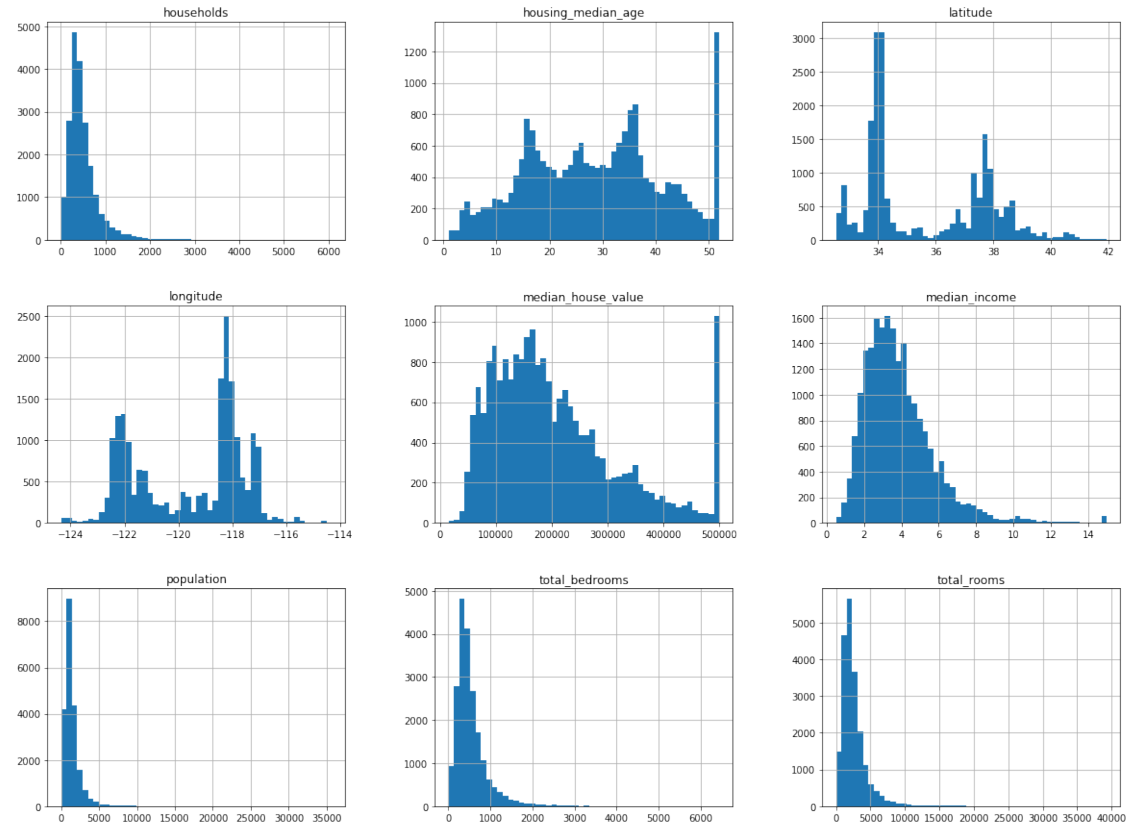
[10]:
```python
# The describe function compiles your typical statistics for each
# column
housing.describe()
```

[10]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000
```

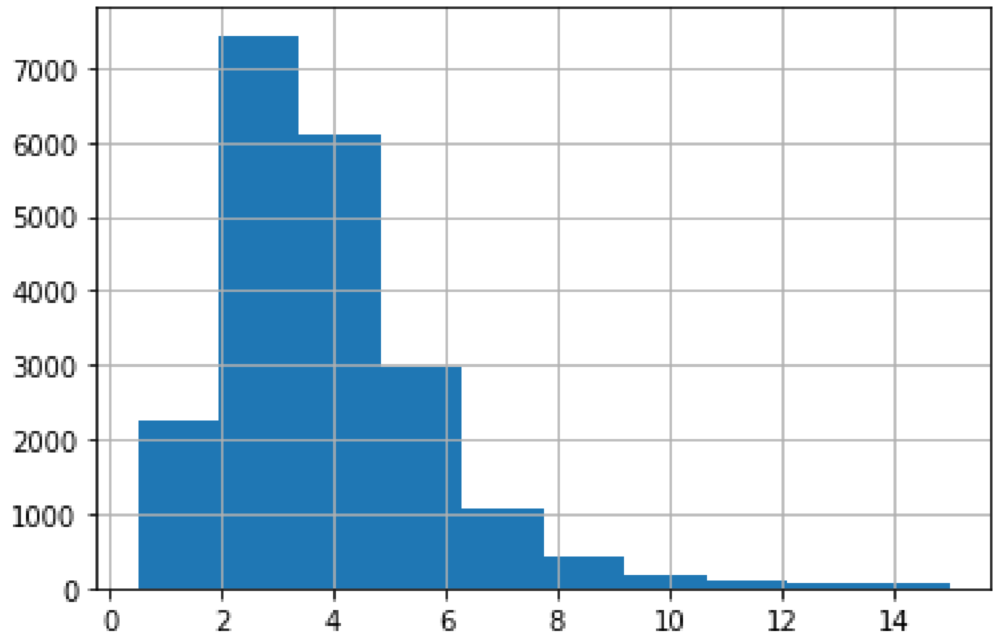
If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section here

0.5 Let's start visualizing the dataset

[11]:
```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
```



[12]:
```python
# if you want to have a histogram on an individual feature:
housing["median_income"].hist()
plt.show()
```



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

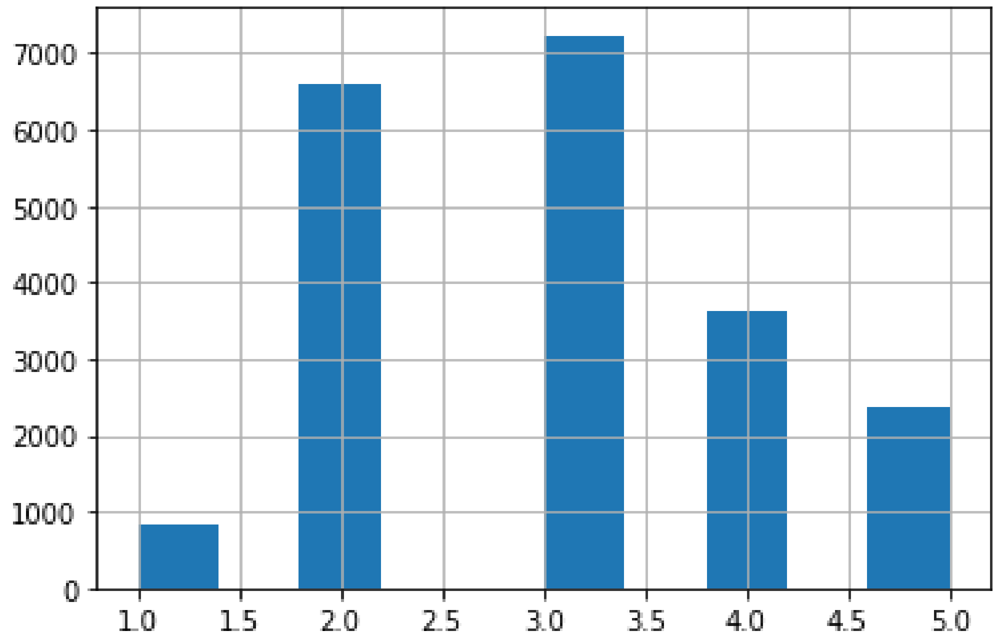
For example, to bin the households based on median_income we can use the pd.cut function

[13]:
```
# assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])

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

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

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

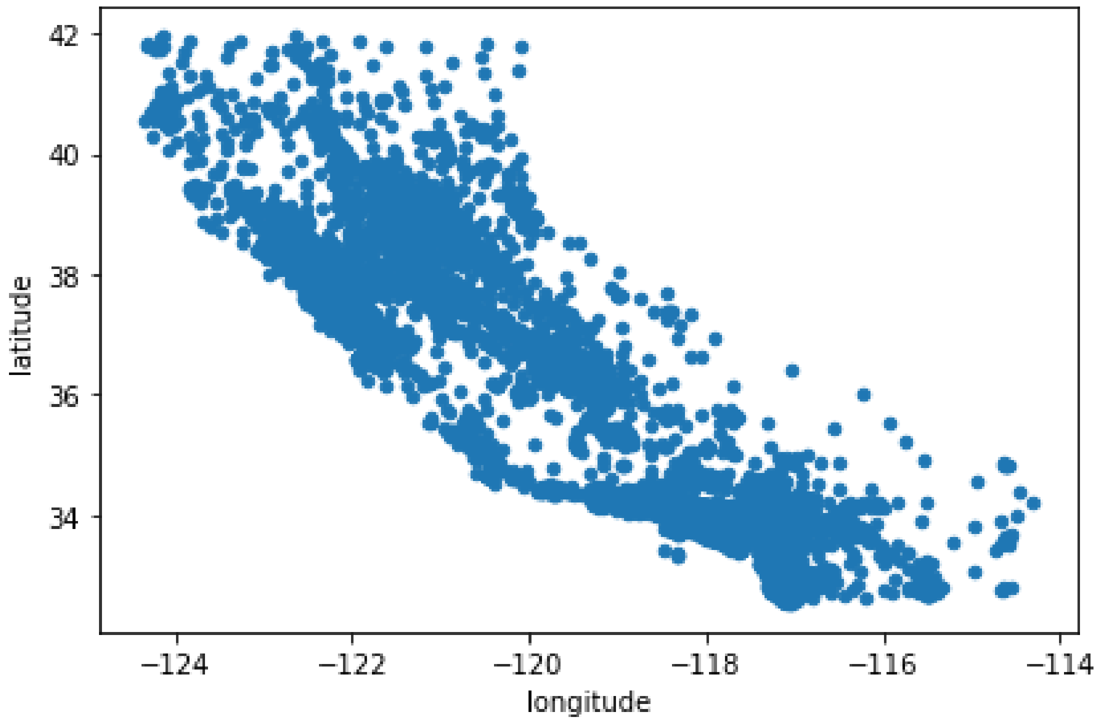
[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a24477d30>
```



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

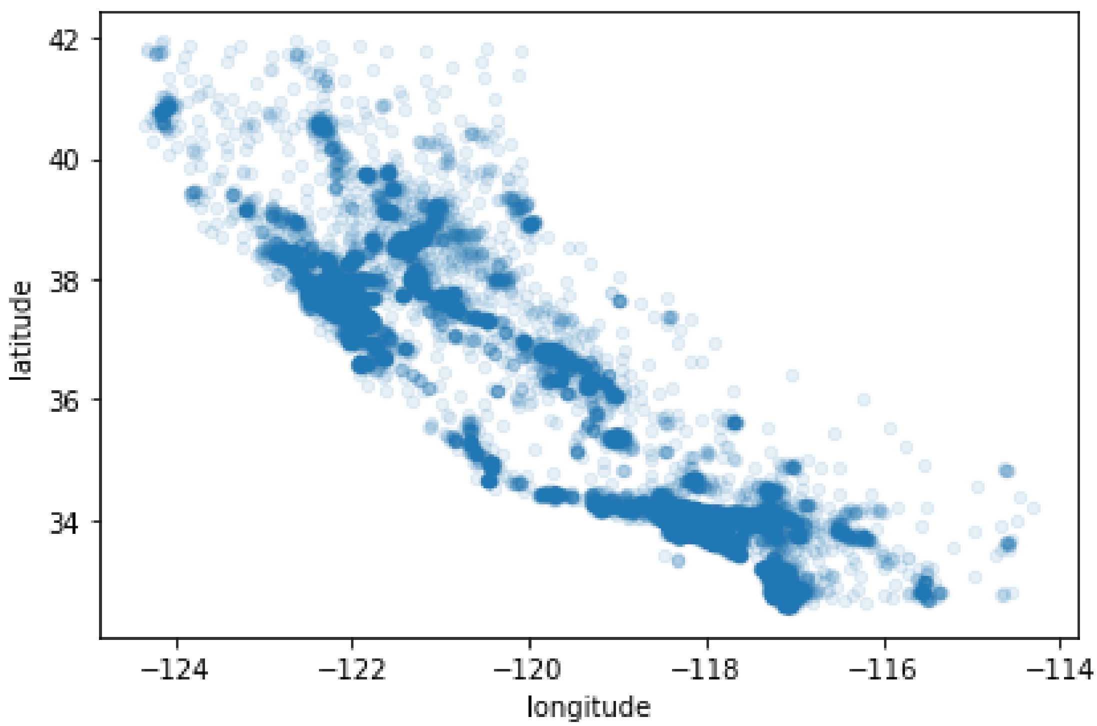
[15]:
```
## here's a not so interestting way plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

```
Saving figure bad_visualization_plot
```



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

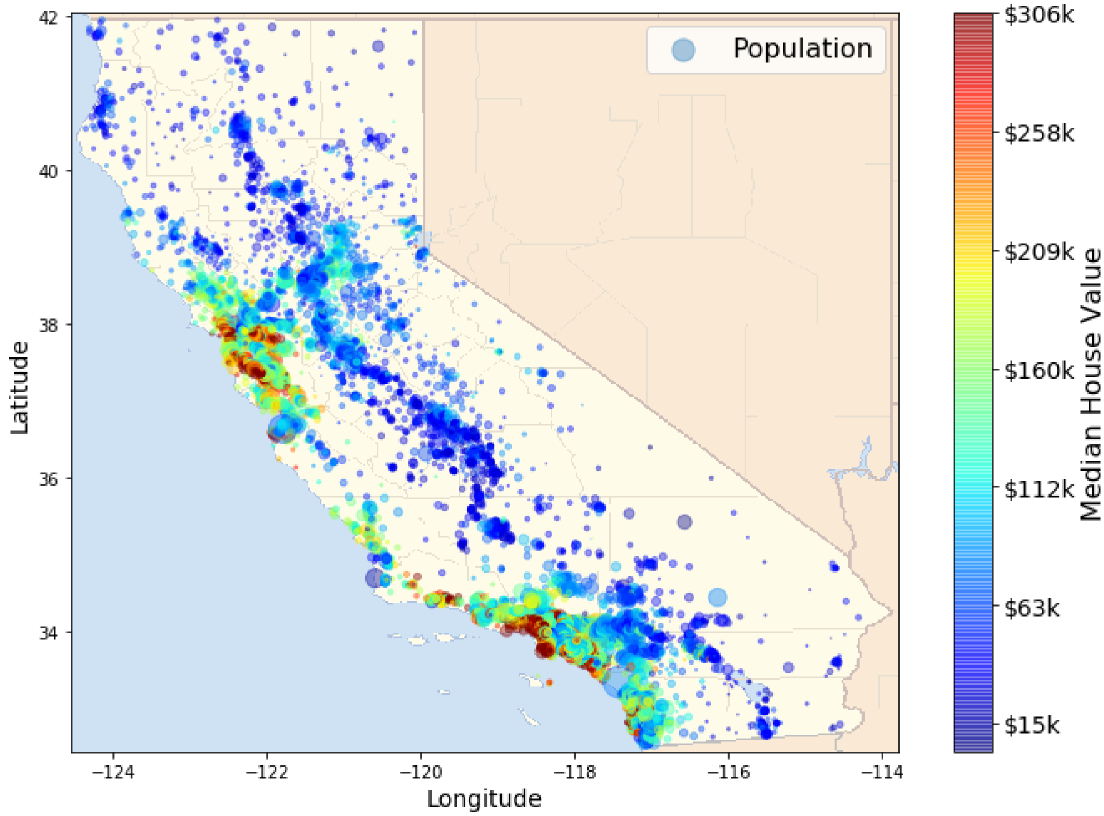
```
Saving figure better_visualization_plot
```



[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)
```

```python
# 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.

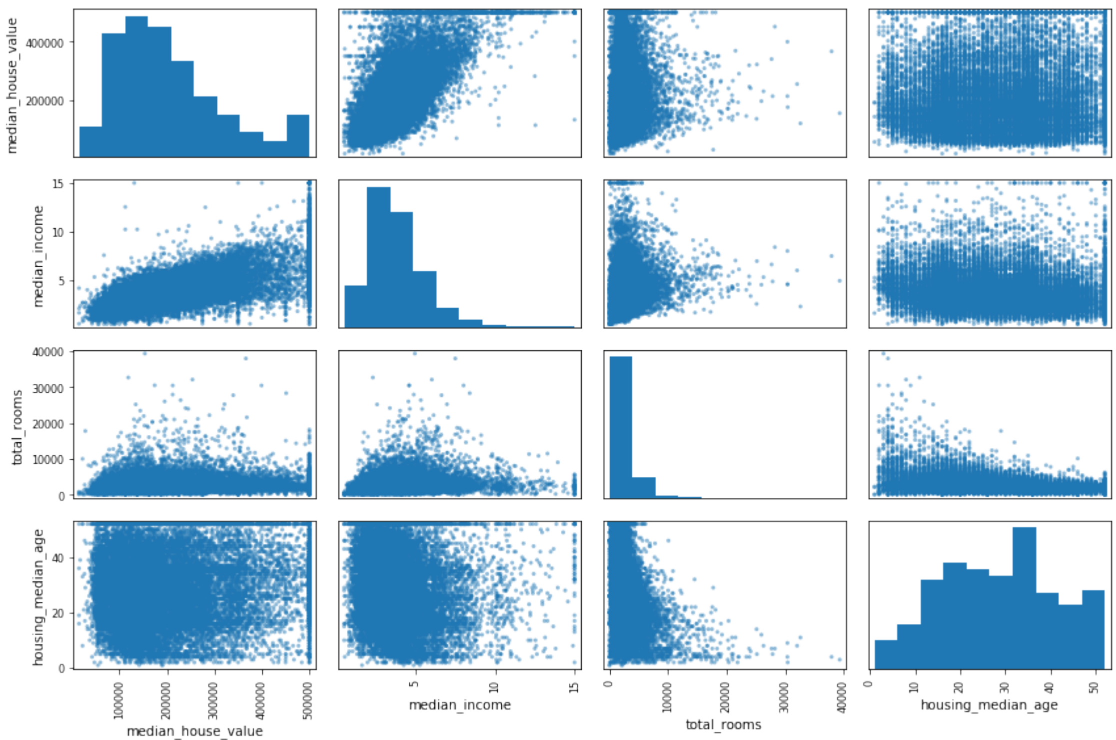
[18]:
```
corr_matrix = housing.corr()
```

[19]:
```
# for example if the target is "median_house_value", most correlated features␣
↪can be sorted
# which happens to be "median_income". This also intuitively makes sense.
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[19]:
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

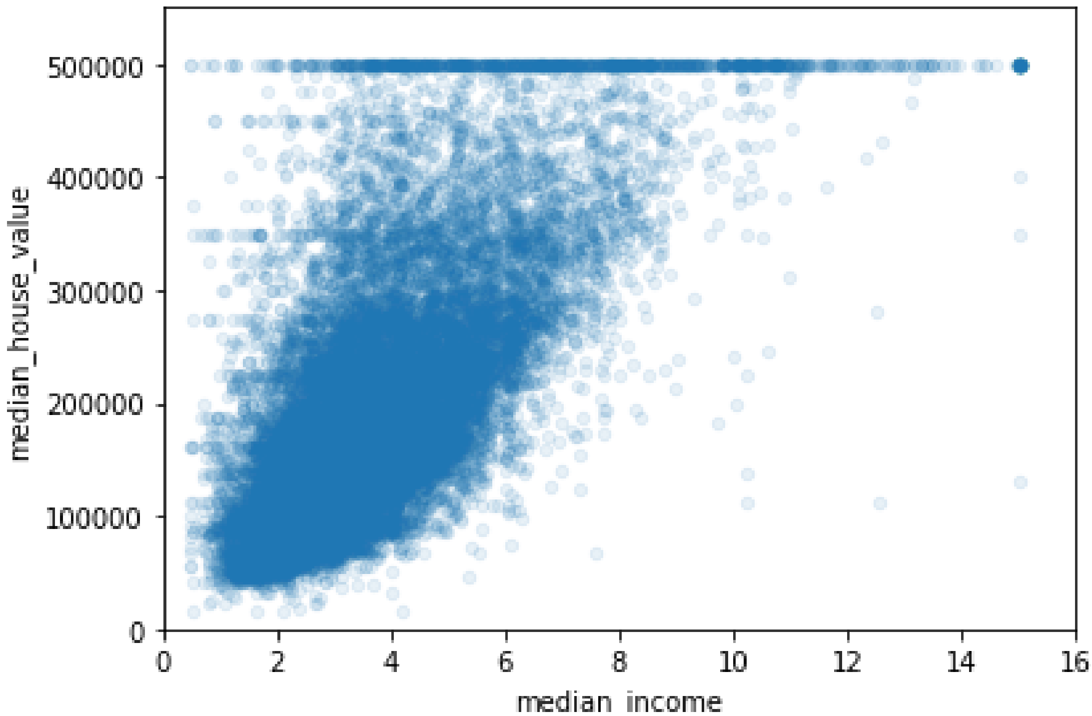
[20]:
```
# the correlation matrix for different attributes/features can also be plotted
# some features may show a positive correlation/negative correlation or
# it may turn out to be completely random!
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",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

```
Saving figure scatter_matrix_plot
```



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

```
Saving figure income_vs_house_value_scatterplot
```



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.

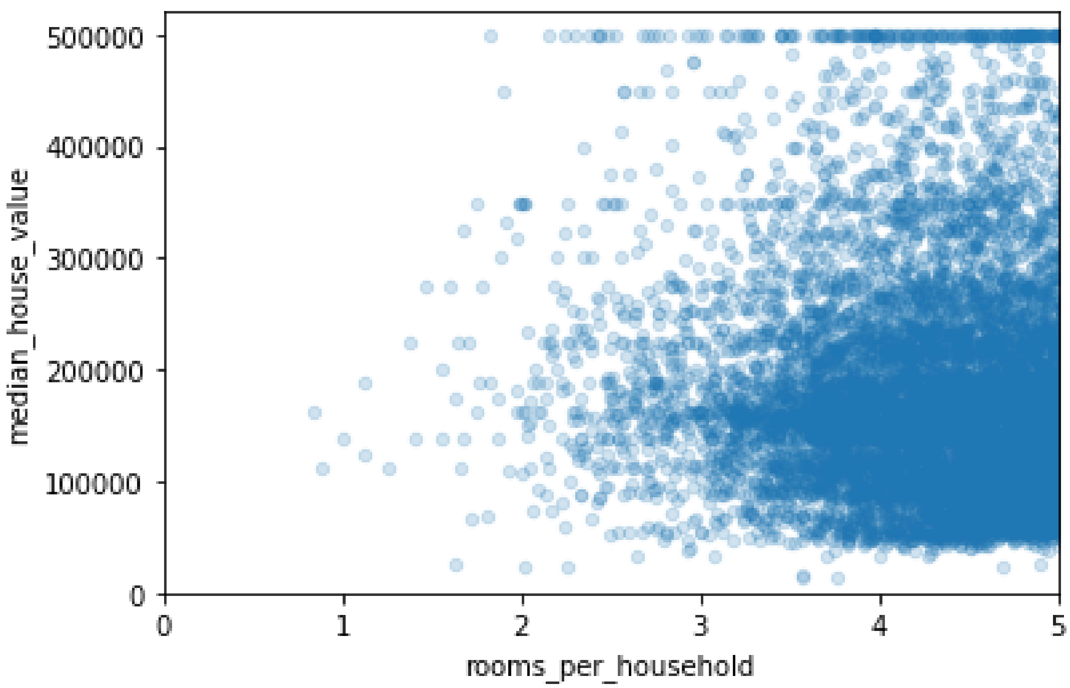
[22]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

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



[25]:
```python
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)
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]:
```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
housing_labels = train_set["median_house_value"].copy()
```

0.6.1 Dealing With Incomplete Data

[28]:
```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
```

[28]:
```
       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
```

[29]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1: simply␣
↪drop rows that have null values
```

[29]:
```
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: []
```

[30]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop␣
↪the complete feature
```

[30]:
```
       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
```

[31]:
```
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
```

[31]:
```
       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

[32]:
```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

#
```

```python
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.

[33]:
```
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.

[34]:
```
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
```

[34]: 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

[35]:
```python
DATASET_PATH = os.path.join("datasets", "airbnb")

def load_airbnb_data(airbnb_path):
    csv_path = os.path.join(airbnb_path, "AB_NYC_2019.csv")
    return pd.read_csv(csv_path)
airbnb = load_airbnb_data(DATASET_PATH)
airbnb = airbnb.drop(columns=['name','host_id','host_name','last_review'])
airbnb.head()
```

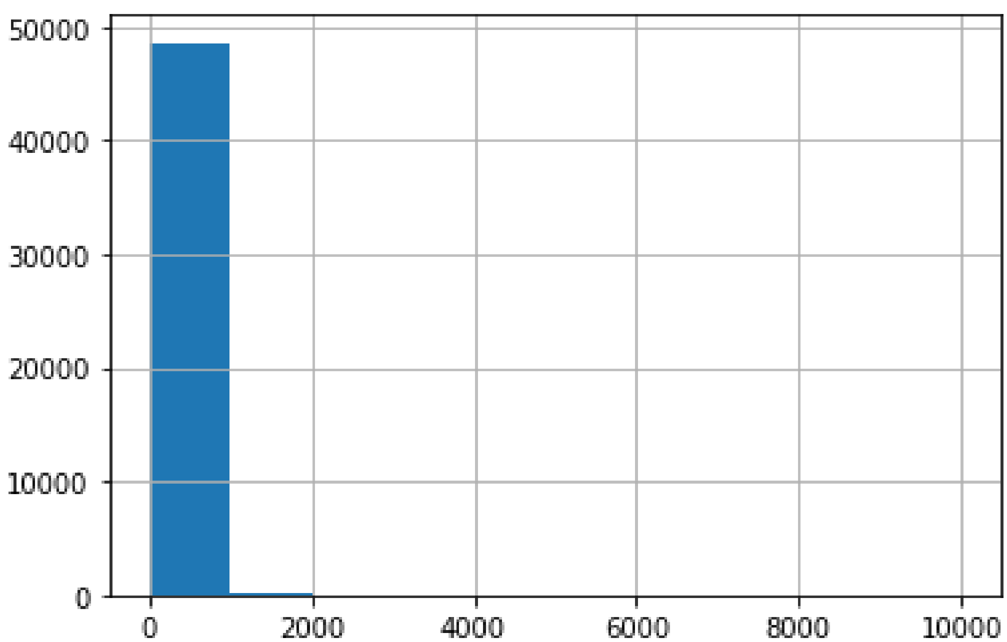
[35]:
```
     id neighbourhood_group neighbourhood  latitude  longitude  \
0  2539            Brooklyn    Kensington  40.64749  -73.97237
1  2595           Manhattan       Midtown  40.75362  -73.98377
2  3647           Manhattan        Harlem  40.80902  -73.94190
3  3831            Brooklyn  Clinton 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
```

[36]:
```python
airbnb.describe()
```

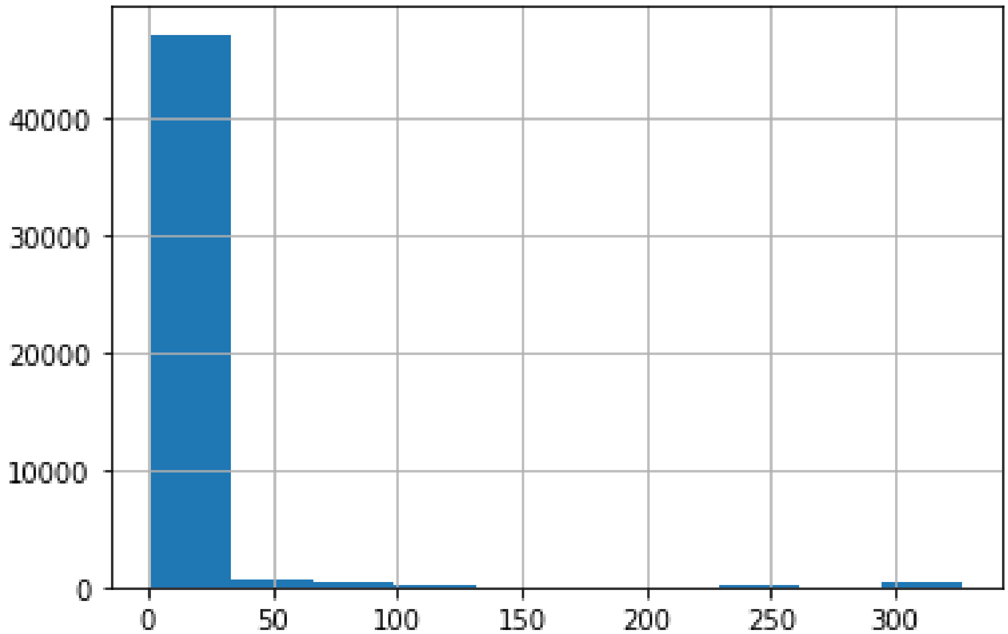
[36]:
```
                 id      latitude     longitude         price  minimum_nights  \
count  4.889500e+04  48895.000000  48895.000000  48895.000000    48895.000000
mean   1.901714e+07     40.728949    -73.952170    152.720687        7.029962
std    1.098311e+07      0.054530      0.046157    240.154170       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
```

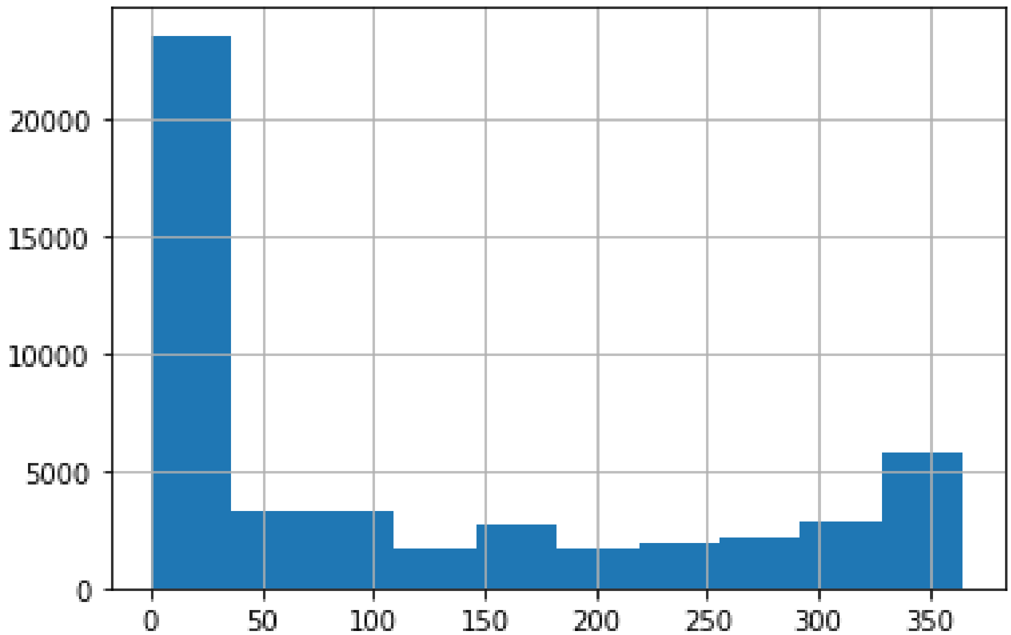
[37]:
```
airbnb["price"].hist()
plt.show()
```



[38]:
```
airbnb["calculated_host_listings_count"].hist()
plt.show()
```



[39]:
```
airbnb["availability_365"].hist()
plt.show()
```

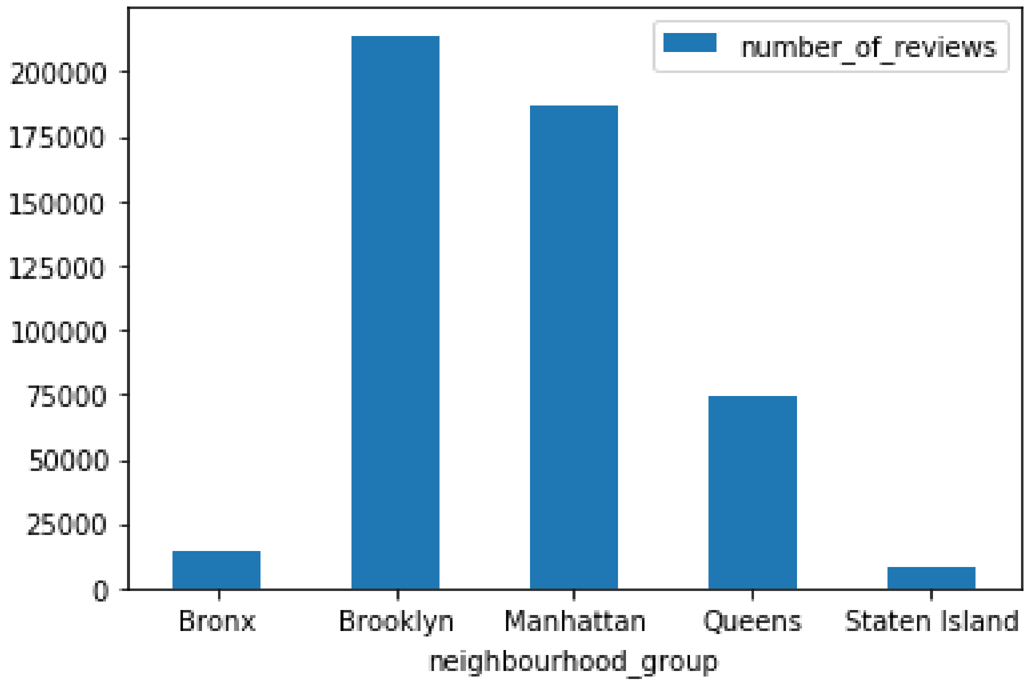


2.0.2 [5 pts] Plot total number_of_reviews per neighbourhood_group

[64]:
```
answer = airbnb_avail.groupby("neighbourhood_group",␣
↪as_index=False)["number_of_reviews"].sum()
answer
```

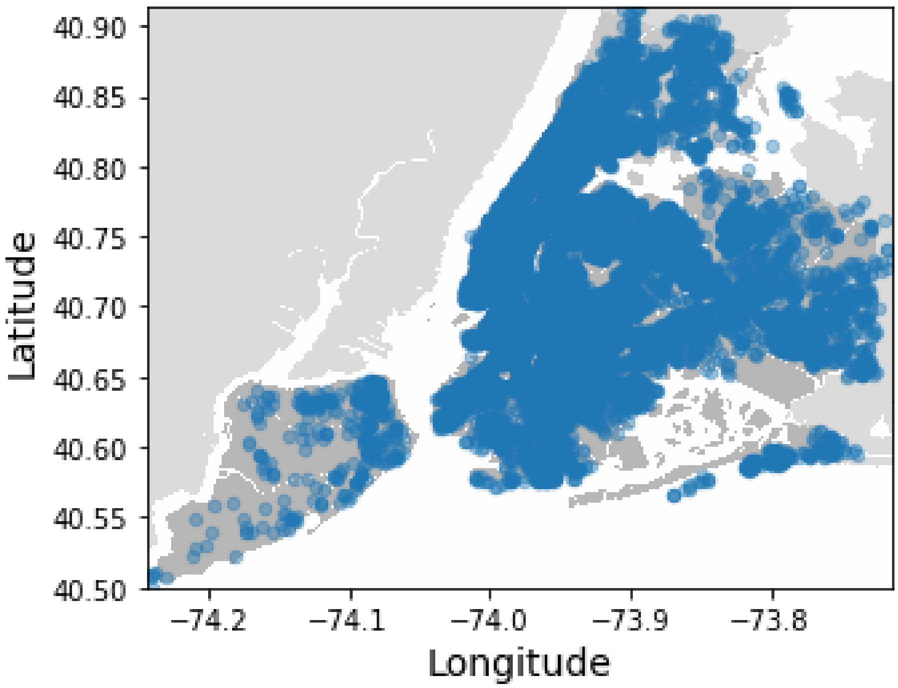
[64]:
```
  neighbourhood_group  number_of_reviews
0               Bronx              14080
1            Brooklyn             214091
2           Manhattan             186786
3              Queens              74420
4       Staten Island               7964
```

[66]:
```
answer_graph = answer.plot.bar(x='neighbourhood_group', y='number_of_reviews',␣
↪rot=0)
```



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 :)).

[40]:
```
images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename="newyorkcity.jpeg"
#import matplotlib.image as mpimg
newyork_img = mpimg.imread(os.path.join(images_path, filename))
ax = airbnb.plot(kind="scatter",x="longitude",y="latitude", alpha=0.4)
plt.imshow(newyork_img, extent=[-74.244420,-73.712990,40.499790,40.
↪913060],alpha=0.5)
plt.ylabel("Latitude",fontsize=14)
plt.xlabel("Longitude",fontsize=14)
plt.show()
```



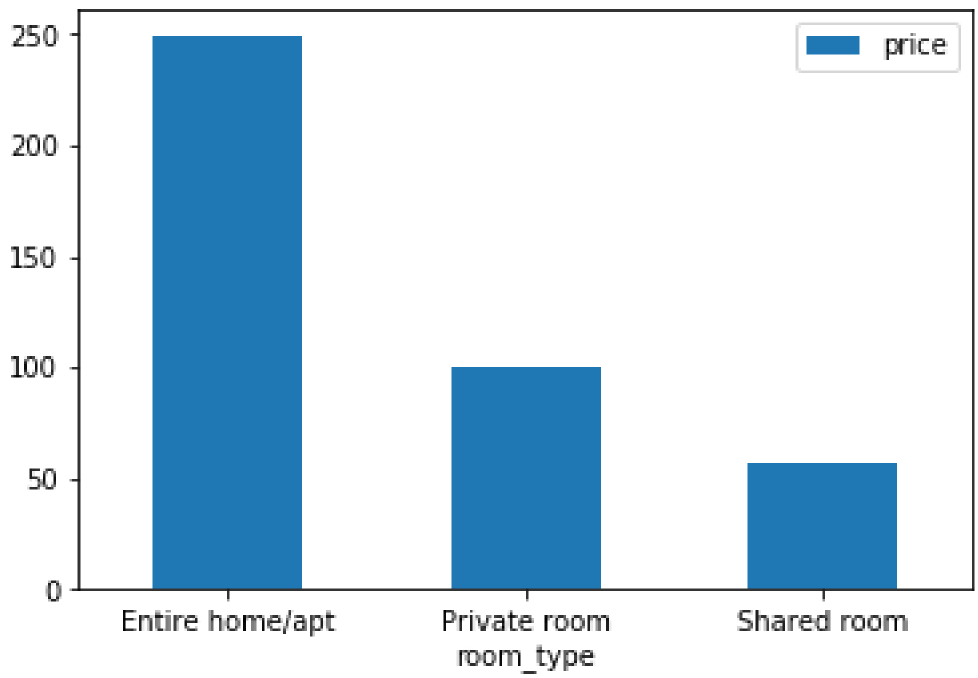
2.0.4 [5 pts] Plot average price of room types who have availability greater than 180 days.

[41]:
```
is_avail = airbnb["availability_365"] > 180
airbnb_avail = airbnb[is_avail]
airbnb_avail.head()
result = airbnb_avail.groupby("room_type", as_index=False)["price"].mean()
result
```

[41]:

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

[67]:
```
result_graph = result.plot.bar(x='room_type', y='price', rot=0)
```



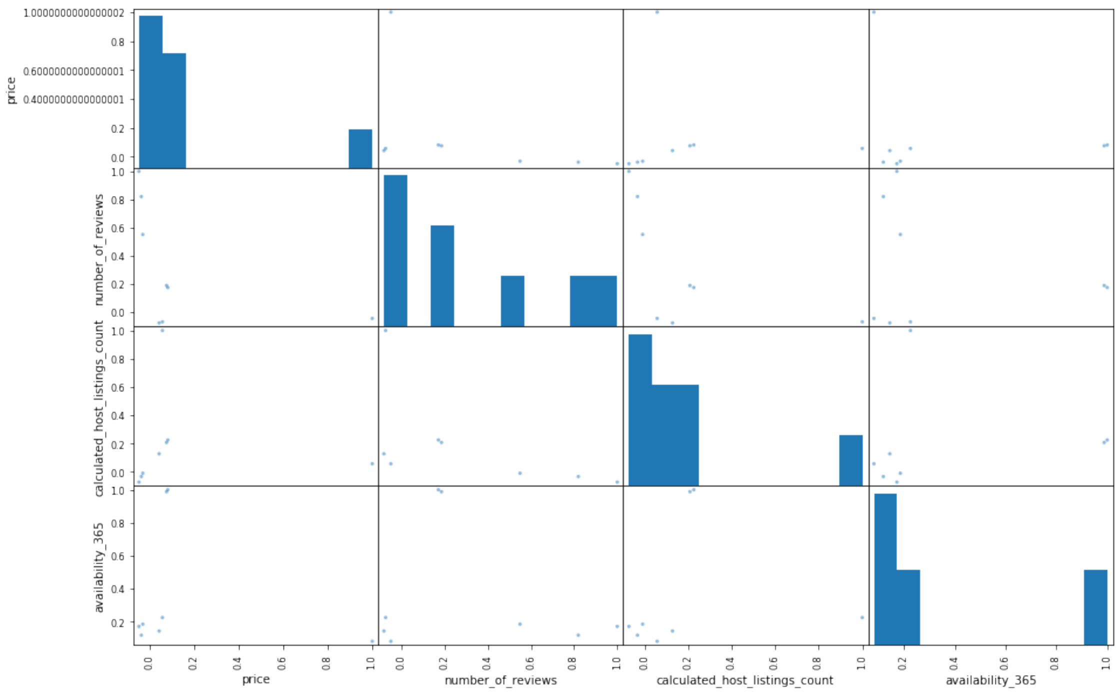
2.0.5 [5 pts] Plot correlation matrix

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

[63]:
```
airbnb_corr_matrix = airbnb.corr()
#from pandas.plotting import scatter_matrix
attributes=["price","number_of_reviews","calculated_host_listings_count","availability_365"]
scatter_matrix(airbnb_corr_matrix[attributes], figsize=(15,10))
```

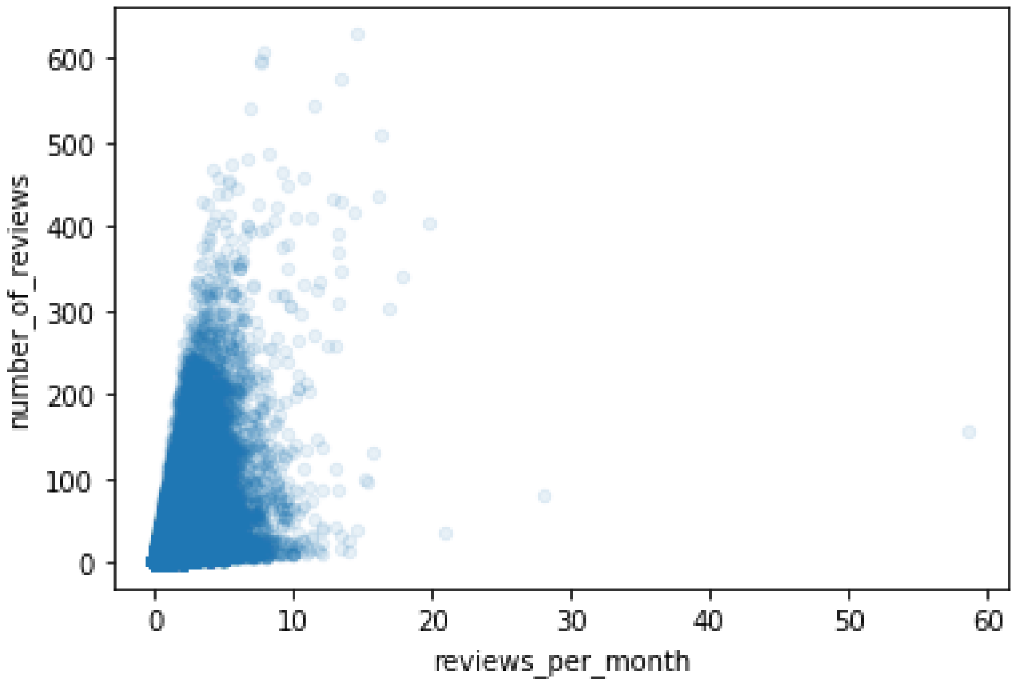
[63]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a2b7b5c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bb14198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bb45748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bb75cf8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a2bbb52e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bbe3898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bc16e48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bc54470>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a2bc544a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bcbbf98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bcf4588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bd27b38>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a2bd67128>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bd966d8>,
```

```
  <matplotlib.axes._subplots.AxesSubplot object at 0x1a2bdc8c88>,
  <matplotlib.axes._subplots.AxesSubplot object at 0x1a2be06278>]],
dtype=object)
```



[44]:
```python
airbnb.plot(kind="scatter", x="reviews_per_month", y="number_of_reviews",
            alpha=0.1)
```

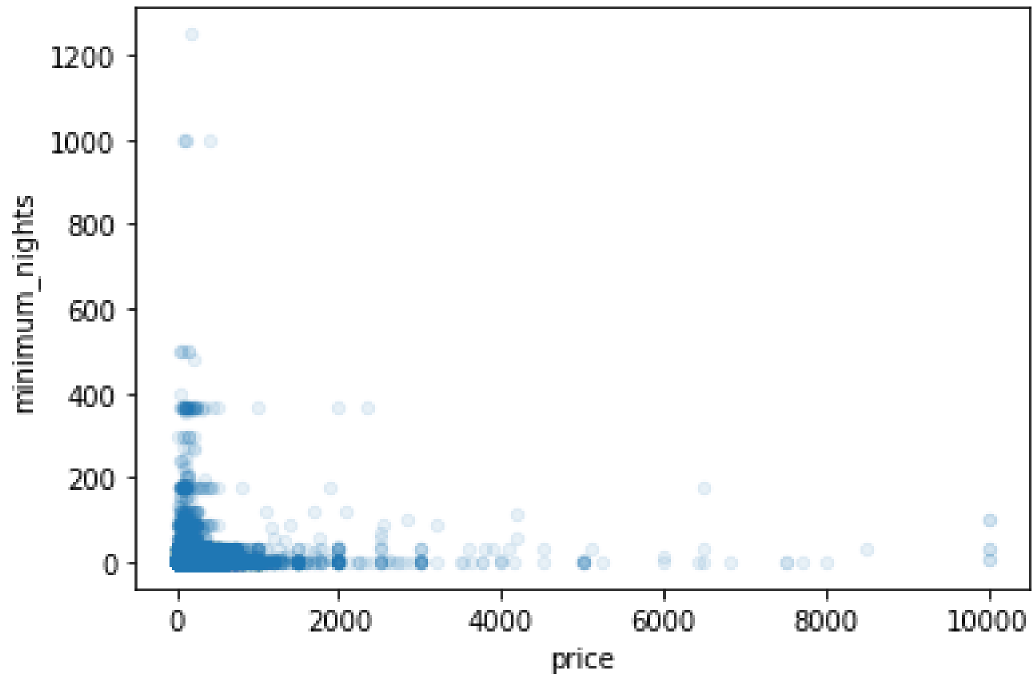
[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a22455278>
```



reviews__per__month and number__of__reviews: positive correlation

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

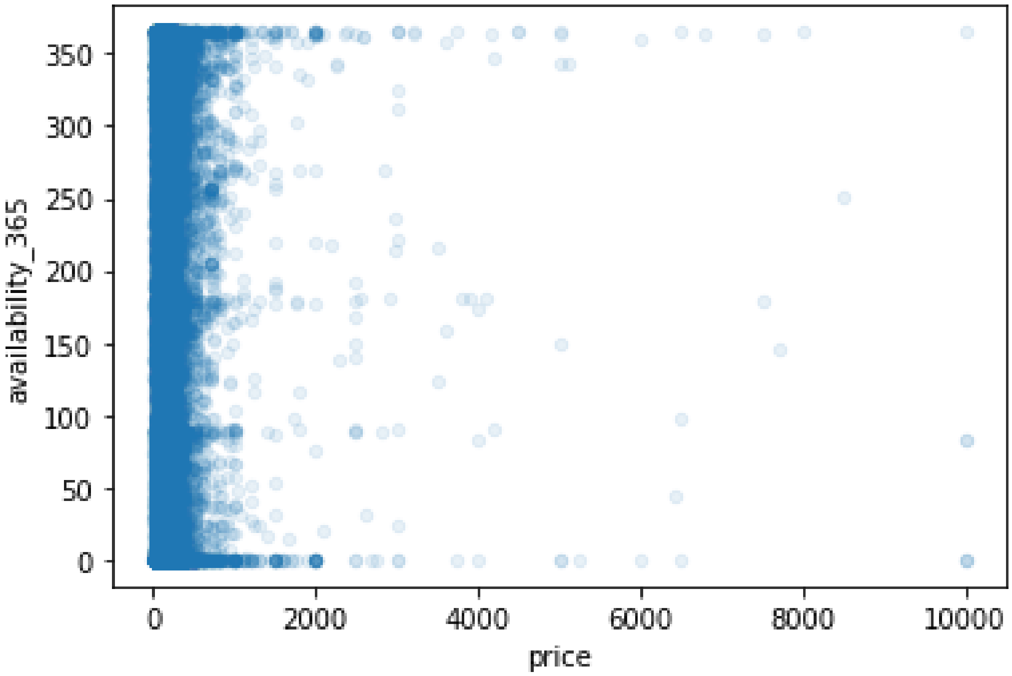
[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2dcc8a58>



price and minimum_nights: positive correlation

[48]:
```
airbnb.plot(kind="scatter", x="price", y="availability_365",
             alpha=0.1)
```

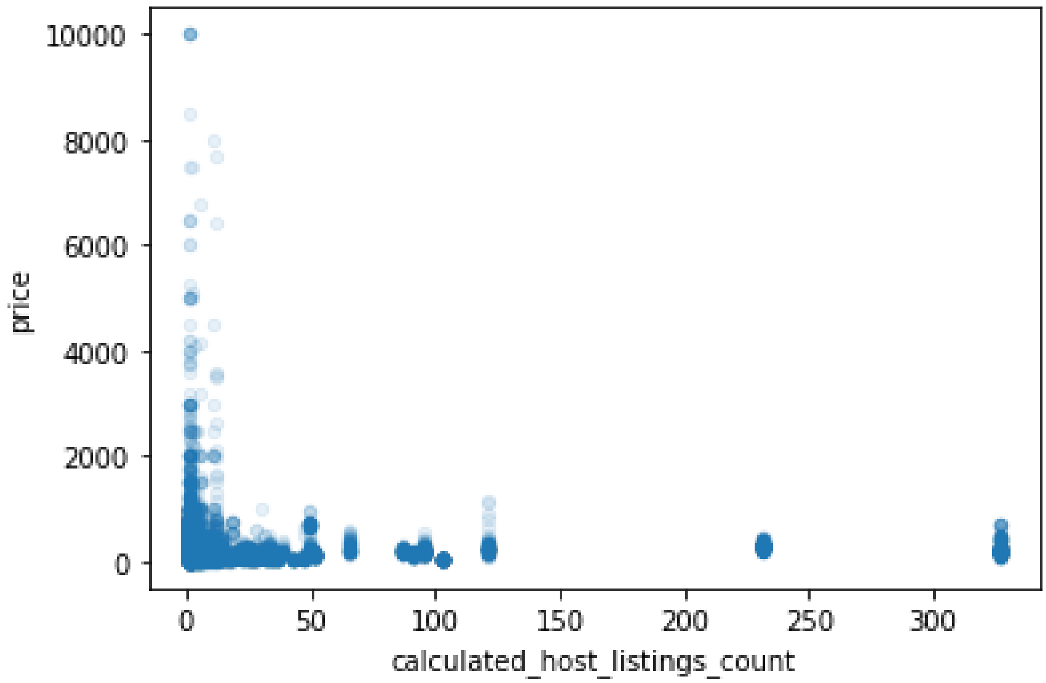
[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a259b22b0>
```



price and availability_365: no correlation

[49]:
```
airbnb.plot(kind="scatter", x="calculated_host_listings_count", y="price",
             alpha=0.1)
```

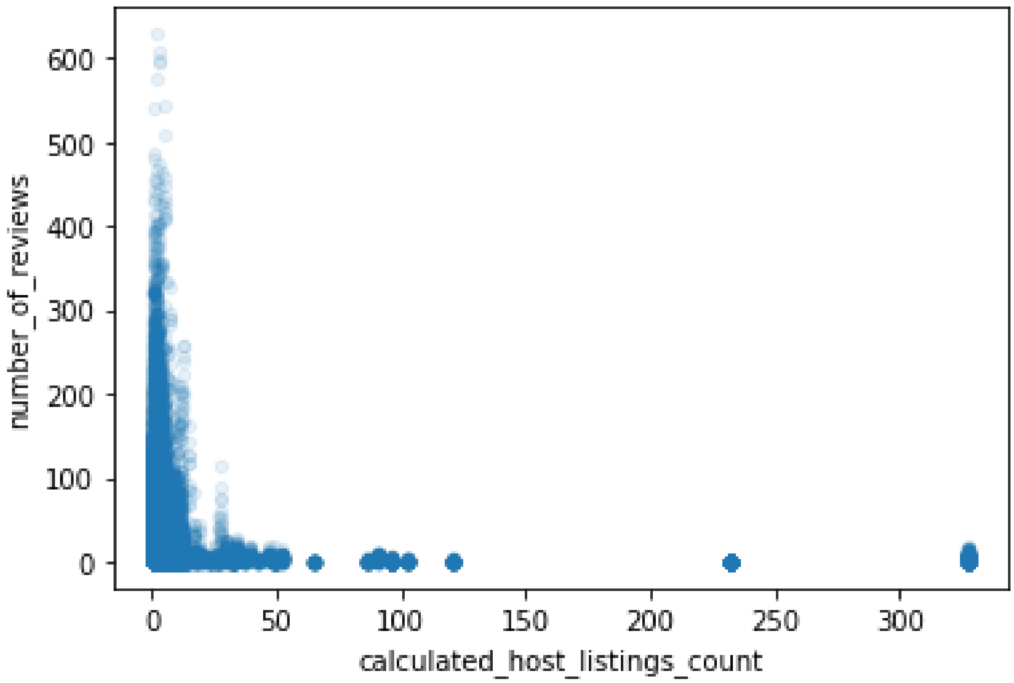
[49]: <matplotlib.axes._subplots.AxesSubplot at 0x1a26077390>



calculated_host_listings_count and price: positive correlation

[50]:
```
airbnb.plot(kind="scatter", x="calculated_host_listings_count",␣
↪y="number_of_reviews",
             alpha=0.1)
```

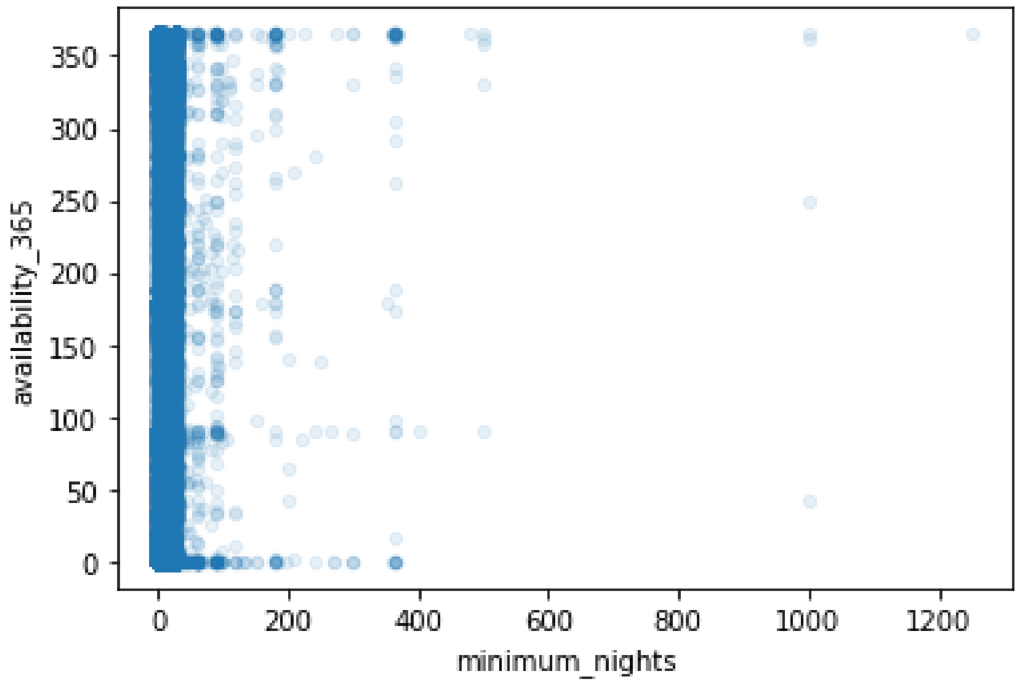
[50]: <matplotlib.axes._subplots.AxesSubplot at 0x1a26292f60>



calculated_host_listings_count and number_of_reviews: positive correlation

[51]:
```
airbnb.plot(kind="scatter", x="minimum_nights", y="availability_365",
             alpha=0.1)
```

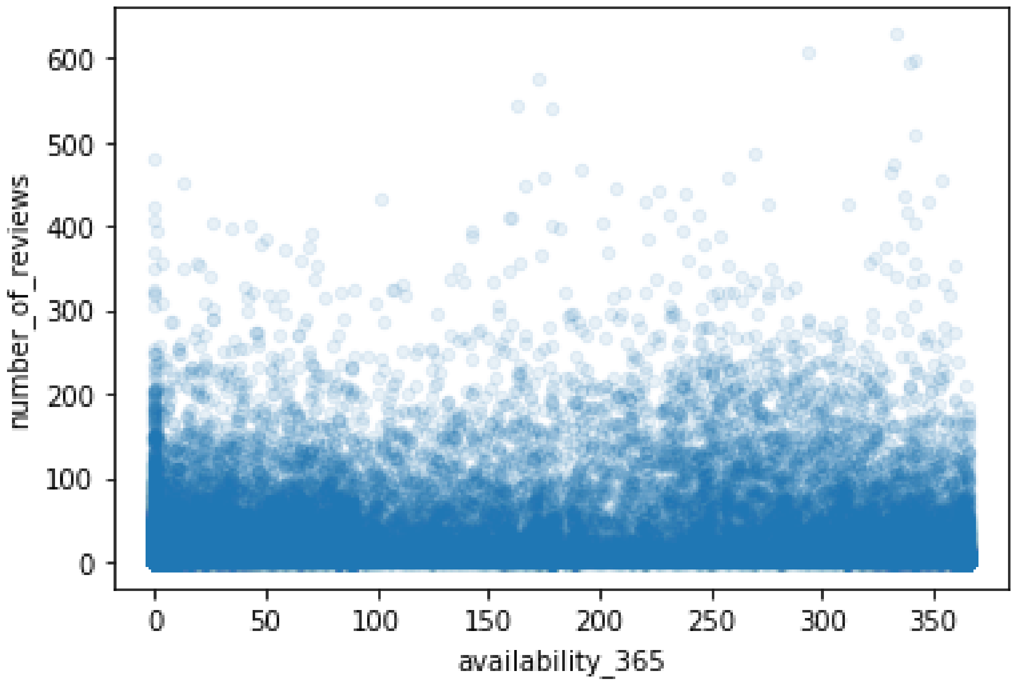
[51]: <matplotlib.axes._subplots.AxesSubplot at 0x1a26c5ddd8>

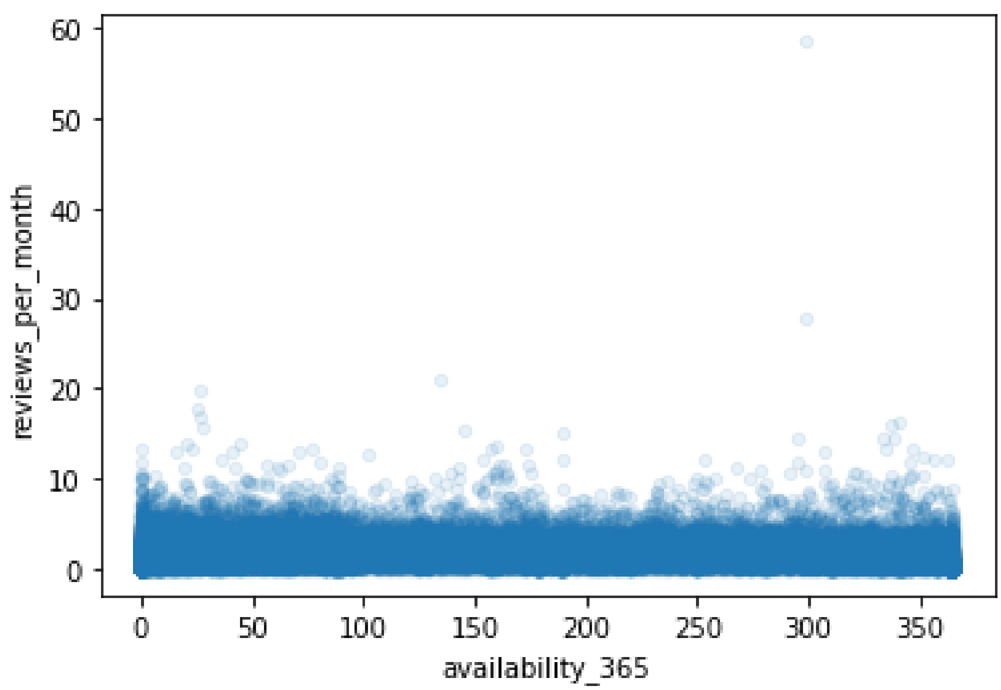


minimum_nights and availability_365: unclear correlation

[52]:
```
airbnb.plot(kind="scatter", x="availability_365", y="number_of_reviews",
            alpha=0.1)
airbnb.plot(kind="scatter", x="availability_365", y="reviews_per_month",
            alpha=0.1)
```

[52]: <matplotlib.axes._subplots.AxesSubplot at 0x1a29f380b8>





availability_365 and number_of_reviews: unclear correlation

availability_365 and reviews_per_month: possibly negative correlation

Reviews per month and the number of reviews appears to have a positive correlation, which makes sense given that a higher number of reviews per month would end up increasing the total number of reviews. Other pairs of features such as minimum nights and price, host listings count and price, and host listings count and number of reviews look to only have slight positive correlation. It can be hard to tell because oftentimes the data will include a density of points that look like a straight line.

There don't appear to be any features that have an overt negative correlation. A case could be made for availability and the number of reviews per month, where a higher number of available days theoretically should correlate to a low number of reviews per month. After all, a higher number of available days means people aren't reserving the listing as much, so a lower number of guests should result in a low number of reviews per month. Given there are other factors that impact the number of reviews per month, like customer dissatisfaction or laziness, the correlation between availability and number of reviews per month is very hard to see on the graph.

3 [25 pts] Prepare the Data

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

[53]:
```
airbnb["reviews_squared_per_month"] = airbnb["number_of_reviews"] *␣
↪airbnb["reviews_per_month"]
airbnb["nights_available_one_stay"] = airbnb["availability_365"] -␣
↪airbnb["minimum_nights"]
```

3.0.2 [5 pts] Set aside 20% of the data as test test (80% train, 20% test).

[54]:
```
from sklearn.model_selection import train_test_split
airbnb = airbnb.drop(columns=['id','neighbourhood','latitude','longitude'])
airbnb_x = airbnb.drop(columns=["price"])
airbnb_y = airbnb["price"]
x_train, x_test, y_train, y_test = train_test_split(airbnb_x, airbnb_y,␣
↪test_size=0.2, random_state=42)
```

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

[55]:
```
incomplete_rows = airbnb[airbnb.isnull().any(axis=1)].head()
incomplete_rows
```

[55]:
```
   neighbourhood_group        room_type  price  minimum_nights  \
2            Manhattan     Private room    150               3
19           Manhattan  Entire home/apt    190               7
```

```
26          Manhattan     Private room     80                4
36           Brooklyn     Private room     35               60
38           Brooklyn     Private room    150                1

    number_of_reviews  reviews_per_month  calculated_host_listings_count  \
2                   0                NaN                               1
19                  0                NaN                               2
26                  0                NaN                               1
36                  0                NaN                               1
38                  0                NaN                               1

    availability_365  reviews_squared_per_month  nights_available_one_stay
2                365                        NaN                        362
19               249                        NaN                        242
26                 0                        NaN                         -4
36               365                        NaN                        305
38               365                        NaN                        364
```

[56]:

```python
incomplete_rows["reviews_per_month"].fillna(0, inplace=True)
incomplete_rows.head()
```

[56]:

```
   neighbourhood_group        room_type  price  minimum_nights  \
2            Manhattan     Private room    150               3
19           Manhattan  Entire home/apt    190               7
26           Manhattan     Private room     80               4
36            Brooklyn     Private room     35              60
38            Brooklyn     Private room    150               1

    number_of_reviews  reviews_per_month  calculated_host_listings_count  \
2                   0                0.0                               1
19                  0                0.0                               2
26                  0                0.0                               1
36                  0                0.0                               1
38                  0                0.0                               1

    availability_365  reviews_squared_per_month  nights_available_one_stay
2                365                        NaN                        362
19               249                        NaN                        242
26                 0                        NaN                         -4
36               365                        NaN                        305
38               365                        NaN                        364
```

I chose to replace the null values with 0. I noticed that the null values occured in the 'reviews_per_month' column and inferred that it was because the value in the 'number_of_reviews' column was 0 and whatever calculation outputted the number of reviews per month would output a null value. When the number of reviews was 0, I thought it was reasonable to set the number of reviews per month to 0 as well instead of a null value.

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

[57]:
```
airbnb_num = x_train.drop(["neighbourhood_group", "room_type"], axis=1)
airbnb_num.head()
nights_ix, num_reviews_ix, reviewspm_ix, avail_ix = 1, 2, 3, 5
```

[58]:
```
airbnb_imputer = SimpleImputer(strategy="constant", fill_value=0)

class Augment_Features1(BaseEstimator, TransformerMixin):
    def __init__(self, add_reviews_squared_per_month = True):
        self.add_reviews_squared_per_month = add_reviews_squared_per_month
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        nights_available_one_stay = X[:, avail_ix] - X[:, nights_ix]
        if self.add_reviews_squared_per_month:
            reviews_squared_per_month = X[:, num_reviews_ix] * X[:,␣
↪reviewspm_ix]
            return np.c_[X, nights_available_one_stay,␣
↪reviews_squared_per_month]
        else:
            return np.c_[X, nights_available_one_stay]
#airbnb_attr_adder = Augment_Features1()
#airbnb_extra_attribs = airbnb_attr_adder.transform(x_train.values)

airbnb_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="constant", fill_value=1)),
    ('attribs_adder', Augment_Features1()),
    ('std_scaler', StandardScaler()),
])
airbnb_num_tr = airbnb_pipeline.fit_transform(airbnb_num)
num_features = list(airbnb_num)
cat_features = ["neighbourhood_group", "room_type"]

a_full_pipeline = ColumnTransformer([
    ("num", airbnb_pipeline, num_features),
    ("cat", OneHotEncoder(), cat_features)
])
airbnb_prepared = a_full_pipeline.fit_transform(x_train)
```

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.

[59]:
```
a_lin_reg = LinearRegression()
a_lin_reg.fit(airbnb_prepared, y_train)

a_data = x_test.iloc[:5]
a_labels = y_train.iloc[:5]
a_data_prepared = a_full_pipeline.transform(a_data)

print("Predictions:", a_lin_reg.predict(a_data_prepared))
print("Actual labels:", list(a_labels))
```

```
Predictions: [182.95703125  56.34765625 112.37890625 266.76953125 223.06640625]
Actual labels: [295, 70, 58, 75, 38]
```

[60]:
```
a_preds = a_lin_reg.predict(airbnb_prepared)
a_mse = mean_squared_error(y_train, a_preds)
a_rmse = np.sqrt(a_mse)
a_rmse
```

[60]: 235.7809852797133

[ ]:
```
```