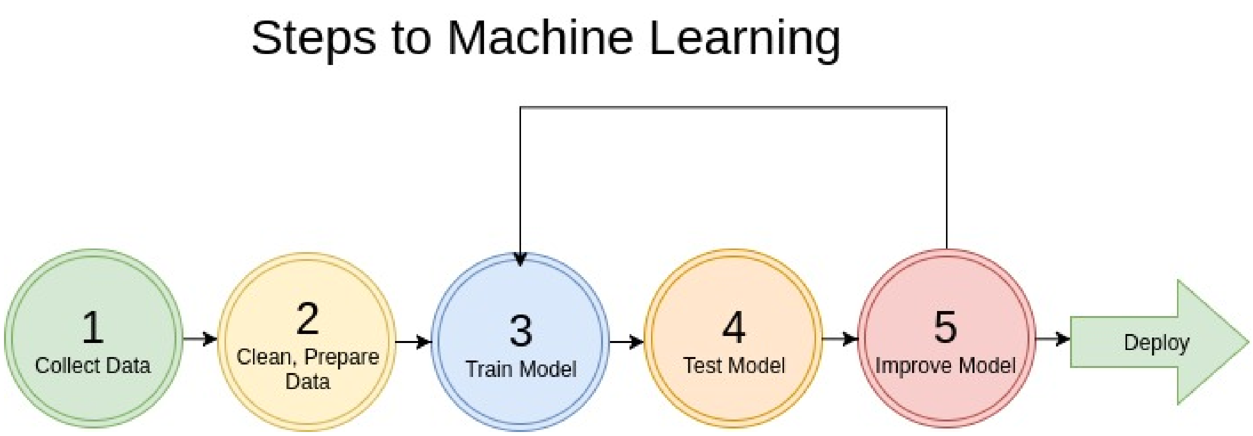
# Introduction

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




# 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 (http://archive.ics.uci.edu/ml/)](http://archive.ics.uci.edu/ml/)
- [Kaggle Datasets (kaggle.com)](kaggle.com)
- [AWS Datasets (https://registry.opendata.aws)](https://registry.opendata.aws)

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

# Setup

In [514]:

```
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=3
    '''
        plt.savefig wrapper. refer to
        https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.savefig.
    '''
    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)
```

In [515]:

```
import os
import tarfile
import urllib
DATASET_PATH = os.path.join("datasets", "housing")
```

# 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 (https://pandas.pydata.org):** is a fast, flexibile and expressive data structure widely used for tabular and multidimensional datasets.
- **Matplotlib (https://matplotlib.org)**: 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 (https://seaborn.pydata.org), ggplot2 (https://ggplot2.tidyverse.org)

In [516]:
```
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)
```

In [517]:
```
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
```

Out[517]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | m |
|---|---|---|---|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322.0 | 126.0 | |
| **1** | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401.0 | 1138.0 | |
| **2** | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496.0 | 177.0 | |
| **3** | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558.0 | 219.0 | |
| **4** | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565.0 | 259.0 | |

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.

In [518]:
```
# 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
```

In [519]:
```
# you can access individual columns similarly
# to accessing elements in a python dict
housing["ocean_proximity"].head() # added head() to avoid printing many col
```

Out[519]:
```
0    NEAR BAY
1    NEAR BAY
2    NEAR BAY
3    NEAR BAY
4    NEAR BAY
Name: ocean_proximity, dtype: object
```

In [520]:
```
# to access a particular row we can use iloc
housing.iloc[1]
```

Out[520]:
```
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
```

In [521]:
```
# one other function that might be useful is
# value_counts(), which counts the number of occurences
# for categorical features
housing["ocean_proximity"].value_counts()
```

Out[521]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [522]:
```
# The describe function compiles your typical statistics for each
# column
housing.describe()
```

Out[522]:

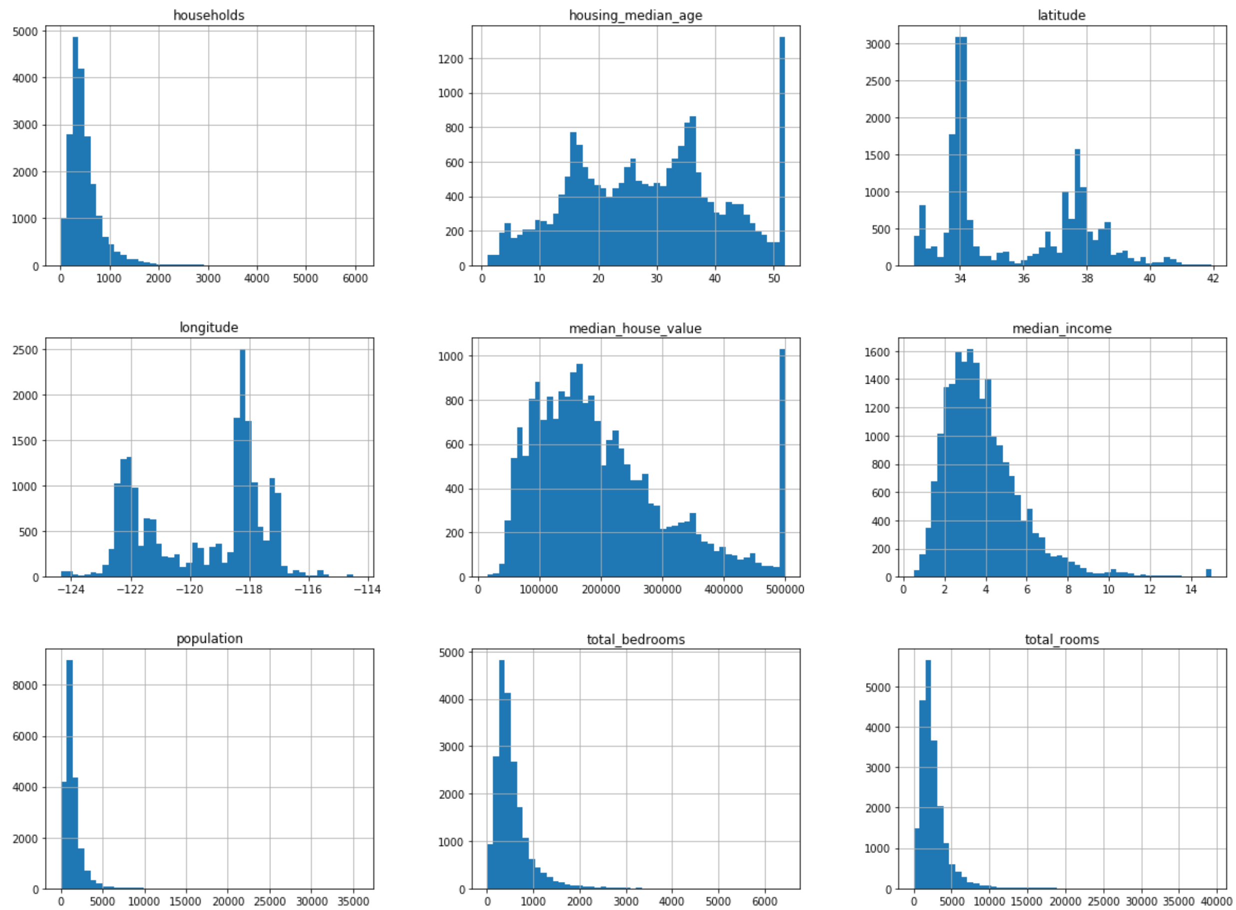
| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | populatio |
|---|---|---|---|---|---|---|
| **count** | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20433.000000 | 20640.00000 |
| **mean** | -119.569704 | 35.631861 | 28.639486 | 2635.763081 | 537.870553 | 1425.47674 |
| **std** | 2.003532 | 2.135952 | 12.585558 | 2181.615252 | 421.385070 | 1132.46212 |
| **min** | -124.350000 | 32.540000 | 1.000000 | 2.000000 | 1.000000 | 3.00000 |
| **25%** | -121.800000 | 33.930000 | 18.000000 | 1447.750000 | 296.000000 | 787.00000 |
| **50%** | -118.490000 | 34.260000 | 29.000000 | 2127.000000 | 435.000000 | 1166.00000 |
| **75%** | -118.010000 | 37.710000 | 37.000000 | 3148.000000 | 647.000000 | 1725.00000 |
| **max** | -114.310000 | 41.950000 | 52.000000 | 39320.000000 | 6445.000000 | 35682.00000 |

**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 (https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html)](https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html)**

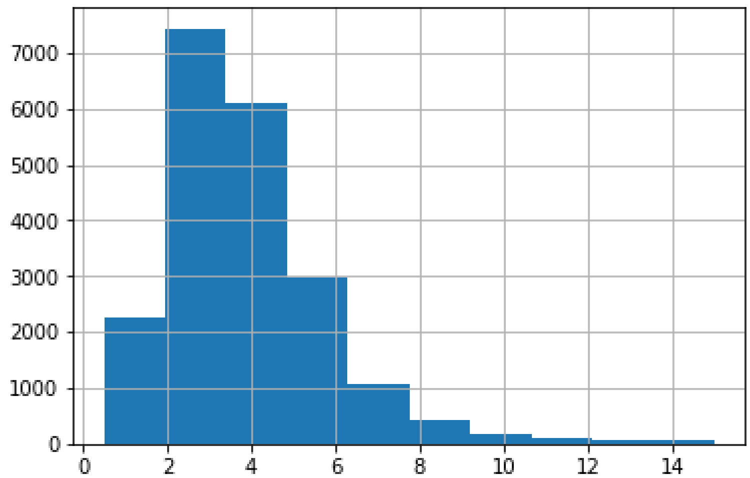
## Let's start visualizing the dataset

In [523]:

```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 figu
           # the show() function must be called
```



In [524]:
```
# 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.

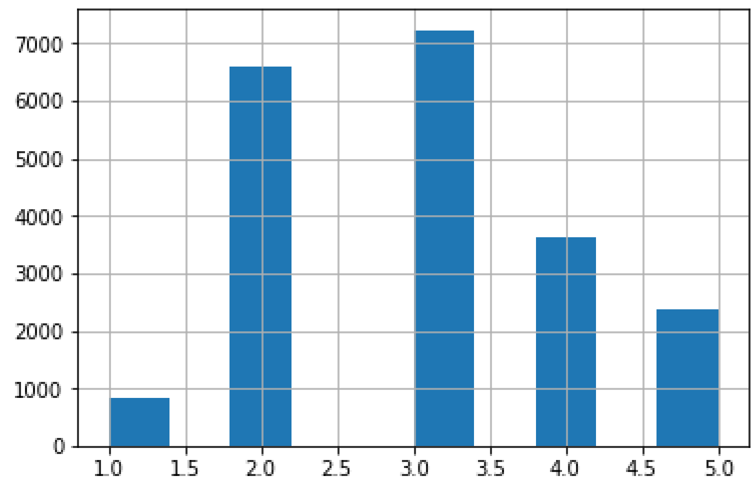
For example, to bin the households based on median_income we can use the pd.cut function

In [525]:
```
# 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()
```

Out[525]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

In [526]:
```python
housing["income_cat"].hist()
```

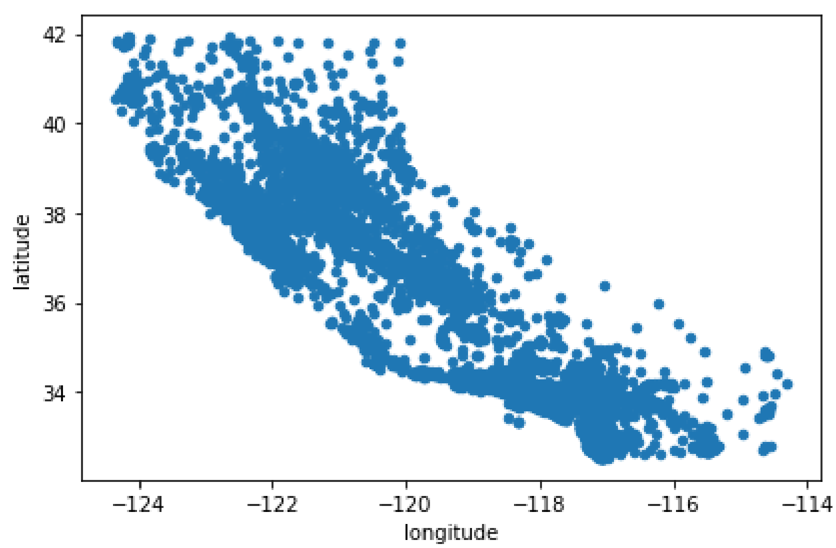
Out[526]: <matplotlib.axes._subplots.AxesSubplot at 0x7f94f46c8278>




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

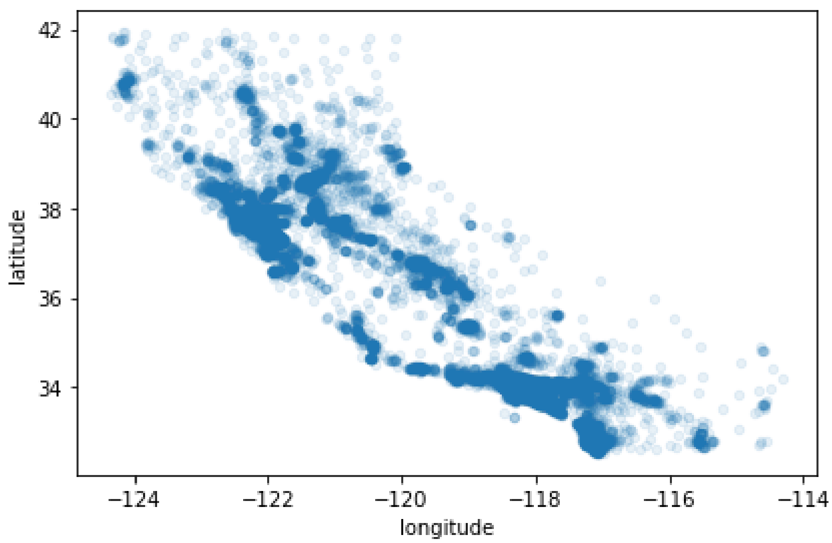
In [527]:
```python
## here's a not so interestting way plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

Saving figure bad_visualization_plot



In [528]:
```
# 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")
```

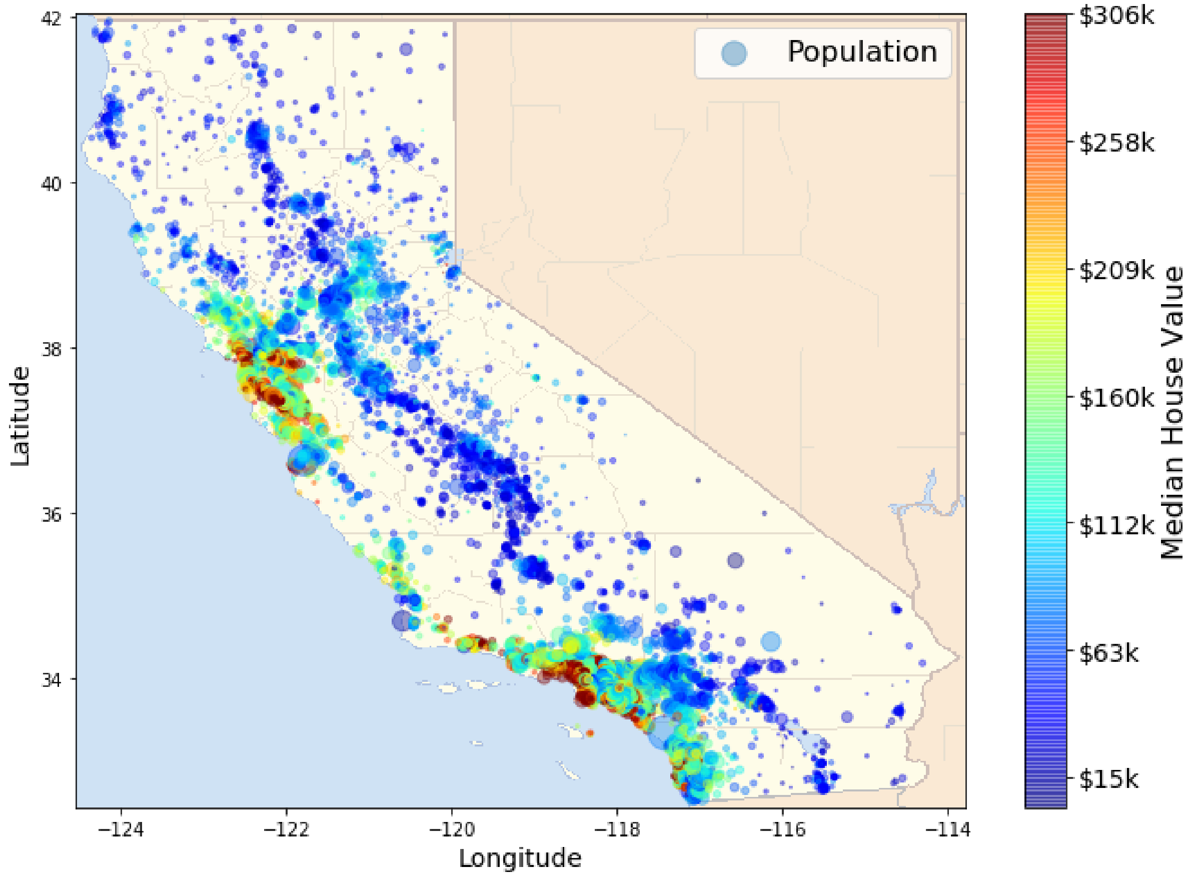
```
Saving figure better_visualization_plot
```



In [529]:

```
# 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,
                       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
           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], fontsi
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.

In [530]:
```
corr_matrix = housing.corr()
```

In [531]:

```python
# for example if the target is "median_house_value", most correlated featur
# which happens to be "median_income". This also intuitively makes sense.
corr_matrix["median_house_value"].sort_values(ascending=False)
```

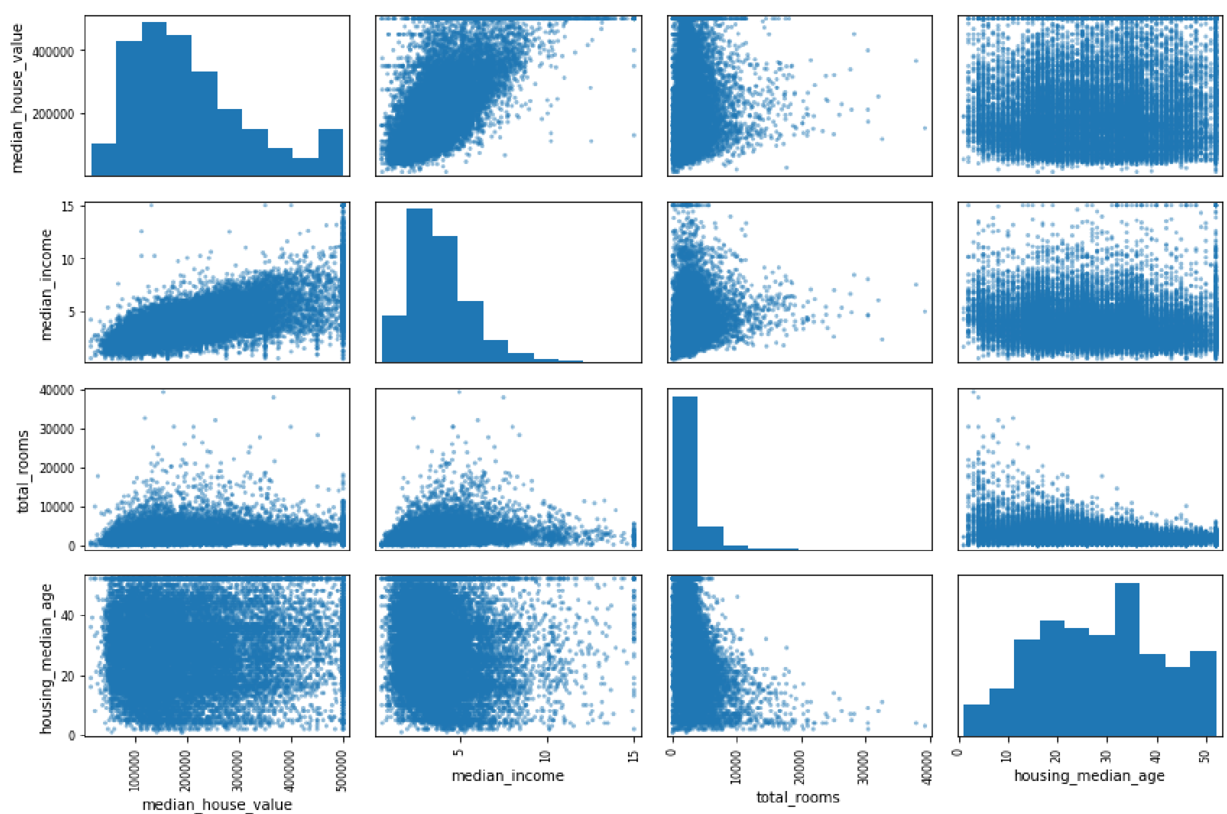
Out[531]:

```
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
```

In [532]:

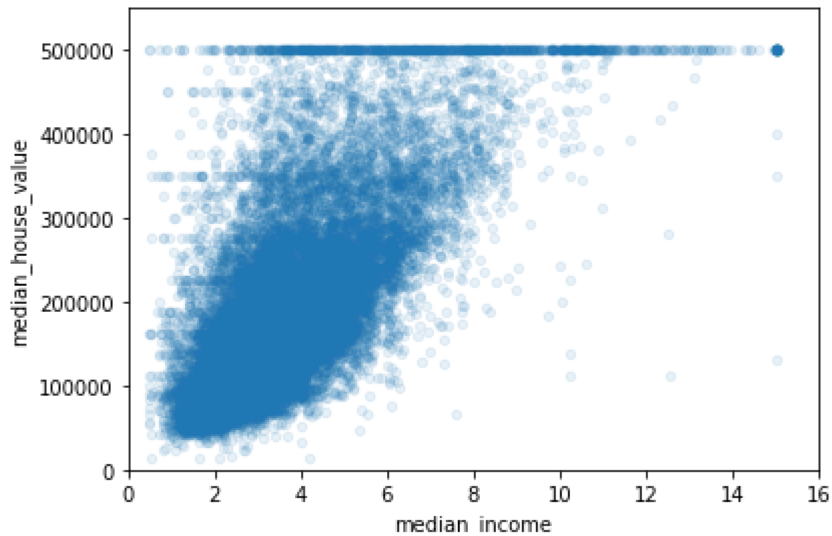
```python
# the correlation matrix for different attributes/features can also be plot
# some features may show a positive correlation/negative correlation or
# it may turn out to be completely random!
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",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

```
Saving figure scatter_matrix_plot
```



In [533]:
```
# median income vs median house vlue plot plot 2 in the first row of top fi
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
```




## Augmenting Features

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

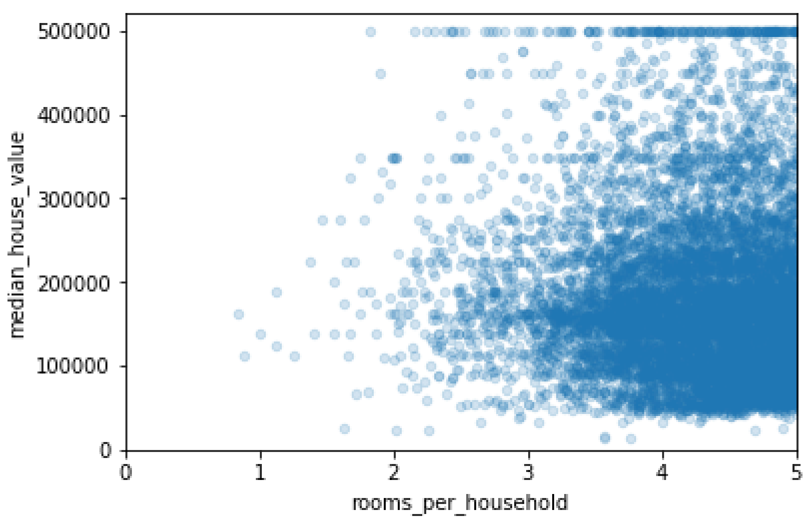
- rooms_per_household = total_rooms / households
- bedrooms_per_room = total_bedrooms / total_rooms
- etc.

In [534]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_roo
housing["population_per_household"]=housing["population"]/housing["househol
```

In [535]:
```
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[535]:
```
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
```

In [536]:
```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



In [537]:
```
housing.describe()
```

Out[537]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | populatio |
|---|---|---|---|---|---|---|
| **count** | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20433.000000 | 20640.00000 |
| **mean** | -119.569704 | 35.631861 | 28.639486 | 2635.763081 | 537.870553 | 1425.47674 |
| **std** | 2.003532 | 2.135952 | 12.585558 | 2181.615252 | 421.385070 | 1132.46212 |
| **min** | -124.350000 | 32.540000 | 1.000000 | 2.000000 | 1.000000 | 3.00000 |
| **25%** | -121.800000 | 33.930000 | 18.000000 | 1447.750000 | 296.000000 | 787.00000 |
| **50%** | -118.490000 | 34.260000 | 29.000000 | 2127.000000 | 435.000000 | 1166.00000 |
| **75%** | -118.010000 | 37.710000 | 37.000000 | 3148.000000 | 647.000000 | 1725.00000 |
| **max** | -114.310000 | 41.950000 | 52.000000 | 39320.000000 | 6445.000000 | 35682.00000 |

# Preparing Dataset 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 (https://scikit-learn.org/stable/)](https://scikit-learn.org/stable/) python package for preprocessing.

Scikit learn is pretty well documented and if you get confused at any point simply look up the function/object!

In [538]:
```
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]
```

In [539]:
```
housing = train_set.drop("median_house_value", axis=1) # drop labels for tr
                                                        # the input to the m
housing_labels = train_set["median_house_value"].copy()
```

## Dealing With Incomplete Data

In [540]:
```
# 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
```

Out[540]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | household |
|---|---|---|---|---|---|---|---|
| **4629** | -118.30 | 34.07 | 18.0 | 3759.0 | NaN | 3296.0 | 1462. |
| **6068** | -117.86 | 34.01 | 16.0 | 4632.0 | NaN | 3038.0 | 727. |
| **17923** | -121.97 | 37.35 | 30.0 | 1955.0 | NaN | 999.0 | 386. |
| **13656** | -117.30 | 34.05 | 6.0 | 2155.0 | NaN | 1039.0 | 391. |
| **19252** | -122.79 | 38.48 | 7.0 | 6837.0 | NaN | 3468.0 | 1405. |

In [541]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])     # option 1: sim
```

Out[541]:

| longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | me |
|---|---|---|---|---|---|---|---|

In [542]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)         # option 2: drc
```

Out[542]:

| | longitude | latitude | housing_median_age | total_rooms | population | households | median_incom |
|---|---|---|---|---|---|---|---|
| **4629** | -118.30 | 34.07 | 18.0 | 3759.0 | 3296.0 | 1462.0 | 2.270 |
| **6068** | -117.86 | 34.01 | 16.0 | 4632.0 | 3038.0 | 727.0 | 5.176 |
| **17923** | -121.97 | 37.35 | 30.0 | 1955.0 | 999.0 | 386.0 | 4.632 |
| **13656** | -117.30 | 34.05 | 6.0 | 2155.0 | 1039.0 | 391.0 | 1.667 |
| **19252** | -122.79 | 38.48 | 7.0 | 6837.0 | 3468.0 | 1405.0 | 3.166 |

In [543]:
```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # opt
sample_incomplete_rows
```

Out[543]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | household |
|---|---|---|---|---|---|---|---|
| **4629** | -118.30 | 34.07 | 18.0 | 3759.0 | 433.0 | 3296.0 | 1462. |
| **6068** | -117.86 | 34.01 | 16.0 | 4632.0 | 433.0 | 3038.0 | 727. |
| **17923** | -121.97 | 37.35 | 30.0 | 1955.0 | 433.0 | 999.0 | 386. |
| **13656** | -117.30 | 34.05 | 6.0 | 2155.0 | 433.0 | 1039.0 | 391. |
| **19252** | -122.79 | 38.48 | 7.0 | 6837.0 | 433.0 | 3468.0 | 1405. |

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

If we knew the dataset was normally distributed, replacing the missing values with the mean could be a good option.

## Prepare Data

In [544]:

```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 inte
# feeding to the model.

# Additionally, categorical values could either be represented as one-hot v
# 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 miss
housing_num = housing.drop("ocean_proximity", axis=1) # remove the categori
# 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["househ
    housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total
    housing["population_per_household"]=housing["population"]/housing["hous
    '''
    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)
```

## 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.

In [545]:
```
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 199614.4828
7493
 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.

In [546]:
```
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
```

Out[546]:
```
67784.32202861732
```

# 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.

# [25 pts] Visualizing Data

### [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

In [547]:
```
def load_airbnb_data(airbnb_path):
    csv_path = os.path.join(airbnb_path, "AB_NYC_2019.csv")
    return pd.read_csv(csv_path)
AIRBNB_DATASET_PATH = os.path.join("datasets", "airbnb")
airbnb = load_airbnb_data(AIRBNB_DATASET_PATH)
```

In [548]:
```
airbnb.head()
```

Out[548]:

| | id | name | host_id | host_name | neighbourhood_group | neighbourhood | latitude | lon |
|---|---|---|---|---|---|---|---|---|
| **0** | 2539 | Clean & quiet apt home by the park | 2787 | John | Brooklyn | Kensington | 40.64749 | -73 |
| **1** | 2595 | Skylit Midtown Castle | 2845 | Jennifer | Manhattan | Midtown | 40.75362 | -73 |
| **2** | 3647 | THE VILLAGE OF HARLEM....NEW YORK ! | 4632 | Elisabeth | Manhattan | Harlem | 40.80902 | -73 |
| **3** | 3831 | Cozy Entire Floor of Brownstone | 4869 | LisaRoxanne | Brooklyn | Clinton Hill | 40.68514 | -73 |
| **4** | 5022 | Entire Apt: Spacious Studio/Loft by central park | 7192 | Laura | Manhattan | East Harlem | 40.79851 | -73 |

In [549]:

```
airbnb_dropped = airbnb.copy()
airbnb_dropped = airbnb_dropped.drop(['name', 'host_id', 'host_name', 'last
airbnb_dropped.head()
```

Out[549]:

| | id | neighbourhood_group | neighbourhood | latitude | longitude | room_type | price | minimum_nig |
|---|---|---|---|---|---|---|---|---|
| **0** | 2539 | Brooklyn | Kensington | 40.64749 | -73.97237 | Private room | 149 | |
| **1** | 2595 | Manhattan | Midtown | 40.75362 | -73.98377 | Entire home/apt | 225 | |
| **2** | 3647 | Manhattan | Harlem | 40.80902 | -73.94190 | Private room | 150 | |
| **3** | 3831 | Brooklyn | Clinton Hill | 40.68514 | -73.95976 | Entire home/apt | 89 | |
| **4** | 5022 | Manhattan | East Harlem | 40.79851 | -73.94399 | Entire home/apt | 80 | |

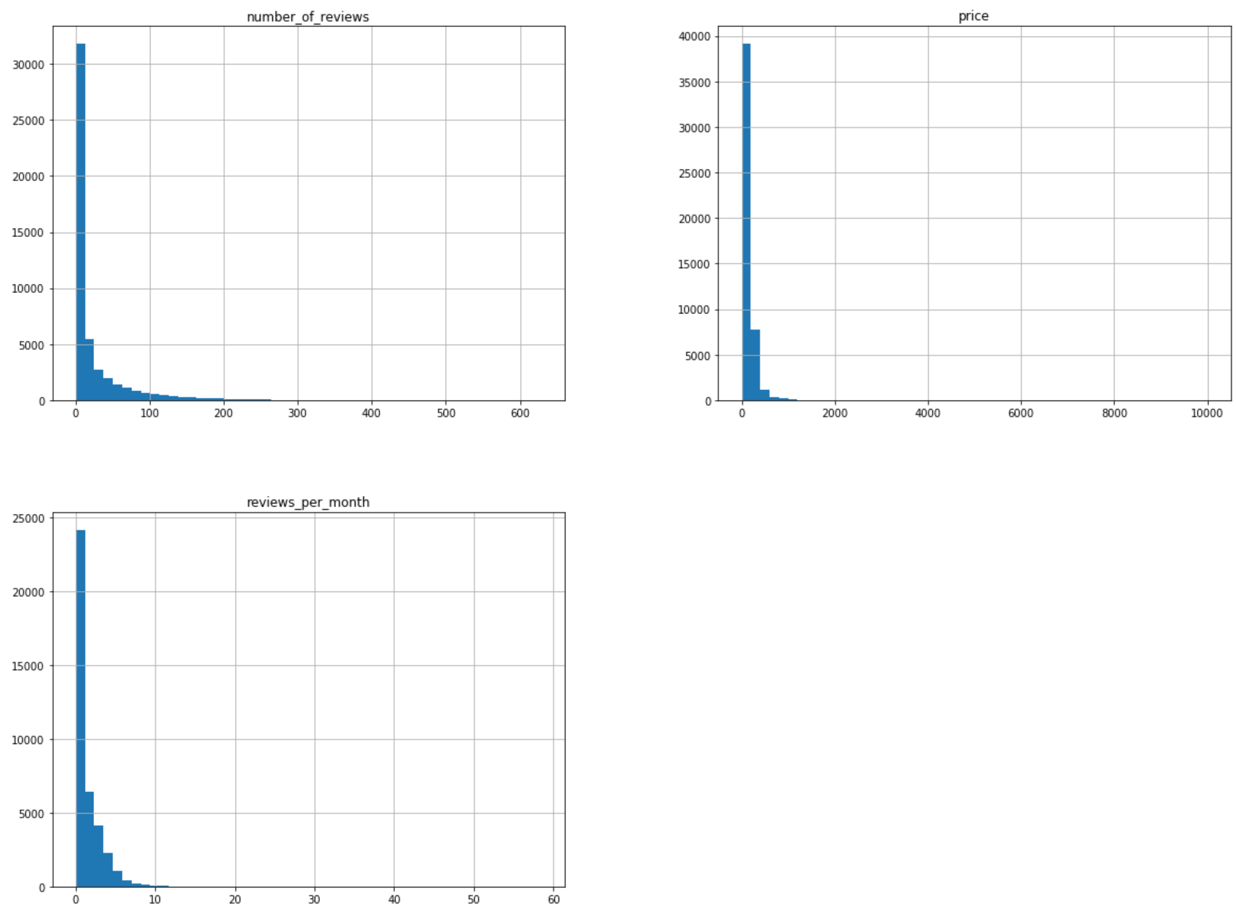
In [550]:

```
airbnb.describe()
```

Out[550]:

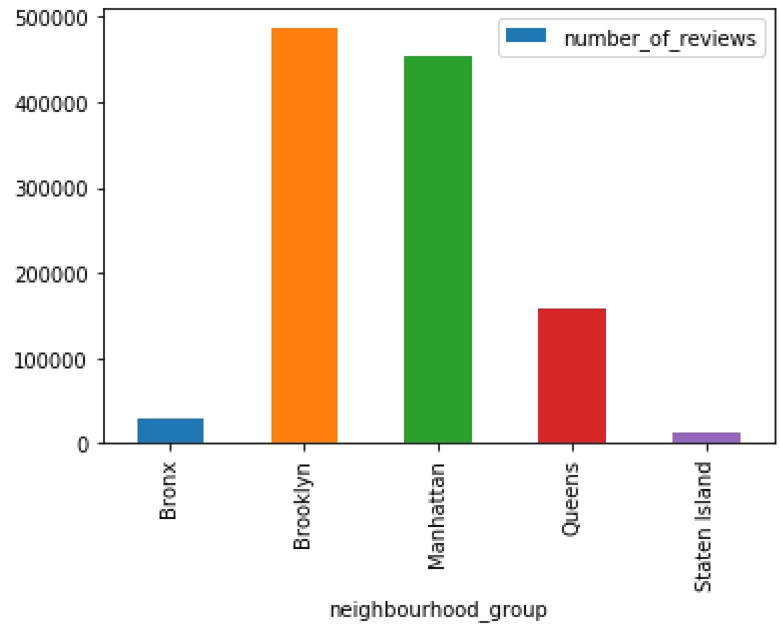
| | id | host_id | latitude | longitude | price | minimum_nights | nu |
|---|---|---|---|---|---|---|---|
| **count** | 4.889500e+04 | 4.889500e+04 | 48895.000000 | 48895.000000 | 48895.000000 | 48895.000000 | |
| **mean** | 1.901714e+07 | 6.762001e+07 | 40.728949 | -73.952170 | 152.720687 | 7.029962 | |
| **std** | 1.098311e+07 | 7.861097e+07 | 0.054530 | 0.046157 | 240.154170 | 20.510550 | |
| **min** | 2.539000e+03 | 2.438000e+03 | 40.499790 | -74.244420 | 0.000000 | 1.000000 | |
| **25%** | 9.471945e+06 | 7.822033e+06 | 40.690100 | -73.983070 | 69.000000 | 1.000000 | |
| **50%** | 1.967728e+07 | 3.079382e+07 | 40.723070 | -73.955680 | 106.000000 | 3.000000 | |
| **75%** | 2.915218e+07 | 1.074344e+08 | 40.763115 | -73.936275 | 175.000000 | 5.000000 | |
| **max** | 3.648724e+07 | 2.743213e+08 | 40.913060 | -73.712990 | 10000.000000 | 1250.000000 | |

In [551]:
```
airbnb.hist(bins=50, figsize=(20,15), column=['price', 'number_of_reviews',
plt.show()
```




## [5 pts] Plot total number_of_reviews per neighbourhood_group

In [552]:
```
airbnb.groupby('neighbourhood_group').number_of_reviews.sum().plot(legend=T
plt.show()
```

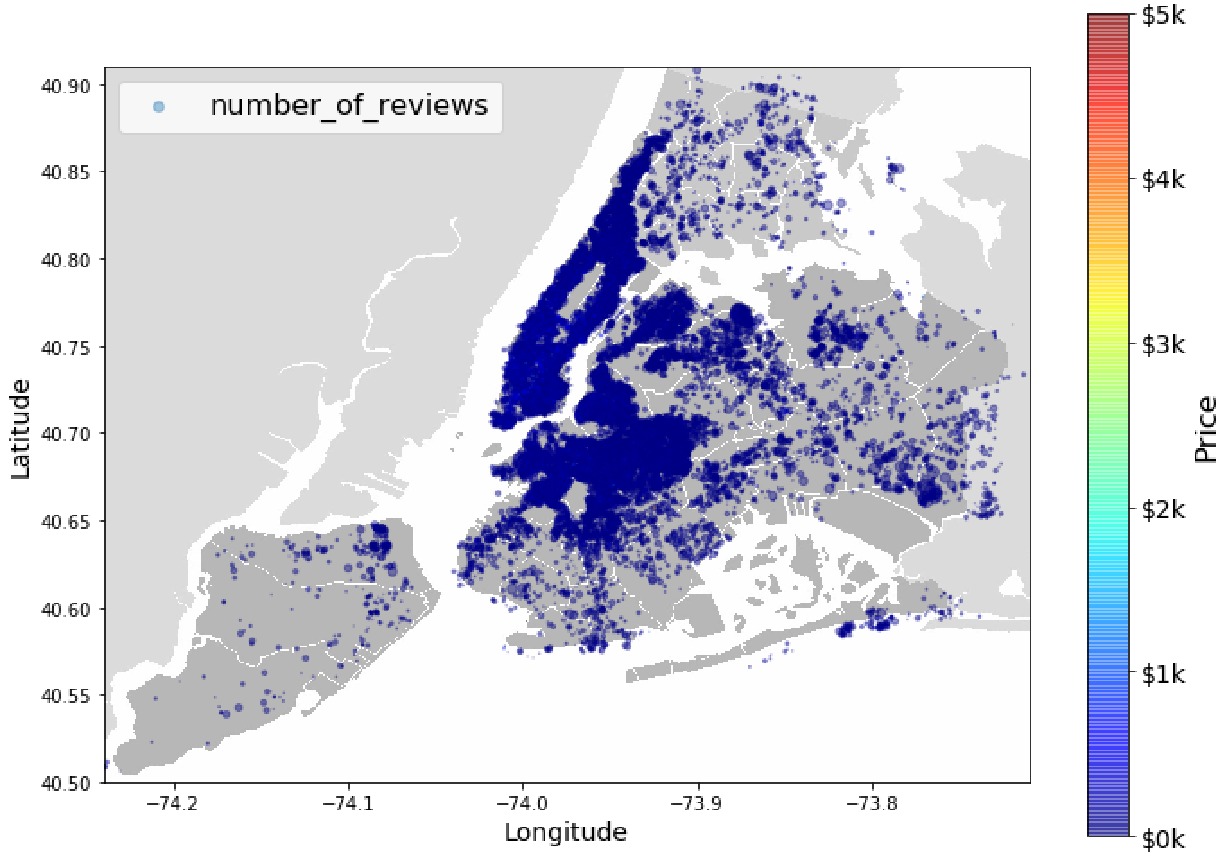



**[5 pts] Plot map of airbnbs throughout New York (if it gets too crowded take a subset of the data, and try to make it look nice if you can :) ).**

In [553]:

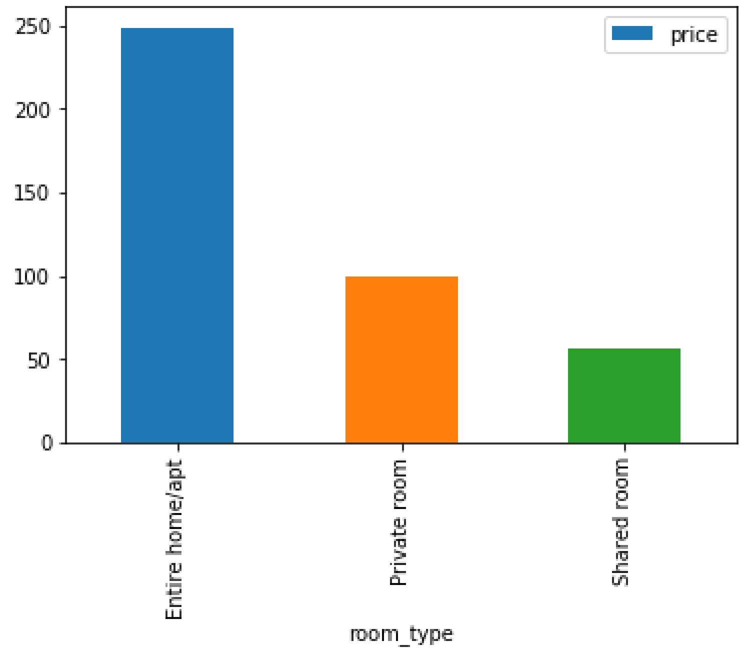
```
# Color code (heatmap) the dots based on price

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

import matplotlib.image as mpimg
newyork_img=mpimg.imread(os.path.join(images_path, filename))
ax = airbnb.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7
                       s=airbnb["number_of_reviews"]/10, label="number_of_r
                       c="price", cmap=plt.get_cmap("jet"),
                       colorbar=False, alpha=0.4,
                      )
# overlay the new york map on the plotted scatter plot
# note: plt.imshow still refers to the most recent figure
# that hasn't been plotted yet.
plt.imshow(newyork_img, extent=[-74.24, -73.71, 40.50, 40.91]
           , alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on price feature
prices = airbnb["price"]
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], fontsi
cb.set_label('Price', fontsize=16)

plt.legend(fontsize=16)
save_fig("newyork_housing_prices_plot")
plt.show()
```

```
Saving figure newyork_housing_prices_plot
```




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

In [554]:
```
airbnb_avail_gt_180 = airbnb.copy()
airbnb_avail_gt_180 = airbnb_avail_gt_180[airbnb_avail_gt_180["availability
airbnb_avail_gt_180.groupby("room_type").price.mean().plot(legend=True, kin
plt.show()
```




### [5 pts] Plot correlation matrix

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

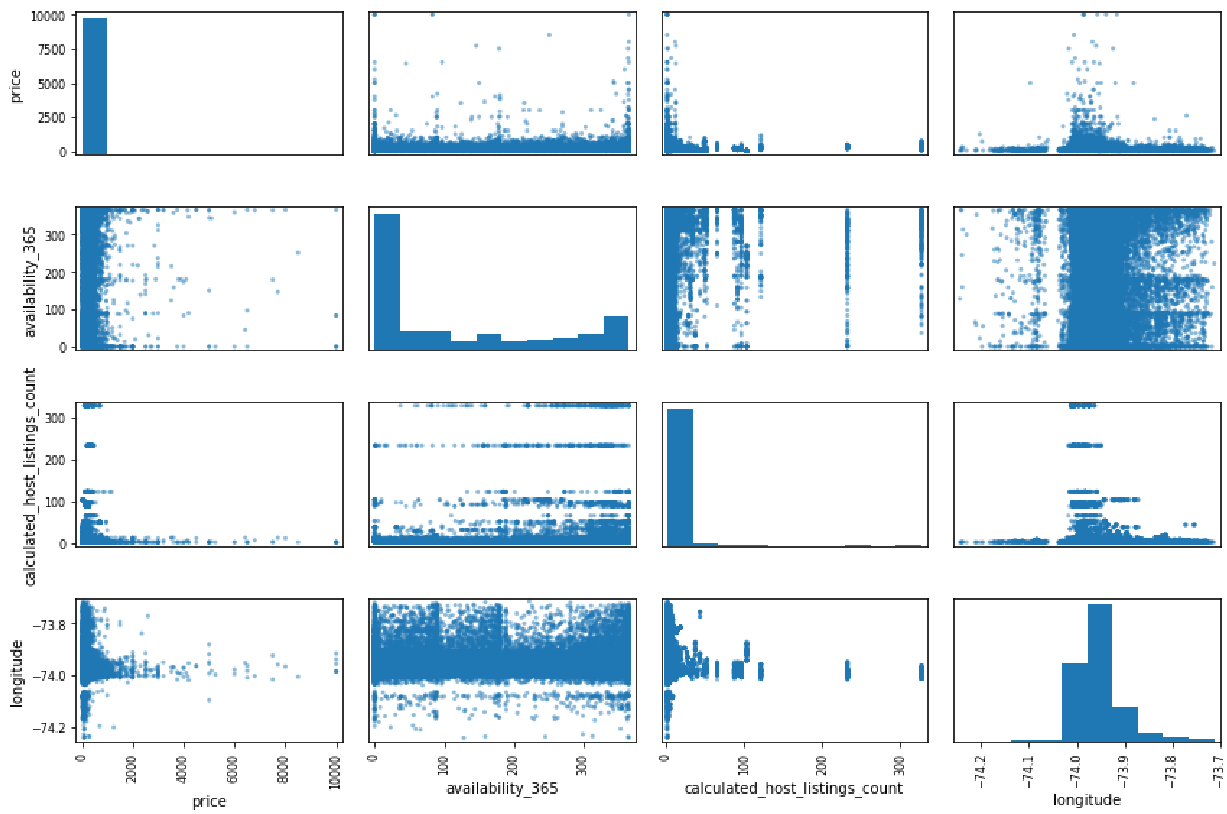
Although it was not obvious to me from the plots of the correlation matrices below, availability_365 and calculated_host_listings_count are the features with the greatest positive correlation to price, and longitude and number_of_reviews are the features with the greatest negative correlation. This is because all of the correlation values are close to 0, which essentially means no correlation.

In [555]:
```
# Determine which attributes to plot in correlation matrix
corr_matrix = airbnb.corr()
corr_matrix["price"].sort_values(ascending=False)
```

Out[555]:
```
price                             1.000000
availability_365                  0.081829
calculated_host_listings_count    0.057472
minimum_nights                    0.042799
latitude                          0.033939
host_id                           0.015309
id                                0.010619
reviews_per_month                -0.030608
number_of_reviews                -0.047954
longitude                        -0.150019
Name: price, dtype: float64
```

In [556]:
```python
from pandas.plotting import scatter_matrix
attributes = ["price", "availability_365", "calculated_host_listings_count"
scatter_matrix(airbnb[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot_airbnb")
```

```
Saving figure scatter_matrix_plot_airbnb
```




# [25 pts] Prepare the Data

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

In [557]:

```
airbnb_augmented = airbnb.copy()
airbnb_augmented["price_per_host_listings_count"] = airbnb_augmented["price
airbnb_augmented["max_bookings"] = airbnb_augmented["availability_365"]/air
airbnb_augmented.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 18 columns):
id                                48895 non-null int64
name                              48879 non-null object
host_id                           48895 non-null int64
host_name                         48874 non-null object
neighbourhood_group               48895 non-null object
neighbourhood                     48895 non-null object
latitude                          48895 non-null float64
longitude                         48895 non-null float64
room_type                         48895 non-null object
price                             48895 non-null int64
minimum_nights                    48895 non-null int64
number_of_reviews                 48895 non-null int64
last_review                       38843 non-null object
reviews_per_month                 38843 non-null float64
calculated_host_listings_count    48895 non-null int64
availability_365                  48895 non-null int64
price_per_host_listings_count     48895 non-null float64
max_bookings                      48895 non-null float64
dtypes: float64(5), int64(7), object(6)
memory usage: 6.7+ MB
```

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

```
As can be seen below, the only numerical feature with missing values is
reviews_per_month. I chose to replace na values with median values because
intuitively looking at the statistics from airbnb.describe(), it seems
plausible reviews_per_month could have outliers on the upper end. For the
next project I will figure out how to confirm this in code, but as a
beginner that seemed out of scope for this project. Also, intuitively
reviews_per_month seems like a good reason hosts could raise the price, so
I didn't want to drop the complete feature.

The categorical features with missing values are name, host_name, and
last_review. I chose to drop the rows with missing values because I didn't
know another technique to deal with this.
```

In [558]:
```python
# Determine which columns have missing values
airbnb_imputed = airbnb_augmented.copy()
airbnb_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 18 columns):
id                                48895 non-null int64
name                              48879 non-null object
host_id                           48895 non-null int64
host_name                         48874 non-null object
neighbourhood_group               48895 non-null object
neighbourhood                     48895 non-null object
latitude                          48895 non-null float64
longitude                         48895 non-null float64
room_type                         48895 non-null object
price                             48895 non-null int64
minimum_nights                    48895 non-null int64
number_of_reviews                 48895 non-null int64
last_review                       38843 non-null object
reviews_per_month                 38843 non-null float64
calculated_host_listings_count    48895 non-null int64
availability_365                  48895 non-null int64
price_per_host_listings_count     48895 non-null float64
max_bookings                      48895 non-null float64
dtypes: float64(5), int64(7), object(6)
memory usage: 6.7+ MB
```

In [559]:
```
median = airbnb_imputed["reviews_per_month"].median()
airbnb_imputed["reviews_per_month"].fillna(median, inplace=True)
airbnb_imputed = airbnb_imputed.dropna(subset=["name", "host_name", "last_r
airbnb_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38821 entries, 0 to 48852
Data columns (total 18 columns):
id                                38821 non-null int64
name                              38821 non-null object
host_id                           38821 non-null int64
host_name                         38821 non-null object
neighbourhood_group               38821 non-null object
neighbourhood                     38821 non-null object
latitude                          38821 non-null float64
longitude                         38821 non-null float64
room_type                         38821 non-null object
price                             38821 non-null int64
minimum_nights                    38821 non-null int64
number_of_reviews                 38821 non-null int64
last_review                       38821 non-null object
reviews_per_month                 38821 non-null float64
calculated_host_listings_count    38821 non-null int64
availability_365                  38821 non-null int64
price_per_host_listings_count     38821 non-null float64
max_bookings                      38821 non-null float64
dtypes: float64(5), int64(7), object(6)
memory usage: 5.6+ MB
```

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

In [560]:

```
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

# use median imputation for missing values
imputer = SimpleImputer(strategy="median")
# remove the categorical and id features
airbnb_num = airbnb.copy()
airbnb_num = airbnb_num.drop(["id", "name", "host_id", "host_name", "neighb
# column index
price_ix, calculated_host_listings_count_ix, availability_365_ix, minimum_n

airbnb = airbnb.dropna(subset=["name", "host_name", "last_review"])

class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    '''
    def __init__(self):
        pass
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        price_per_host_listings_count = X[:, price_ix] / X[:, calculated_ho
        max_bookings = X[:, availability_365_ix] / X[:, minimum_nights_ix]
        return np.c_[X, price_per_host_listings_count, max_bookings]

attr_adder = AugmentFeatures()

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

numerical_features = list(airbnb_num)
categorical_features = ["name", "host_name", "neighbourhood_group", "neighb

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])
airbnb_prepared = full_pipeline.fit_transform(airbnb)
```

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

In [562]:
```
from sklearn.model_selection import train_test_split

airbnb_X = airbnb.drop("price", axis=1)
airbnb_y = airbnb["price"].copy()
X_train, X_test, y_train, y_test = train_test_split(airbnb_X, airbnb_y, tes
```

## [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.

In [563]:
```
# Test set

from sklearn.linear_model import LinearRegression

airbnb_labels = airbnb_y

lin_reg = LinearRegression()
lin_reg.fit(airbnb_prepared, airbnb_labels)

data = X_test
labels = y_test
data_prepared = full_pipeline.transform(data)

from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(data_prepared)
mse = mean_squared_error(labels, preds)
mse
```

```
/Users/stewart/anaconda3/lib/python3.6/site-packages/pandas/core/indexin
g.py:1472: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-
reindex-listlike (https://pandas.pydata.org/pandas-docs/stable/indexing.h
tml#deprecate-loc-reindex-listlike)
  return self._getitem_tuple(key)
```

Out[563]: 35169.35341014199

In [564]:
```python
# Train set

from sklearn.linear_model import LinearRegression

airbnb_labels = airbnb_y

lin_reg = LinearRegression()
lin_reg.fit(airbnb_prepared, airbnb_labels)

data = X_train
labels = y_train
data_prepared = full_pipeline.transform(data)

from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(data_prepared)
mse = mean_squared_error(labels, preds)
mse
```

```
/Users/stewart/anaconda3/lib/python3.6/site-packages/pandas/core/indexin
g.py:1472: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-
reindex-listlike (https://pandas.pydata.org/pandas-docs/stable/indexing.h
tml#deprecate-loc-reindex-listlike)
  return self._getitem_tuple(key)
```

Out[564]: 41846.880097970534

In [ ]: