Introduction

Welcome to **CS188 - Data Science Fundamentals!** This course is designed to equip you with the tools and experiences necessary to start you off on a life-long exploration of datascience. We do not assume a prerequisite knowledge or experience in order to take the course.

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

1. Familiarize you with the development environment for doing datascience
2. Get you comfortable with the python coding required to do datascience
3. Provide you with an sample end-to-end project to help you visualize the steps needed to complete a project on your own
4. Ask you to recreate a similar project on a separate dataset

In this project you will work through an example project end to end. Many of the concepts you will encounter will be unclear to you. That is OK! The course is designed to teach you these concepts in further detail. For now our focus is simply on having you replicate the code successfully and seeing a project through from start to finish.

Here are the main steps:

1. Get the data
2. Visualize the data for insights
3. Preprocess the data for your machine learning algorithm
4. Select a model and train
5. Does it meet the requirements? Fine tune the model

 steps

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

Submission Instructions

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

Example Datascience Exercise

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

Setup

In [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)
```

In [2]:
```
import os
import tarfile
import urllib
DATASET_PATH = os.path.join("datasets", "housing")
```

Step 1. Getting the data

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

In [3]:
```
import pandas as pd

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

In [4]:
```
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[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	mec
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 [5]:
```
# to see a concise summary of data types, null values, and counts
# use the info() method on the dataframe
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   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
```

In [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..
```

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

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

Out[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
```

In [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()
```

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

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

Out[9]:

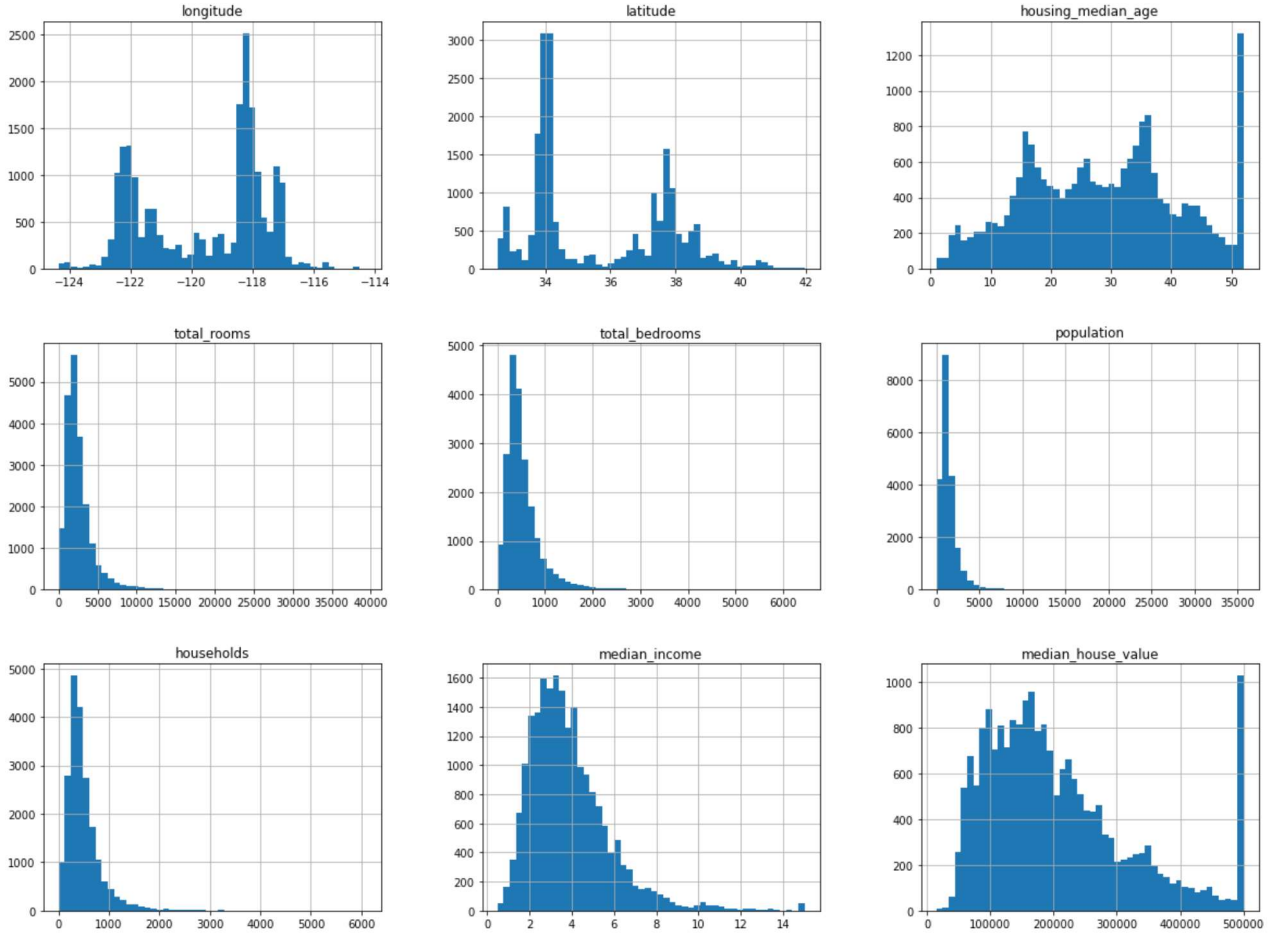
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	2(
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	

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

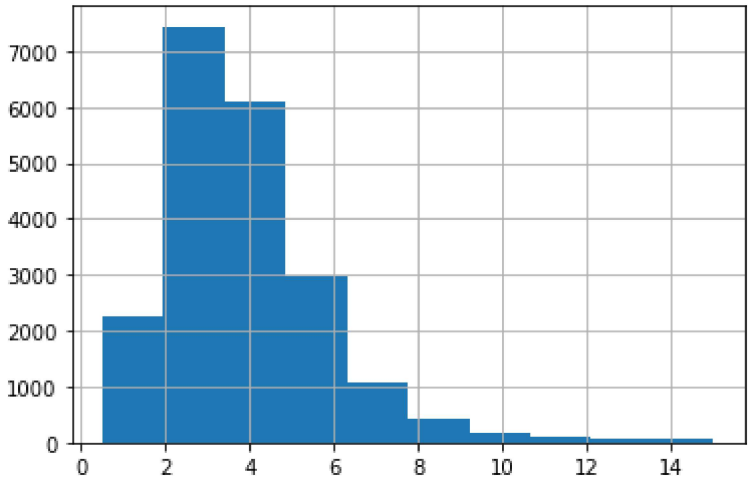
Step 2. Visualizing the data

Let's start visualizing the dataset

In [10]:
```
# 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
```



In [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.

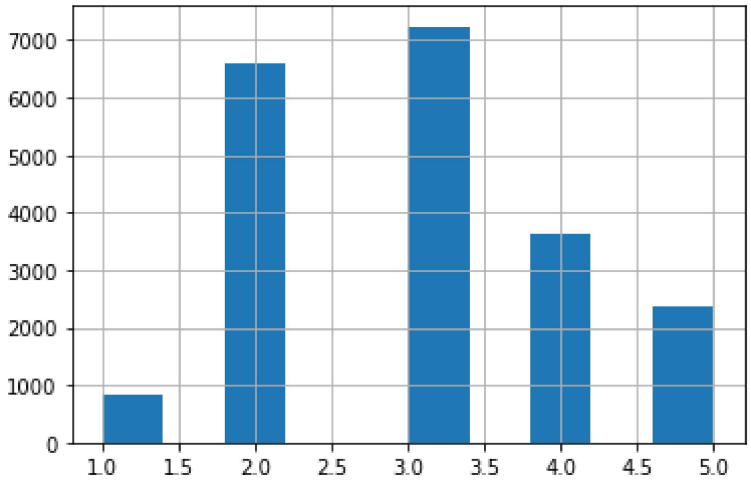
For example, to bin the households based on median_income we can use the pd.cut function

In [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()
```

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

In [13]:
```
housing["income_cat"].hist()
```

Out[13]:
```
<AxesSubplot:>
```

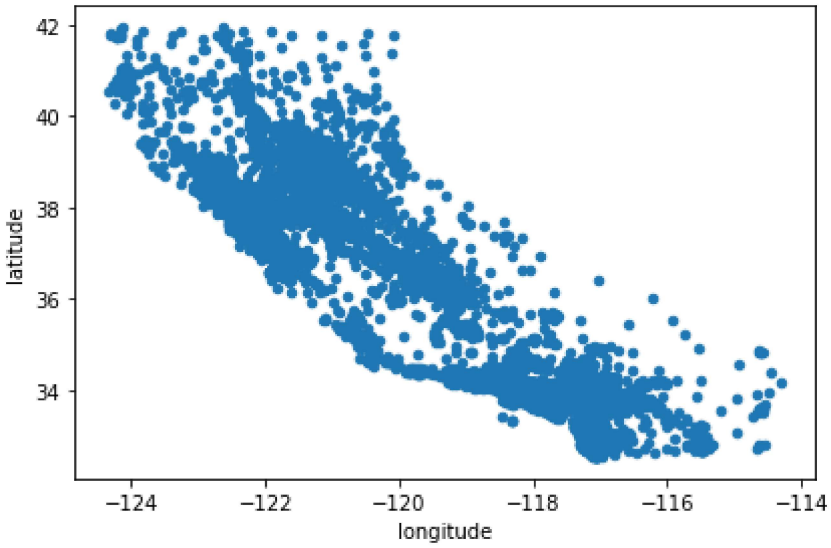


Next let's visualize the household incomes based on latitude & longitude coordinates

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

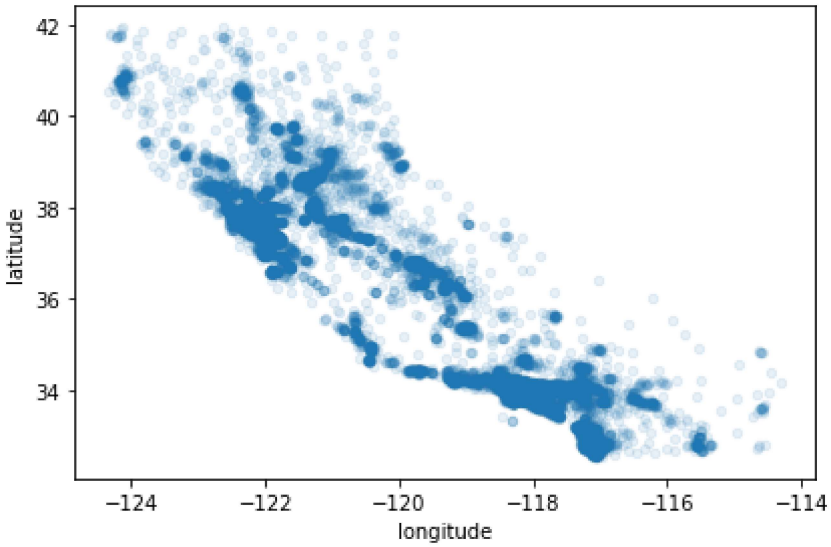
```
save_fig("bad_visualization_plot")
```

```
Saving figure bad_visualization_plot
```



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

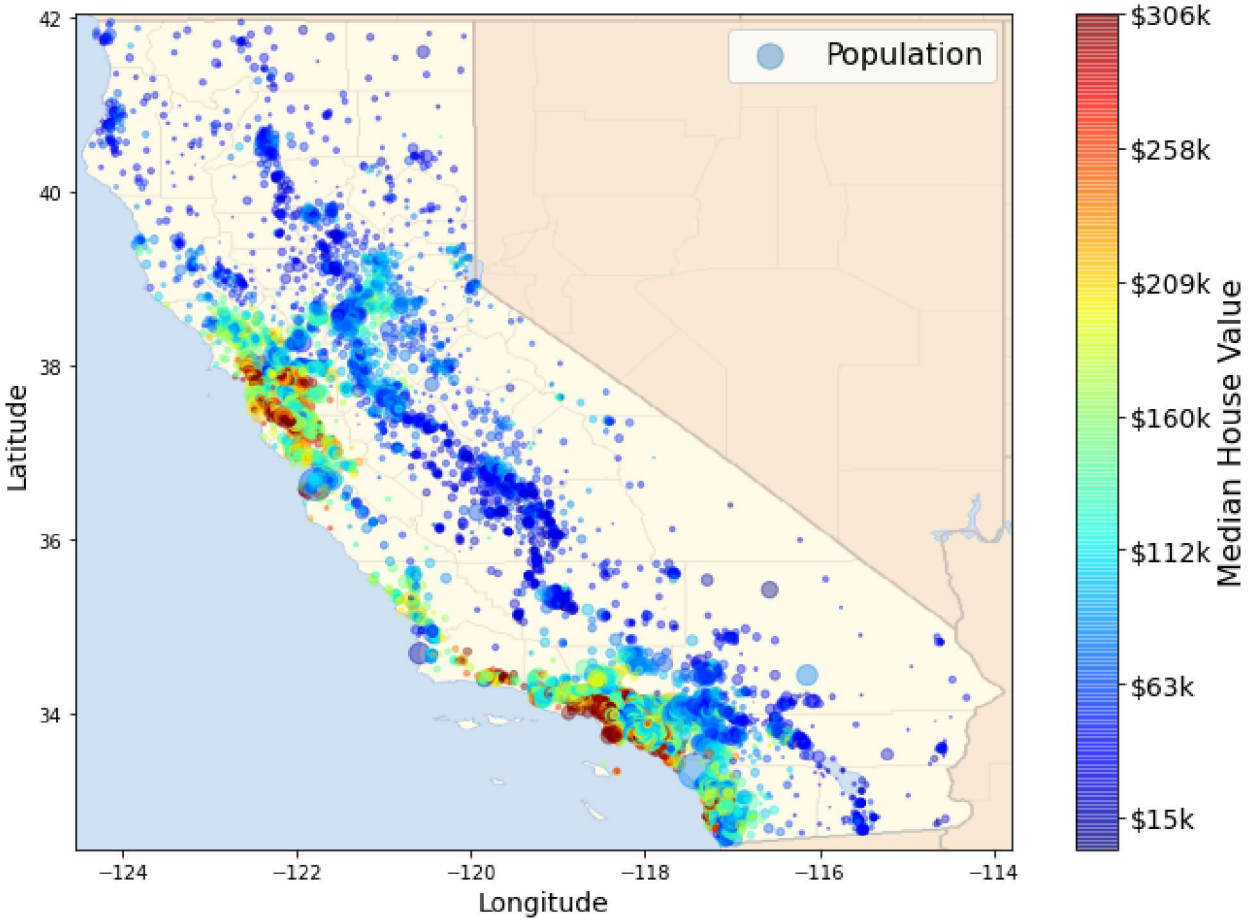
```
Saving figure better_visualization_plot
```



In [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()
```

```
<ipython-input-16-30a6f1a2327a>:28: UserWarning: FixedFormatter should only be used toge
ther with FixedLocator
  cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
Saving figure california_housing_prices_plot
```



Not suprisingly, we can see that the most expensive houses are concentrated around the San Francisco/Los Angeles areas.

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

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

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

None the less we can explore this using correlation matrices.

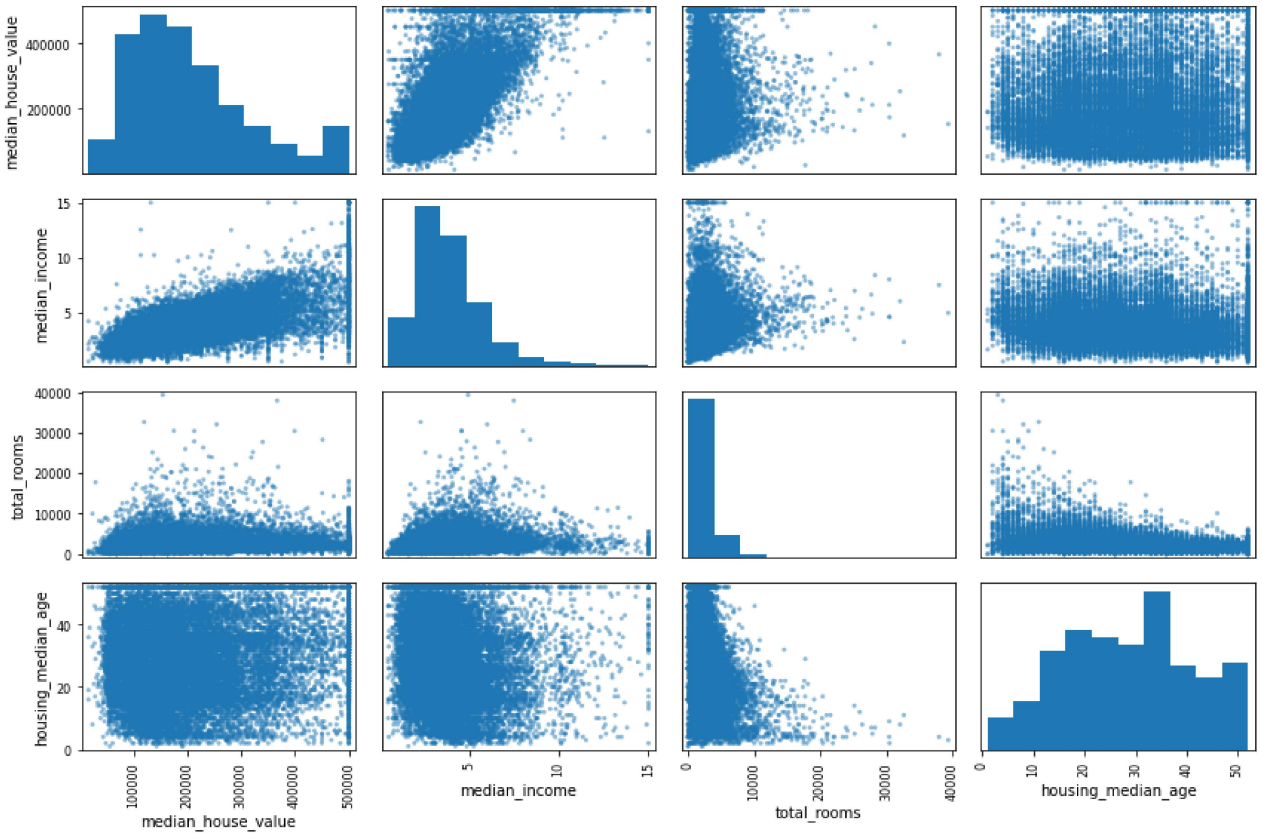
In [17]:
```
corr_matrix = housing.corr()
```

In [18]:
```
# for example if the target is "median_house_value", most correlated features can be so
# which happens to be "median_income". This also intuitively makes sense.
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[18]:
```
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 [19]:
```
# 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")
```

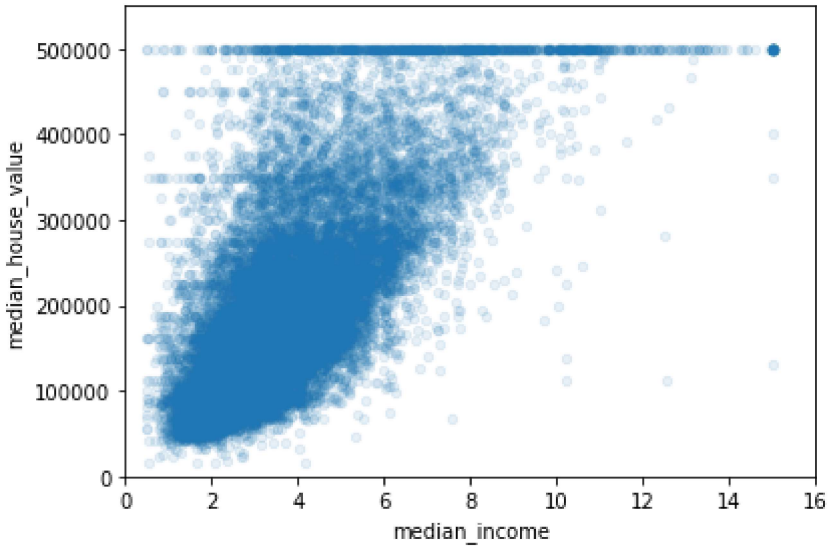
```
Saving figure scatter_matrix_plot
```



In [20]:
```
# 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
```



Augmenting Features

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

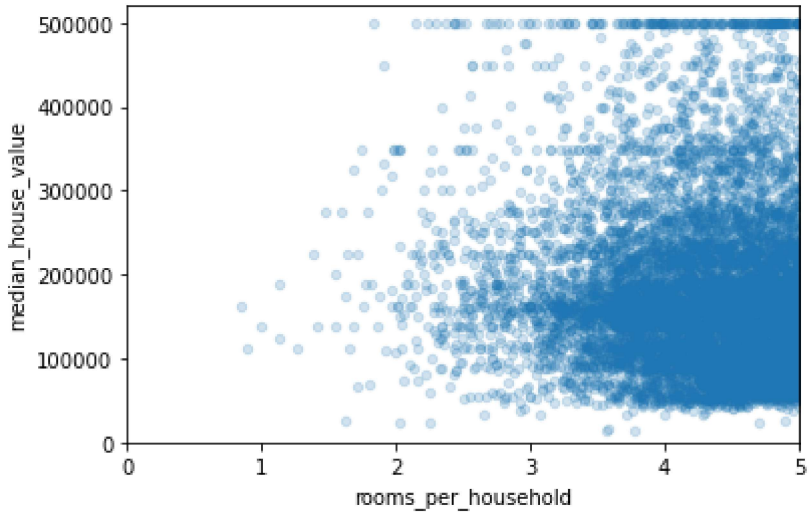
- rooms_per_household = total_rooms / households
- bedrooms_per_room = total_bedrooms / total_rooms
- etc.

In [21]:
```
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"]
```

In [22]:
```
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[22]:
```
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 [23]:
```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



In [24]:
```
housing.describe()
```

Out[24]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	2(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	(

Step 3. Preprocess the data for your machine learning algorithm

Once we've visualized the data, and have a certain understanding of how the data looks like. It's time to clean!

Most of your time will be spent on this step, although the datasets used in this project are relatively nice and clean... in the real world it could get real dirty.

After having cleaned your dataset you're aiming for:

- train set
- test set

In some cases you might also have a validation set as well for tuning hyperparameters (don't worry if you're not familiar with this term yet..)

In supervised learning setting your train set and test set should contain (**feature**, **target**) tuples.

- **feature**: is the input to your model
- **target**: is the ground truth label
  - when target is categorical the task is a classification task
  - when target is floating point the task is a regression task

We will make use of **scikit-learn** python package for preprocessing.

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

Dealing With Incomplete Data

In [25]:
```
# have you noticed when looking at the dataframe summary certain rows
# contained null values? we can't just leave them as nulls and expect our
# model to handle them for us so we'll have to devise a method for dealing with them...
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

Out[25]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
290	-122.16	37.77	47.0	1256.0	NaN	570.0	218.0	
341	-122.17	37.75	38.0	992.0	NaN	732.0	259.0	
538	-122.28	37.78	29.0	5154.0	NaN	3741.0	1273.0	
563	-122.24	37.75	45.0	891.0	NaN	384.0	146.0	
696	-122.10	37.69	41.0	746.0	NaN	387.0	161.0	

In [26]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1: simply drop row
```

Out[26]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	medi

In [27]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop the comple
```

Out[27]:

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	m
290	-122.16	37.77	47.0	1256.0	570.0	218.0	4.3750	
341	-122.17	37.75	38.0	992.0	732.0	259.0	1.6196	
538	-122.28	37.78	29.0	5154.0	3741.0	1273.0	2.5762	
563	-122.24	37.75	45.0	891.0	384.0	146.0	4.9489	
696	-122.10	37.69	41.0	746.0	387.0	161.0	3.9063	

In [28]:
```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3: repla
sample_incomplete_rows
```

Out[28]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
290	-122.16	37.77	47.0	1256.0	435.0	570.0	218.0	
341	-122.17	37.75	38.0	992.0	435.0	732.0	259.0	
538	-122.28	37.78	29.0	5154.0	435.0	3741.0	1273.0	
563	-122.24	37.75	45.0	891.0	435.0	384.0	146.0	
696	-122.10	37.69	41.0	746.0	435.0	387.0	161.0	

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

Prepare Data

In [29]:
```
housing_unlabeled = housing.drop("median_house_value", axis=1) # drop labels for traini
                                                               # the input to the model should
housing_labels = housing["median_house_value"].copy()
```

In [30]:
```
housing_unlabeled.head()
```

Out[30]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	mec
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	

In [31]:
```
# 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
# feeding to the model.

# Additionally, categorical values could either be represented as one-hot vectors or si
# Here we encode them using one hot vectors.

# DO NOT WORRY IF YOU DO NOT UNDERSTAND ALL THE STEPS OF THIS PIPELINE. CONCEPTS LIKE N
# ONE-HOT ENCODING ETC. WILL ALL BE COVERED IN DISCUSSION

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin
```

```
imputer = SimpleImputer(strategy="median") # use median imputation for missing values
housing_num = housing_unlabeled.drop("ocean_proximity", axis=1) # remove the categorica
# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

#
class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
    housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
    housing["population_per_household"]=housing["population"]/housing["households"]
    '''
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

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

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

housing_num_tr = num_pipeline.fit_transform(housing_num)
numerical_features = list(housing_num)
categorical_features = ["ocean_proximity"]

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

housing_prepared = full_pipeline.fit_transform(housing_unlabeled)
```

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 [32]:
```
from sklearn.model_selection import train_test_split
data_target = housing['median_house_value']
train, test, target, target_test = train_test_split(housing_prepared, data_target, test
```

Splitting our dataset

First we need to carve out our dataset into a training and testing cohort. To do this we'll use train_test_split, a very elementary tool that arbitrarily splits the data into training and testing cohorts.

In [33]:
```
from sklearn.linear_model import LinearRegression

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

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

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

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

In [34]:
```
from sklearn.metrics import mean_squared_error

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

Out[34]:
```
67879.86844243007
```

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.

[35 pts] Visualizing Data

[5 pts] Load the data + statistics

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

In [54]:
```
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.head()
```

Out[54]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitud

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitud
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.972
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.983
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.941
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.959
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.943

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

In [55]:
```
airbnb.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
```

The data entries that are personally assiggned and have no impact on the actual room and price, including id, name, host_id, and host_name will be problematic in feeding into my model, as it will add needless increased complexity that may result in nonsensical correlations.

Convertign between ints, objects, and floats can also lead to many simple errors while funelling a combination of these types into my dataset.

Missing data in some of the columns, indicated by Nan values, also must be fixed before inputting into model.

The other qualitative entries will also be problematic when feeding into my model, as I must use some sort of encoding to create a real number mapping for them, however the values of this real number mapping should not

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

In [56]:
```
airbnb = airbnb.drop(['name', 'host_id', 'host_name', 'last_review'], axis=1)
#axis (0 = row, 1 = column)?
```

In [57]:
```
airbnb.describe()
```

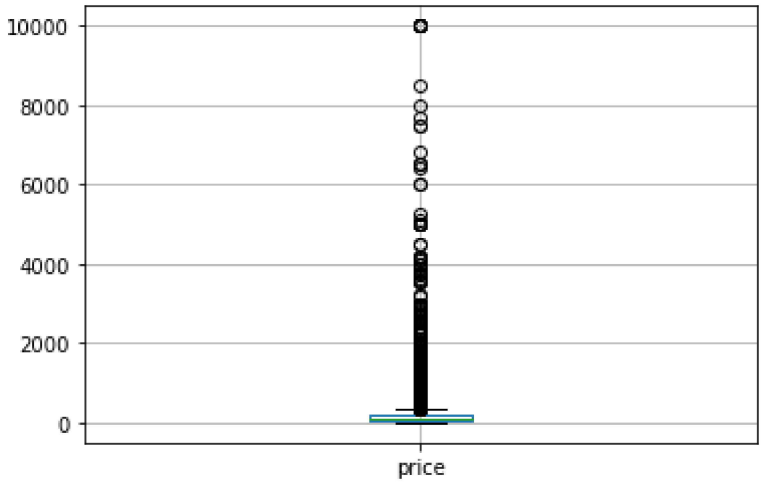
Out[57]:

	id	latitude	longitude	price	minimum_nights	number_of_reviews	re
count	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	
mean	1.901714e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	
std	1.098311e+07	0.054530	0.046157	240.154170	20.510550	44.550582	
min	2.539000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	
25%	9.471945e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	
50%	1.967728e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	
75%	2.915218e+07	40.763115	-73.936275	175.000000	5.000000	24.000000	
max	3.648724e+07	40.913060	-73.712990	10000.000000	1250.000000	629.000000	

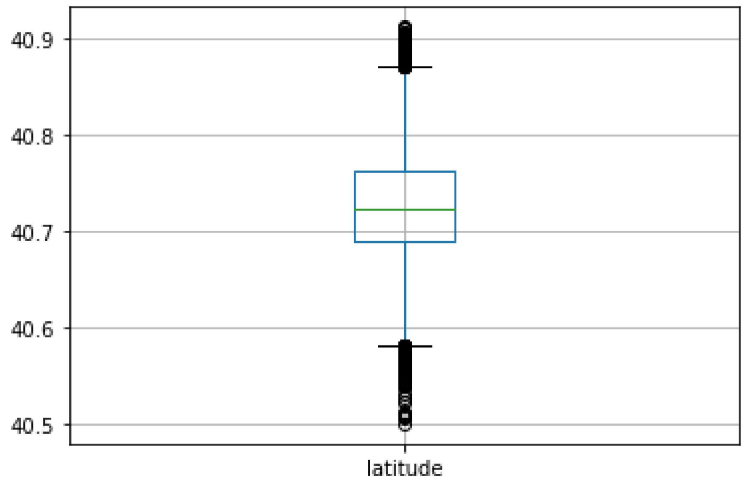
[5 pts] Boxplot 3 features of your choice

- plot boxplots for 3 features of your choice

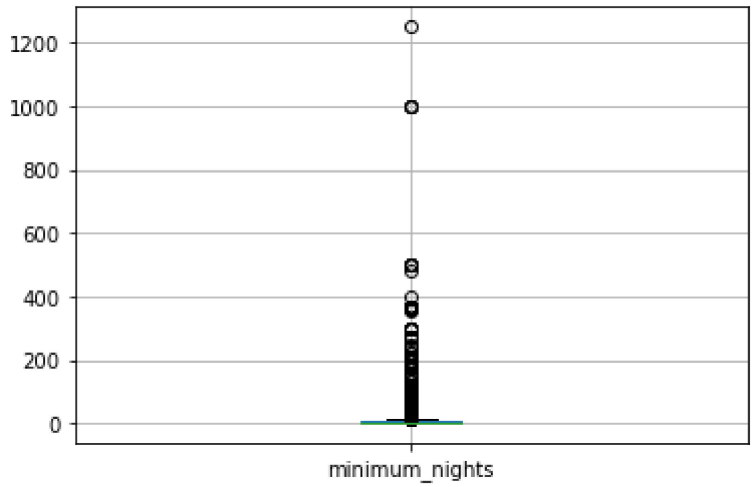
In [58]:
```
boxplot = airbnb.boxplot(column = ['price'])
```



In [59]:
```
boxplot = airbnb.boxplot(column = ['latitude'])
```



In [60]:
```
boxplot = airbnb.boxplot(column = ['minimum_nights'])
```



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

With the price boxplot, I expected to see high variability with the majority of data points in the 0-1000 range for price. The box plot validated my thoughts, but I did not expect the extent of the spread upwards due to a cluster of extreme prices in the 4,000-8,000 range and one extreme outlier at 10,000.

With the latitude boxplot, I expected to see a much lower variance and a tighter cluster in the mid-40 range, as latitude was given as 40.x for each data plot, as the data is all for the New York area which all falls in the same general latitude range. Latitude is a larger measure meant to cover the entire world, so applying to to a specific area will yield similar latitudes for each entry.

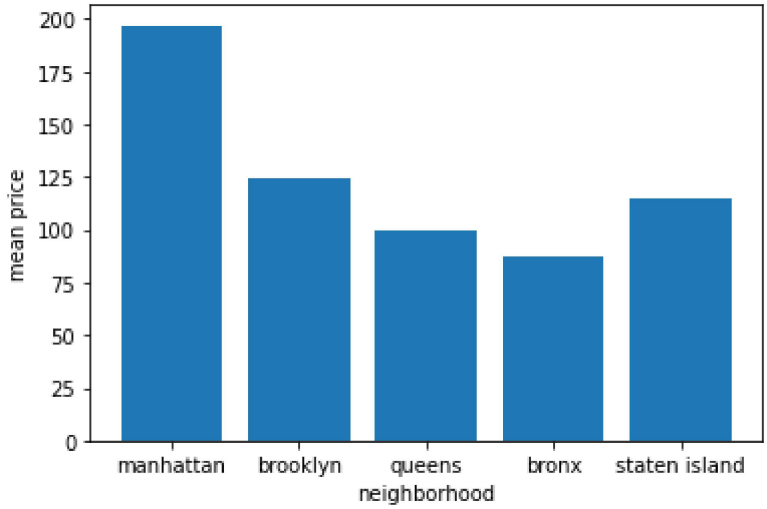
With the minimum nights feauture, I expected a tight grouping with small variance around with a mean and median around 1-2. I was completely suprised with the results of the box plot, and can't help but to think that much of the data is corrupted or invalid. Theres a whole range of points between 100-400 which seems inconsistent with the airbnb business model. Not even mentioning the extreme outliers of 1,000 and around 1200. Like what owner puts their house on airbnb for a 3 year stay?!

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

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

In [61]:
```
airbnb['neighbourhood_group'].value_counts()
manhattan = airbnb[airbnb['neighbourhood_group']=='Manhattan']
brooklyn = airbnb[airbnb['neighbourhood_group']=='Brooklyn']
queens = airbnb[airbnb['neighbourhood_group']=='Queens']
bronx = airbnb[airbnb['neighbourhood_group']=='Bronx']
sewer = airbnb[airbnb['neighbourhood_group']=='Staten Island']

price_per_neighbourhood = [manhattan['price'].mean(), brooklyn['price'].mean(), queens[
plt.bar(['manhattan', 'brooklyn', 'queens', 'bronx', 'staten island'],price_per_neighbo
plt.xlabel("neighborhood")
plt.ylabel("mean price")
```

Out[61]:
```
Text(0, 0.5, 'mean price')
```



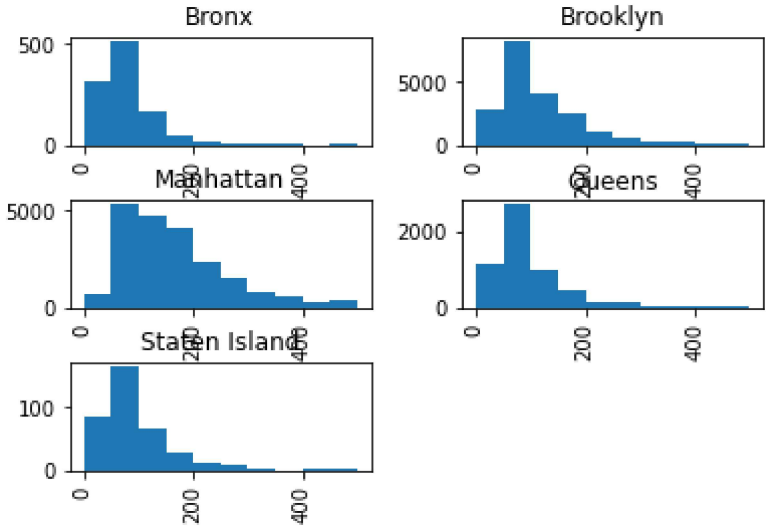
- describe what you expected to see with these features and what you actually observed

Based on my limited knowledge of New York neighborhoods, I expected brooklyn and manhattan to have the highest average prices per night with queens and the bronx having the least. I expected staten island to be somewhere in the middle. Most of my predictions matched what I observed, ecxept for the queens having a higher average rent per night than the bronx. It should be noted that there is some bias, as certain areas of these neihborhoods have been gentrified in recent years, and those areas might be more likely to list there houses on airbnb than poorer areas that have a family in a single room who cannot travel to their summer house and lease out their home.

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

In [62]:
```
#ended up using instructor answer on piazaa post @48
airbnb.hist(column='price', by='neighbourhood_group', range=[0, 500])

#price on x axis, frequency on y
```

Out[62]:
```
array([[<AxesSubplot:title={'center':'Bronx'}>,
        <AxesSubplot:title={'center':'Brooklyn'}>],
       [<AxesSubplot:title={'center':'Manhattan'}>,
        <AxesSubplot:title={'center':'Queens'}>],
       [<AxesSubplot:title={'center':'Staten Island'}>, <AxesSubplot:>]],
      dtype=object)
```

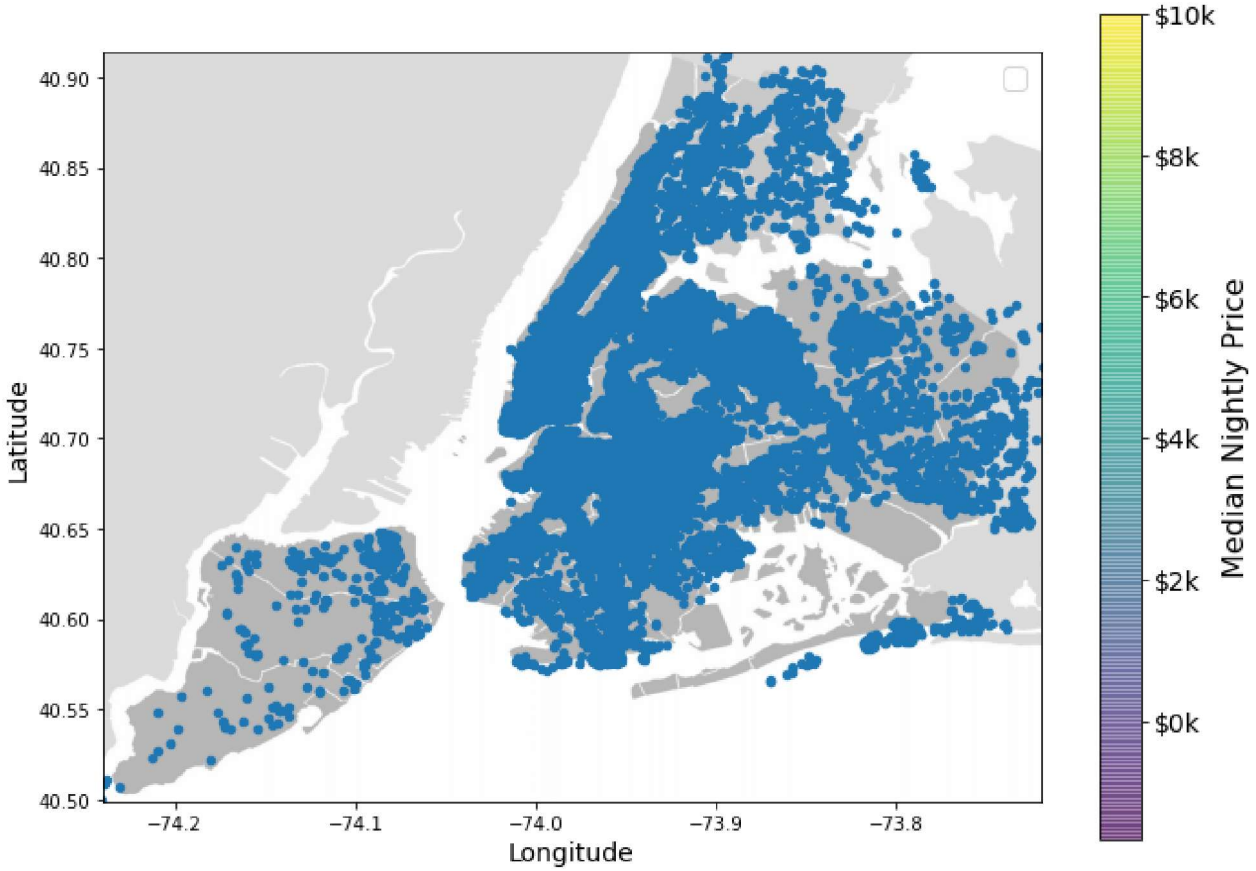


[5 pts] Plot map of airbnbs throughout New York (if it gets too crowded take a subset of the data, and try to make it look nice if you can :)).

In [63]:
```
# A more interesting plot is to color code (heatmap) the dots
# based on income. The code below achieves this

# 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)
                       )
# 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, -73.72, 40.499, 40.914], alpha=0.5)
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on median_house_value feature
prices = airbnb["price"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cb = plt.colorbar()
cb.ax.set_yticklabels(["$%dk"%(round(v/500)) for v in tick_values], fontsize=14)
cb.set_label('Median Nightly Price', fontsize=16)

plt.legend(fontsize=16)
save_fig("newYork_airbnb_prices_plot")
plt.show()


# print(airbnb['latitude'].min())
```

```
# print(airbnb['latitude'].max())
# print(airbnb['longitude'].min())
# print(airbnb['longitude'].max())
```

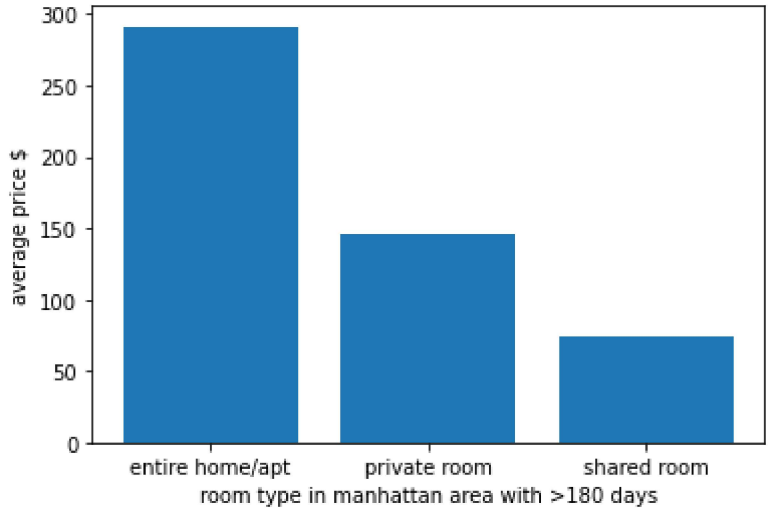
```
<ipython-input-63-77b2d760b457>:24: UserWarning: FixedFormatter should only be used toge
ther with FixedLocator
  cb.ax.set_yticklabels(["$%dk"%(round(v/500)) for v in tick_values], fontsize=14)
No handles with labels found to put in legend.
Saving figure newYork_airbnb_prices_plot
```



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

In [64]:
```
manhattan.head()
over_half = manhattan[manhattan['availability_365'] > 180]
# over_half.hist(by='room_type')
entire = over_half[over_half['room_type']=='Entire home/apt']
private = over_half[over_half['room_type']=='Private room']
shared = over_half[over_half['room_type']=='Shared room']
room_split_averages = [entire['price'].mean(), private['price'].mean(), shared['price']
plt.bar(['entire home/apt', 'private room', 'shared room'], room_split_averages)
plt.xlabel('room type in manhattan area with >180 days')
plt.ylabel('average price $')
```

Out[64]:
```
Text(0, 0.5, 'average price $')
```



[5 pts] Plot correlation matrix

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

In [65]:
```
corr_matrix = airbnb.corr()
corr_matrix["price"].sort_values(ascending=False)
```

Out[65]:
```
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
```

Positive correlations to price: availability_365 0.081829 calculated_host_listings_count 0.057472 minimum_nights 0.042799 latitude 0.033939 id 0.010619 (This one is bs)

Negative correlations to price: reviews_per_month -0.030608 number_of_reviews -0.047954 longitude -0.150019

[30 pts] Prepare the Data

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

In [66]:
```
import math

#airbnb["minimum_total_cost"] = airbnb['minimum_nights']*airbnb['price']
airbnb["max_yearly_bookings"] = airbnb['availability_365']/airbnb['minimum_nights']
airbnb['months_on_airbnb'] = airbnb['number_of_reviews']/airbnb['reviews_per_month'] #a
```

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

In [67]:
```
#I chose to remove the rows with missing data, as each feauture is neccisary, but indiv
#I choose not to replace them with a median, as if the data was missing some values it
#to believe that the entry was corrupted, and I could not trust the rest of the feature
#it all came from the same listing, so if it is missing some feauture data, odds are so
#with airbnb's data collection for that listing so it should not be trusted
#I also finally dropped ID, as it is useless an not assosiated with price

airbnb = airbnb.drop(['id'], axis=1)
airbnb = airbnb.dropna(axis=0)



airbnb = airbnb[airbnb.availability_365 != 0] #had a divide by 0 error in my pipeline t
#decided to just remove entiries with 0 availability throughout the year, as it means t
#rented through airbnb, so why bother use it to predict airbnb prices
airbnb.head(30)
```

Out[67]:

	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	nu
0	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	1	
1	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	1	
3	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	1	
5	Manhattan	Murray Hill	40.74767	-73.97500	Entire home/apt	200	3	
7	Manhattan	Hell's Kitchen	40.76489	-73.98493	Private room	79	2	
9	Manhattan	Chinatown	40.71344	-73.99037	Entire home/apt	150	1	
10	Manhattan	Upper West Side	40.80316	-73.96545	Entire home/apt	135	5	
11	Manhattan	Hell's Kitchen	40.76076	-73.98867	Private room	85	2	
12	Brooklyn	South Slope	40.66829	-73.98779	Private room	89	4	
13	Manhattan	Upper West Side	40.79826	-73.96113	Private room	85	2	
15	Brooklyn	Williamsburg	40.70837	-73.95352	Entire home/apt	140	2	
16	Brooklyn	Fort Greene	40.69169	-73.97185	Entire home/apt	215	2	
17	Manhattan	Chelsea	40.74192	-73.99501	Private room	140	1	
18	Brooklyn	Crown Heights	40.67592	-73.94694	Entire home/apt	99	3	
21	Brooklyn	Park Slope	40.68069	-73.97706	Private room	130	2	

	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	nu
22	Brooklyn	Park Slope	40.67989	-73.97798	Private room	80	1	
23	Brooklyn	Park Slope	40.68001	-73.97865	Private room	110	2	
24	Brooklyn	Bedford-Stuyvesant	40.68371	-73.94028	Entire home/apt	120	2	
25	Brooklyn	Windsor Terrace	40.65599	-73.97519	Private room	60	1	
27	Manhattan	Hell's Kitchen	40.76715	-73.98533	Entire home/apt	150	10	
28	Manhattan	Inwood	40.86482	-73.92106	Private room	44	3	
29	Manhattan	East Village	40.72920	-73.98542	Entire home/apt	180	14	
30	Manhattan	Harlem	40.82245	-73.95104	Private room	50	3	
31	Manhattan	Harlem	40.81305	-73.95466	Private room	52	2	
32	Brooklyn	Greenpoint	40.72219	-73.93762	Private room	55	4	
33	Manhattan	Harlem	40.82130	-73.95318	Private room	50	3	
34	Brooklyn	Bedford-Stuyvesant	40.68310	-73.95473	Private room	70	1	
35	Brooklyn	South Slope	40.66869	-73.98780	Private room	89	4	
37	Brooklyn	Bushwick	40.70186	-73.92745	Entire home/apt	85	2	
39	Manhattan	Lower East Side	40.71401	-73.98917	Shared room	40	1	

[15 pts] Code complete data pipeline using sklearn mixins

In [68]:
```
airbnb_unlabeled = airbnb.drop("price", axis=1)
airbnb_labels = airbnb['price'].copy()
airbnb_unlabeled.head()
```

Out[68]:

	neighbourhood_group	neighbourhood	latitude	longitude	room_type	minimum_nights	number_of
0	Brooklyn	Kensington	40.64749	-73.97237	Private room	1	
1	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	1	

	neighbourhood_group	neighbourhood	latitude	longitude	room_type	minimum_nights	number_of
3	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	1	
5	Manhattan	Murray Hill	40.74767	-73.97500	Entire home/apt	3	
7	Manhattan	Hell's Kitchen	40.76489	-73.98493	Private room	2	

In [69]:
```
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

## REMOVE CATEGORIAL FEAUTURES
imputer = SimpleImputer(strategy="median") # use median imputation for missing values
airbnb_num = airbnb_unlabeled.drop(["neighbourhood_group", "neighbourhood", "room_type"


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


airbnb_num_tr = num_pipeline.fit_transform(airbnb_num)
numerical_features = list(airbnb_num)
categorical_features = ["neighbourhood", "neighbourhood_group", "room_type"]

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

airbnb_prepared = full_pipeline.fit_transform(airbnb_unlabeled)

airbnb_prepared
```

Out[69]:
```
<26155x237 sparse matrix of type '<class 'numpy.float64'>'
        with 392325 stored elements in Compressed Sparse Row format>
```

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

In [71]:
```
from sklearn.model_selection import train_test_split
train, test, target, target_test = train_test_split(airbnb_unlabeled, airbnb_labels, te

print("train shape:", train.shape)
print("target shape:", target.shape)
print("test shape:", test.shape)
print("target_test shape:",target_test.shape)
```

```
train shape: (20924, 12)
```

```
target shape: (20924,)
test shape: (5231, 12)
target_test shape: (5231,)
```

[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 [73]:
```
# from sklearn.linear_model import LinearRegression

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

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

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

In [74]:
```
test.head()
```

Out[74]:

	neighbourhood_group	neighbourhood	latitude	longitude	room_type	minimum_nights	numb
45348	Queens	Astoria	40.76417	-73.92898	Private room	2	
25222	Queens	Richmond Hill	40.69584	-73.84727	Private room	1	
2730	Brooklyn	Williamsburg	40.71215	-73.95637	Private room	3	
10719	Manhattan	East Village	40.72255	-73.97813	Private room	5	
5507	Manhattan	East Village	40.73256	-73.98558	Entire home/apt	30	

In [77]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

test_drop = test.drop([ "neighbourhood", "neighbourhood_group", "room_type"], axis=1)
train_drop = train.drop(["neighbourhood", "neighbourhood_group", "room_type"], axis=1)

lin_reg = LinearRegression()

lin_reg.fit(train_drop, target)



preds = lin_reg.predict(train_drop)
mse = mean_squared_error(target, preds)
print ("Train MSE: ", mse)

preds = lin_reg.predict(test_drop)
```

```
mse = mean_squared_error(target_test, preds)
print ("Test MSE: ", mse)
```

```
Train MSE:  38870.54913959547
Test MSE:  33598.94442304889
```

In [78]:
```
rmse = np.sqrt(mse)
rmse
```

Out[78]:
```
183.3001484534284
```

In [ ]:
```

```