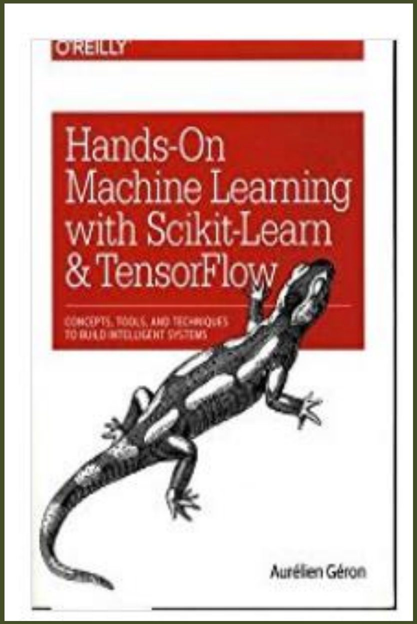


CHAPTER 2: END-TO-END MACHINE LEARNING PROJECT

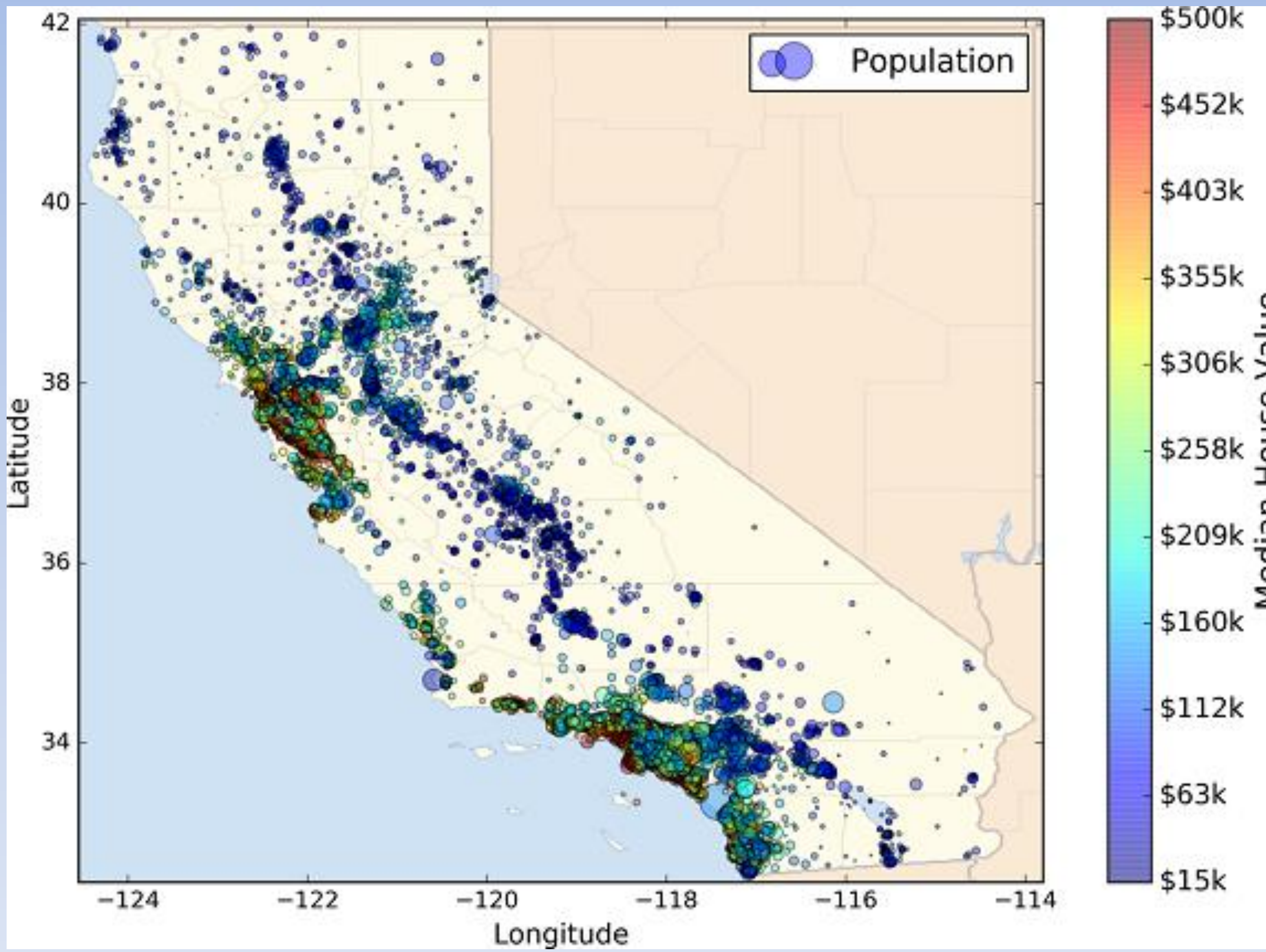
New Hire: Data Scientist

- Real Estate Company

Project High Level Steps

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model. Present your solution.
7. Launch, monitor, and maintain your system.

The Data!



Population
Latitude
Longitude
Median House Value
42
40
38
36
34
−124
−122
−120
−118
−116
−114
$500k
$452k
$403k
$355k
$306k
$258k
$209k
$160k
$112k
$63k
$15k

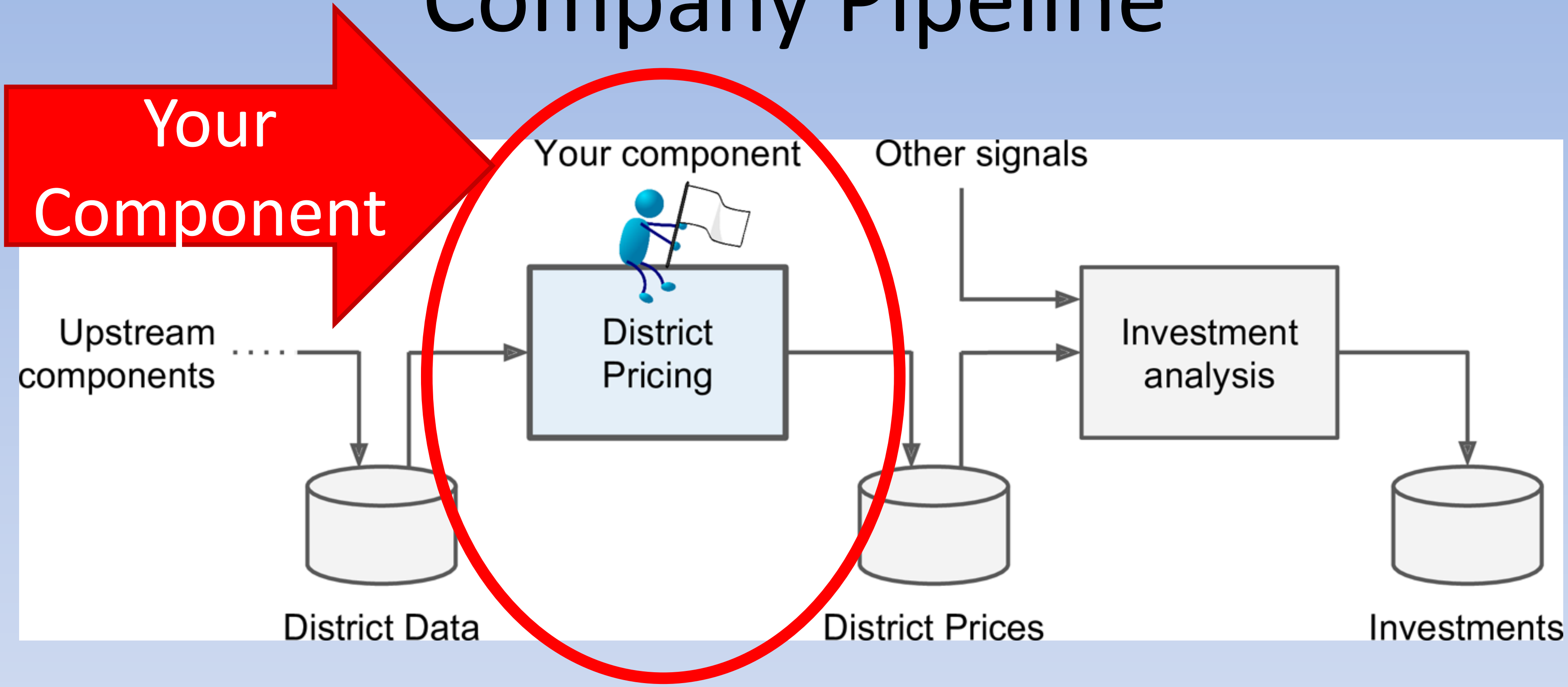
Welcome to Machine Learning Housing Corporation!

- The first task: build a model of housing prices in California using the California census data.
- Metrics such as: population, median income, median housing price, and so on for each block group in California.
  - Block groups are the smallest geographical unit for which the US Census Bureau publishes sample data
  - (a block group typically has a population of 600 to 3,000 people).
  - We will just call them “districts” for short.
- Your model should learn from this data:
  - Be able to predict the median housing price in any district, given all the other metrics.

Framing Problem

- The first question: what exactly is the business objective;
  - building a model is probably not the end goal.
  - How does the company expect to use and benefit from this model?
- This will determine:
  - how you frame the problem,
  - what algorithms you will select,
  - what performance measure you will use to evaluate your model,
  - how much effort you should spend tweaking it.
- Boss' answer:
  - model's output (a prediction of a district's median housing price) will be fed to another Machine Learning system, along with many other signals.
  - This downstream system will determine whether it is worth investing in a given area or not.
  - Getting this right is critical, as it directly affects revenue.

Company Pipeline



How About Current Solution??

- Current solution will give you a reference performance, as well as insights on how to solve the problem.
- Your boss answers that the district housing prices are currently estimated manually by experts:
  - a team gathers up-to-date information about a district, and when they cannot get the median housing price, they estimate it using complex rules.
  - Costly and time-consuming, and their estimates are not great;
  - in cases where they manage to find out the actual median housing price, they often realize that their estimates were off by more than 10%.
- Company thinks that it would be useful to train a model to predict a district’s median housing price given other data about that district.
- The census data looks like a great dataset to exploit for this purpose
  - it includes the median housing prices of thousands of districts, as well as other data.

System Design

- Type of Task:
  - is it supervised, unsupervised, or Reinforcement Learning?
  - Is it a classification task, a regression task, or something else?
  - Should you use batch learning or online learning techniques?

- Typical supervised learning task since you are given labeled training examples
  - each instance comes with the expected output,
    - i.e., the district’s median housing price.
- Typical regression task, since you are asked to predict a value.
  - Multivariate regression problem since the system will use multiple features to make a prediction
    - it will use the district’s population, the median income, etc.
- Finally, batch learning should do just fine
  - there is no continuous flow of data coming in the system,
  - there is no particular need to adjust to changing data rapidly
  - the data is small enough to fit in memory, so plain.

Performance Measures

- Need to decide on measures of accuracy of model.
- Root-Mean-Square Error
- Mean Absolute Error

Performance Measure
Root-Mean-Square Error

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

Performance Measure
Mean Absolute Error

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m} \mid h(\mathbf{x}^{(i)}) - y^{(i)} \mid$$

Performance Measures

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}$$

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left| h(\mathbf{x}^{(i)}) - y^{(i)} \right|$$

Verify Assumptions

- Lastly, list and verify assumptions made (by you or others);
  - this can catch serious issues early on.
- For example, district prices that your system outputs are going to be fed into a downstream Machine Learning system, and we assume that these prices are going to be used as such.
  - But what if the downstream system actually converts the prices into categories (e.g., “cheap,” “medium,” or “expensive”) and then uses those categories instead of the prices themselves?
  - In this case, getting the price perfectly right is not important at all; your system just needs to get the category right.
  - If that’s so, then the problem should have been framed as a classification task, not a regression task.
- You don’t want to find out a classification system was needed after working on a regression system for months.

Now Code

everestso / handson-ml
forked from ageron/handson-ml

Unwatch 1 | Star 0 | Fork 2,432

Code | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Branch: master | handson-ml / 02_end_to_end_machine_learning_project.ipynb | Find file | Copy path

Kxrr Fix incorrect description in chapter 2 — 91b142c on Nov 24, 2017

4 contributors

1.29 MB | Download | History

Chapter 2 – End-to-end Machine Learning project

Welcome to Machine Learning Housing Corp.! Your task is to predict median house values in Californian districts, given a number of features from these districts.

This notebook contains all the sample code and solutions to the exercices in chapter 2.

Note: You may find little differences between the code outputs in the book and in these Jupyter notebooks: these slight differences are mostly due to the random nature of many training algorithms: although I have tried to make these notebooks' outputs as constant as possible, it is impossible to guarantee that they will produce the exact same output on every platform. Also, some data structures (such as dictionaries) do not preserve the item order. Finally, I fixed a few minor bugs (I added notes next to the concerned cells) which lead to slightly different results, without changing the ideas presented in the book.

GitHub, Inc. [US] https://github.com/everestso/handson-ml/tree/master/datasets/housing

myfsu email OneDrive nbviewer VitalSource mit Nitrous GitHub zyBooks | Stanford Lagunita csuf aol LinkedIn

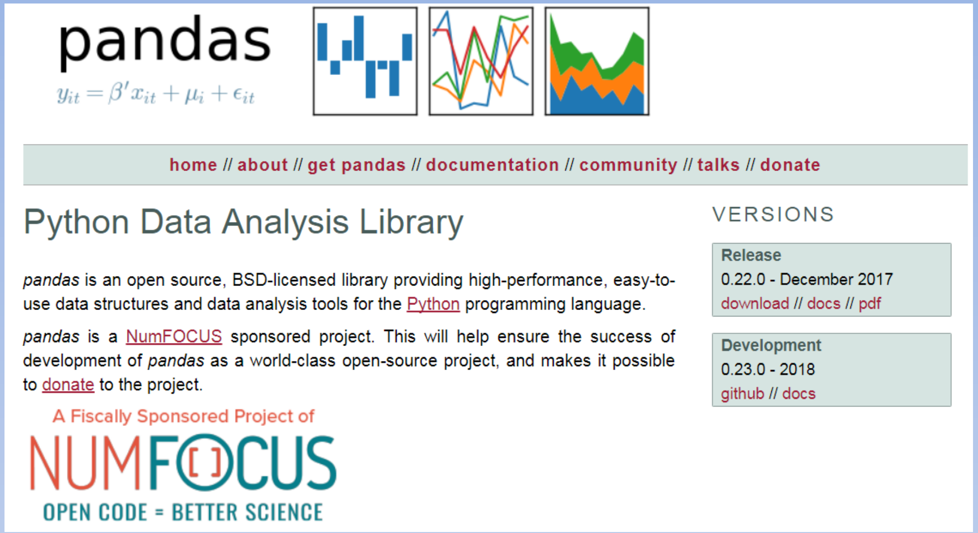
California Housing

Source

This dataset is a modified version of the California Housing dataset available from [http://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html](Luís Torgo's page) (University of Porto). Luís Torgo obtained it from the StatLib repository (which is closed now). The dataset may also be downloaded from StatLib mirrors.

This dataset appeared in a 1997 paper titled *Sparse Spatial Autoregressions* by Pace, R. Kelley and Ronald Barry, published in the *Statistics and Probability Letters* journal. They built it using the 1990 California census data. It contains one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

Pandas for Data



- Read in our data.
- Explore the data a bit.
- Download a tgz file, uzip, and save a csv

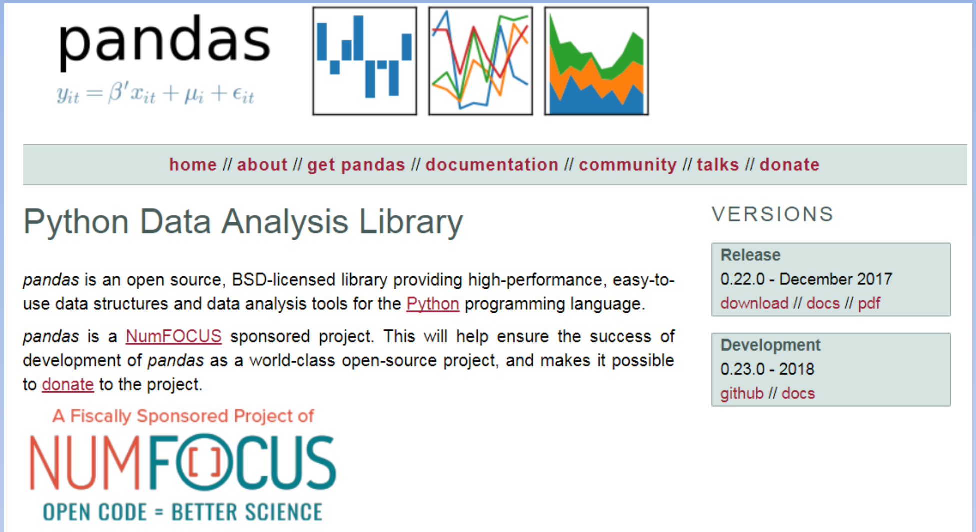
Python Code

```
In [4]: DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
        HOUSING_PATH = os.path.join("datasets", "housing")
        HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

        print (HOUSING_URL)
        print (HOUSING_PATH)
        def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
            if not os.path.isdir(housing_path):
                os.makedirs(housing_path)
            tgz_path = os.path.join(housing_path, "housing.tgz")
            urllib.request.urlretrieve(housing_url, tgz_path)
            housing_tgz = tarfile.open(tgz_path)
            housing_tgz.extractall(path=housing_path)
            housing_tgz.close()

        https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.tgz
        datasets/housing

In [5]: fetch_housing_data()
```

Now the data

	A	B	C	D	E	F	G	H	I	J
1	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_prox
2	-122.23	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY
20614	-121.56	39.08	26	1377	289	761	267	1.4934	48300	INLAND
20615	-121.55	39.09	31	1728	365	1167	384	1.4958	53400	INLAND
20616	-121.54	39.08	26	2276	460	1455	474	2.4695	58000	INLAND
20617	-121.54	39.08	23	1076	216	724	197	2.3598	57500	INLAND
20618	-121.53	39.08	15	1810	441	1157	375	2.0469	55100	INLAND
20619	-121.53	39.06	20	561	109	308	114	3.3021	70800	INLAND
20620	-121.55	39.06	25	1332	247	726	226	2.25	63400	INLAND
20621	-121.56	39.01	22	1891	340	1023	296	2.7303	99100	INLAND
20622	-121.48	39.05	40	198	41	151	48	4.5625	100000	INLAND
20623	-121.47	39.01	37	1244	247	484	157	2.3661	77500	INLAND
20624	-121.44	39	20	755	147	457	157	2.4167	67000	INLAND
20625	-121.37	39.03	32	1158	244	598	227	2.8235	65500	INLAND
20626	-121.41	39.04	16	1698	300	731	291	3.0739	87200	INLAND
20627	-121.52	39.12	37	102	17	29	14	4.125	72000	INLAND
20628	-121.43	39.18	36	1124	184	504	171	2.1667	93800	INLAND
20629	-121.32	39.13	5	358	65	169	59	3	162500	INLAND
20630	-121.48	39.1	19	2043	421	1018	390	2.5952	92400	INLAND
20631	-121.39	39.12	28	10035	1856	6912	1818	2.0943	108300	INLAND
20632	-121.32	39.29	11	2640	505	1257	445	3.5673	112000	INLAND
20633	-121.4	39.33	15	2655	493	1200	432	3.5179	107200	INLAND
20634	-121.45	39.26	15	2319	416	1047	385	3.125	115600	INLAND
20635	-121.53	39.19	27	2080	412	1082	382	2.5495	98300	INLAND
20636	-121.56	39.27	28	2332	395	1041	344	3.7125	116800	INLAND
20637	-121.09	39.48	25	1665	374	845	330	1.5603	78100	INLAND
20638	-121.21	39.49	18	697	150	356	114	2.5568	77100	INLAND
20639	-121.22	39.43	17	2254	485	1007	433	1.7	92300	INLAND
20640	-121.32	39.43	18	1860	409	741	349	1.8672	84700	INLAND
20641	-121.24	39.37	16	2785	616	1387	530	2.3886	89400	INLAND

Pandas for Data



- Read the CSV into Pandas Dataframe

Read the csv file

```
In [12]: housing = pd.read_csv("datasets/housing/housing.csv")
         print (housing.info())
         print (housing.head(4))
         print (housing.tail(4))

<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
```

Create a Histogram w/ House Prices

Histogram of Data w/ Pandas & Matplotlib

pandas.DataFrame.hist

`DataFrame.hist`(*data, column=None, by=None, grid=True, xlabelsize=None, xrot=None, ylabelsize=None, yrot=None, ax=None, sharex=False, sharey=False, figsize=None, layout=None, bins=10, **kwds*) [source]

Draw histogram of the DataFrame's series using matplotlib / pylab.

Parameters:

data : *DataFrame*

column : *string or sequence*

If passed, will be used to limit data to a subset of columns

by : *object, optional*

If passed, then used to form histograms for separate groups

grid : *boolean, default True*

Whether to show axis grid lines

xlabelsize : *int, default None*

If specified changes the x-axis label size

xrot : *float, default None*

rotation of x axis labels

ylabelsize : *int, default None*

If specified changes the y-axis label size

yrot : *float, default None*

rotation of y axis labels

ax : *matplotlib axes object, default None*

sharex : *boolean, default True if ax is None else False*

In case subplots=True, share x axis and set some x axis labels to invisible; defaults to True if ax is None otherwise False if an ax is passed in; Be aware, that passing in both an ax and sharex=True will alter all x axis labels for all subplots in a figure!

sharey : *boolean, default False*

In case subplots=True, share y axis and set some y axis labels to invisible

figsize : *tuple*

The size of the figure to create in inches by default

layout : *tuple, optional*

Tuple of (rows, columns) for the layout of the histograms

bins : *integer, default 10*

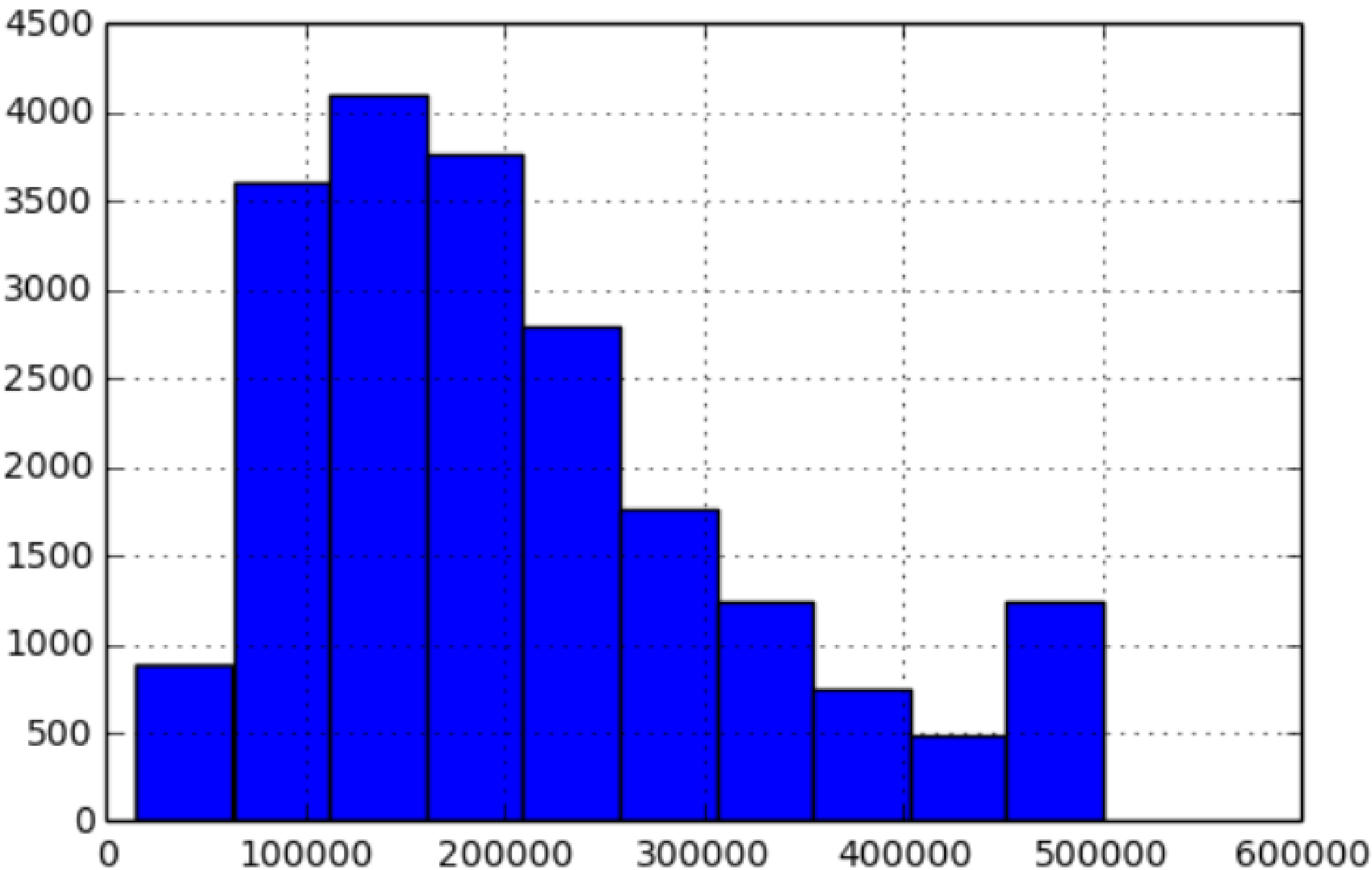
Number of histogram bins to be used

kwds : *other plotting keyword arguments*

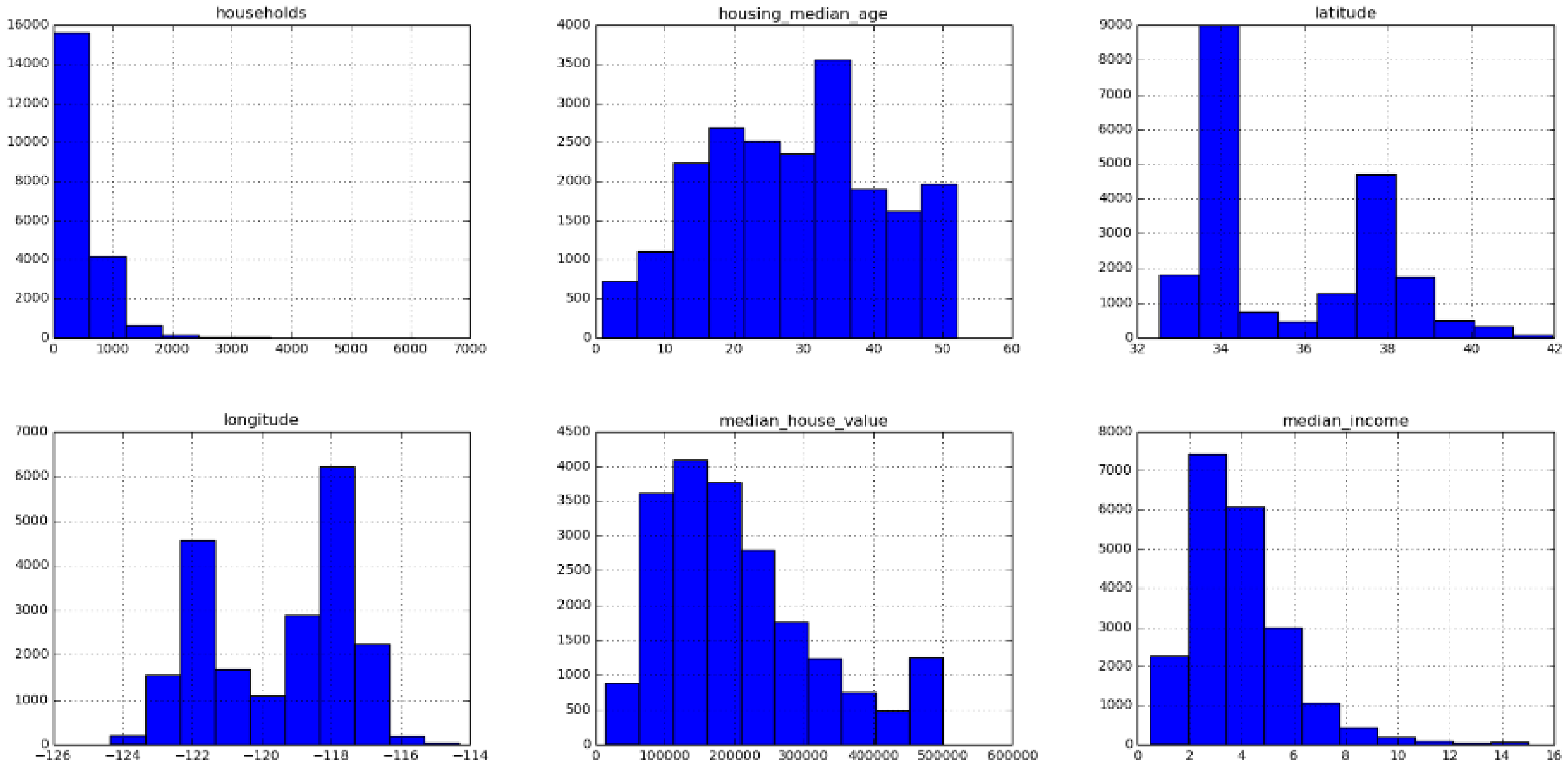
To be passed to hist function

```
housing["median_house_value"].hist()
```

```
<matplotlib.axes.AxesSubplot at 0x7f57e886f850>
```



```
In [21]: housing.hist(figsize=(20,15))

Out[21]: array([[<matplotlib.axes.AxesSubplot object at 0x7f57e803fc90>,
                 <matplotlib.axes.AxesSubplot object at 0x7f57e7676890>,
                 <matplotlib.axes.AxesSubplot object at 0x7f57e757c750>],
                [<matplotlib.axes.AxesSubplot object at 0x7f57e7564150>,
                 <matplotlib.axes.AxesSubplot object at 0x7f57e74d9fd0>,
                 <matplotlib.axes.AxesSubplot object at 0x7f57e7440cd0>],
                [<matplotlib.axes.AxesSubplot object at 0x7f57e73c5b90>,
                 <matplotlib.axes.AxesSubplot object at 0x7f57e737fc10>,
                 <matplotlib.axes.AxesSubplot object at 0x7f57e6aed710>]], dtype=object)
```



pandas.DataFrame.describe

`DataFrame.describe`(*percentiles=None*, *include=None*, *exclude=None*) [source]

Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding `NaN` values.

Analyzes both numeric and object series, as well as `DataFrame` column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

Parameters:

percentiles : *list-like of numbers, optional*

The percentiles to include in the output. All should fall between 0 and 1. The default is `[.25, .5, .75]`, which returns the 25th, 50th, and 75th percentiles.

include : *'all', list-like of dtypes or None (default), optional*

A white list of data types to include in the result. Ignored for `Series`. Here are the options:

- 'all' : All columns of the input will be included in the output.
- A list-like of dtypes : Limits the results to the provided data types. To limit the result to numeric types submit `numpy.number`. To limit it instead to object columns submit the `numpy.object` data type. Strings can also be used in the style of `select_dtypes` (e.g. `df.describe(include=['O'])`). To select pandas categorical columns, use `'category'`
- None (default) : The result will include all numeric columns.

exclude : *list-like of dtypes or None (default), optional,*

A black list of data types to omit from the result. Ignored for `Series`. Here are the options:

- A list-like of dtypes : Excludes the provided data types from the result. To exclude numeric types submit `numpy.number`. To exclude object columns submit the data type `numpy.object`. Strings can also be used in the style of `select_dtypes` (e.g. `df.describe(include=['O'])`). To exclude pandas categorical columns, use `'category'`
- None (default) : The result will exclude nothing.

Returns: *summary: Series/DataFrame of summary statistics*

See also: `DataFrame.count`, `DataFrame.max`, `DataFrame.min`, `DataFrame.mean`, `DataFrame.std`, `DataFrame.select_dtypes`

```python
In [23]: print (housing.describe())
```

```
          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
```

Ocean_Proximity

```
In [26]: print(housing["ocean_proximity"].value_counts())

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

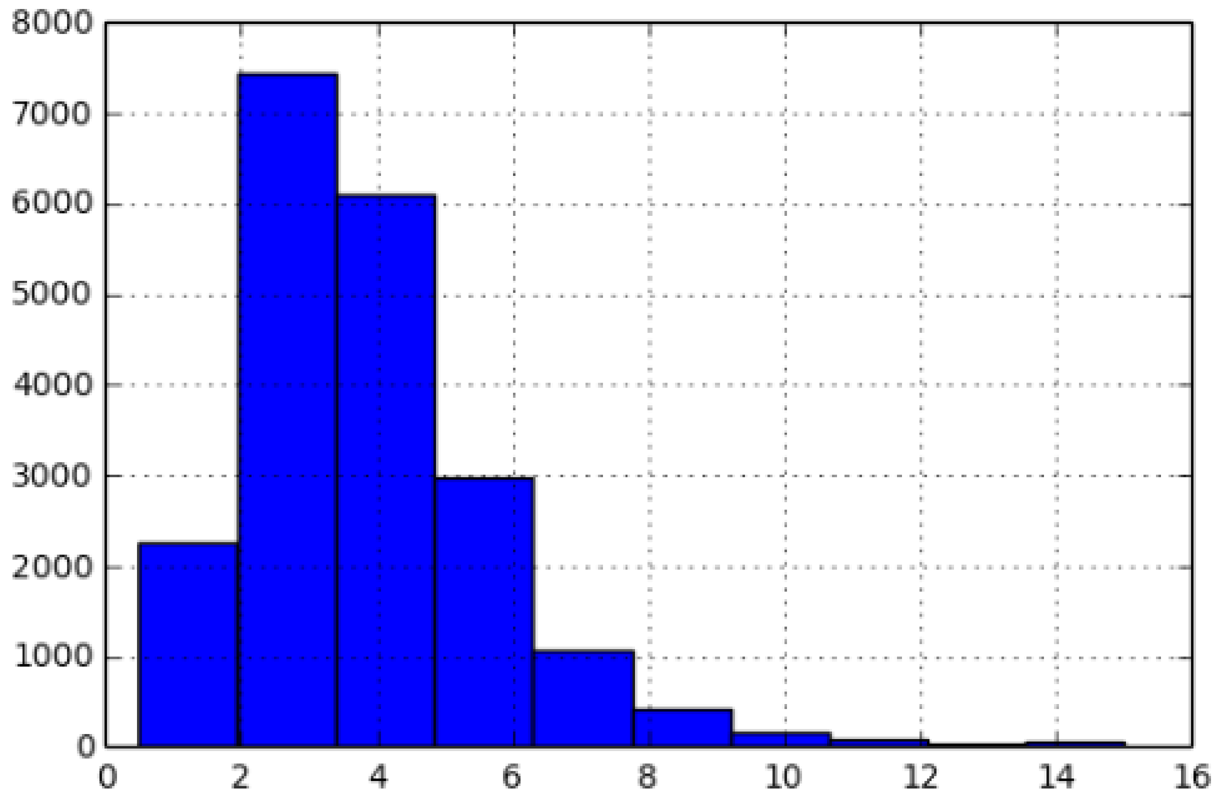
What Have We Noticed?

- Median income attribute does not look like it is expressed in US dollars (USD).

In [29]:
```
print(housing["median_income"].describe())
housing["median_income"].hist()
```

```
count    20640.000000
mean         3.870671
std          1.899822
min          0.499900
25%          2.563400
50%          3.534800
75%          4.743250
max         15.000100
Name: median_income, dtype: float64
```

Out[29]: <matplotlib.axes.AxesSubplot at 0x7f57e7b6b450>



What Have We Noticed?

- Median income attribute does not look like it is expressed in US dollars (USD).
- After checking with the team that collected the data, you are told that the data has been scaled and capped at
  - 15 (actually 15.0001) for higher median incomes,
  - 0.5 (actually 0.4999) for lower median incomes.
- Working with preprocessed attributes is common in Machine Learning, and it is not necessarily a problem, but you should try to understand how the data was computed.

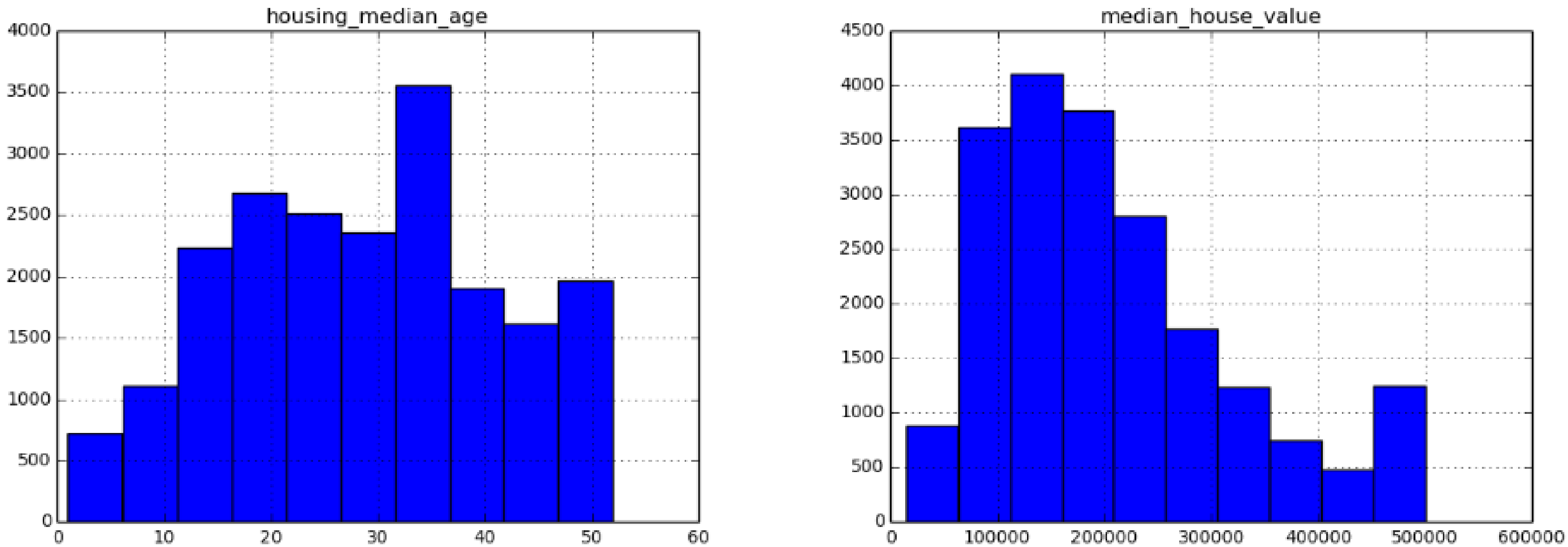
Housing Median Age / Median House Value

- The housing median age and the median house value were also capped.

In [40]:
```
print(housing[["housing_median_age", "median_house_value"]].describe())
housing[["housing_median_age", "median_house_value"]].hist(figsize=(15,5))
```

```
       housing_median_age  median_house_value
count        20640.000000        20640.000000
mean            28.639486       206855.816909
std             12.585558       115395.615874
min              1.000000        14999.000000
25%             18.000000       119600.000000
50%             29.000000       179700.000000
75%             37.000000       264725.000000
max             52.000000       500001.000000
```

Out[40]:
```
array([[<matplotlib.axes.AxesSubplot object at 0x7f57e4423c90>,
        <matplotlib.axes.AxesSubplot object at 0x7f57e4337890>]], dtype=object)
```



Median House Value Issue?

- The housing median age and the median house value were also capped.
- The latter may be a serious problem since it is your target attribute (your labels).
- Your Machine Learning algorithms may learn that prices never go beyond that limit.
- You need to check with your client team (the team that will use your system’s output) to see if this is a problem or not.
- If they tell you that they need precise predictions even beyond $500,000, then you have mainly two options:
  - Collect proper labels for the districts whose labels were capped.
  - Remove those districts from the training set (and also from the test set, since your system should not be evaluated poorly if it predicts values beyond $500,000).

Machine Learning Issues?

- These attributes have very different scales. We will discuss this later in this chapter when we explore feature scaling.
- Finally, many histograms are tail heavy:
  - they extend much farther to the right of the median than to the left.
- This may make it a bit harder for some Machine Learning algorithms to detect patterns.
- We will try transforming these attributes later on to have more bell-shaped distributions.

Pause!

- We've taken a quick look at data
- Geron recommends we quickly create a test set.
- Your brain is an amazing pattern detection system, which means that it is highly prone to overfitting:
  - if you look at the test set, you may stumble upon some seemingly interesting pattern in the test data that leads you to select a particular kind of Machine Learning model.
- When you estimate the generalization error using the test set, your estimate will be too optimistic and you will launch a system that will not perform as well as expected.
- This is called data snooping bias.

Scikit Learn

Scikit-Learn provides a few functions to split datasets into multiple subsets in various ways. The simplest function is `train_test_split`, which does pretty much the same thing as the function `split_train_test` defined earlier, with a couple of additional features. First there is a `random_state` parameter that allows you to set the random generator seed as explained previously, and second you can pass it multiple datasets with an identical number of rows, and it will split them on the same indices (this is very useful, for example, if you have a separate DataFrame for labels):

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

```
[52]: from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

print(train_set.describe())
print( "\nTraining Percentages:\n" )
print(train_set.count()/housing.count())
```

```
          longitude      latitude  housing_median_age   total_rooms  \
count  16512.000000  16512.000000        16512.000000  16512.000000
mean    -119.582290     35.643149           28.608285   2642.004784
std        2.005654      2.136665           12.602499   2174.646744
min     -124.350000     32.550000            1.000000      2.000000
25%     -121.810000     33.930000           18.000000   1454.000000
50%     -118.510000     34.260000           29.000000   2129.000000
75%     -118.010000     37.720000           37.000000   3160.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    16512.000000  16512.000000  16512.000000   16512.000000
mean       538.496851   1426.453004    499.986919       3.880754
std        419.007096   1137.056380    380.967964       1.904294
min          1.000000      3.000000      1.000000       0.499900
25%        296.750000    789.000000    280.000000       2.566700
50%        437.000000   1167.000000    410.000000       3.545800
75%        647.000000   1726.000000    606.000000       4.773175
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        16512.000000
mean        207194.693738
std         115622.626448
min          14999.000000
25%         119800.000000
50%         179850.000000
75%         265125.000000
max         500001.000000

Training Percentages:

longitude             0.800000
latitude              0.800000
housing_median_age    0.800000
total_rooms           0.800000
total_bedrooms        0.808105
population            0.800000
households            0.800000
median_income         0.800000
median_house_value    0.800000
ocean_proximity       0.800000
dtype: float64
```

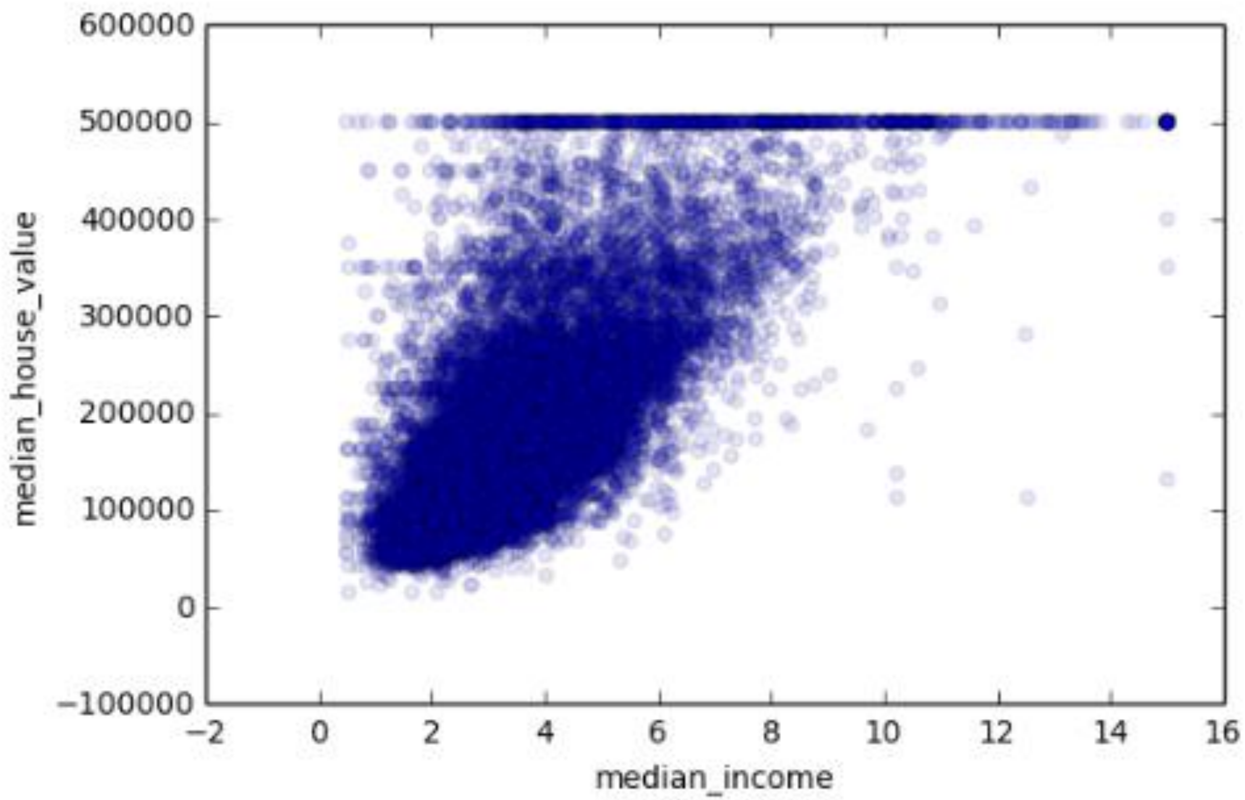
Insuring Good Distributions

Suppose you chatted with experts who told you that the median income is a very important attribute to predict median housing prices. You may want to ensure that the test set is representative of the various categories of incomes in the whole dataset. Since the median income is a continuous numerical attribute, you first need to create an income category attribute. Let's look at the median income histogram more closely (back in Figure 2-8): most median income values are clustered around $20,000–$50,000, but some median incomes go far beyond $60,000. It is important to have a sufficient number of instances in your dataset for each stratum, or else the estimate of the stratum's importance may be biased. This means that you should not have too many strata, and each stratum should be large enough. The following code creates an income category attribute by dividing the median income by 1.5 (to limit the number of income categories), and rounding up using `ceil` (to have discrete categories), and then merging all the categories greater than 5 into category 5:

```
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

Explore Data

```
In [53]: train_set.plot(kind="scatter", x="median_income", y="median_house_value",
                      alpha=0.1)

Out[53]: <matplotlib.axes.AxesSubplot at 0x7f57d291c210>

/usr/lib/pymodules/python2.7/matplotlib/collections.py:548: FutureWarning: e
ad, but in the future will perform elementwise comparison
  if self._edgecolors == 'face':
```



Prepare For Machine Learning Algorithms

- Write functions for several good reasons:
  - This will allow you to reproduce these transformations easily on any dataset (e.g., the next time you get a fresh dataset).
  - You will gradually build a library of transformation functions that you can reuse in future projects.
  - You can use these functions in your live system to transform the new data before feeding it to your algorithms.
  - This will make it possible for you to easily try various transformations and see which combination of transformations works best.

Preparation Tasks

- Cleaning Data
- Handling Text and Categorical Attributes
- Feature Scaling

Scikit-Learn Design Principles Consistency w/ Interface

- *Estimators*. Any object that can estimate some parameters based on a dataset is called an *estimator*
  - Estimation is performed by the fit()method, and it takes only a dataset as a parameter
    - or two for supervised learning algorithms; the second dataset contains the labels.
  - Any other parameter needed to guide the estimation process is considered a hyperparameter and it must be set as an instance variable (generally via a constructor parameter).
- *Transformers*. Some estimators can also transform a dataset;
  - Transformation is performed by the transform() method with the dataset a parameter.
  - It returns the transformed dataset.
  - Transformation generally relies on the learned parameters
  - All transformers also have a convenience method called fit_transform() that is equivalent to calling fit() and then transform()
- *Predictors*. Finally, some estimators are capable of making predictions given a dataset;
  - For example, the LinearRegression model
  - predictor has a predict() method that takes a dataset of new instances and returns a dataset of corresponding predictions.
  - It also has a score() method that measures the quality of the predictions given a test set (and the corresponding labels in the case of supervised learning algorithms)

Now In Kaggle