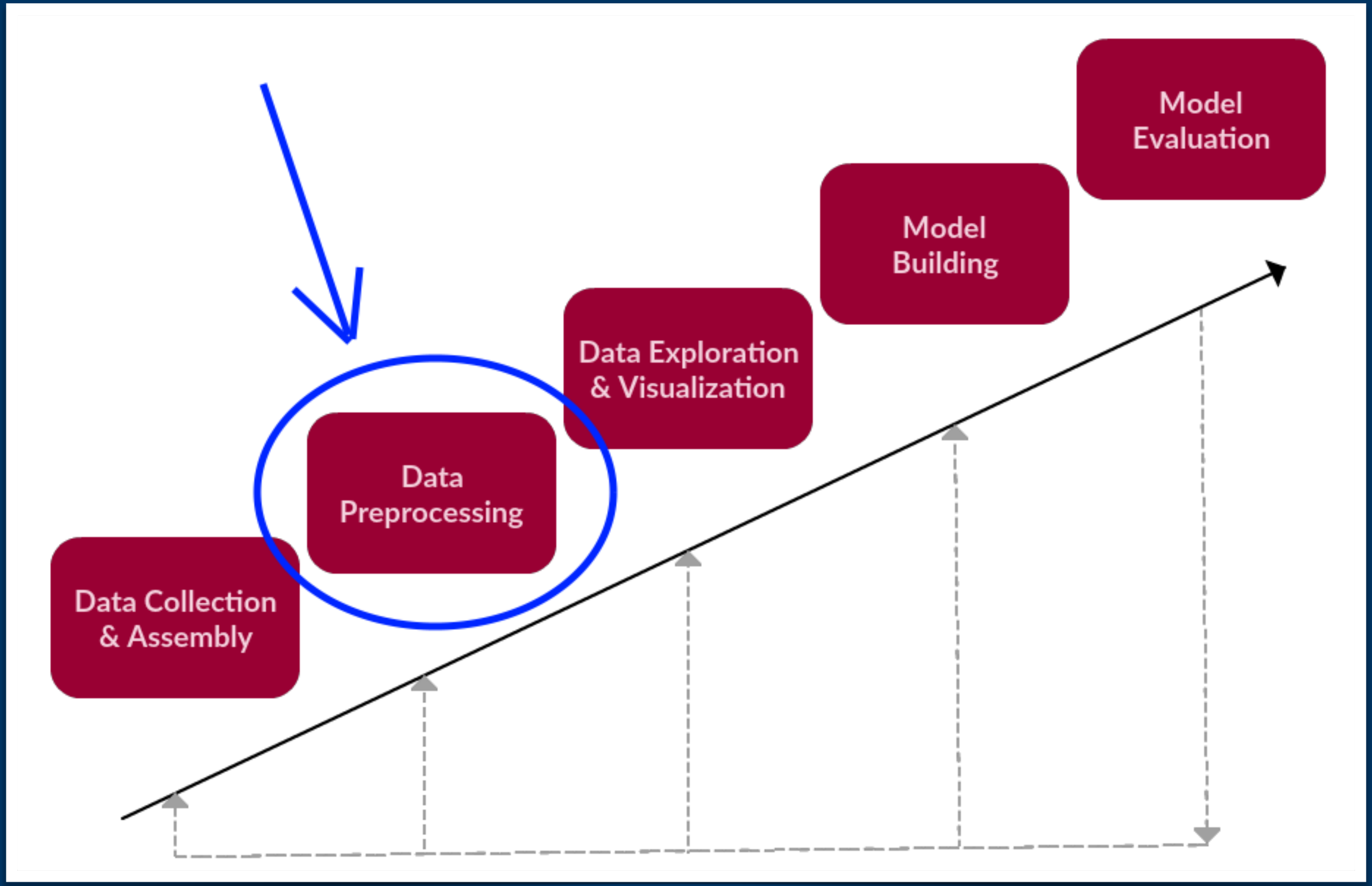


Data Pre-processing

CSCI-P556 Applied Machine Learning
Lecture 4

D.S. Williamson

Agenda and Learning Outcomes

Today’s Topics

- **Topics**:
  - Finish Data design discussion
  - Discuss ways to get data
  - Loading and initial analysis of data in Python
  - Data splitting

Data Selection for Experimental Design

Group practice - Zoom Break out rooms

- **Scenario**: You are a machine learning scientists and you want to develop an approach that can classify whether a given fruit on a farm is ripe or not. You want your approach to work for as many fruits and farms as possible.
- **Task**: You need to gather/collect data to train your system. In a group, think of the type of data that you need, including how/where/when it is collected or gathered from. Be sure that you consider the bad data problem.
    - In each group, one person take notes to summarize your approach
    - We'll come back and discuss in 5-10 minutes
- **Assumptions**:
    - You have the perfect classifier to use and metrics for evaluation
    - A drone can be used to fly over the farm(s) to see the fruits
    - Data will be annotated accurately once gathered

Data Selection for Experimental Design

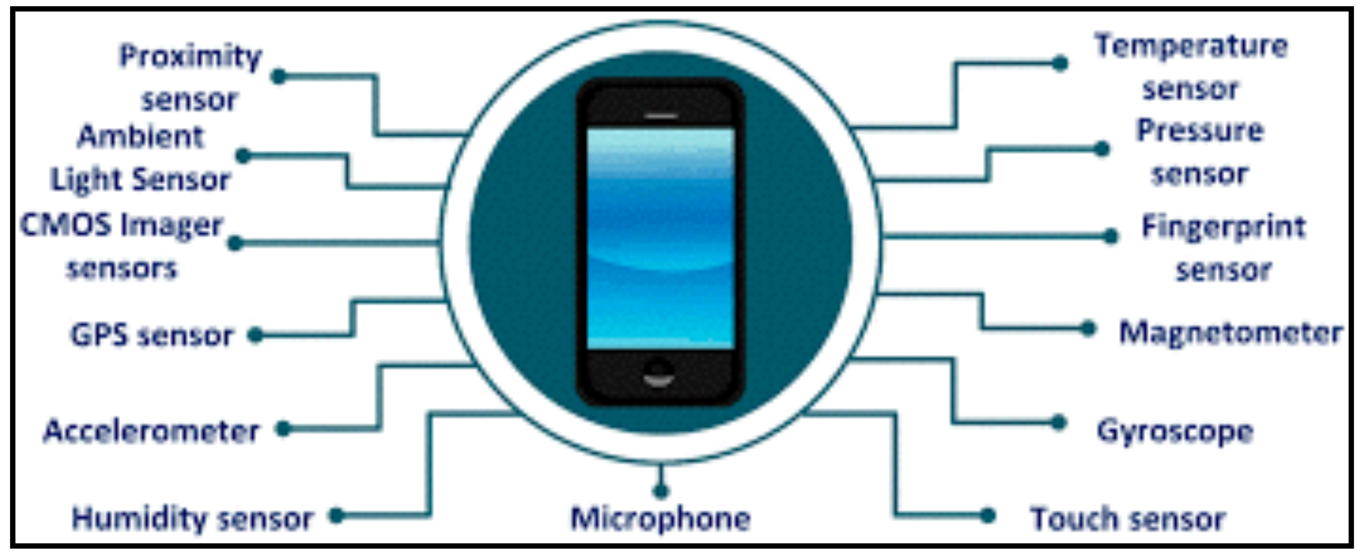
Questions to answer

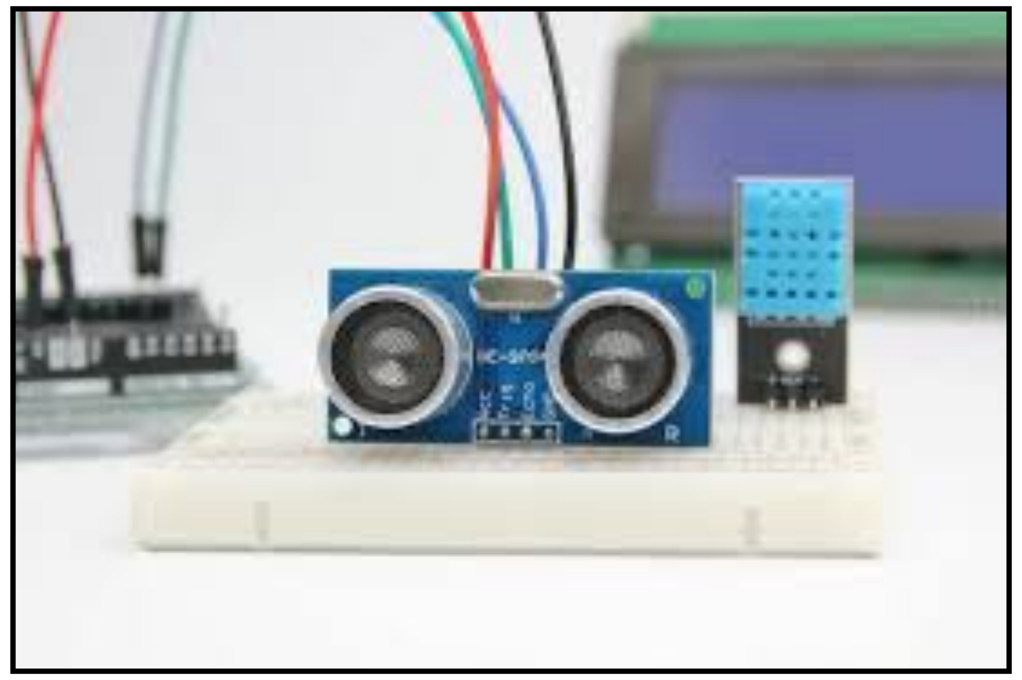
- What type of fruit do I need to consider?
- What seasons are these fruits generally available?
- What are physical indicators of ripeness for the different types of fruit?
- Will there be possible occlusions that impact captured data (e.g. people, machinery, weather,…)?
- How different are the farms in terms of layout and location?
- How many different angles, distances, and etc. between the fruits and drones should be considered?
- …

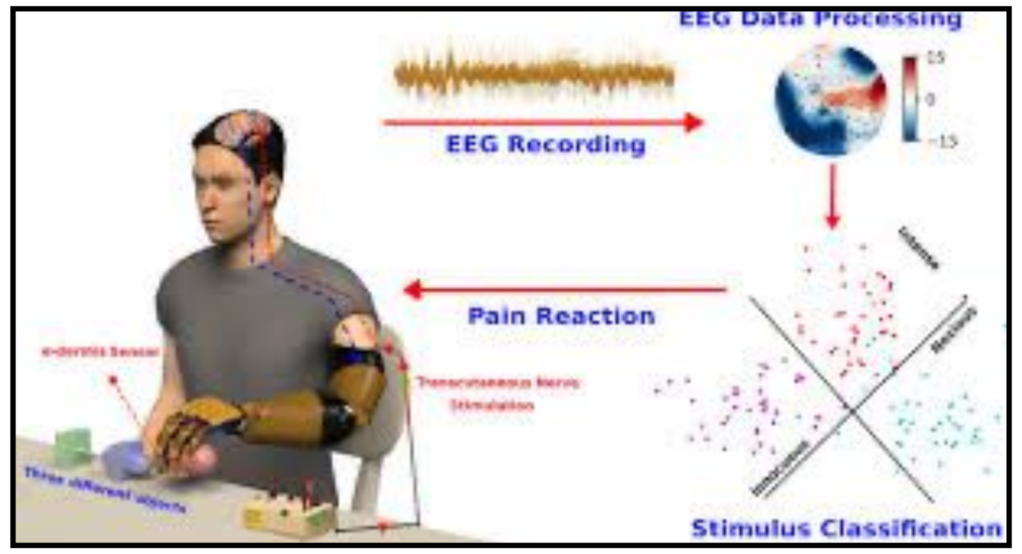
How do we get Data?

Collect it Ourselves

- Sensors for data collection are everywhere!
- Carefully designed experiments must be conducted.
- Must consider data challenges
  - **<u>Inaccurate, noisy and/or missing data</u>**: Over collect when possible
  - **<u>Data imbalance and bias</u>**: must ensure “enough” data from ALL classes, scenarios, and environments are collected







How do we get Data?

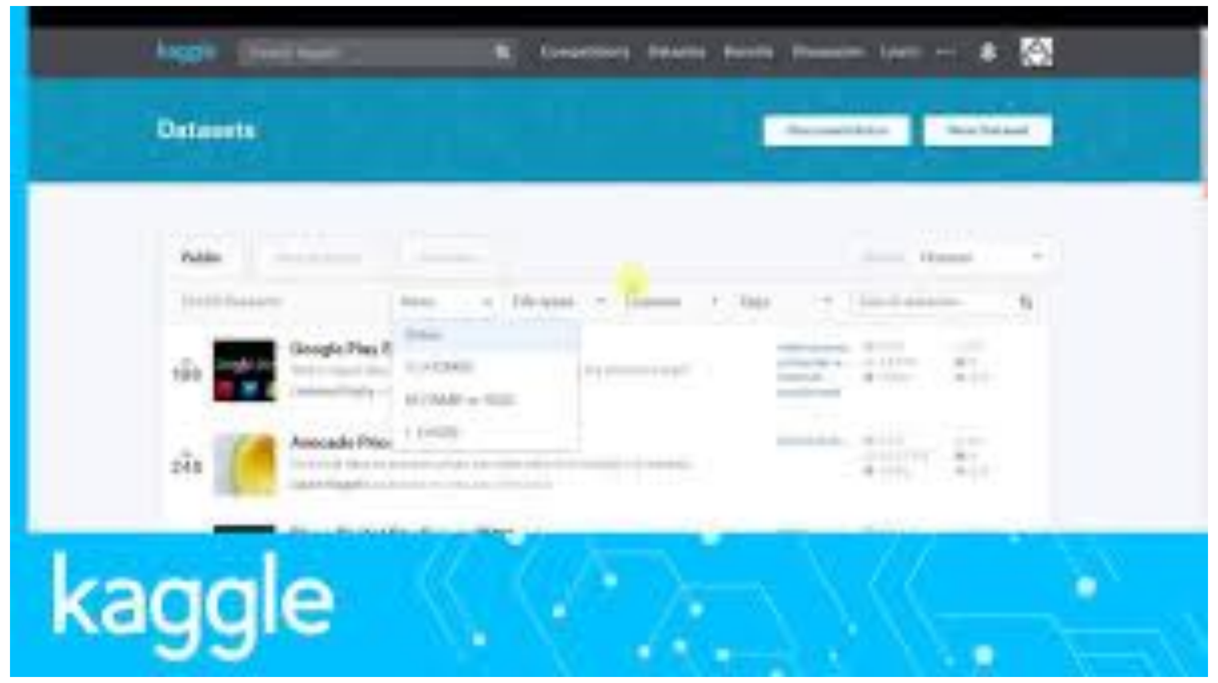
Collect it Ourselves

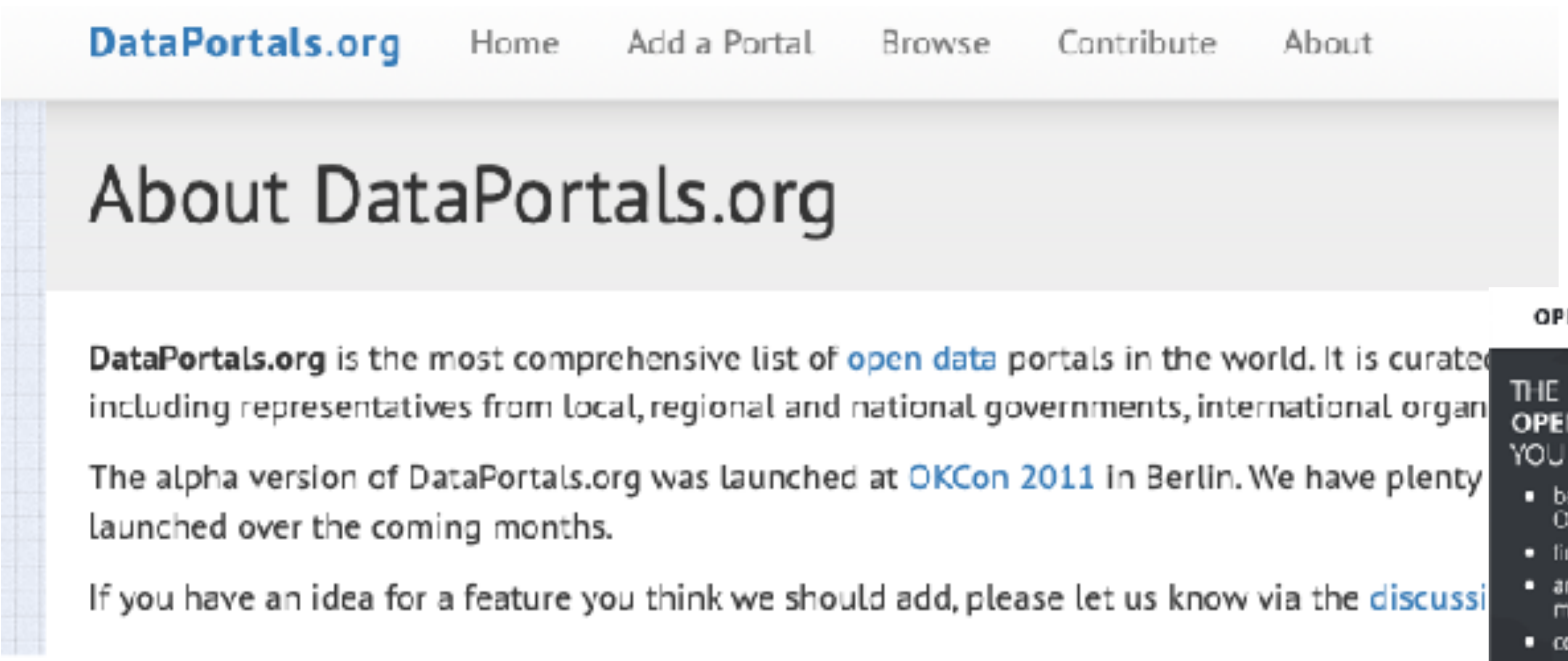
- Now that you have data, it must be annotated!
- This is the hard, costly and time-consuming part!
- For example, objects in pictures must be denoted. Transcriptions of audio must be provided
- You may have to do this yourself! Luckily online crowdsourcing has helped with this.
    - This is perfect, right?
    - Subject matter experts (Medical doctors, linguists, …) can provide this info too

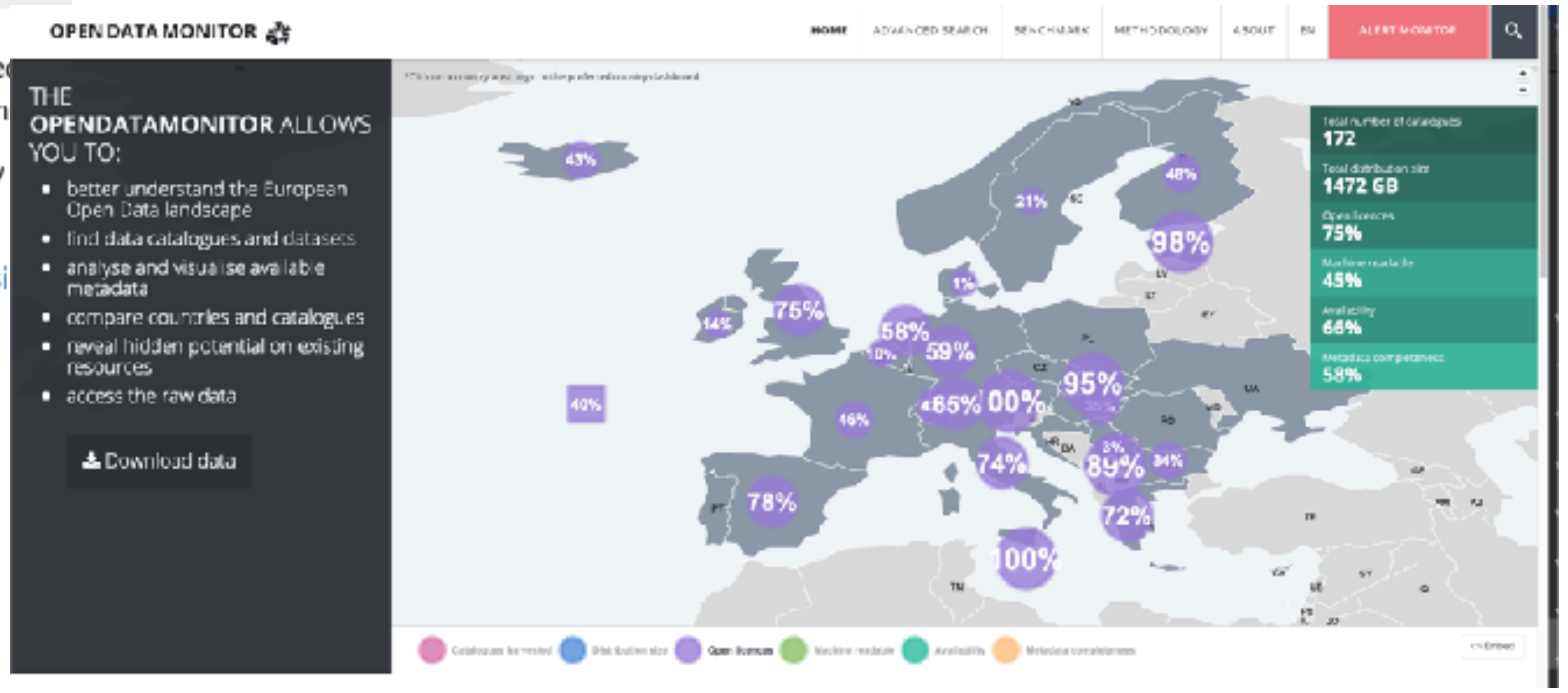
How do we get Data?

Online Sources

- Thankfully, there are many downloadable datasets that we can start with!







Data Pre-processing

Now that we have data, what's next? An Example Case

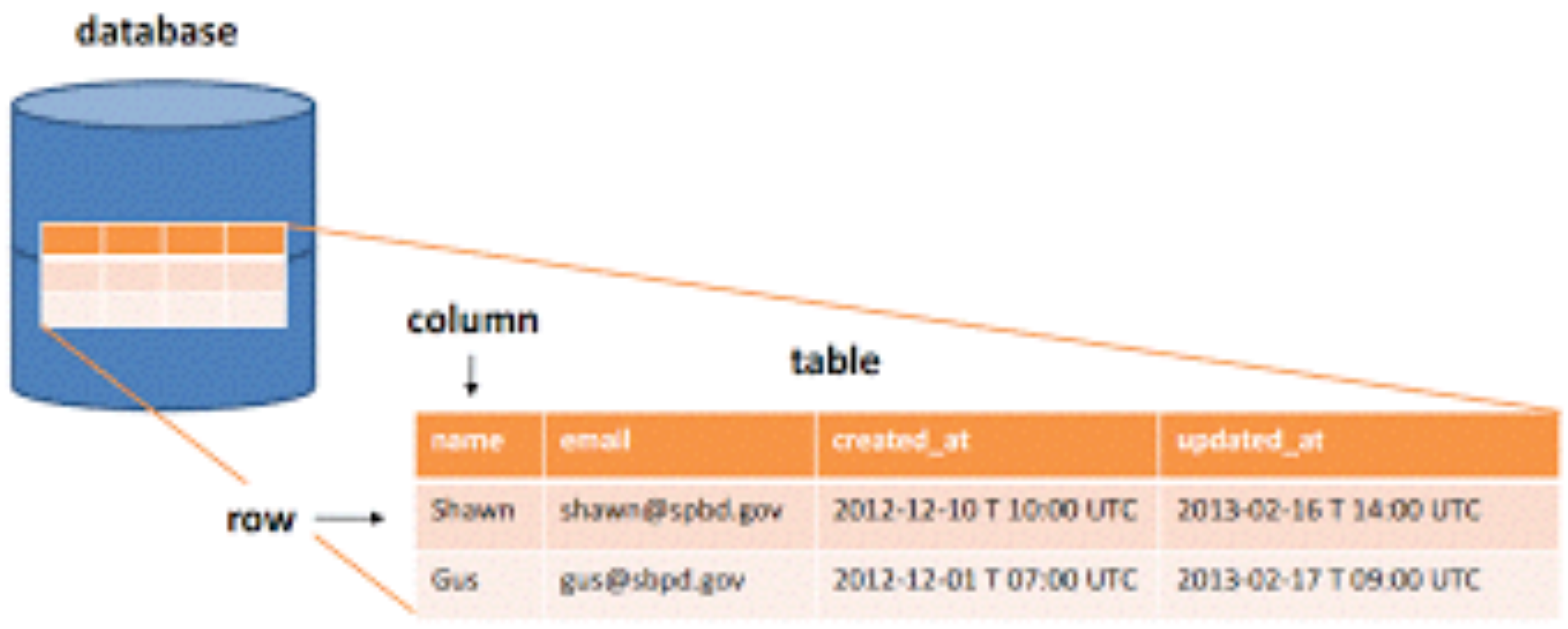
- Suppose you are a Data Scientist at a Housing Corporation. Your boss wants you to build a prediction model of median housing prices in California using their census data

- **Data has info about**: population, median income, median housing prices, … for each block group or district in California.

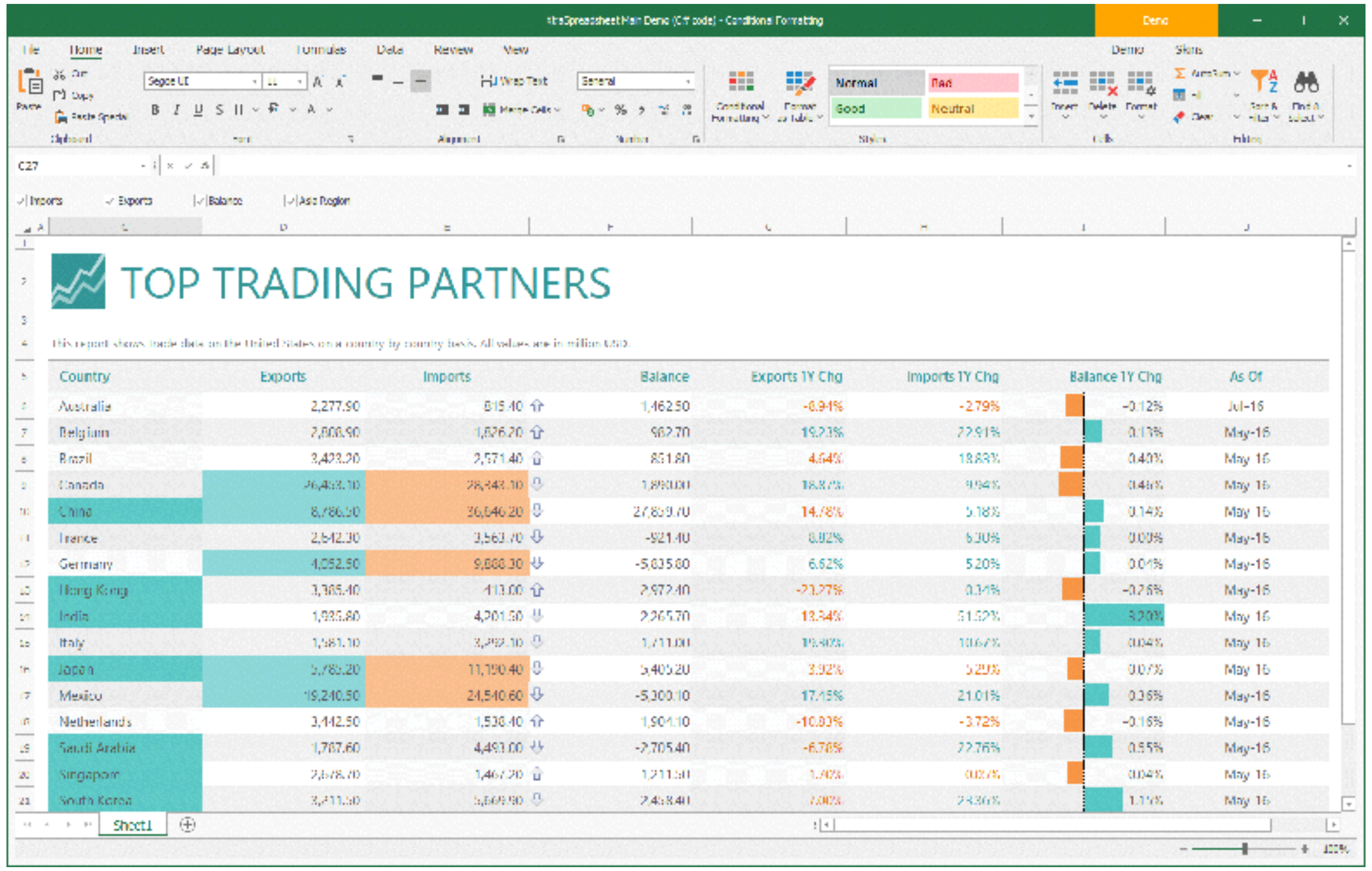
- **How should this problem be framed**?
  - Supervised Learning, Unsupervised learning, Reinforcement Learning? Why?
  - Classification, Regression, Other? Why?
  - Batch vs. Online?

Format of Data

Anything can be considered as data



- Data comes in multiple formats
  - Relational databases
  - Folders/files: 1 data, 1 for labels
  - Spreadsheets (Excel or CSV)
- Python can handle each option.
  - Today, we'll go through and example using a CSV file



1. Download data using a script

Python can work directly with tar files

```
In [2]: import os
        import tarfile
        from six.moves import urllib

        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"

        def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
            os.makedirs(housing_path, exist_ok=True)
            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()
```

Import needed modules

Define necessary paths

Create directory in workspace
Download tar file (housing.tgz)
Extract csv file (housing.csv)

```
In [3]: fetch_housing_data()
```

Call function when ready

2. Load Data using Pandas

Read CSV file

```
In [4]: import pandas as pd

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

Import module to use functions

Function to access CSV file and read

- We can now look at the top few rows using the DataFrame's head() method

```
In [5]: housing = load_housing_data()
        housing.head()
```

Out[5]:

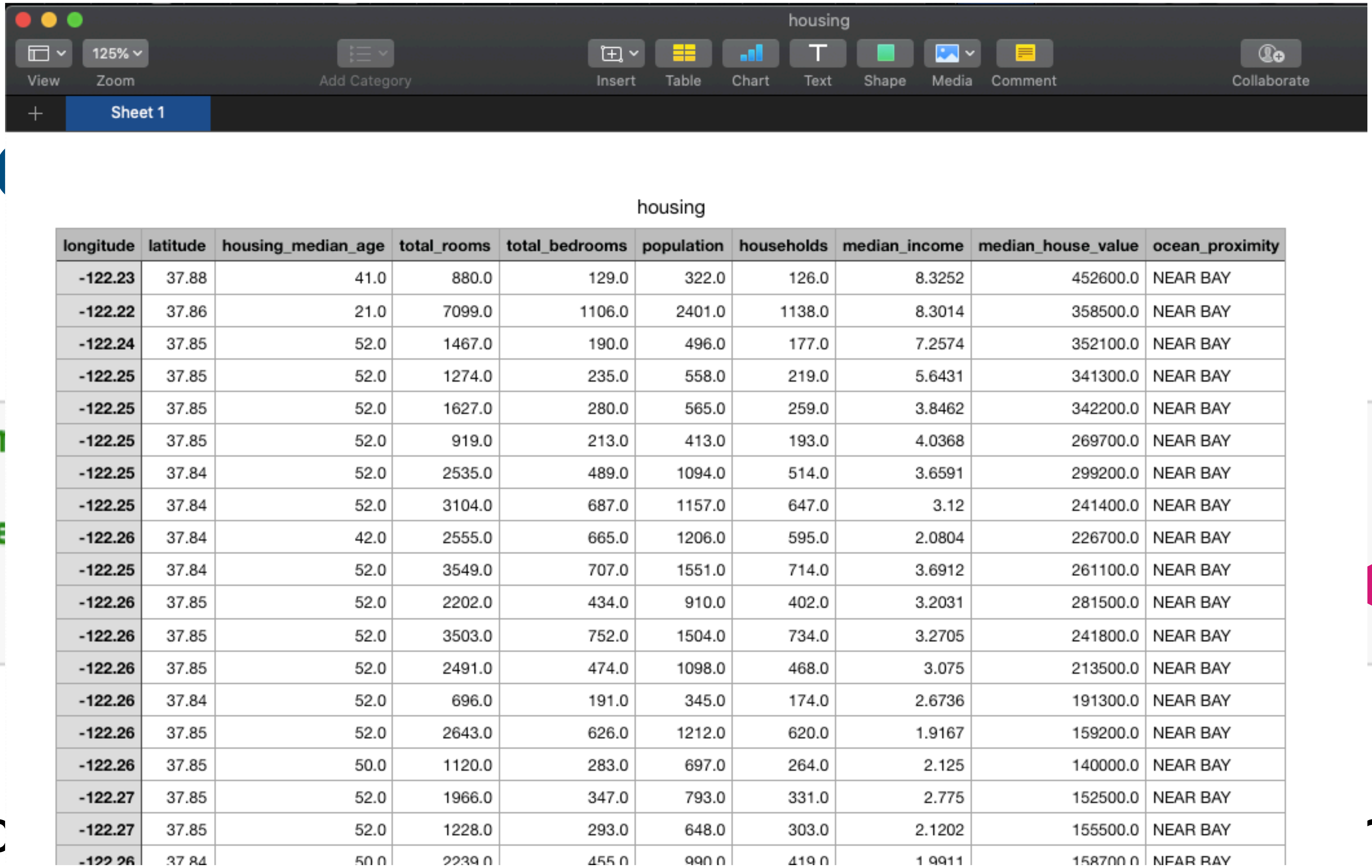
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

2. L

Read

In [4]: im

de



housing

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.12	241400.0	NEAR BAY
-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY
-122.26	37.85	52.0	2202.0	434.0	910.0	402.0	3.2031	281500.0	NEAR BAY
-122.26	37.85	52.0	3503.0	752.0	1504.0	734.0	3.2705	241800.0	NEAR BAY
-122.26	37.85	52.0	2491.0	474.0	1098.0	468.0	3.075	213500.0	NEAR BAY
-122.26	37.84	52.0	696.0	191.0	345.0	174.0	2.6736	191300.0	NEAR BAY
-122.26	37.85	52.0	2643.0	626.0	1212.0	620.0	1.9167	159200.0	NEAR BAY
-122.26	37.85	50.0	1120.0	283.0	697.0	264.0	2.125	140000.0	NEAR BAY
-122.27	37.85	52.0	1966.0	347.0	793.0	331.0	2.775	152500.0	NEAR BAY
-122.27	37.85	52.0	1228.0	293.0	648.0	303.0	2.1202	155500.0	NEAR BAY
-122.26	37.84	50.0	2239.0	455.0	990.0	419.0	1.9911	158700.0	NEAR BAY

Notice the contents of the CSV file and from the header match!

- We c ame's head() method

In [5]:
```
housing = load_housing_data()
housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

2. Load the Data

Examine data

- This data has ten attributes (e.g. observations/labels) for each housing district.

In [5]:
```
housing = load_housing_data()
housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Attributes/
Labels

3. Analyze the Data

Look at information

- Use info() to get information about the data, including formats of attributes/ labels

In [6]:
```
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
```

Number of data samples (e.g. entries)

Format of each attribute

total_bedrooms has null values (e.g. missing data)

?

3. Analyze the Data

Look at ocean_proximity feature

- Use info() to get information about the data, including formats of attributes/ labels

```
In [7]: housing["ocean_proximity"].value_counts()

Out[7]: <1H OCEAN     9136
        INLAND        6551
        NEAR OCEAN    2658
        NEAR BAY      2290
        ISLAND           5
        Name: ocean_proximity, dtype: int64
```

Find out possible values for this attribute the counts

- What happens if you do this for a different attribute? One with continuous values

```
In [9]: housing["households"].value_counts()
```

3. Analyze the Data

Summarize statistics of data by attributes

- Use describe() to show statistical values for each attribute

```
In [10]: housing.describe()
```

Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

3. Analyze the Data - Group Activity

Loot at the visual characteristics of the data

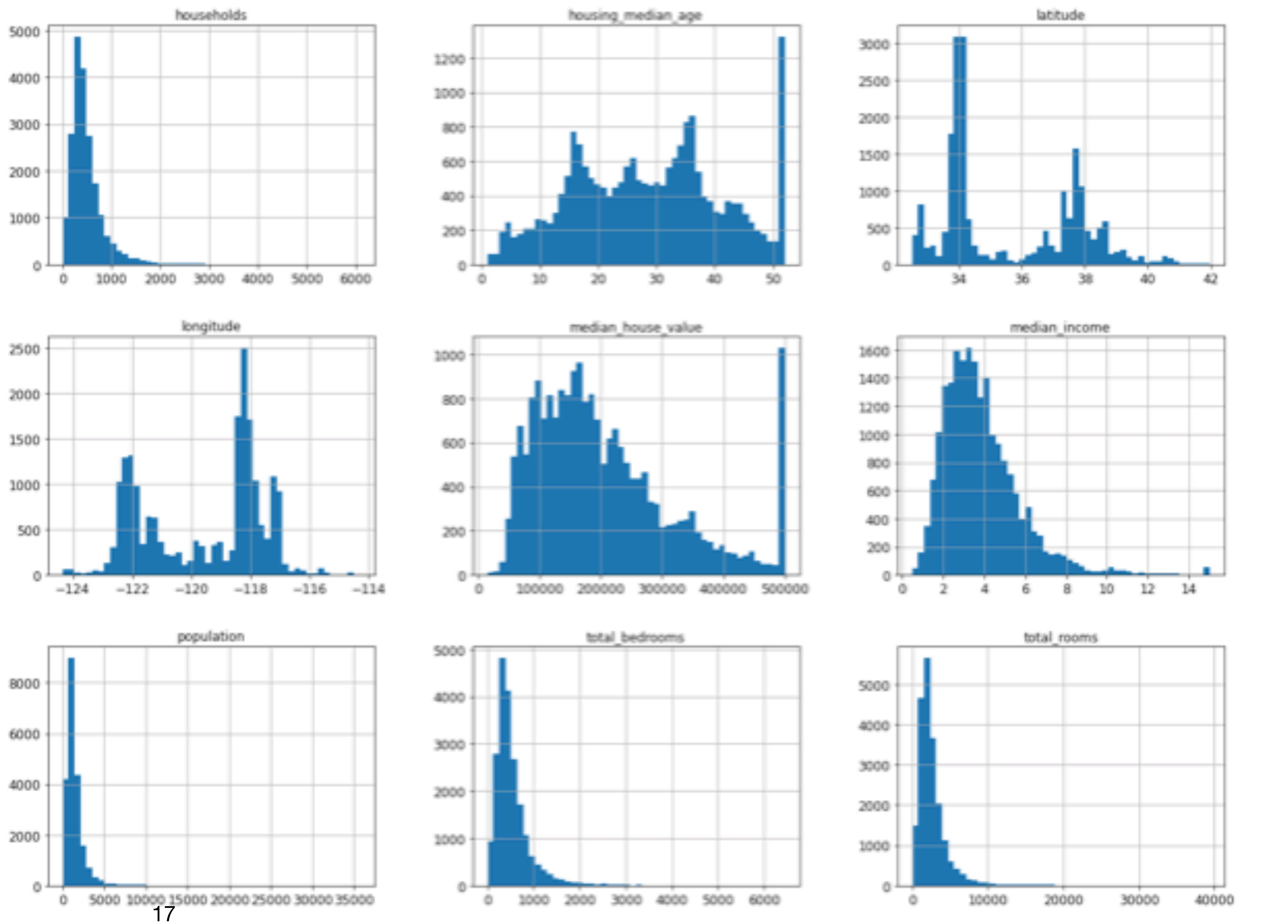
- Plot the histogram of the different attributes.

- Approximates the data distribution, which may come in handy later

In [12]:

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
#save_fig("attribute_histogram_plots")
plt.show()
```

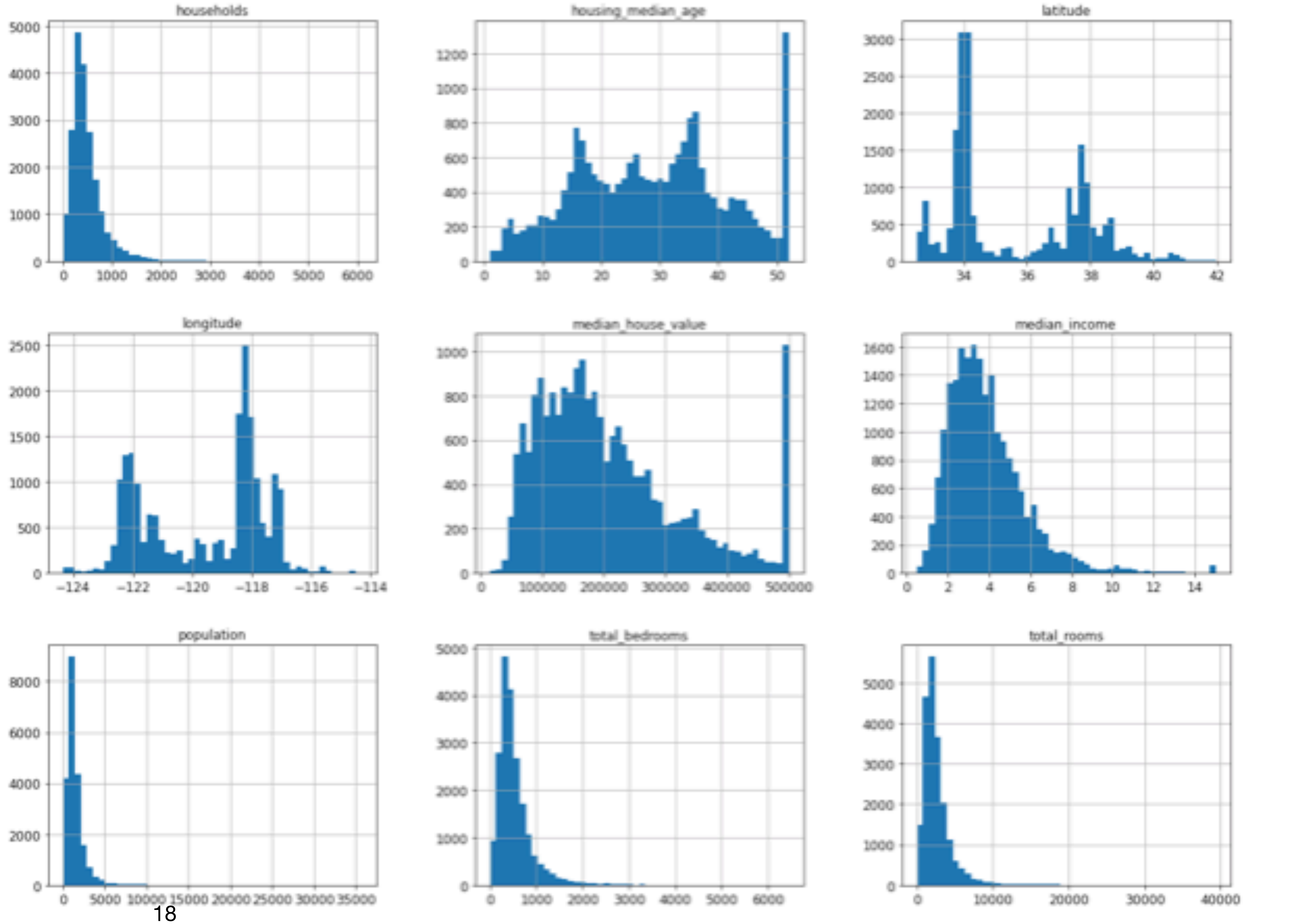
- What do the histograms tell you about the data? What ***bad data problems*** could this present?



3. Analyze the Data - Group Activity

Loot at the visual characteristics of the data

- Median age and house values were capped.
    - This may impact generalization
- Most attributes follow different “distributions”
- Four attributes have heavy tails.
    - May complicate ML
    - May need to be transformed



Supervised Learning

Split data into training and testing sets

- Train the learning algorithm using the training set
- Test generalization with the testing set. Don't peak!
- Designate a random percentage for testing, and 1 - percentage for training

- A third (development set) may also be needed. More on this later.

Splitting the data in Python

Simple Solution: Training and Testing Sets

```
In [13]: # to make this notebook's output identical at every run
         np.random.seed(42)
```

```
In [14]: import numpy as np

         # For illustration only. Sklearn has train_test_split()
         def split_train_test(data, test_ratio):
             shuffled_indices = np.random.permutation(len(data))
             test_set_size = int(len(data) * test_ratio)
             test_indices = shuffled_indices[:test_set_size]
             train_indices = shuffled_indices[test_set_size:]
             return data.iloc[train_indices], data.iloc[test_indices]
```

- Function for splitting data, given data and test ratio
- Randomly shuffle data (or indices) and select testing data
- Return testing and training sets

```
In [15]: train_set, test_set = split_train_test(housing, 0.2)
         print(len(train_set), "train +", len(test_set), "test")

         16512 train + 4128 test
```

Check Data Split

Splitting the data in Python

Scikit-Learn's Solution: Training and Testing Sets

In [17]:
```python
from sklearn.model_selection import train_test_split

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

In [18]:
```python
test_set.head()
```

Out[18]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250

Data Spliting using Random Sampling

- Any problems with randomly splitting the da

A Famous Example of Sampling Bias

Perhaps the most famous example of sampling bias happened during the US presidential election in 1936, which pitted Landon against Roosevelt: the *Literary Digest* conducted a very large poll, sending mail to about 10 million people. It got 2.4 million answers, and predicted with high confidence that Landon would get 57% of the votes.

24 | Chapter 1: The Machine Learning Landscape

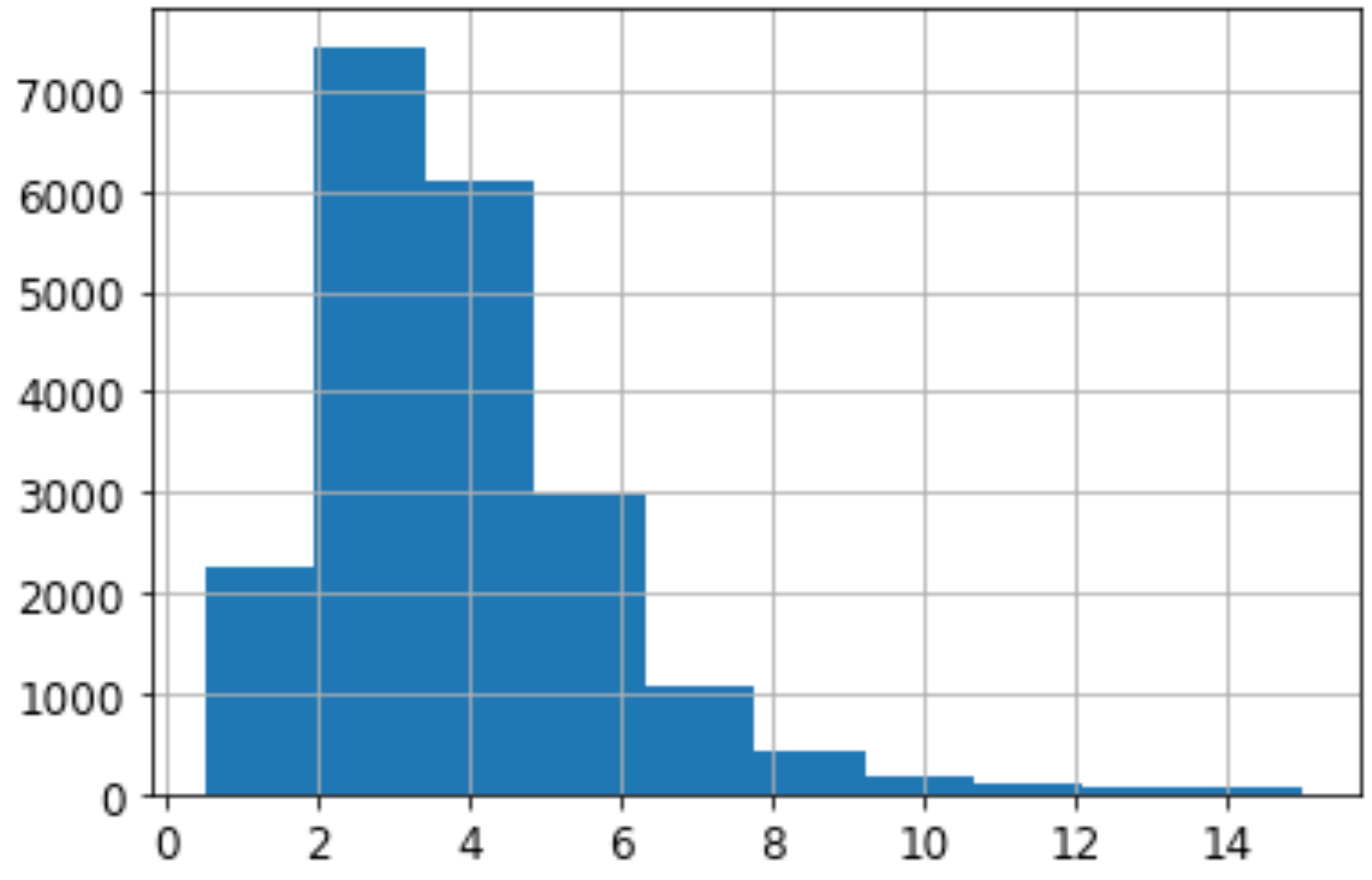
  - Potential for Sampling Bias
  - Need training/testing data to be representative
- Instead, maintain “appropriate and representative” ratios of data in both sets. This is called ***stratified sampling***, since the data is divided into homogenous subgroups called strata where the right number of instances is sampled from each stratum (or subgroup)
  - Let’s see this through an example

Stratified Sampling

Housing Example Continued

- Let's look at the "median_income" attribute

```
In [19]: housing["median_income"].hist()
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8ad8f104f0>
```

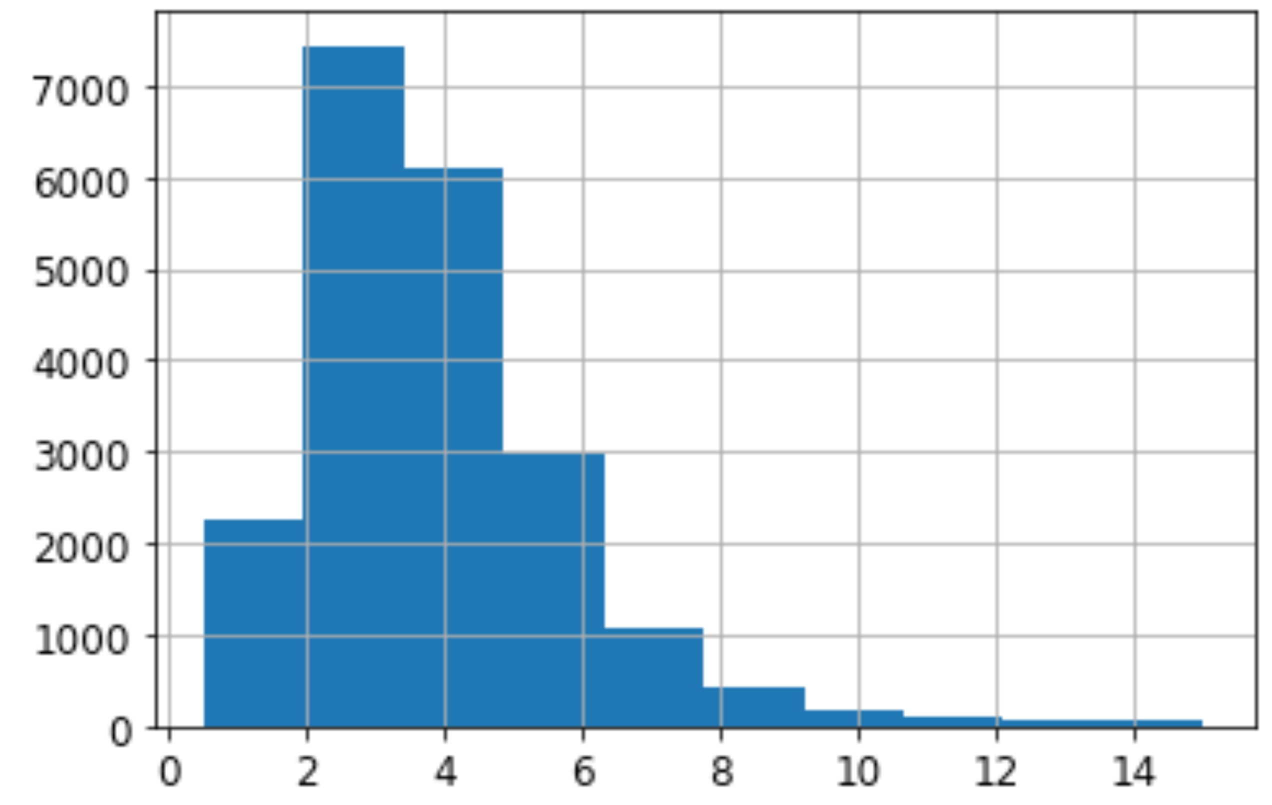


- Most data is between 2 and 5, but some goes beyond this
- Need instances from each stratum, or bias will occur

Stratified Sampling

Housing Example Continued

```
In [19]: housing["median_income"].hist()
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8ad8f104f0>
```



- We can: (1) Limit the number of strata and (2) Ensure each strata has enough examples (e.g. merge instances where income > 6 into one strata)

```
In [20]: 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])
```

Divide into Strata (using 1.5 spacing)

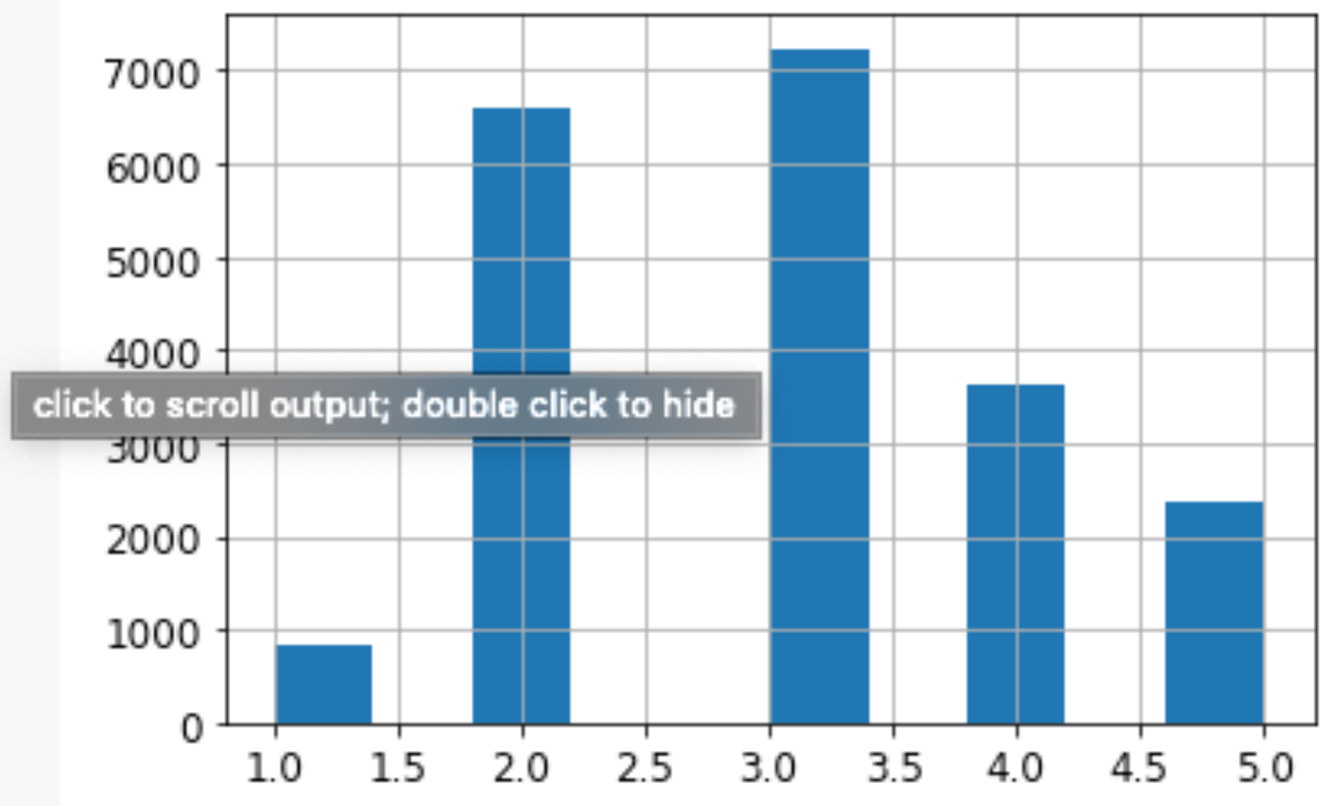
```
In [21]: housing["income_cat"].value_counts()
Out[21]: 3    7236
         2    6581
         4    3639
         5    2362
         1     822
         Name: income_cat, dtype: int64
```

Stratified Sampling

Housing Example Continued

- Finally performing stratified sampling

```
In [22]: housing["income_cat"].hist()
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8ad91709a0>
```




```
In [23]: from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

split.split(): Generate indices to split data into training and test sets

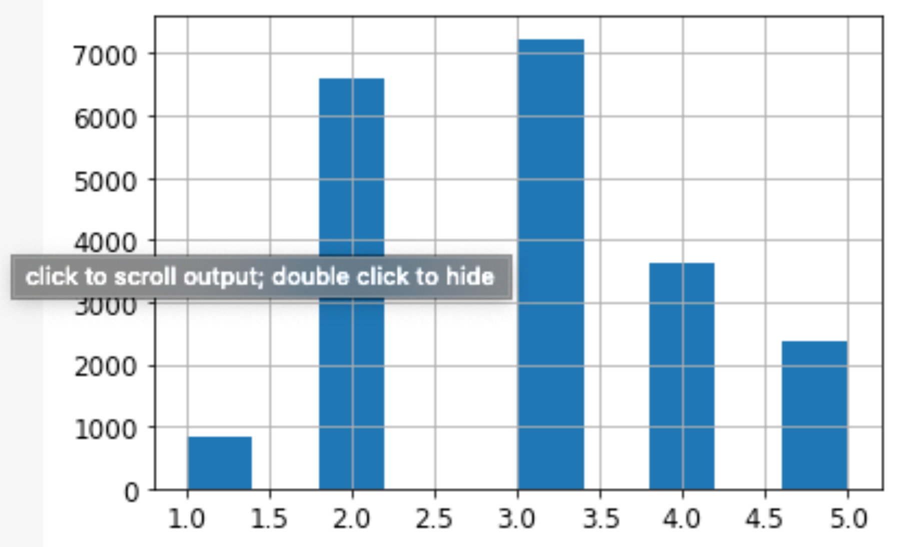
Specifies training data

Specifies variable/ attribute used for stratification

Stratified Sampling

Housing Example Continued

```
In [22]: housing["income_cat"].hist()
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8ad91709a0>
```



- Comparing data split for testing set, training set and original data

Percentages by incoming category match

```
In [24]: strat_test_set["income_cat"].value_counts() / len(strat_test_set)
Out[24]: 3    0.350533
         2    0.318798
         4    0.176357
         5    0.114583
         1    0.039729
         Name: income_cat, dtype: float64
```

```
In [26]: strat_train_set["income_cat"].value_counts() / len(strat_train_set)
Out[26]: 3    0.350594
         2    0.318859
         4    0.176296
         5    0.114402
         1    0.039850
         Name: income_cat, dtype: float64
```

```
In [25]: housing["income_cat"].value_counts() / len(housing)
Out[25]: 3    0.350581
         2    0.318847
         4    0.176308
         5    0.114438
         1    0.039826
         Name: income_cat, dtype: float64
```

Stratified Sampling

Housing Example Continued

- Remove stratified variable attribute “income_cat”, since we only used it to have representative data splits (we don’t really want to use it as an attribute)

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

In [44]:
```
strat_train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 17606 to 15775
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           16512 non-null  float64
 1   latitude            16512 non-null  float64
 2   housing_median_age  16512 non-null  float64
 3   total_rooms         16512 non-null  float64
 4   total_bedrooms      16354 non-null  float64
 5   population          16512 non-null  float64
 6   households          16512 non-null  float64
 7   median_income       16512 non-null  float64
 8   median_house_value  16512 non-null  float64
 9   ocean_proximity     16512 non-null  object
 10  income_cat          16512 non-null  category
dtypes: category(1), float64(9), object(1)
memory usage: 1.4+ MB
```

After Removal →

In [46]:
```
strat_train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 17606 to 15775
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           16512 non-null  float64
 1   latitude            16512 non-null  float64
 2   housing_median_age  16512 non-null  float64
 3   total_rooms         16512 non-null  float64
 4   total_bedrooms      16354 non-null  float64
 5   population          16512 non-null  float64
 6   households          16512 non-null  float64
 7   median_income       16512 non-null  float64
 8   median_house_value  16512 non-null  float64
 9   ocean_proximity     16512 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.4+ MB
```

Next Class

More on Data Pre-processing