

---

## End-to-End Machine Learning Project



# Project

---

## End-to-End Machine Learning Project

# Objective

---

- In this session
  - We'll walk you through with the complete cycle of a Machine Learning project
- It's okay if you do not understand few code snippets
  - We'll cover them in details as we go forward in the course

# Objective

---

Let's Start

# Prepare the Environment

---

- Open Jupyter in CloudxLab's "My Lab"
- New Terminal
- Clone the repository if not yet cloned
  - `git clone https://github.com/cloudxlab/ml.git`
- Else, update the repository
  - `cd ml && git pull origin master`
- Notebook for this project is located at
  - `machine_learning/end_to_end_project.ipynb`
- Goto Files

# Checklist for Machine Learning Projects

---

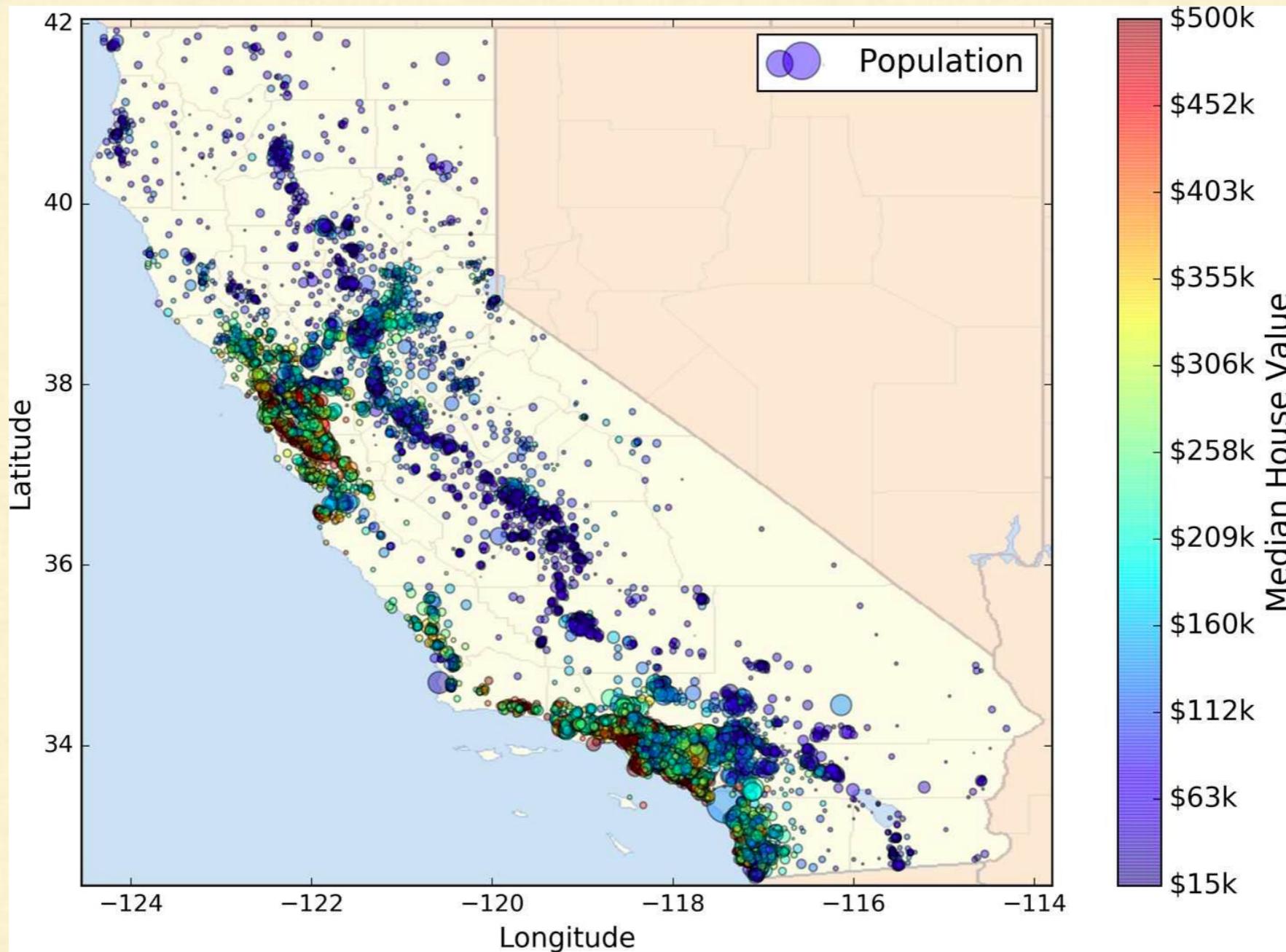
1. Frame the problem and look at the big picture
2. Get the data
3. Explore the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Explore many different models and short-list the best ones
6. Fine-tune model
7. Present the solution
8. Launch, monitor, and maintain the system

# End-to-End Machine Learning Project

---

Build a model of housing prices in California  
using the California census data

# End-to-End Machine Learning Project



Dataset is based on data from the 1990 California census  
Circle Size: Population, Price: Blue to Red

# Let's Have a Look at Data

---

```
>>> import pandas as pd  
>>> import os  
  
>>> HOUSING_PATH = 'datasets/housing/'  
  
>>> def load_housing_data(housing_path=HOUSING_PATH):  
    csv_path = os.path.join(housing_path,  
    "housing.csv")  
    return pd.read_csv(csv_path)  
  
>>> housing = load_housing_data()  
>>> housing.head()
```

Run this in notebook

# Checklist for Machine Learning Projects

---

- 1. Frame the problem and look at the big picture**
2. Get the data
3. Explore the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Explore many different models and short-list the best ones
6. Fine-tune model
7. Present the solution
8. Launch, monitor, and maintain the system

# Look at the Big Picture

---

- Dataset has following attributes for each block group in California
  - Population
  - Median income
  - Median housing price
  - And more....

# Look at the Big Picture

---

- 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
- Let's call them **districts**

# Look at the Big Picture

---

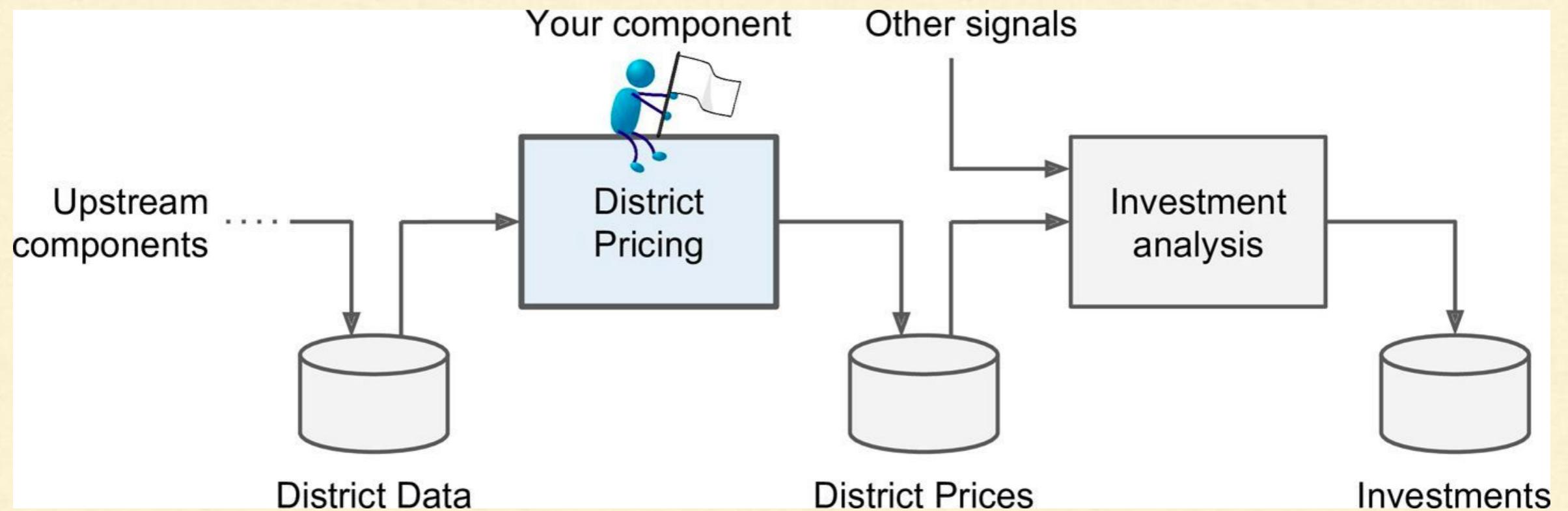
Our model should learn from this data and be able to predict  
the median housing price in any district

# Frame the Problem - Questions?

---

- What exactly is the business objective?
- How does the company expect to use and benefit from this model?
- Above questions helps in determining
  - How to frame the problem?
  - Which algorithm to select?
  - Performance measure to evaluate the model
  - Effort required to tweaking it

# Frame the Problem - Answers



# Frame the Problem - Answers

---

- Model's output (a prediction of a district's median housing price) will be fed
  - To another Machine Learning system
- The system will determine
  - Whether it is worth investing in a given area or not
  - Getting this right is critical as
  - It directly affects the revenue

# Frame the Problem - Current solution?

---

- What the current solution looks like (if any)
- It often gives a reference performance, as well as insights on how to solve the problem

# Frame the Problem - Answer

---

- District housing prices are currently estimated manually by experts
  - Team gathers up-to-date information about a district
  - Experts use complex rules to come up with an estimate
  - This is costly and time-consuming
  - Estimates are not great
  - Typical error rate is about 15%

# Frame the Problem - Type of Learning?

---

- Is this problem
  - I. Supervised, Unsupervised or Reinforcement Learning?
  - 2. Classification task, Regression task, or something else?
  - 3. Should we use batch learning or online learning techniques?

# Frame the Problem - Answers

---

## I. Supervised learning task

- Data has labeled training examples
- Each row in the data has the expected output(the district's median housing price)

## 2. Regression task

- We have to predict a value
- Multivariate regression
- System will use multiple features to make a prediction

# Frame the Problem - Answers

---

## 3. Should we use batch learning or online learning techniques?

- There is no continuous flow of data coming in the system
- Data is small enough to fit in memory
- Plain batch learning should do just fine

# Frame the Problem - Answers

---

## Select a Performance Measure - Root Mean Square Error

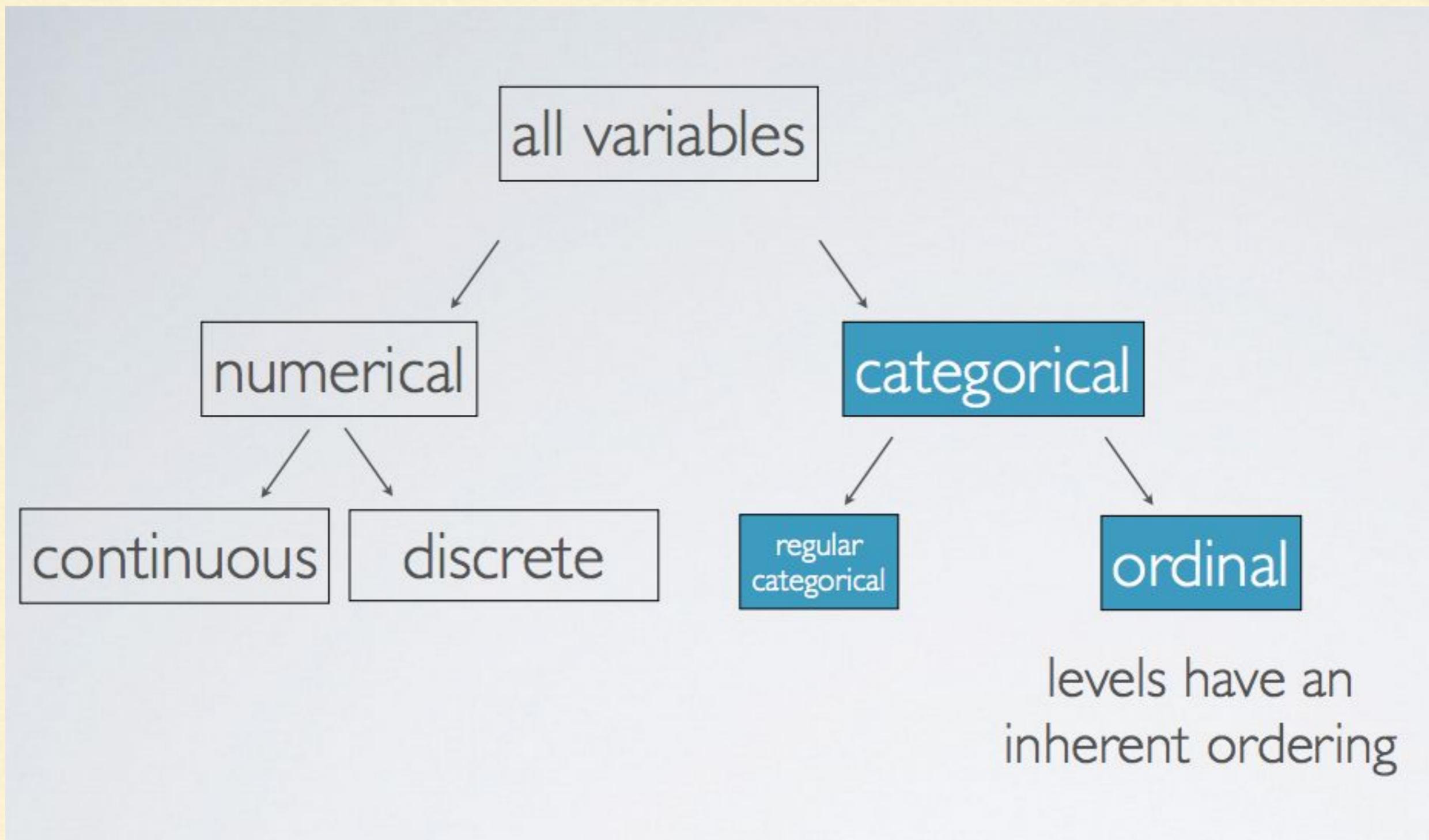
- Let's say we want to predict house value based on house area in square feet
- We will use univariate linear regression
  - Since there is only one feature **house area**

# Statistical Inference

---

Brief introduction to Statistical Inference

# Types of Variables

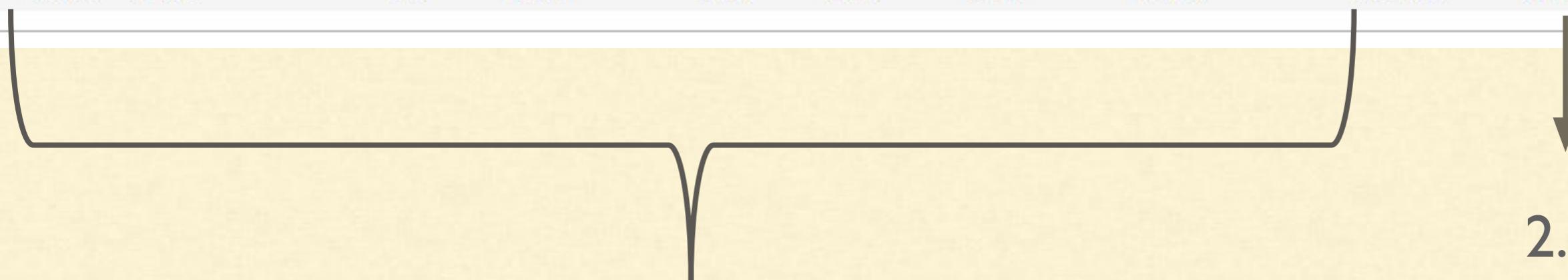


# Types of Variables

Each Column is Variable

Each Row is observation

	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



1. ??

2. ??

# Types of Variables

Each Column is Variable

Each Row is observation

	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



Numerical

Categorical

# Types of Variables

Each Column is Variable

Each Row is observation

country	cr_req	cr_comply	ud_req	ud_comply	...	hemisphere	hdi
Argentina	21	100	134	32	...	southern	very high
Australia	10	40	361	73	...	southern	very high
Belgium	<10	100	90	67	...	northern	very high
Brazil	224	67	703	82	...	southern	high
...	...	...	...	...	...	...	...
United States	92	63	5950	93	...	northern	very high

A. ??

B. ??

C. ??

D. ??

# Types of Variables

Each Column is Variable

Each Row is observation

country	cr_req	cr_comply	ud_req	ud_comply	...	hemisphere	hdi
Argentina	21	100	134	32	...	southern	very high
Australia	10	40	361	73	...	southern	very high
Belgium	<10	100	90	67	...	northern	very high
Brazil	224	67	703	82	...	southern	high
...	...	...	...	...	...	...	...
United States	92	63	5950	93	...	northern	very high

Discrete numerical

Continuous Numerical

Regular  
Categorical

Ordinal  
Categorical

# Random Process

In a random process we know what outcomes could happen, but we don't know which particular outcome will happen.



# Probability

---

$P(A)$  = Probability of event A

$$0 \leq P(A) \leq 1$$

# Probability

---

## Question -

In a village, there has been a tradition that people keep on having children till they get a boy. What will be the ratio of male to female in that village?

# Probability

---

**Answer -**

| : |

# Probability - example

What is probability that the sum of pair of fair dice when rolled is 4?



# Probability - example

What is probability that the sum of pair of fair dice when rolled is 4?



Total Possible ways in which the pair can be rolled = ??

Instances when the sum is 4 = ??

Probability = ??

# Probability - example

What is probability that the sum of pair of fair dice when rolled is 4?



Total Possible ways in which the pair can be rolled =  $6 \times 6 = 36$

Instances when the sum is 4 = ??

Probability = ??

# Probability - example

What is probability that the sum of pair of fair dice when rolled is 4?



Total Possible ways in which the pair can  
be rolled =  $6 \times 6 = 36$

Instances when the sum is 4 = (1,3), (3,1),  
(2,2) = 3

Probability = ??

# Probability - example

What is probability that the sum of pair of fair dice when rolled is 4?



Total Possible ways in which the pair can  
be rolled =  $6 \times 6 = 36$

Instances when the sum is 4 = (1,3), (3,1),  
(2,2) = 3

Probability =  $3/36 = 1/12$

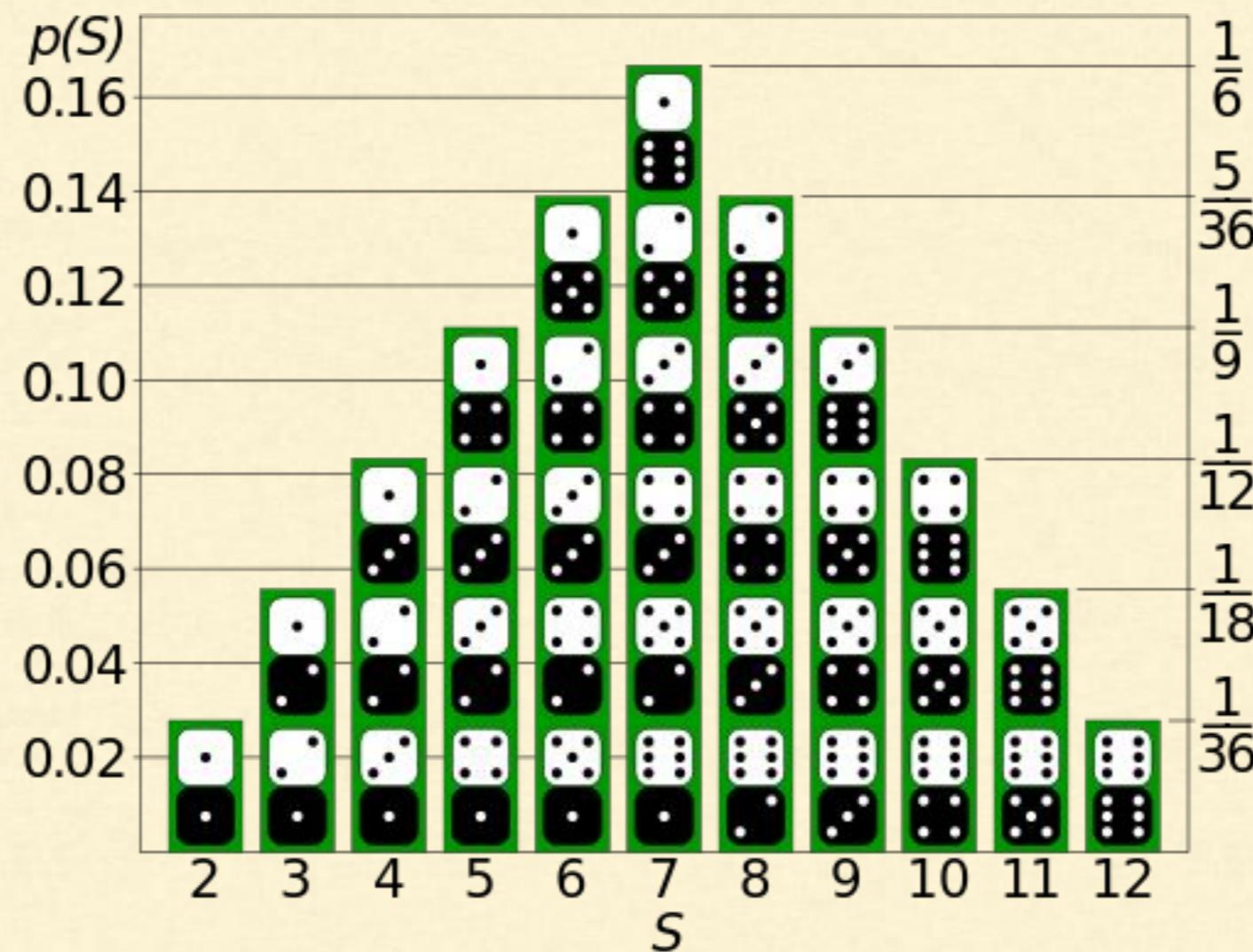
# Random Variable

---

- Random variable is a variable
  - Whose value is unknown or
  - A function that assigns values to each of an experiment's outcomes
- Can be of multiple types
- Depending upon the type of the quantity measured i.e.
  - Continuous
  - Discrete
  - Categorical etc

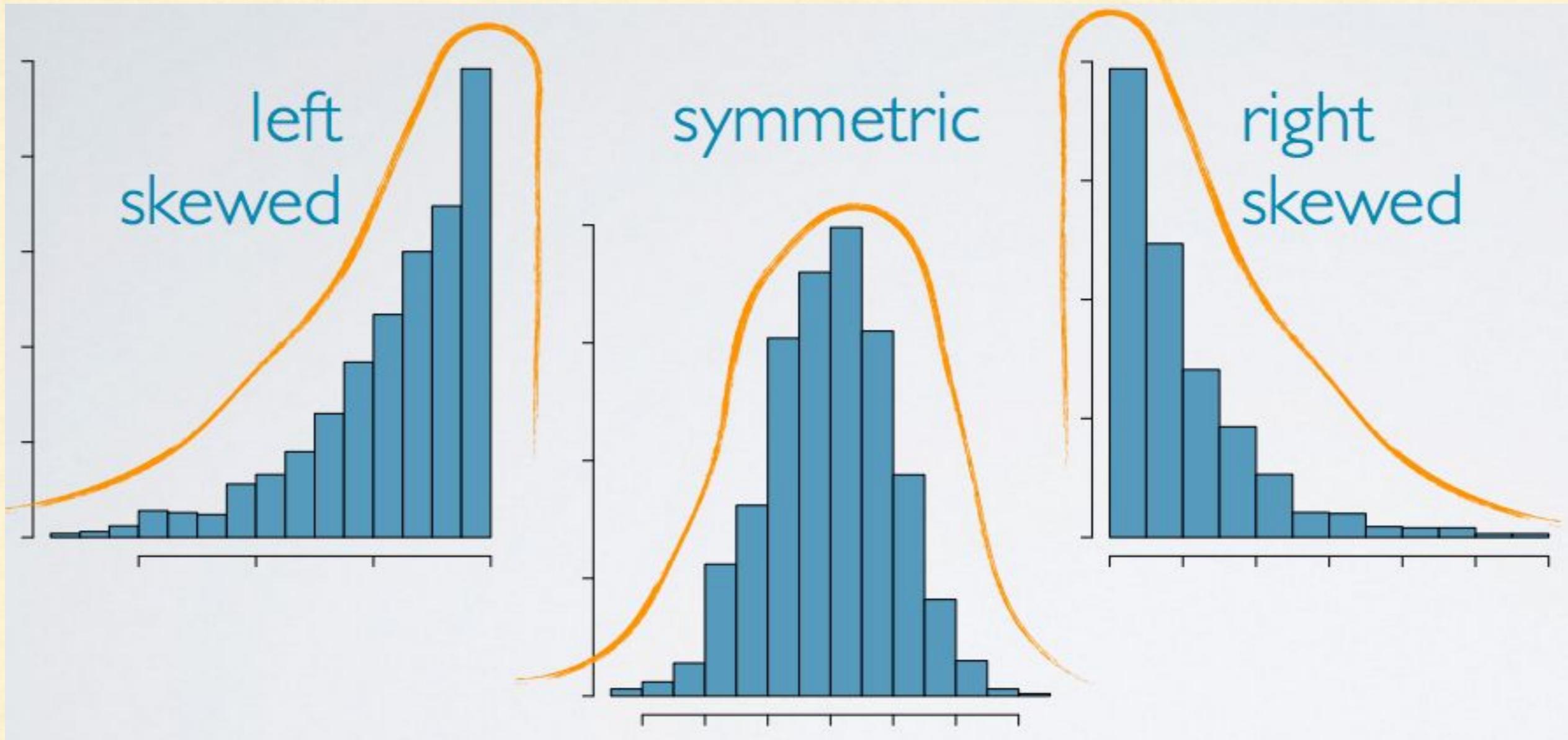
# Probability Distribution

- A probability distribution assigns a
  - Probability to each of the possible outcomes of
  - An random experiment, survey, or procedure of statistical inference



# Normality - Skewness

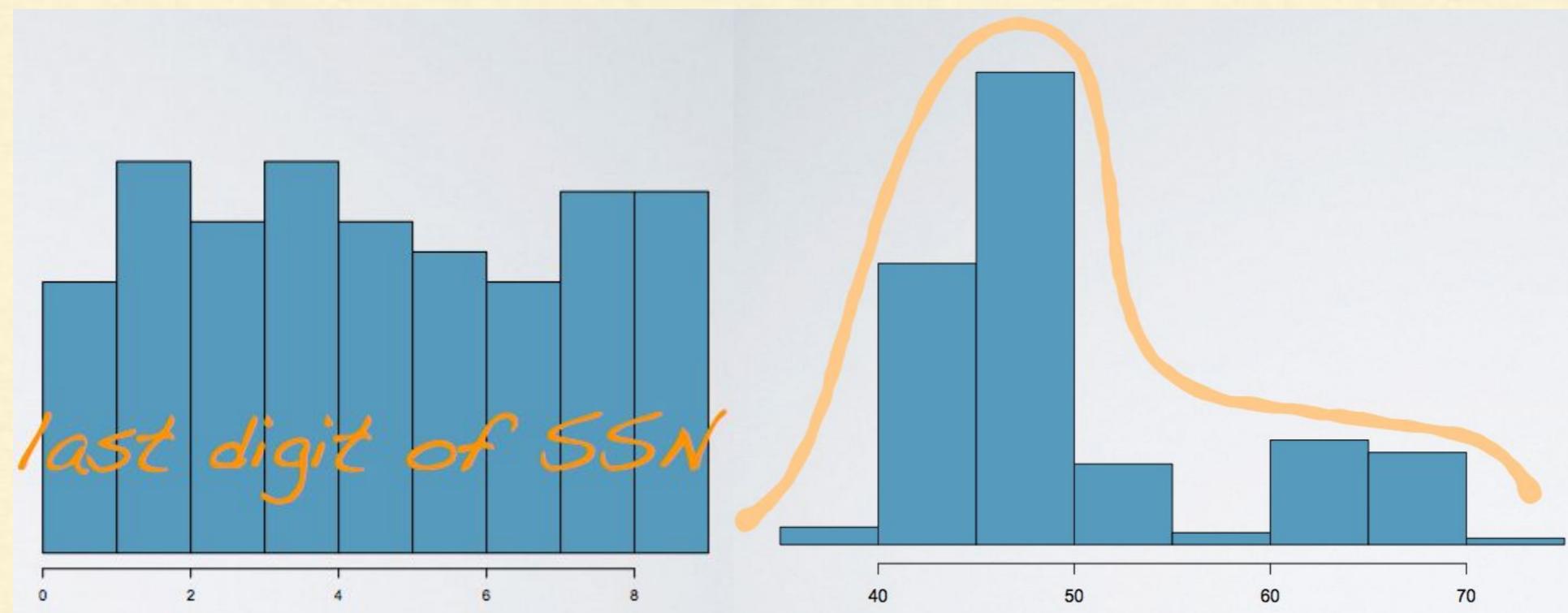
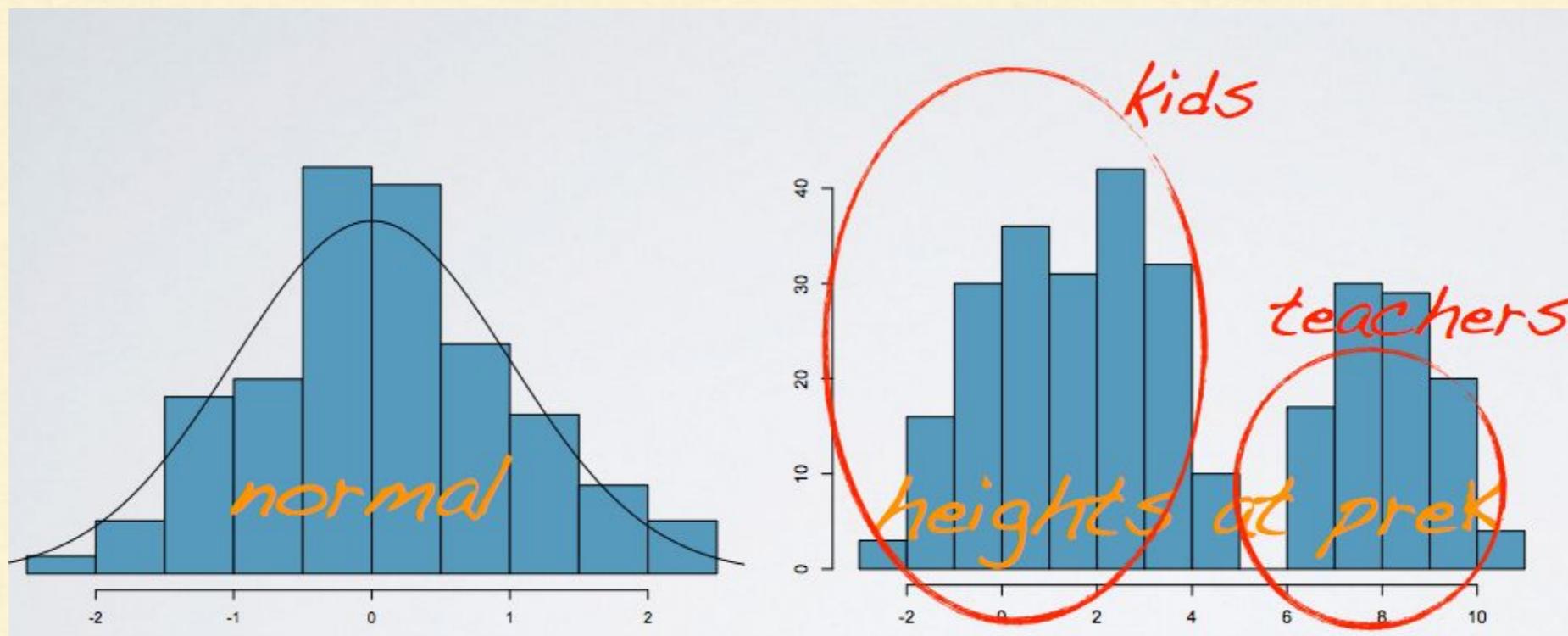
Distributions are skewed to the side of the long tail



# Normality - Modality

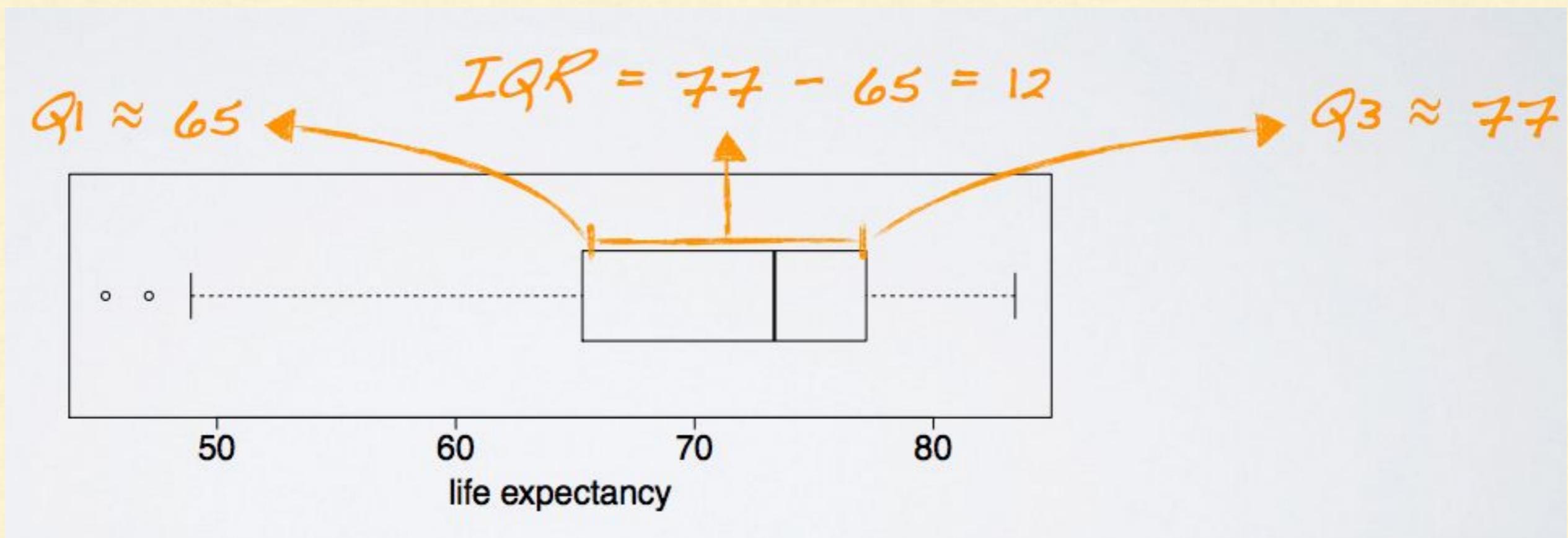


# Normality - Modality

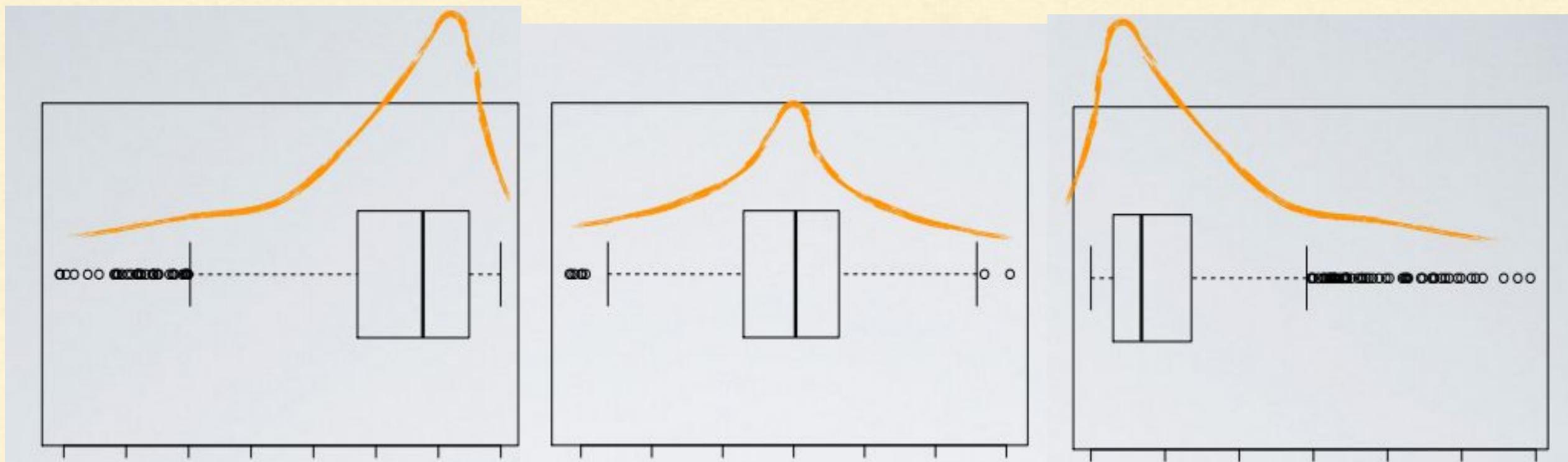


# Box Plot

Range of the middle 50% of the data, distance between the first quartile (25th percentile) and third quartile (75th percentile)



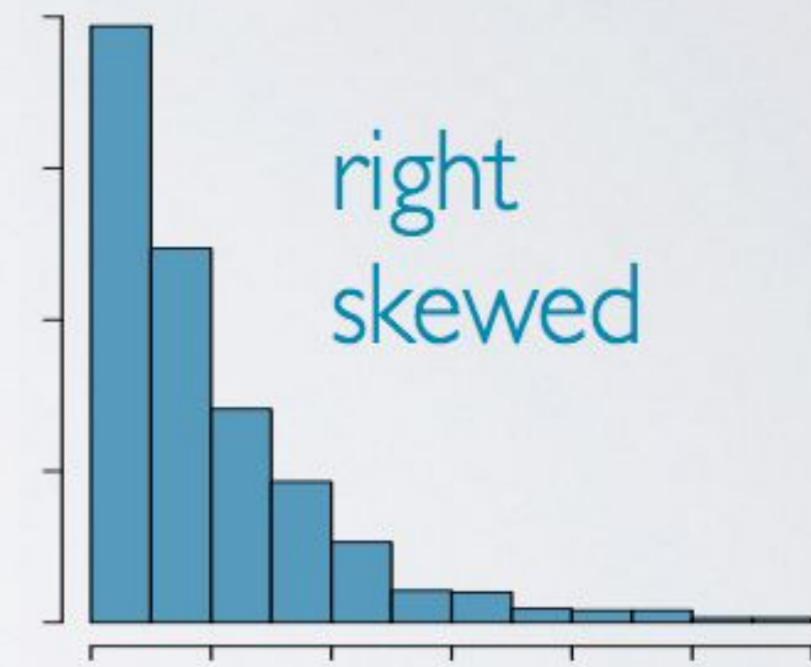
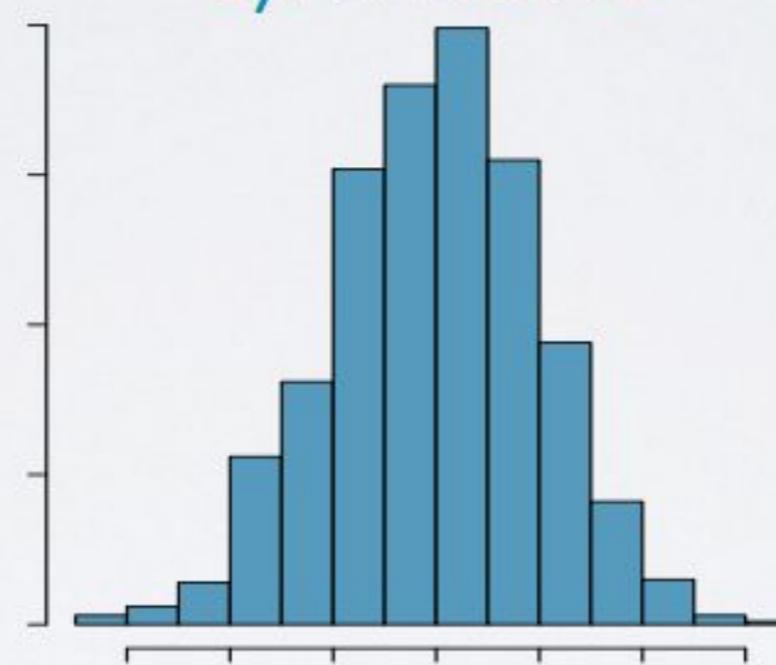
# Normality - Box Plots



symmetric

left  
skewed

right  
skewed



# Measures of Central Tendencies

---

How do you define center of certain data?

# Measures of Central Tendencies

## mean

arithmetic average

$\bar{x}$  sample mean

$\mu$  population mean

## median

midpoint of the distribution  
(50th percentile)

## mode

most frequent observation

sample statistic

point estimate

population parameter

# Measures of Central Tendencies

---

Which one would you choose?  
Mean, Median or mode?

# Measures of Central Tendencies

75, 69, 88, 93, 95, 54, 87, 88, 27

mean:  $\frac{75+69+88+93+95+54+87+88+27}{9} = 75.11$

mode: 88

median: 27, 54, 69, 75, 87, 88, 88, 93, 95

# Robust Statistics

Measures on which extreme observations have little effect

example

data	mean	median
1, 2, 3, 4, 5, 6	3.5	3.5
1, 2, 3, 4, 5, 1000	169	3.5

	robust	non-robust
center	median	mean
spread	IQR	SD, range

*skewed,  
with extreme  
observations*

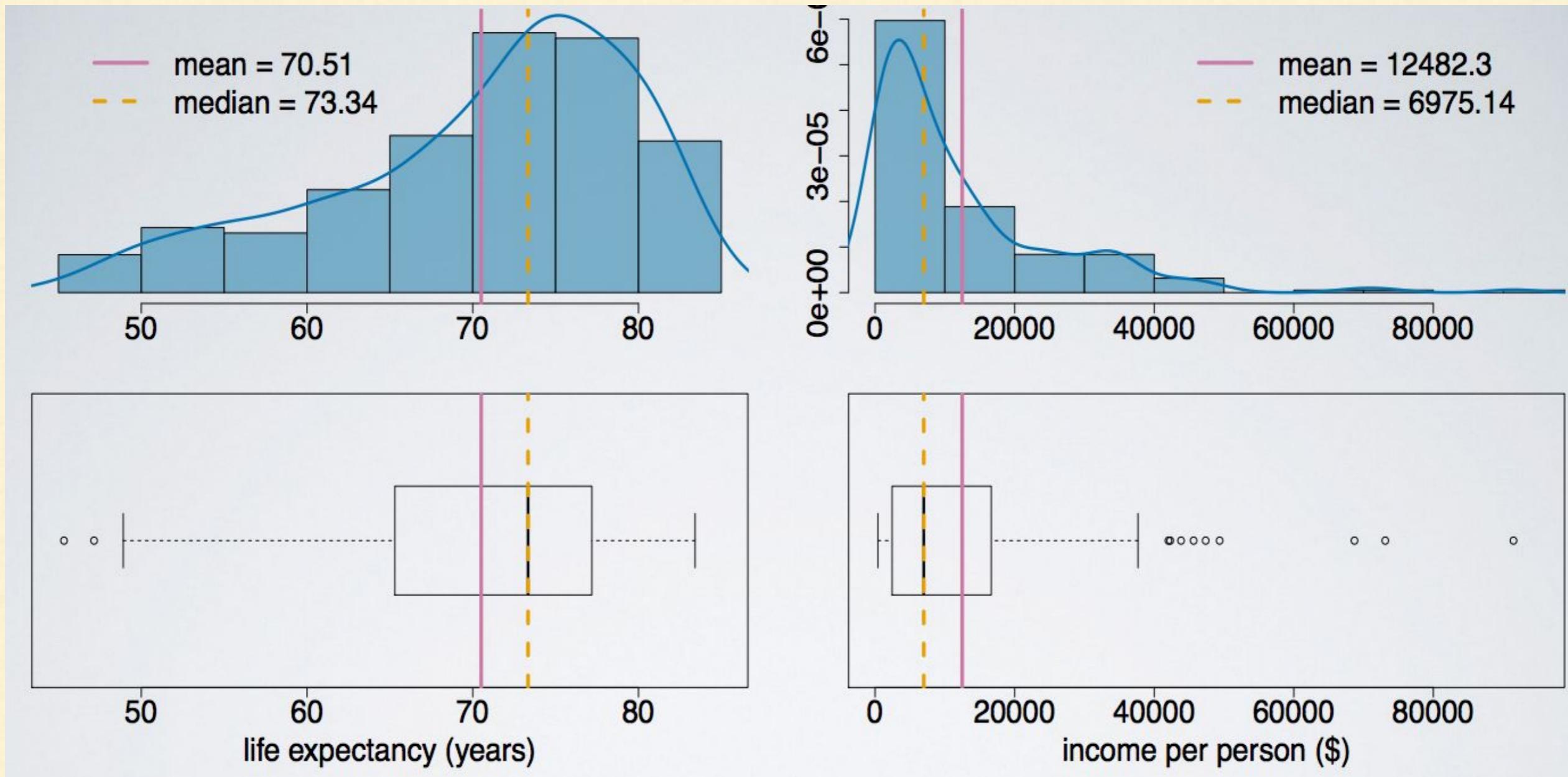
*symmetric*

# Measures of Central Tendencies

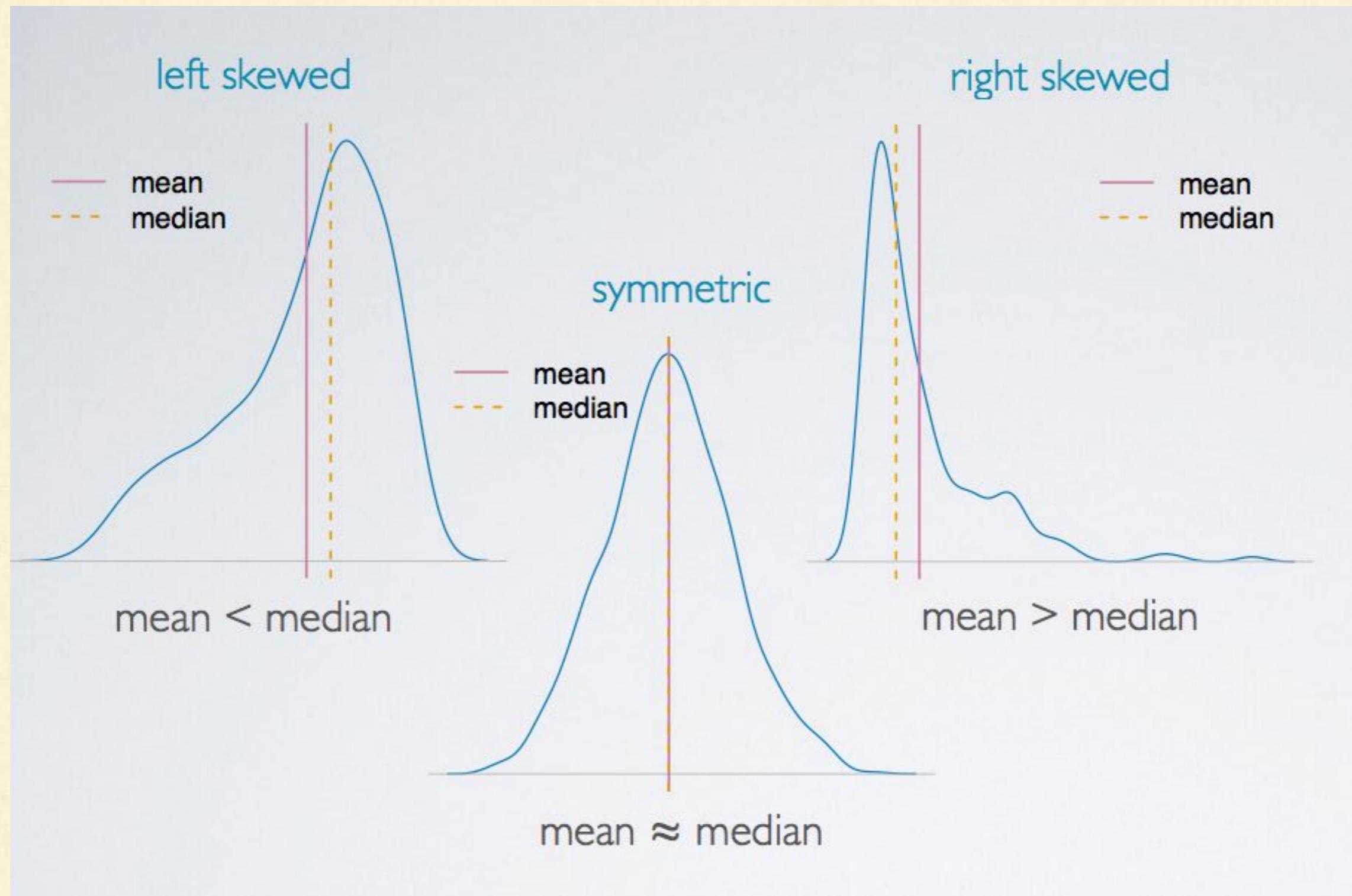
data	income per person (\$, 2012)	life expectancy (years, 2012)
Afghanistan	1359.7	60.254
Albania	6969.3	77.185
Algeria	6419.1	70.874
...	...	...
Zimbabwe	545.3	58.142

Source: [gapminder.com](http://gapminder.com)

# Measures of Central Tendencies



# Measures of Central Tendencies



# Measures of Spread

variance

roughly the average squared deviation from the mean

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

example

Given that the average life expectancy is 70.5, and there are 201 countries in the dataset:

$$\begin{aligned}s^2 &= \frac{(60.3 - 70.5)^2 + (77.2 - 70.5)^2 + \dots + (58.1 - 70.5)^2}{201 - 1} \\ &= 83.06 \text{ years}^2\end{aligned}$$

	country	life exp
1	Afghanistan	60.3
2	Albania	77.2
3	Algeria	70.9
	...	...
201	Zimbabwe	58.1

# Measures of Spread

Why do we square the differences?

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

# Measures of Spread

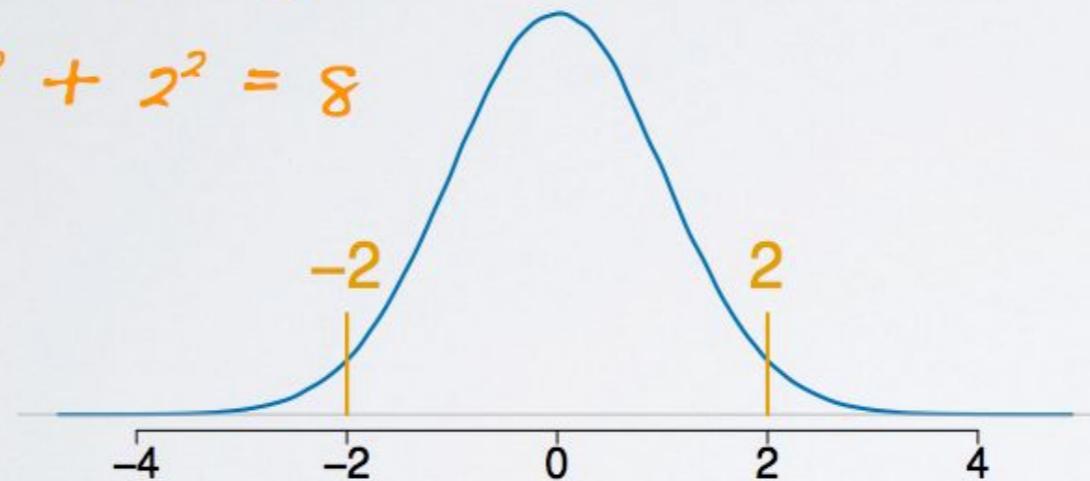
Why do we square the differences?

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

- ▶ get rid of negatives so that negatives and positives don't cancel each other when added together

$$(-2) + 2 = 0$$

$$(-2)^2 + 2^2 = 8$$



# Measures of Spread

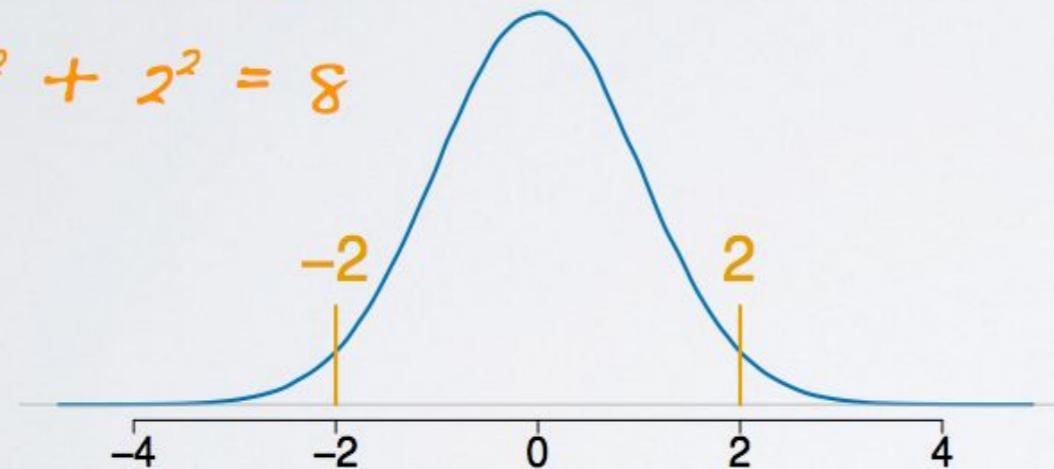
Why do we square the differences?

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

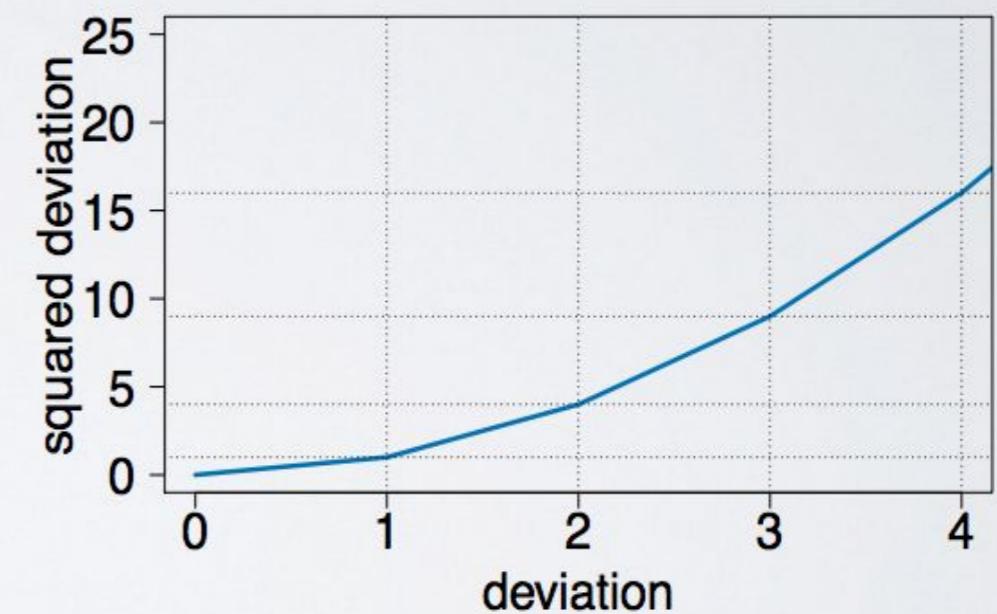
- ▶ get rid of negatives so that negatives and positives don't cancel each other when added together

$$(-2) + 2 = 0$$

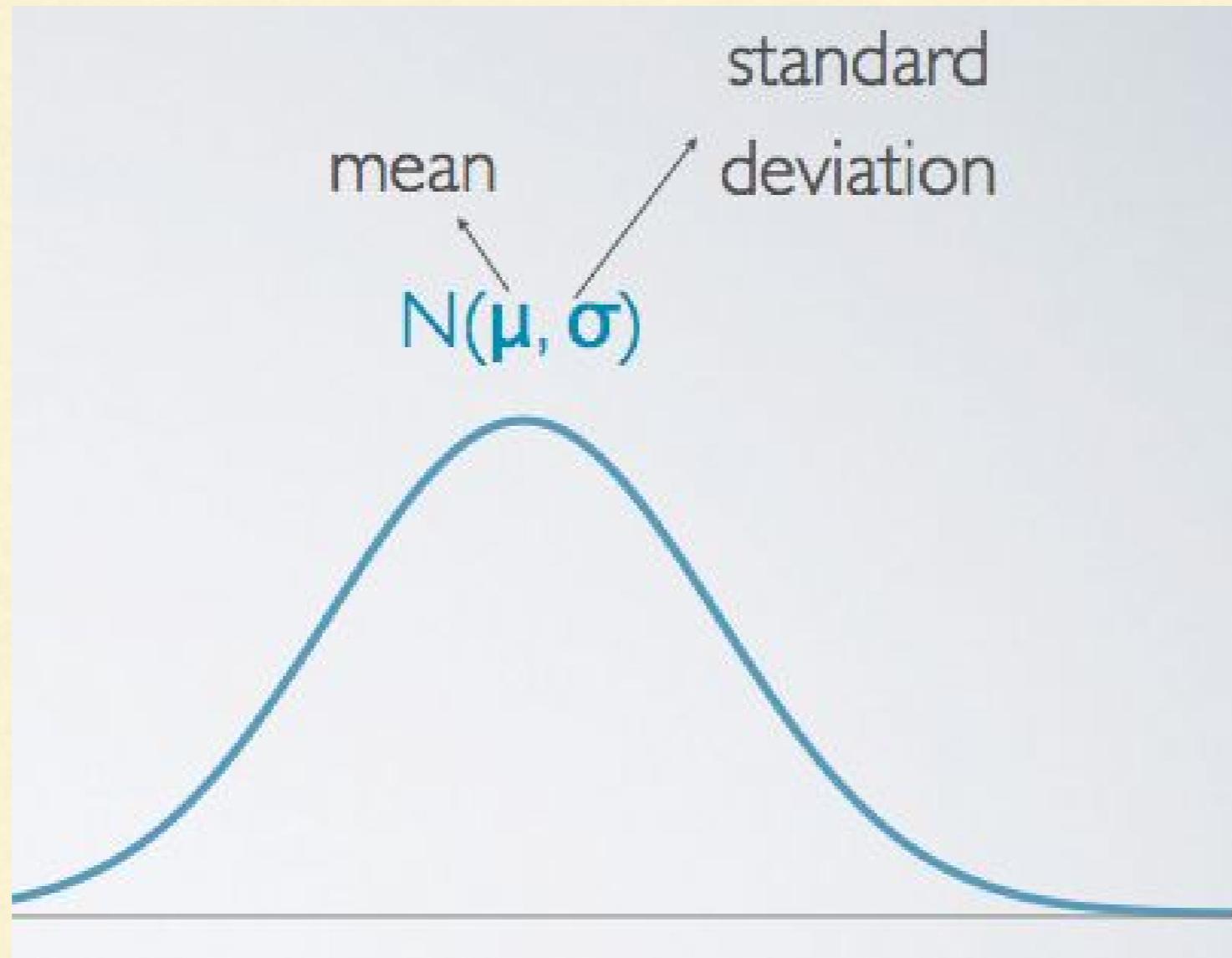
$$(-2)^2 + 2^2 = 8$$



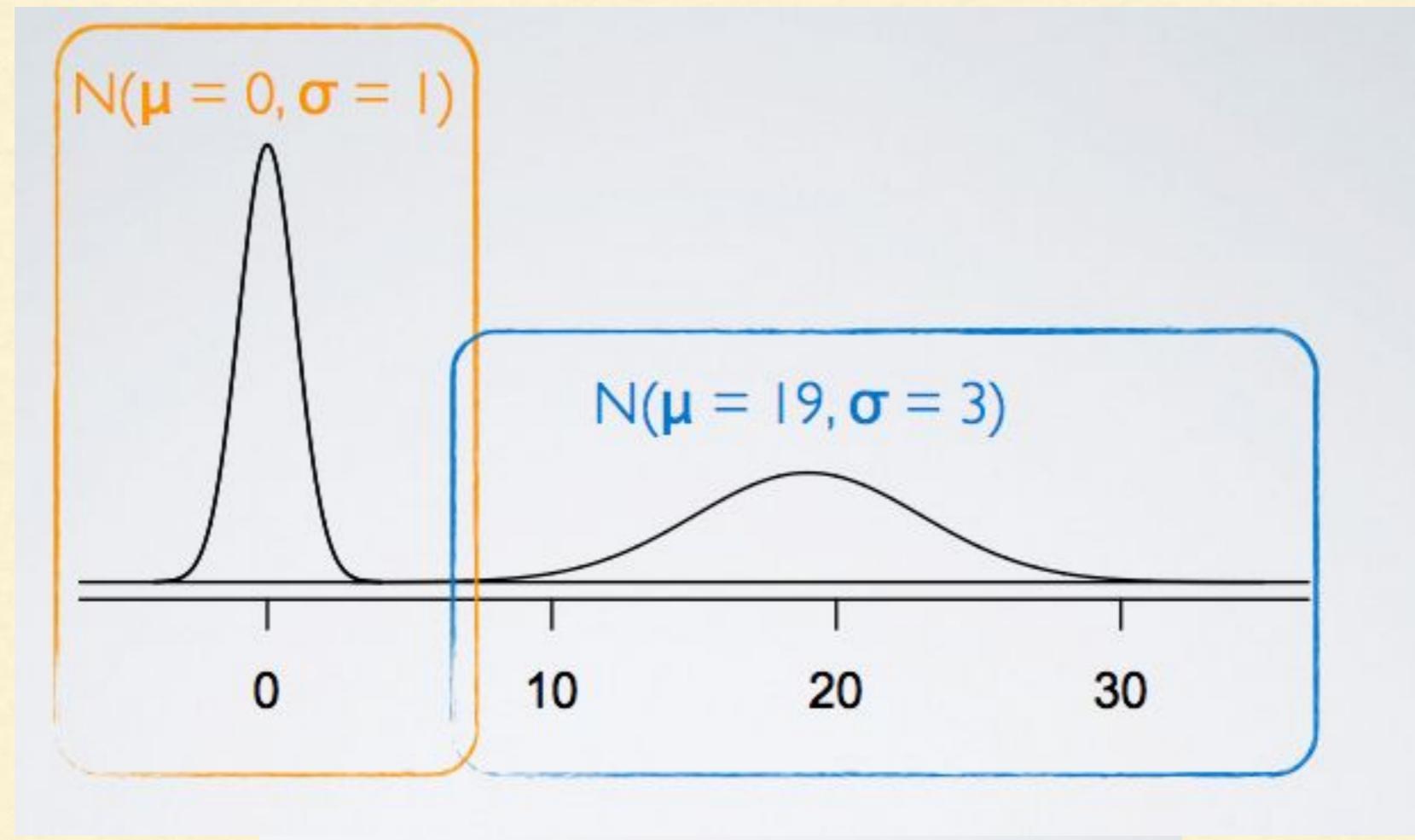
- ▶ increase larger deviations more than smaller ones so that they are weighed more heavily



# Normal Distribution



# Normal Distribution



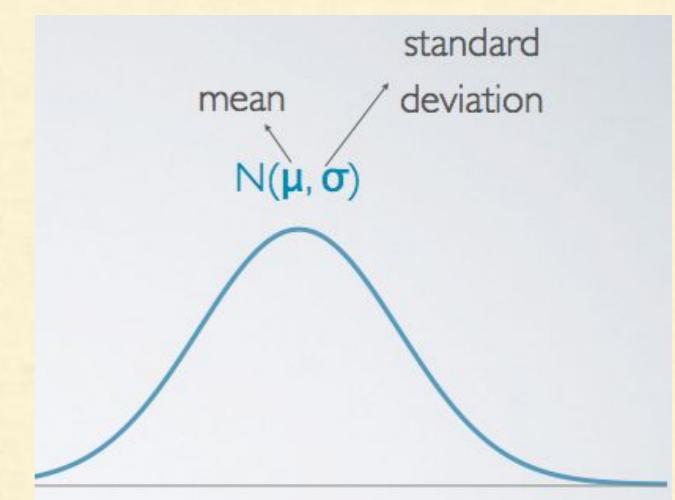
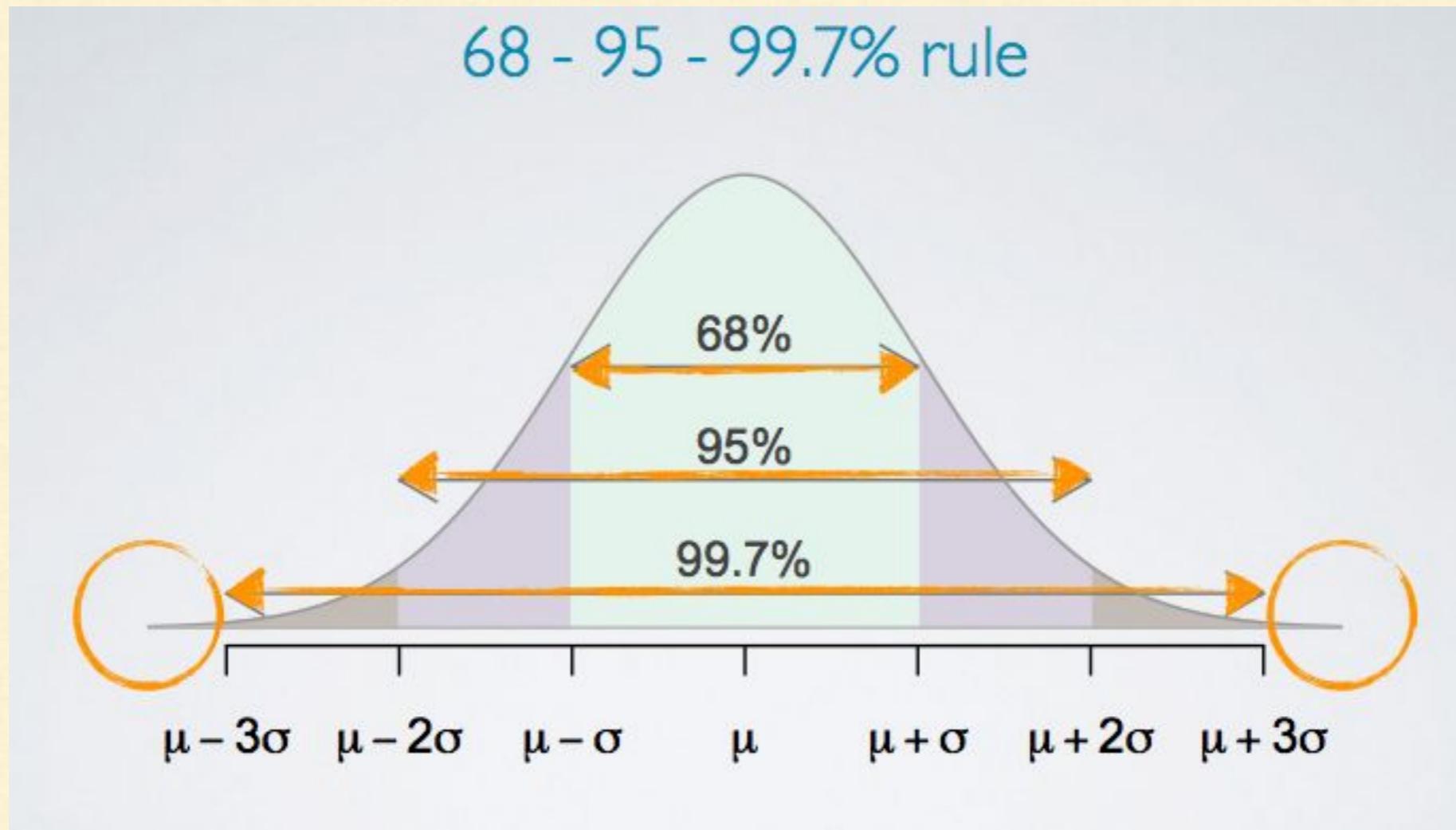
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2}$$

# Normal Distribution

---

- Unimodal and symmetric
- For continuous variables
- Follows very strict guidelines about
  - How variably the data are distributed around the mean

# Normal Distribution



# Normal Distribution

---

Question - A college admissions officer wants to determine which of the two applicants scored better on their standardized test with respect to the other test takers: Pam, who earned an 1800 on her SAT, or Jim, who scored a 24 on his ACT?

SAT scores  $\sim N(\text{mean} = 1500, \text{SD} = 300)$

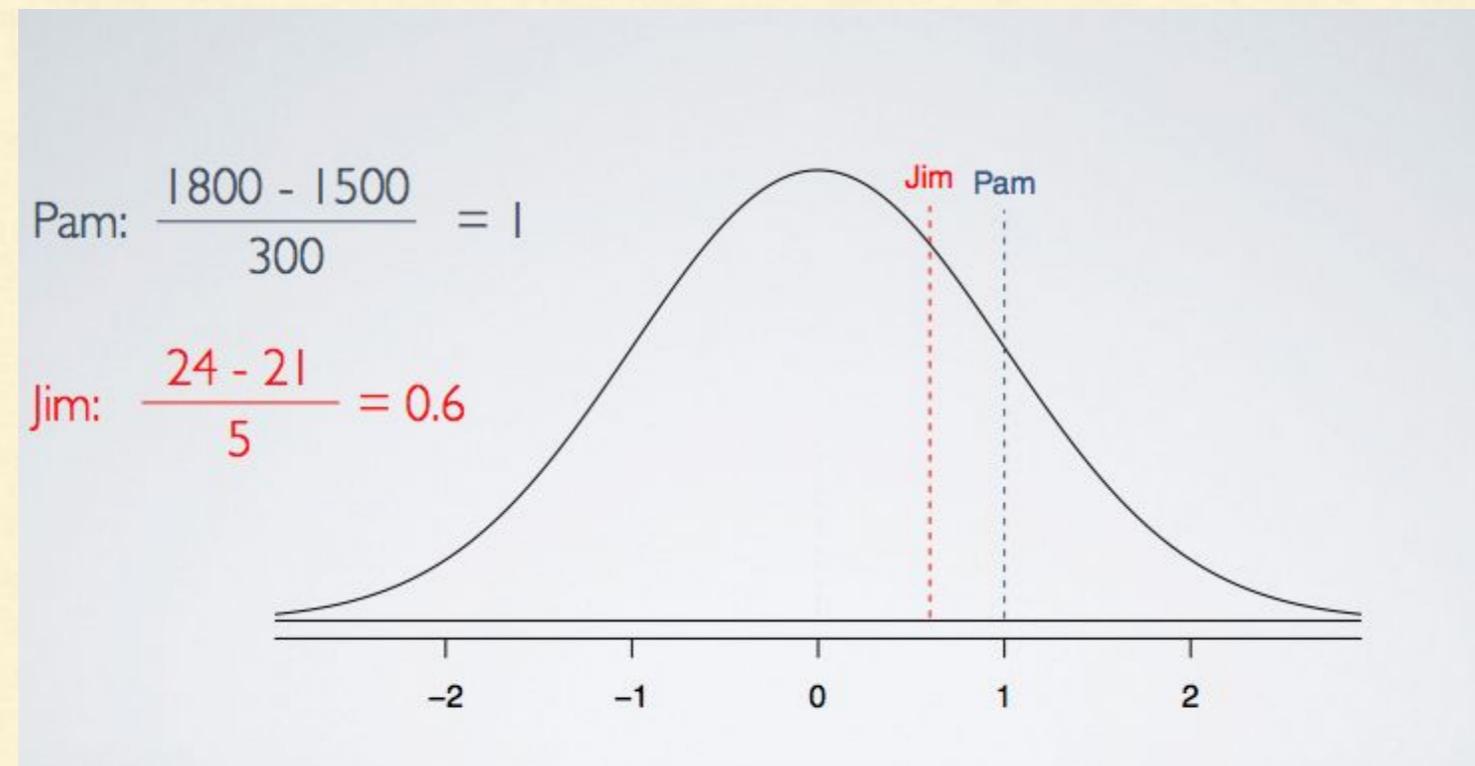
ACT scores  $\sim N(\text{mean} = 21, \text{SD} = 5)$

# Normal Distribution

A college admissions officer wants to determine which of the two applicants scored better on their standardized test with respect to the other test takers: Pam, who earned an 1800 on her SAT, or Jim, who scored a 24 on his ACT?

SAT scores  $\sim N(\text{mean} = 1500, \text{SD} = 300)$

ACT scores  $\sim N(\text{mean} = 21, \text{SD} = 5)$



---

---

# Back to: End to End Project

# Frame the Problem - Answers

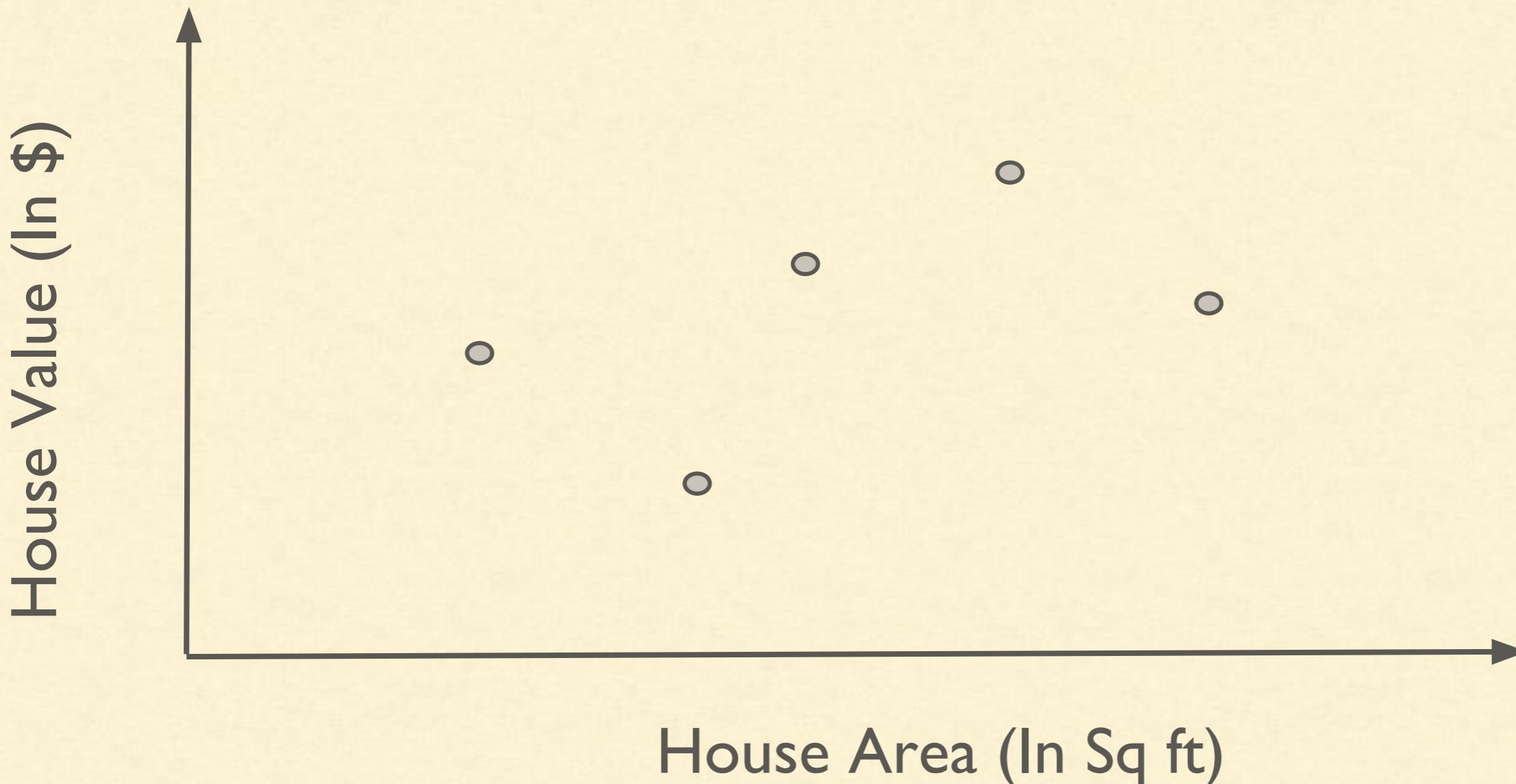
---

## Select a Performance Measure - Root Mean Square Error

- Let's say we want to predict house value based on house area in square feet
- We will use univariate linear regression
  - Since there is only one feature **house area**

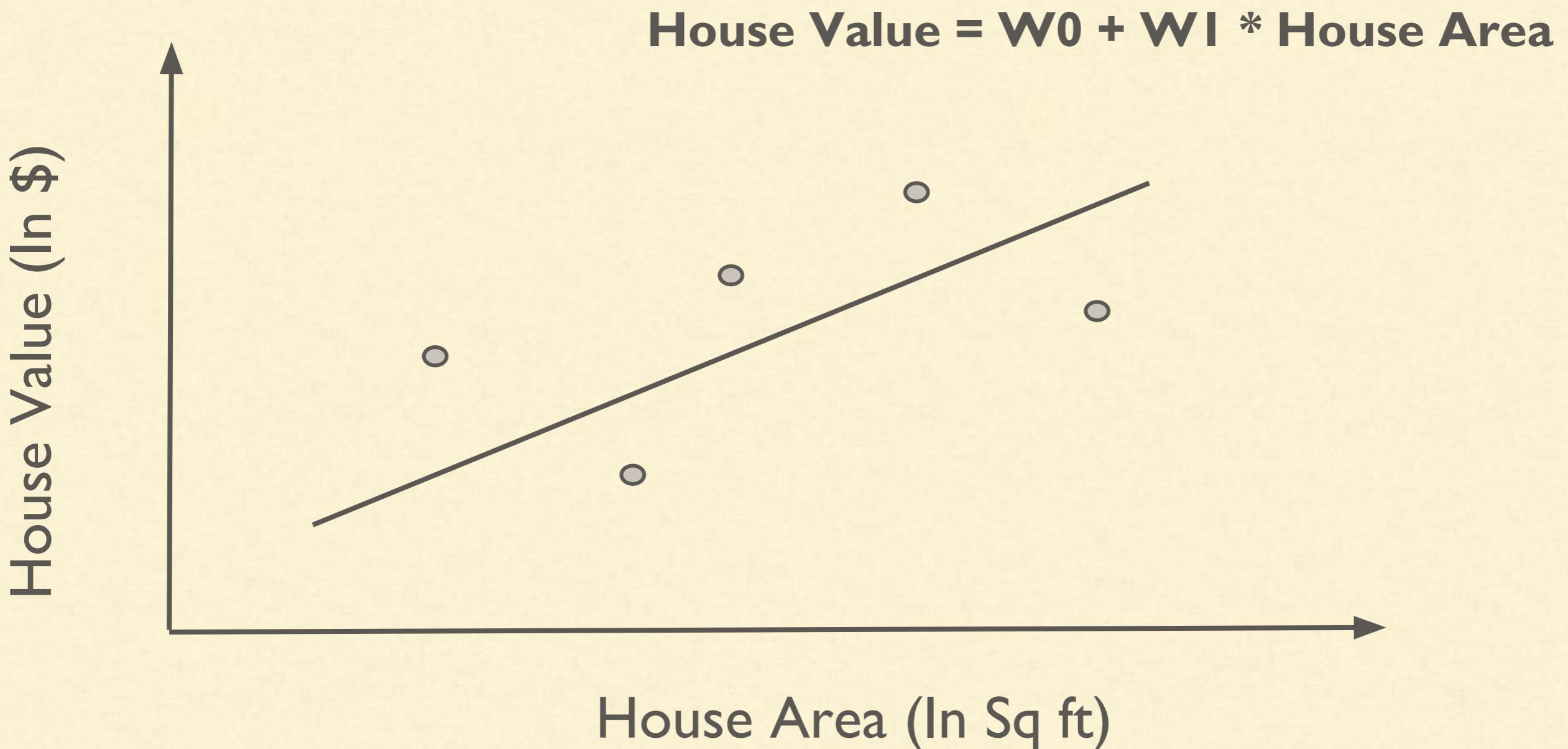
# Frame the Problem - Answers

## House Value vs House Area - Actual Data



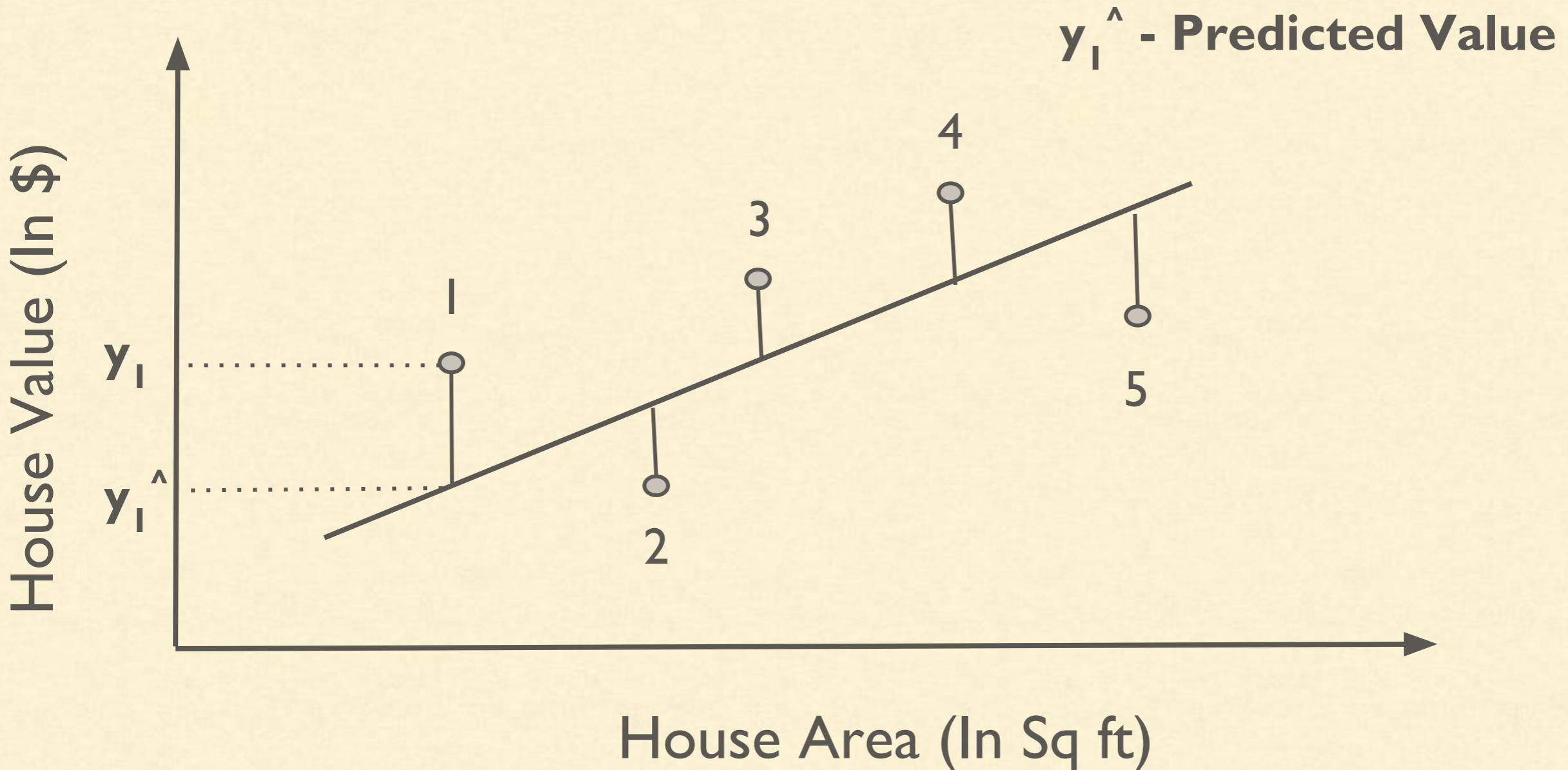
# Frame the Problem - Answers

## Fit a Line



# Frame the Problem - Answers

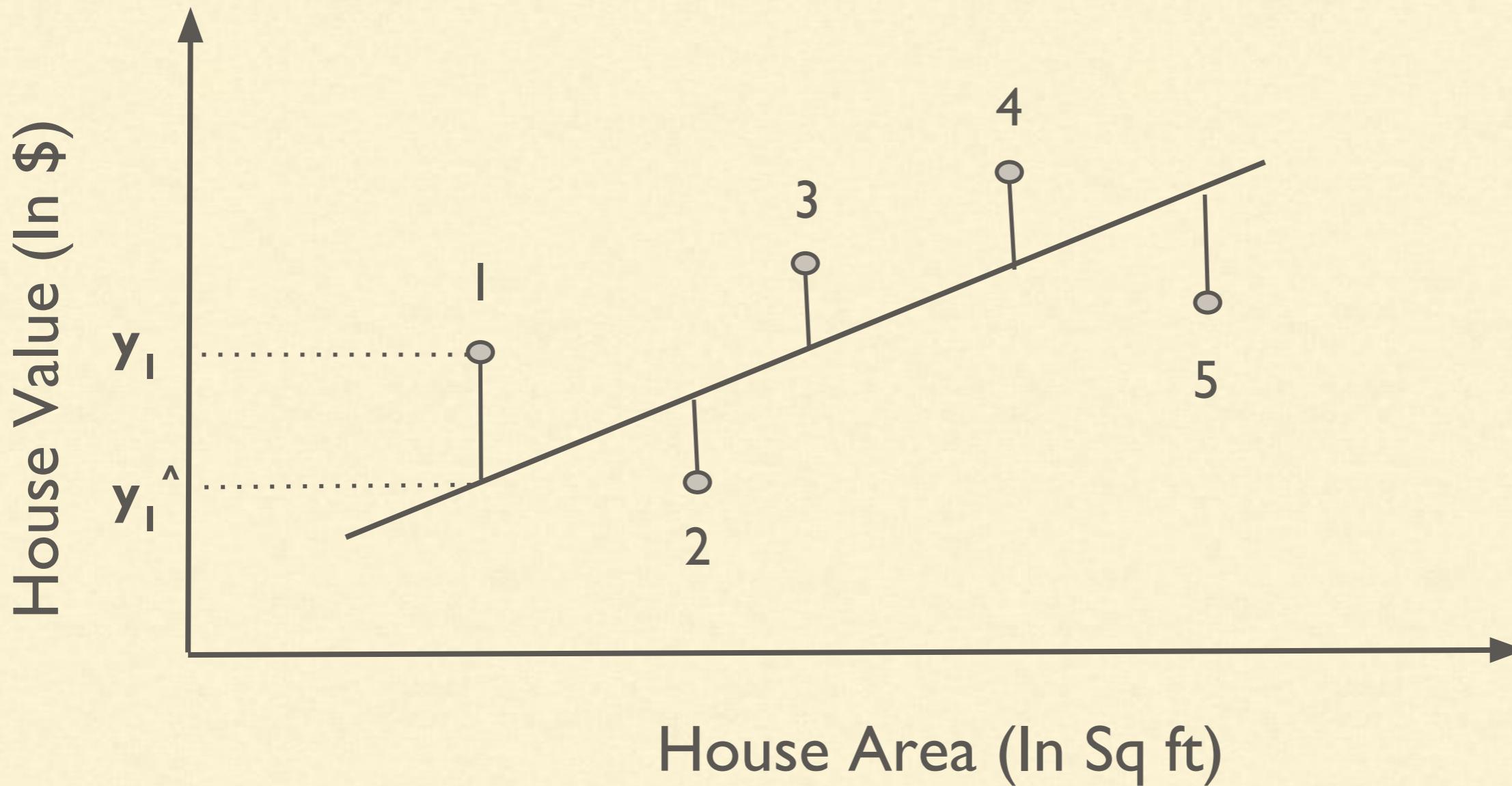
Fit a Line



# Frame the Problem - Answers

Error in Prediction

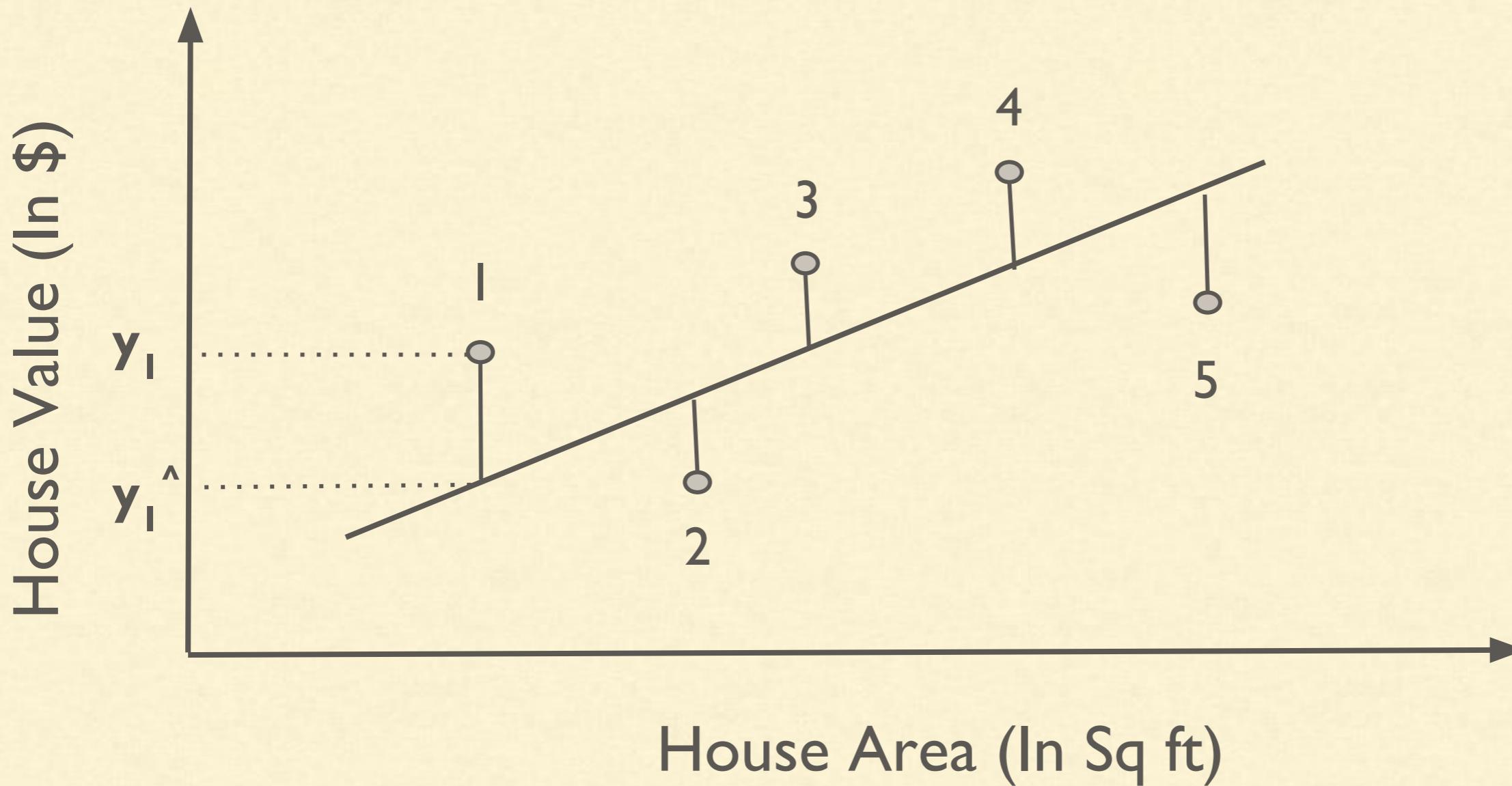
$$(y_i^{\wedge} - y_i)$$



# Frame the Problem - Answers

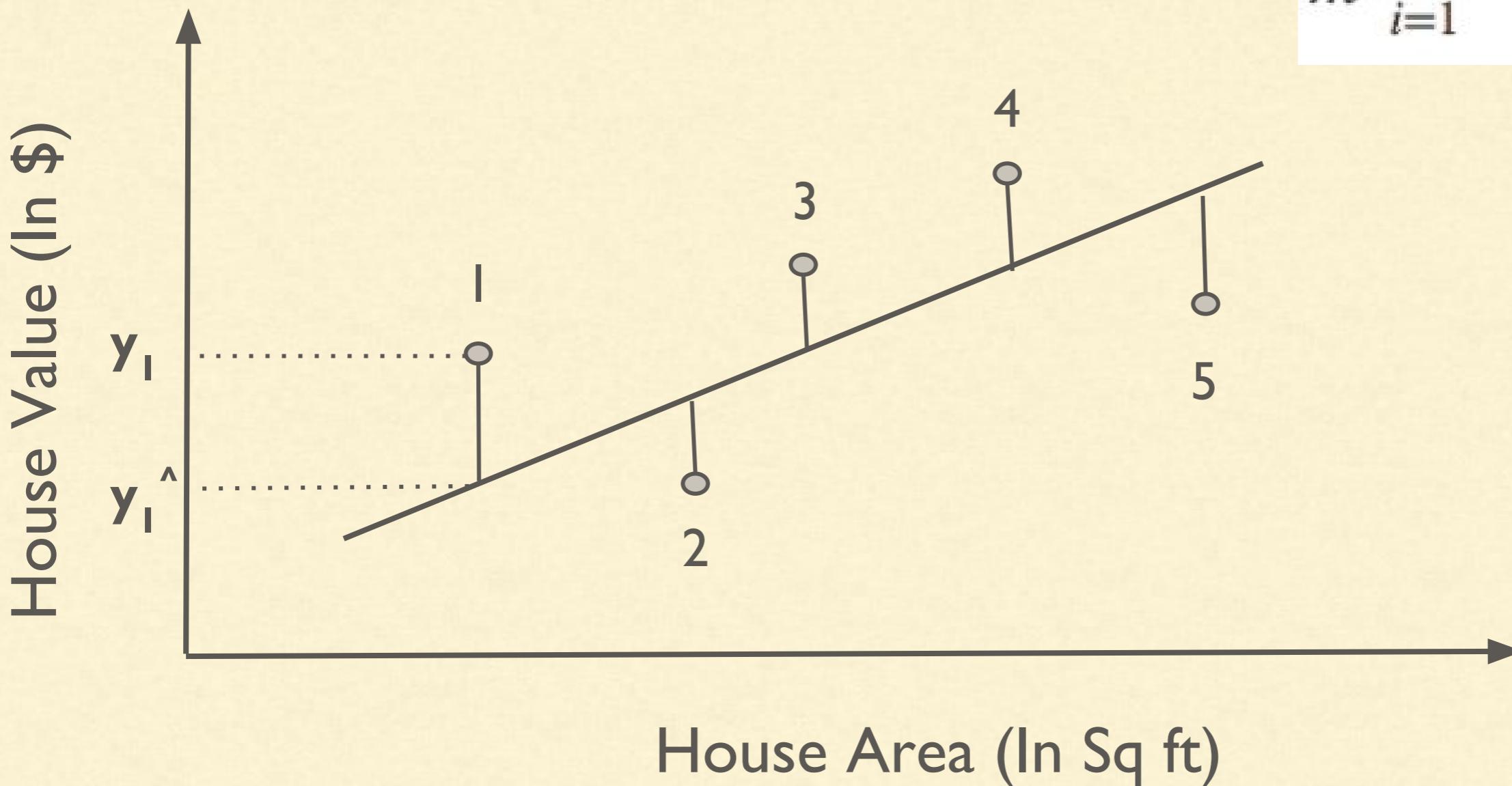
Squared Error

$$(y_i^{\wedge} - y_i)^2$$



# Frame the Problem - Answers

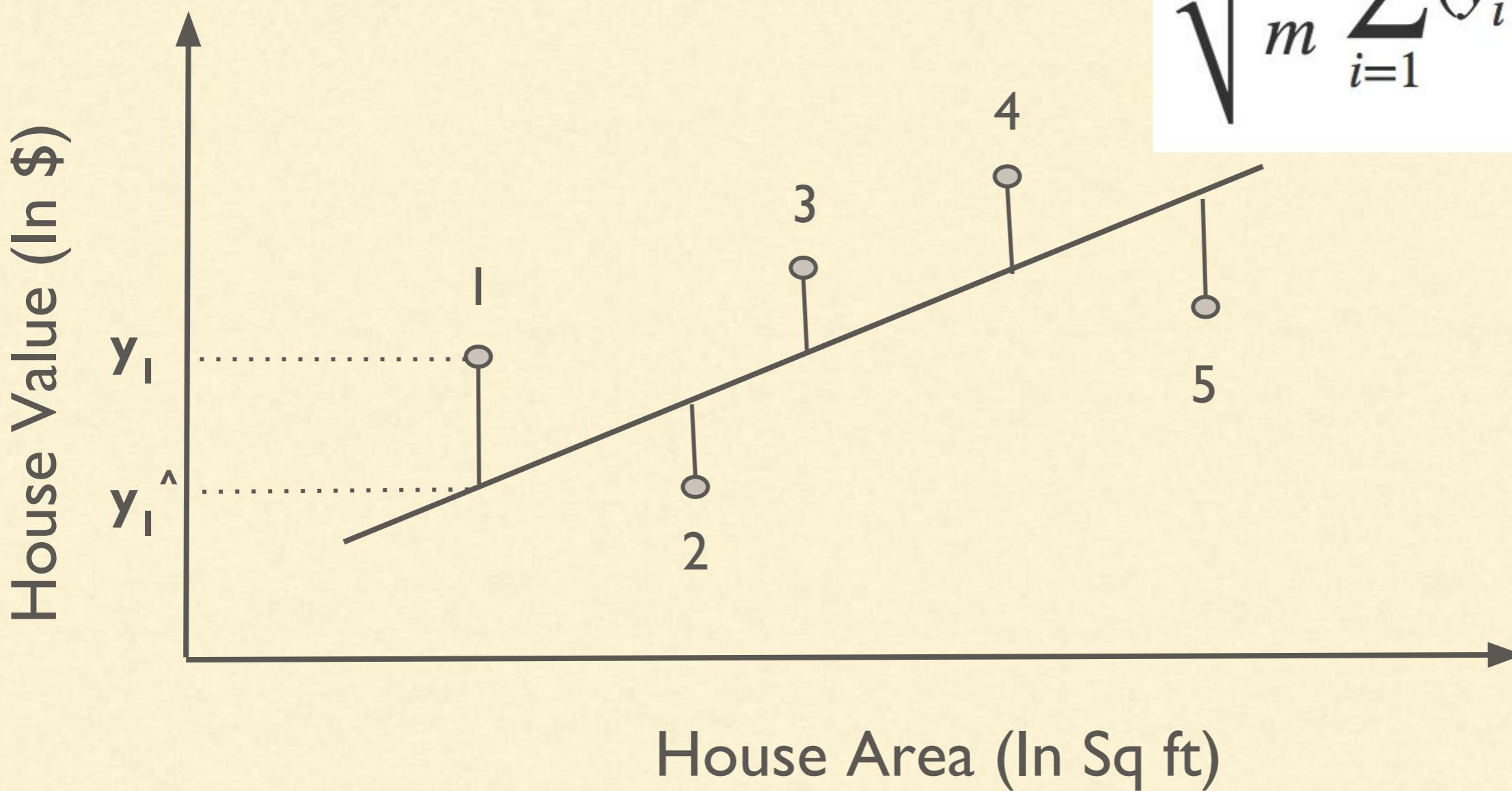
## Mean Square Error



$$\frac{1}{m} \sum_{i=1}^m (y_i^{\wedge} - y_i)^2$$

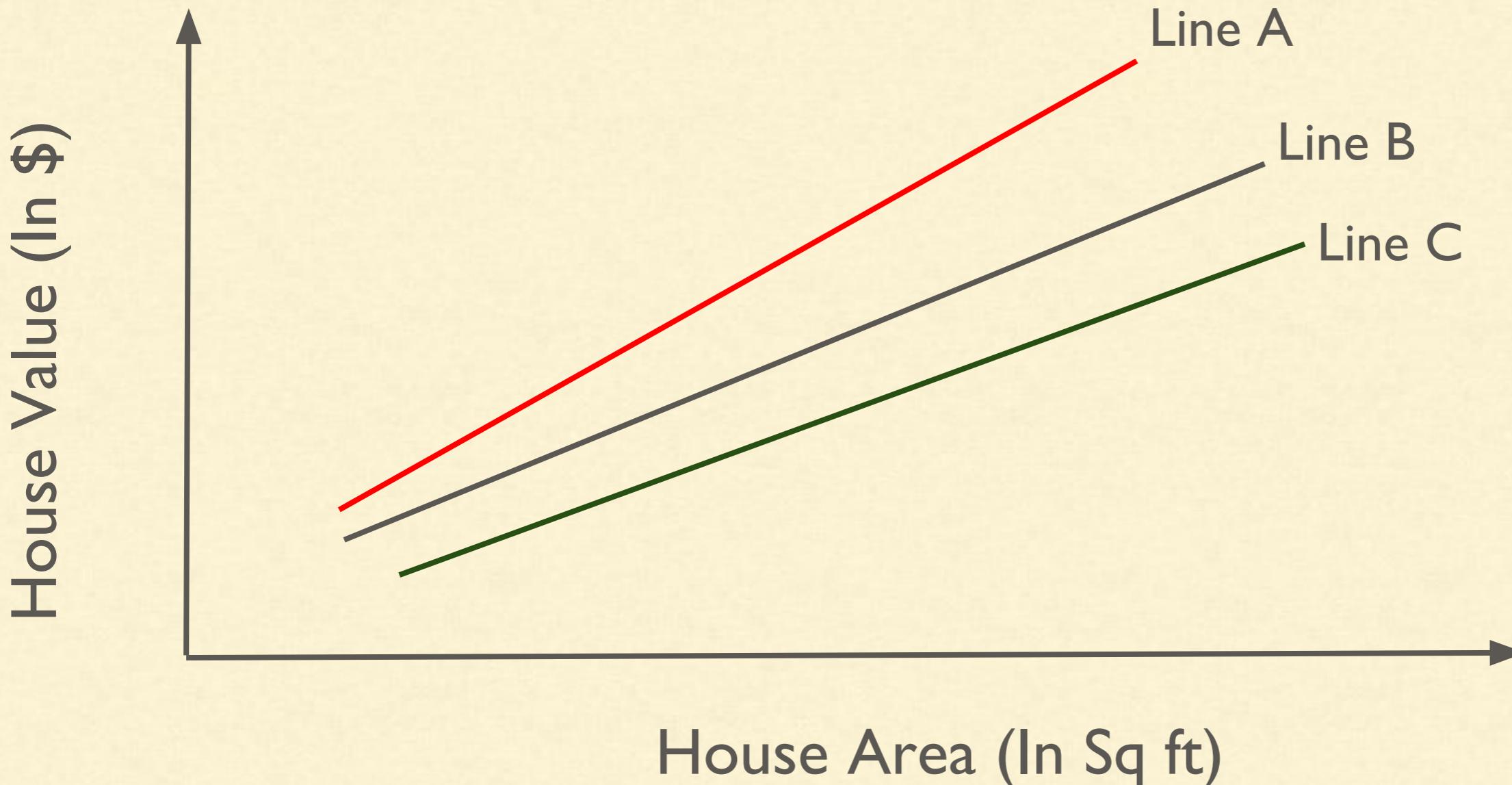
# Frame the Problem - Answers

## Root Mean Square Error (RMSE)



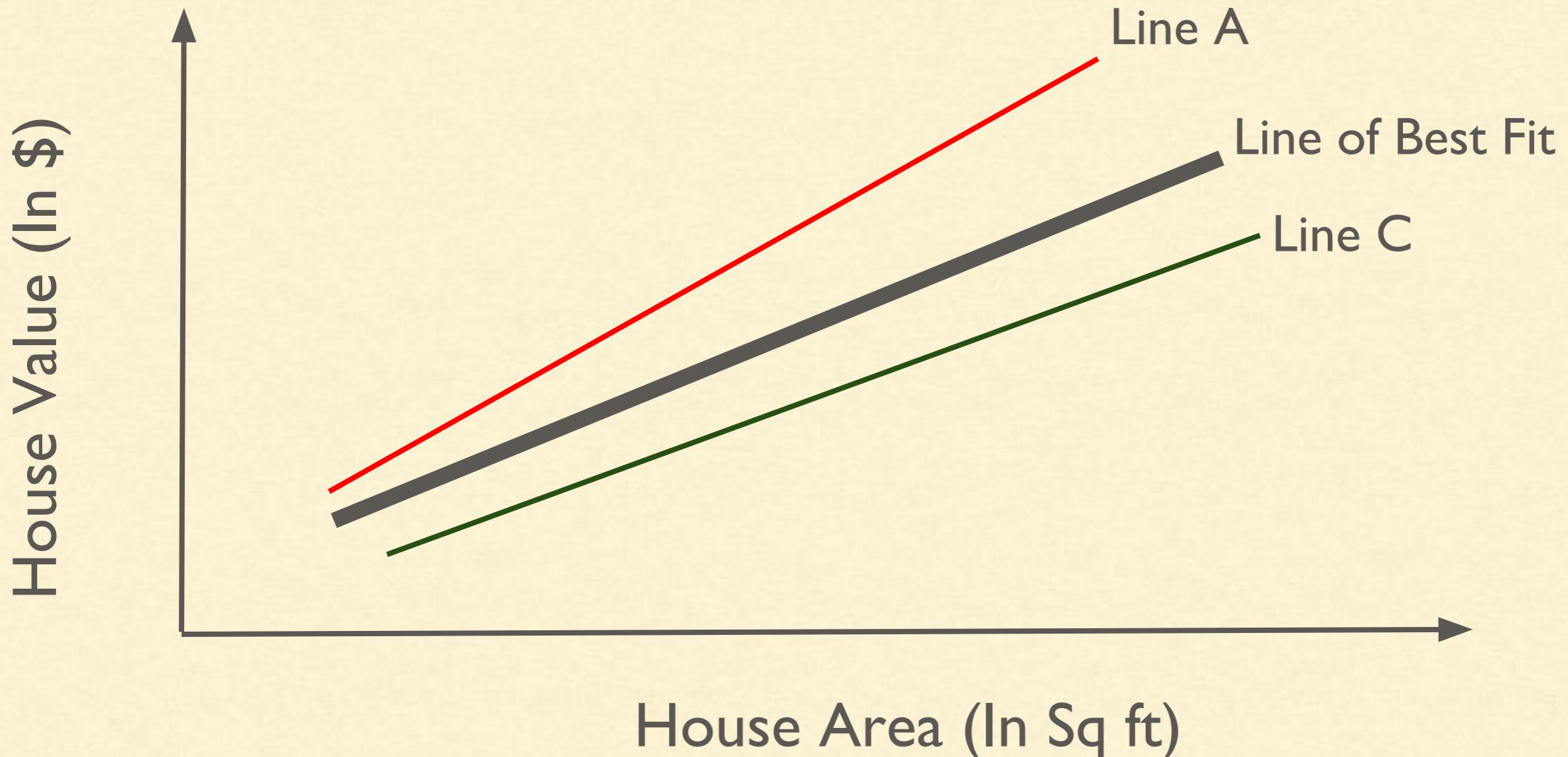
# Frame the Problem - Answers

**Algorithm tries many lines to fit the data**



# Frame the Problem - Answers

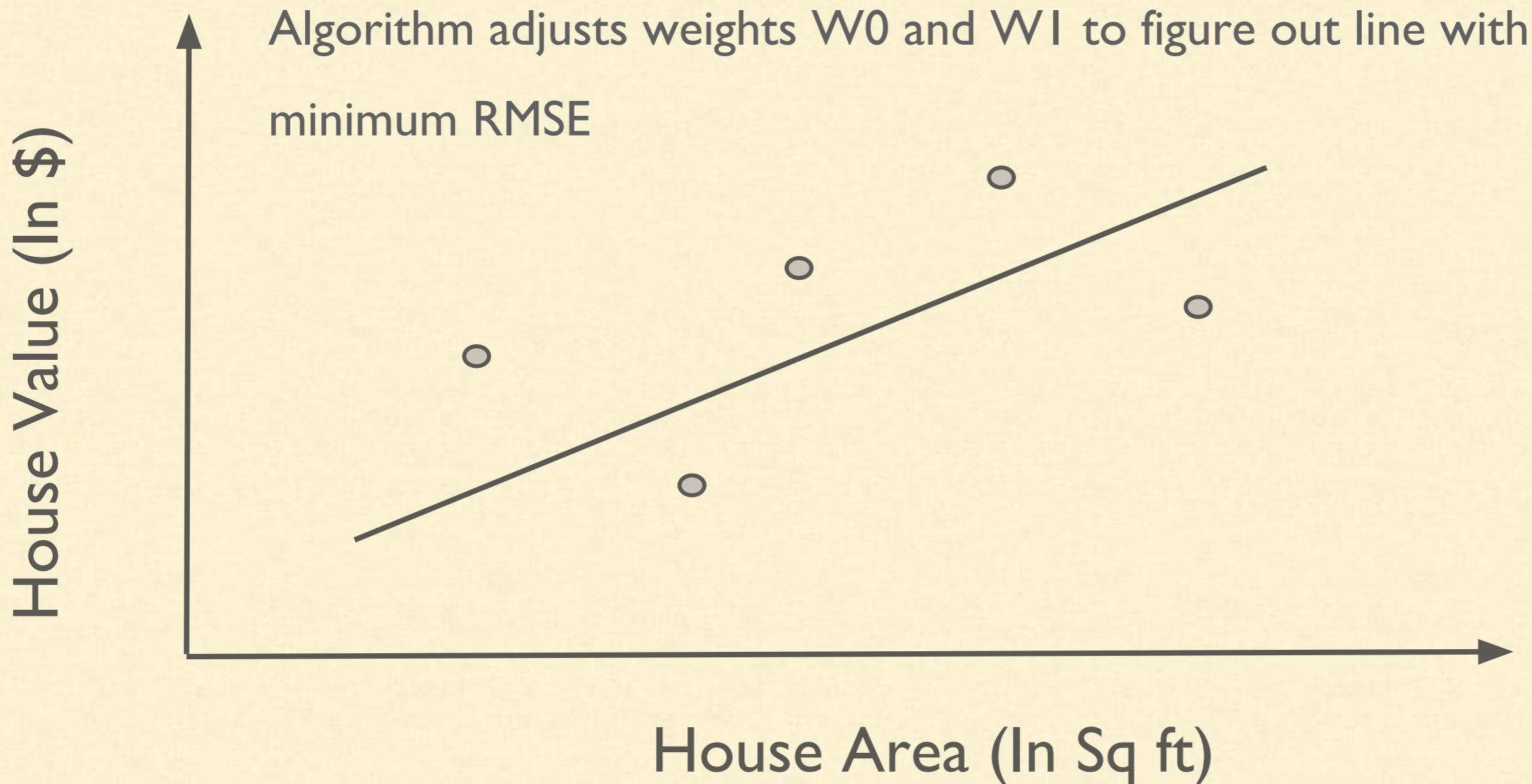
## Line of Best Fit - Line with a Minimum RMSE



# Frame the Problem - Answers

**Line of Best Fit**

**House Value = W0 + WI \* House Area**



# Checklist for Machine Learning Projects

---

1. Frame the problem and look at the big picture
2. **Get the data**
3. Explore the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Explore many different models and short-list the best ones
6. Fine-tune model
7. Present the solution
8. Launch, monitor, and maintain the system

# Get the Data

---

```
>>> import pandas as pd  
>>> import os  
  
>>> HOUSING_PATH = 'datasets/housing/'  
  
>>> def load_housing_data(housing_path=HOUSING_PATH):  
    csv_path = os.path.join(housing_path,  
"housing.csv")  
    return pd.read_csv(csv_path)  
  
>>> housing = load_housing_data()  
>>> housing.head()  
  
# Run this in notebook
```

# Get the Data

	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

Check this in Notebook

# Get the Data

---

- Each row represents one district
- There are 10 attributes:
  - longitude
  - latitude
  - housing\_median\_age
  - total\_rooms
  - total\_bedrooms
  - population
  - households
  - median\_income
  - median\_house\_value
  - ocean\_proximity

# Get the Data

---

Continue in the Notebook until plotting the histogram

# Get the Data

---

## Plot histogram

- Plot a histogram to get the feel of type of data we are dealing with
- We can plot histogram only for numerical attributes

# Get the Data

---

## Plot histogram

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

# Get the Data

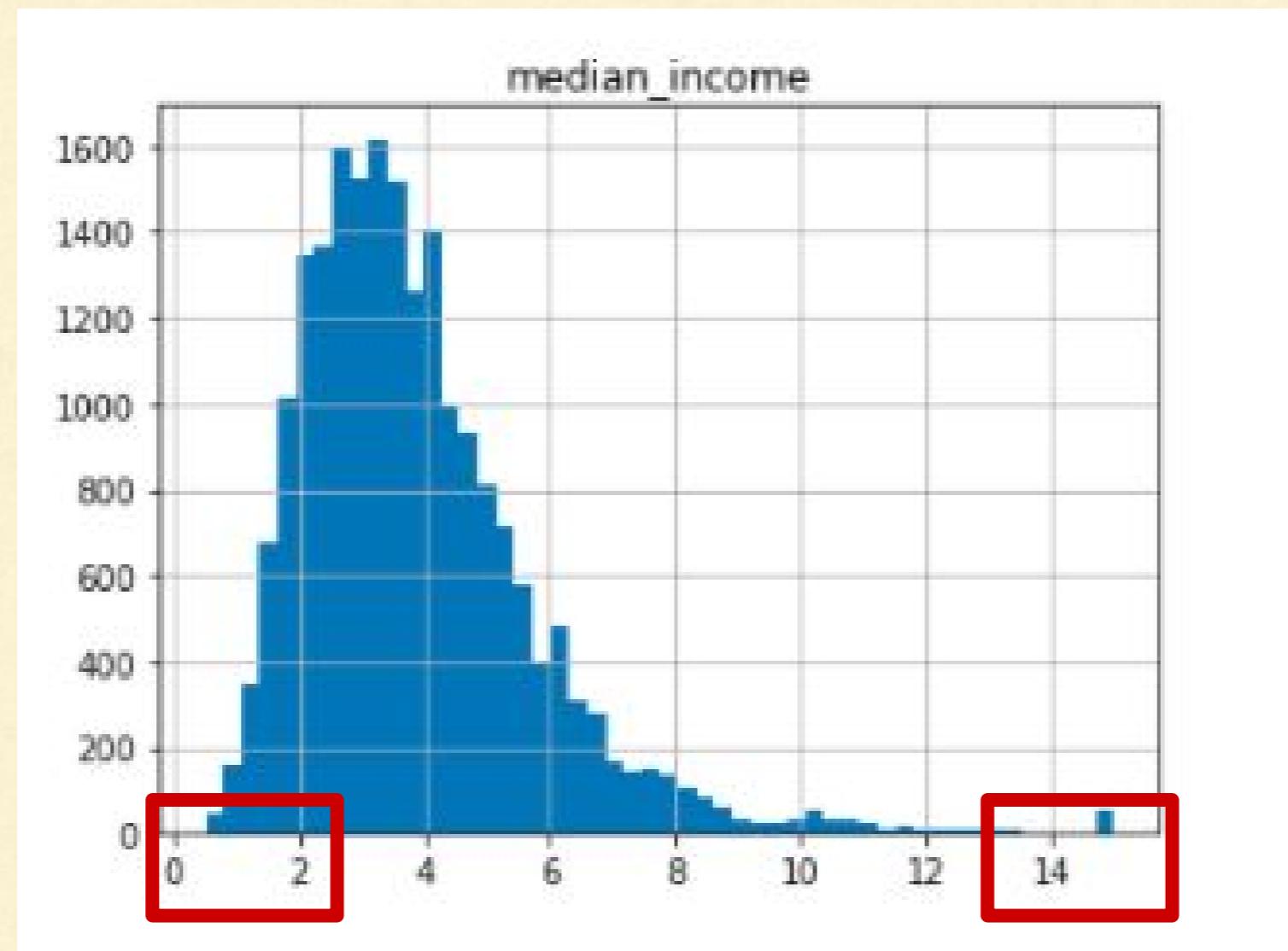
---

**Plot histogram**

**Observations?**

# Things to Note in Histogram - One

Plot histogram



# Things to Note in Histogram - One

---

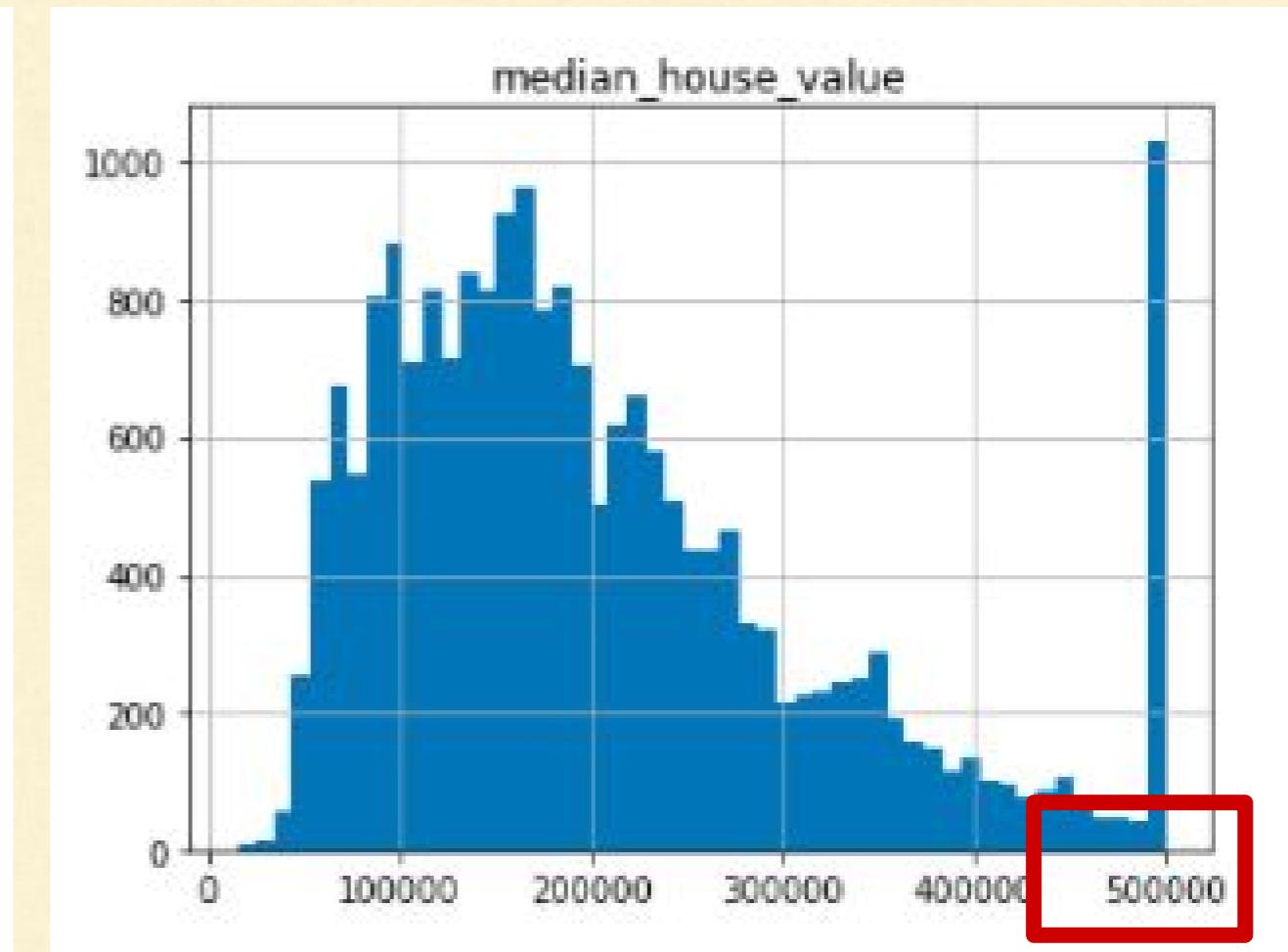
## Data is Capped

- The 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, and
  - At 0.5 (actually 0.4999) for lower median incomes

**Lesson: It is important to understand how your data was computed**

# Things to Note in Histogram - Two

## Plot histogram



# Things to Note in Histogram - Two

- The following are also capped
  - Median age - 50
  - Median house value - 500, 000
- Machine Learning algorithms may learn that prices never go beyond that limit

Serious problem since median\_house\_value is the target attribute



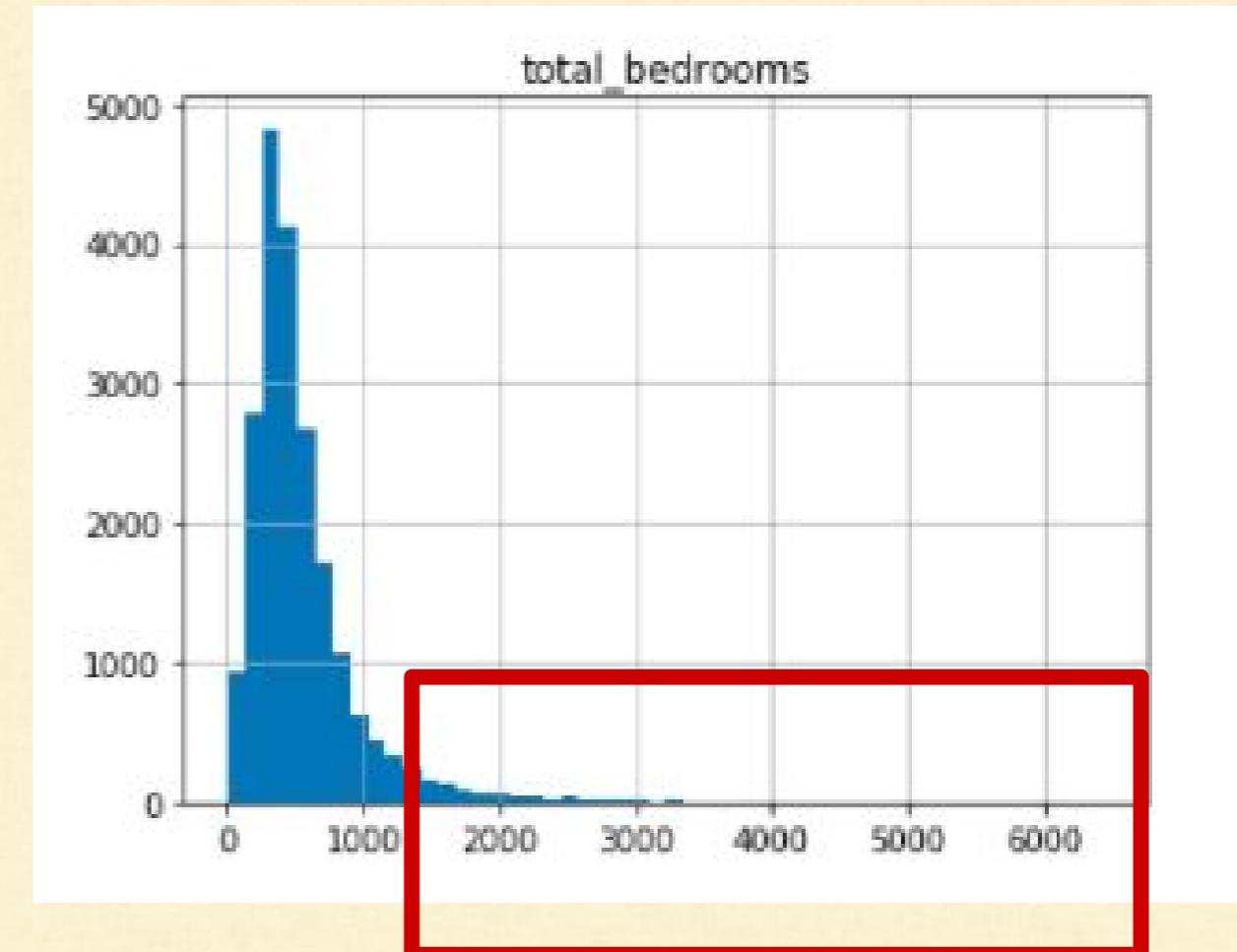
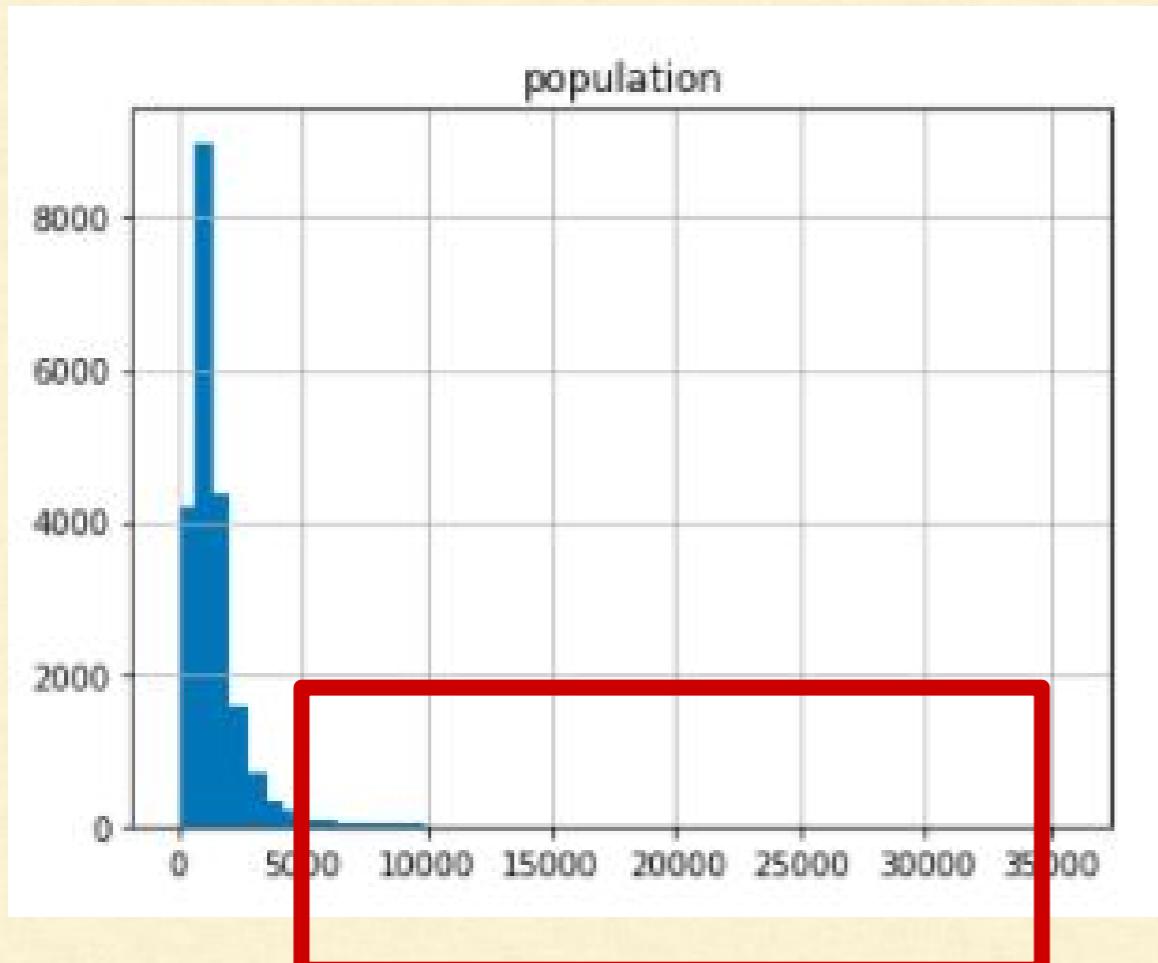
# Things to Note in Histogram - Two

## How do we solve it?

- Check with the team if this is a problem or not
- If team says they need precise predictions even beyond \$500,000 then
  - Collect proper labels for districts whose labels were capped
  - OR
  - Remove those labels from training as well as test dataset

# Things to Note in Histogram - Three

## Plot histogram



# Things to Note in Histogram - Three

---

## Tail heavy histograms

- Many histograms are tail heavy
- Harder for Machine Learning algorithms to detect patterns
- We will transform these attributes to more bell-shaped distributions

# Create a Test Set

---

## Training and Test set

- We generally split the data into
  - **Training set** - Contains 80% of the data
  - **Test set** - Contains remaining 20% of the data
- We train the model on training set
- And evaluate the performance of the model on test set

# Create a Test Set

---

- We've just seen the data and
- Still have to learn a lot more about it to decide which algorithm to use

**Why create a test set right now?**

# Create a Test Set

---

## Data snooping bias

- Brain is an amazing pattern detection system
- Highly prone to overfitting
- We may find interesting patterns in test data
- And select a particular Machine Learning model
- Our model will be too optimistic
- And will not perform as well as expected

# Create a Test Set

---

- In this project, we will use
  - Scikit-learn `train_test_split()` function
  - To create test set
- Over the next few slides, we will learn
  - How to create our own function
  - To create test set

# Create a Test Set

---

## Splitting Train and Test set

```
>>> np.random.seed(42)

>>> import numpy as np

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]
```

# Create a Test Set

---

## Splitting Train and Test set

- Remember to pass a seed to `np.random.permutation` always
- Seed is important because we want our model
  - To be trained and evaluated on same train and test set every time
- Else on every run of `split_train_test` function
  - Train and test set will be different
  - Over time our machine learning model may see the whole dataset during training
    - Which we should to avoid

# Create a Test Set

---

## Splitting Train and Test set - Seed

- Seed can be any number
- Remember to pass same seed every time
- To make sure same rows goes to train and test set
- Everytime we run `split_train_test` function

# Create a Test Set

---

**Let's run `split_train_test` in notebook**

```
>>> train_set, test_set = split_train_test(housing, 0.2)
>>> print(len(train_set), "train +", len(test_set),
"test")
```

**Output -**

**16512 train + 4128 test**

# Create a Test Set

---

**Question -**

**What is the problem with `split_train_test` function?**

# Create a Test Set

---

**Answer -**

This solution will break next time when we fetch an updated dataset

# Create a Test Set

---

## Other Solution -

- If each instance in the dataset has a
  - Unique and Immutable Identifier
  - We can decide on the basis of identifier
  - If it should go into test set or not

# Create a Test Set

---

## Other Solution Example-

- Compute hash of each instance's identifier
- Take only last byte of hash
- If last byte value is lower or equal to 51 (20% of 256)
  - Put instance in the test set

# Create a Test Set

---

## Other Solution Example-

- Test set using previous example
  - Will be consistent across multiple runs
  - Even if we refresh the dataset
  - Will contain 20% of the instances
  - Will not contain any instance that was previously in the training set

# Create a Test Set

---

## Implementation-

```
>>> import hashlib

>>> def test_set_check(identifier, test_ratio, hash):
        return hash(np.int64(identifier)).digest()[-1] < 256 * 
test_ratio

>>> def split_train_test_by_id(data, test_ratio, id_column,
hash=hashlib.md5):

    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_,
test_ratio, hash))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

# Create a Test Set

---

## Problems?

- Housing dataset does not have an identifier column
- Simplest solution is to use row index as ID

```
>>> housing_with_id = housing.reset_index()  
  
>>> train_set, test_set =  
split_train_test_by_id(housing_with_id, 0.2, "index")
```

Run it on Notebook

# Create a Test Set

---

**Problems with row index?**

# Create a Test Set

---

## Problem with row index?

- Make sure new data is appended to the end of the dataset
- No row ever is deleted

# Create a Test Set

---

## Problem with row index?

- A district's latitude and longitude are guaranteed to be stable for a few million years
- We can combine them into an ID

# Create a Test Set

---

## Problem with row index?

```
>>> housing_with_id["id"] = housing["longitude"] * 1000  
+ housing["latitude"]  
>>> train_set, test_set =  
split_train_test_by_id(housing_with_id, 0.2, "id")
```

Run it in Notebook

# Create a Test Set

---

## Using Scikit-learn for the same

- Sklearn provides **train\_test\_split** function
  - Same as **split\_train\_test** defined earlier with two more features
  - There is `random_state` parameter which sets the random generator seed
  - We can pass it on a multiple datasets with an identical number of rows
  - **split\_train\_test** splits them on the same indices
  - Useful if we have separate DataFrame for labels

# Create a Test Set

---

**Using Scikit-learn for the same**

```
>>> from sklearn.model_selection import train_test_split  
>>> train_set, test_set = train_test_split(housing,  
test_size=0.2, random_state=42)
```

**Run it in notebook**

# Create a Test Set

---

## Sampling Bias

- Till now, we have considered pure **random sampling methods**
- **Random sampling methods** work fine
  - If dataset is large enough
  - Else we are at risk of introducing significant **sampling bias**

# Create a Test Set

---

## Sampling Bias - Example

- Let's say a survey company in US decides to call 1,000 people to ask them few questions

**What is the best approach to pick 1,000 people?**

# Create a Test Set

---

## Sampling Bias - Example

- Let's say a survey company in US decides to call 1,000 people to ask them few questions

**We can not just pick 1,000 people randomly**

# Create a Test Set

---

## **Sampling Bias - Important Point**

Samples must be representative of the whole  
population

# SAMPLING BIAS

## Convenience sample

Individuals who are easily accessible are more likely to be included in the sample

## Non-response

If only a (non-random) fraction of the randomly sampled people respond to a survey such that the sample is no longer representative of the population

## Voluntary response

Occurs when the sample have strong opinions on the issue

Should the West intervene in Syria?

Yes

No

[VOTE](#)

or view results

Should the West intervene in Syria?

Yes 34% 534

No 66% 1038

Total Votes: 1572

This is not a scientific poll

# SAMPLING METHODS

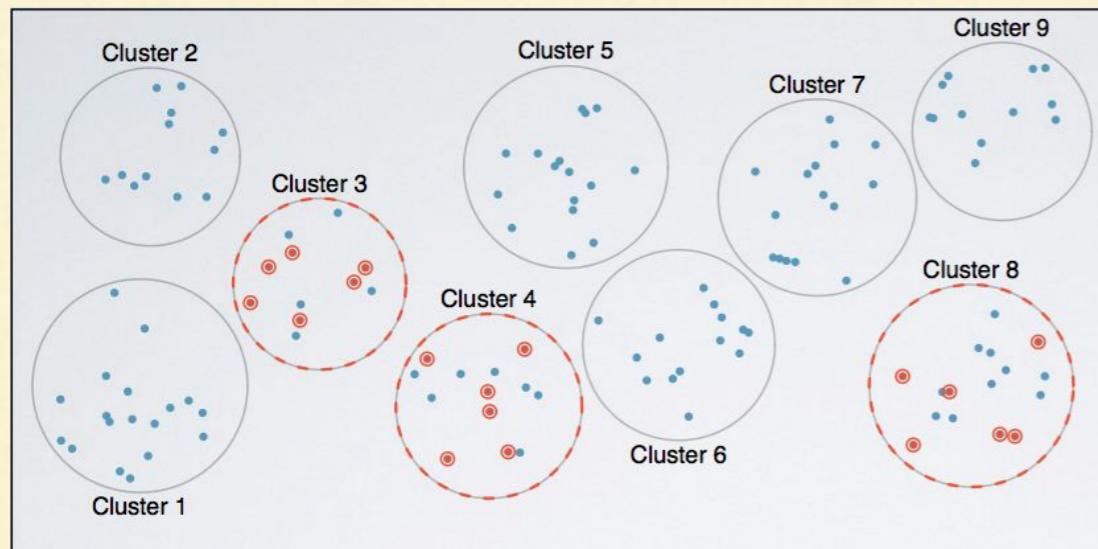
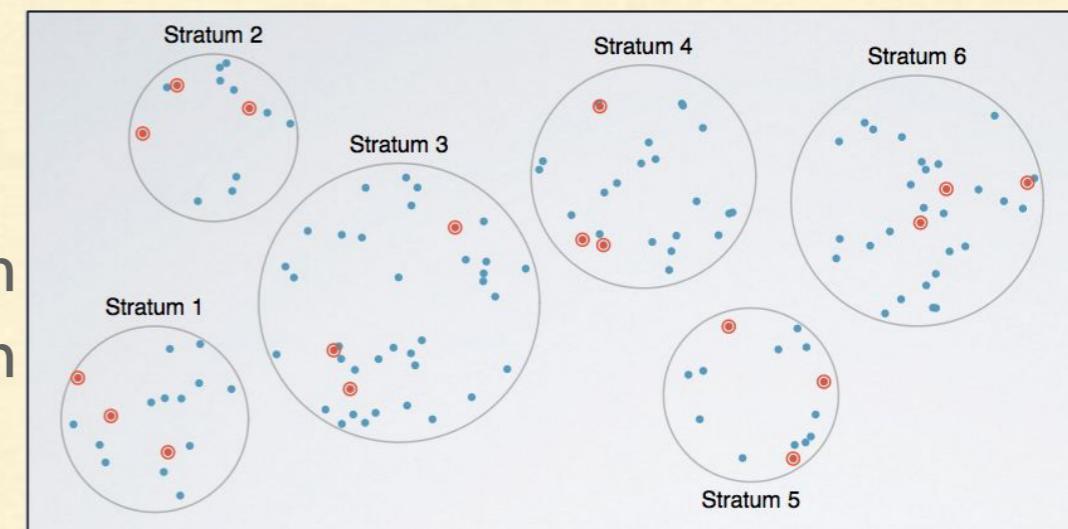


## Simple Random Sample (SRS)

Each case is equally likely to be selected

## Stratified Sample

Divide the population into homogenous strata, then randomly sample from within each stratum



## Cluster Sample

Divide the population clusters, randomly sample a few clusters, then randomly sample from within these clusters

# Create a Test Set

---

## Sampling Bias - Example

- US population is composed of
  - 51.3% female
  - 48.7% male

# Create a Test Set

---

## Stratified Sampling

- A well-conducted survey on 1000 people in the US should
  - Try to maintain the ratio in the sample
  - 513 female
  - 487 male
- This is called **stratified sampling**

# Create a Test Set

---

## Stratified Sampling

- The population is divided into
  - Homogeneous subgroups called **strata**
- And the right number of instances is
  - Sampled from each **stratum**
- Else the survey results would be significantly biased

# Create a Test Set

---

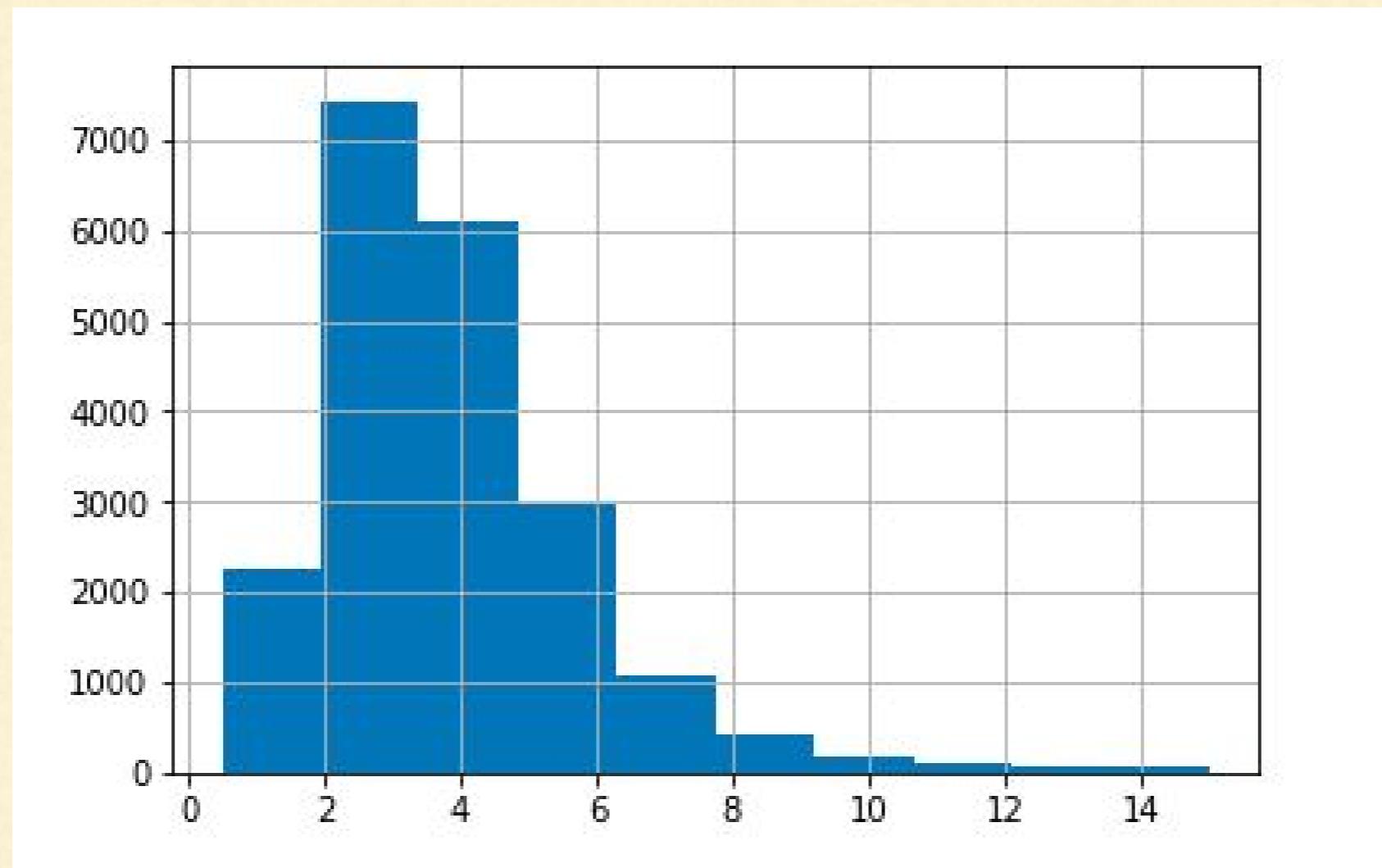
## Median Income

- Let's say some experts advised us that
  - The median income is a very important attribute to predict median housing prices
- Then the test set should be representative of
  - Various categories of median income

# Create a Test Set

## Create Histogram of Median Income

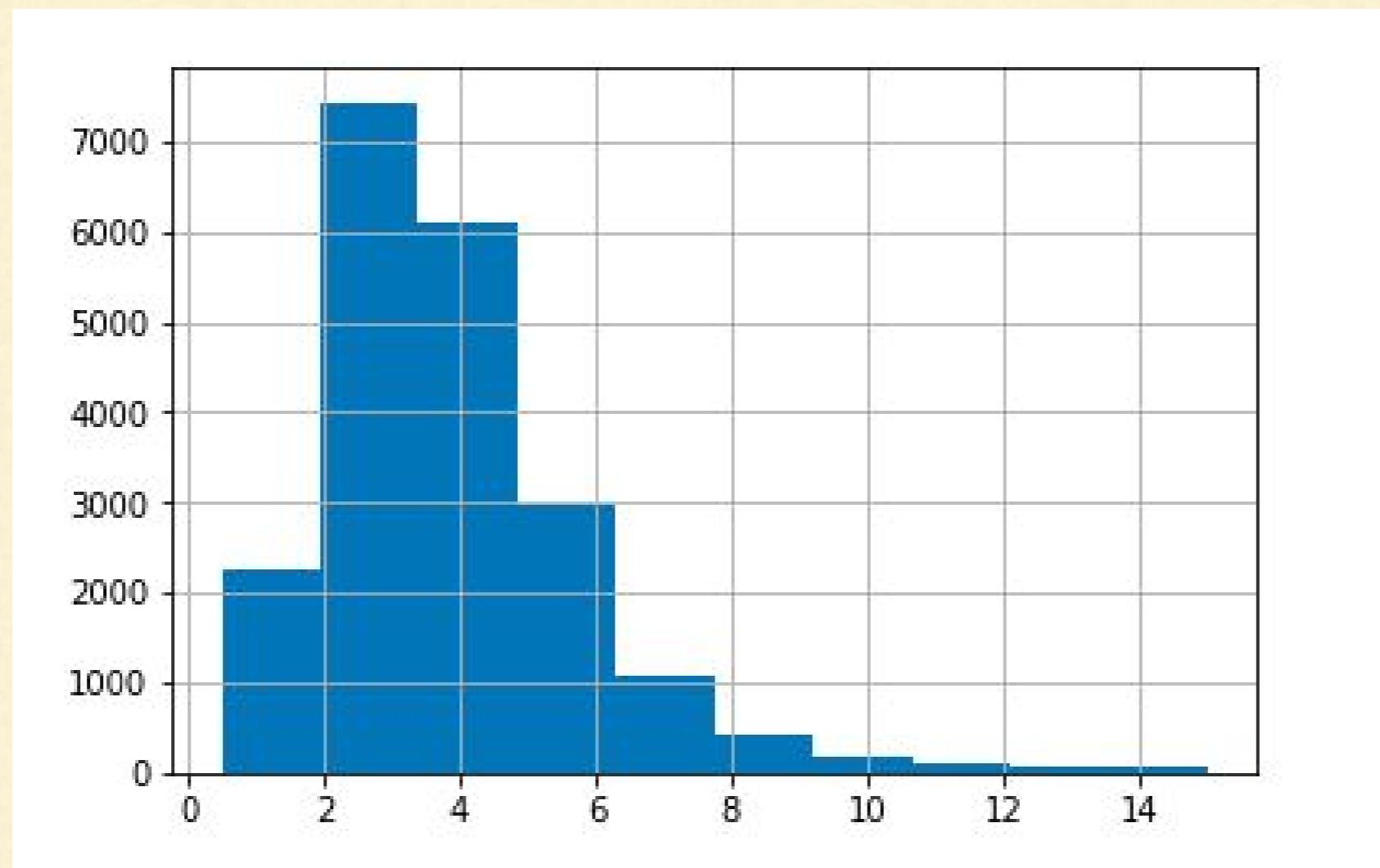
```
>>> housing["median_income"].hist()
```



# Create a Test Set

## Create Histogram of Median Income

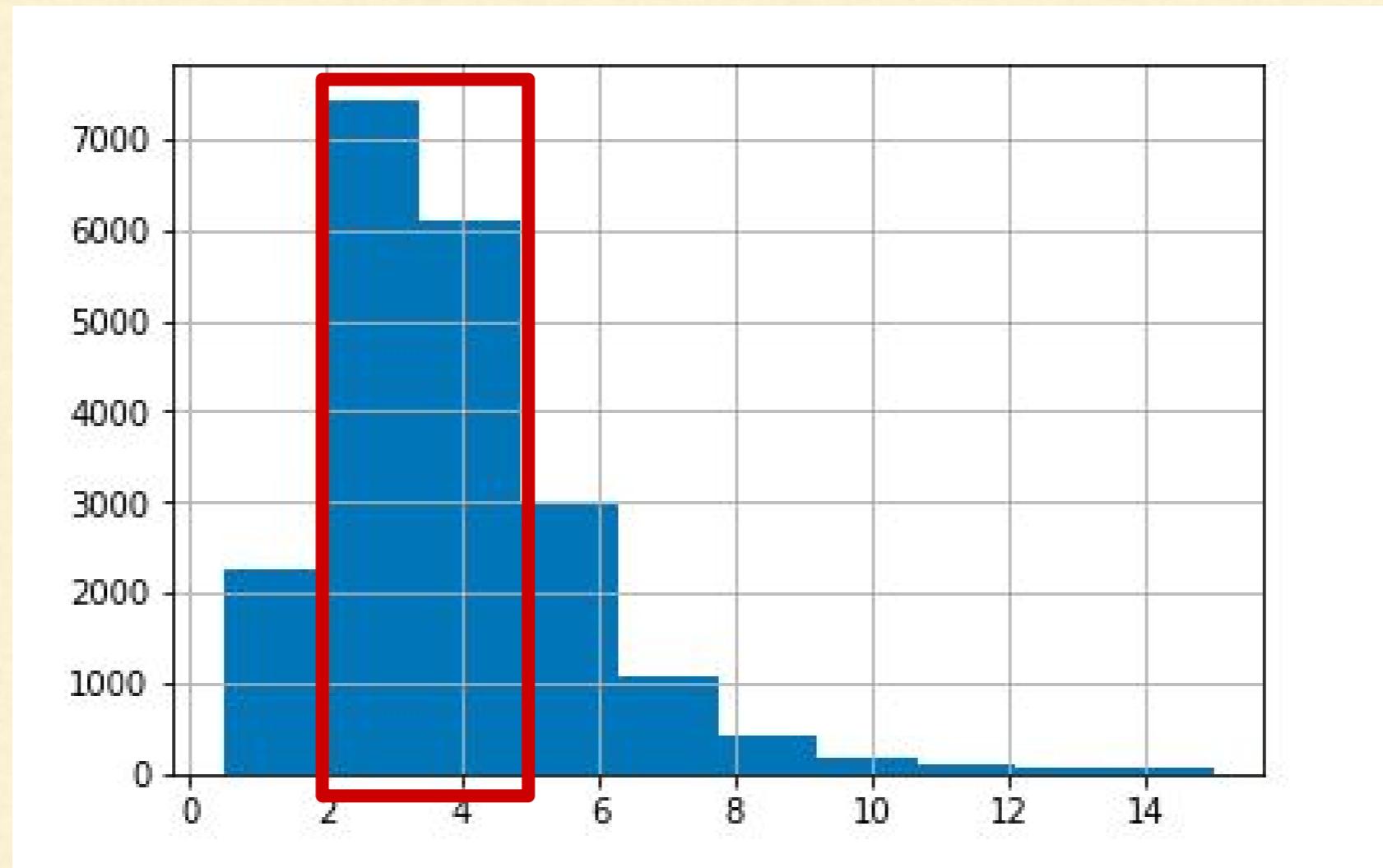
Observations?



# Create a Test Set

## Create Histogram of Median Income

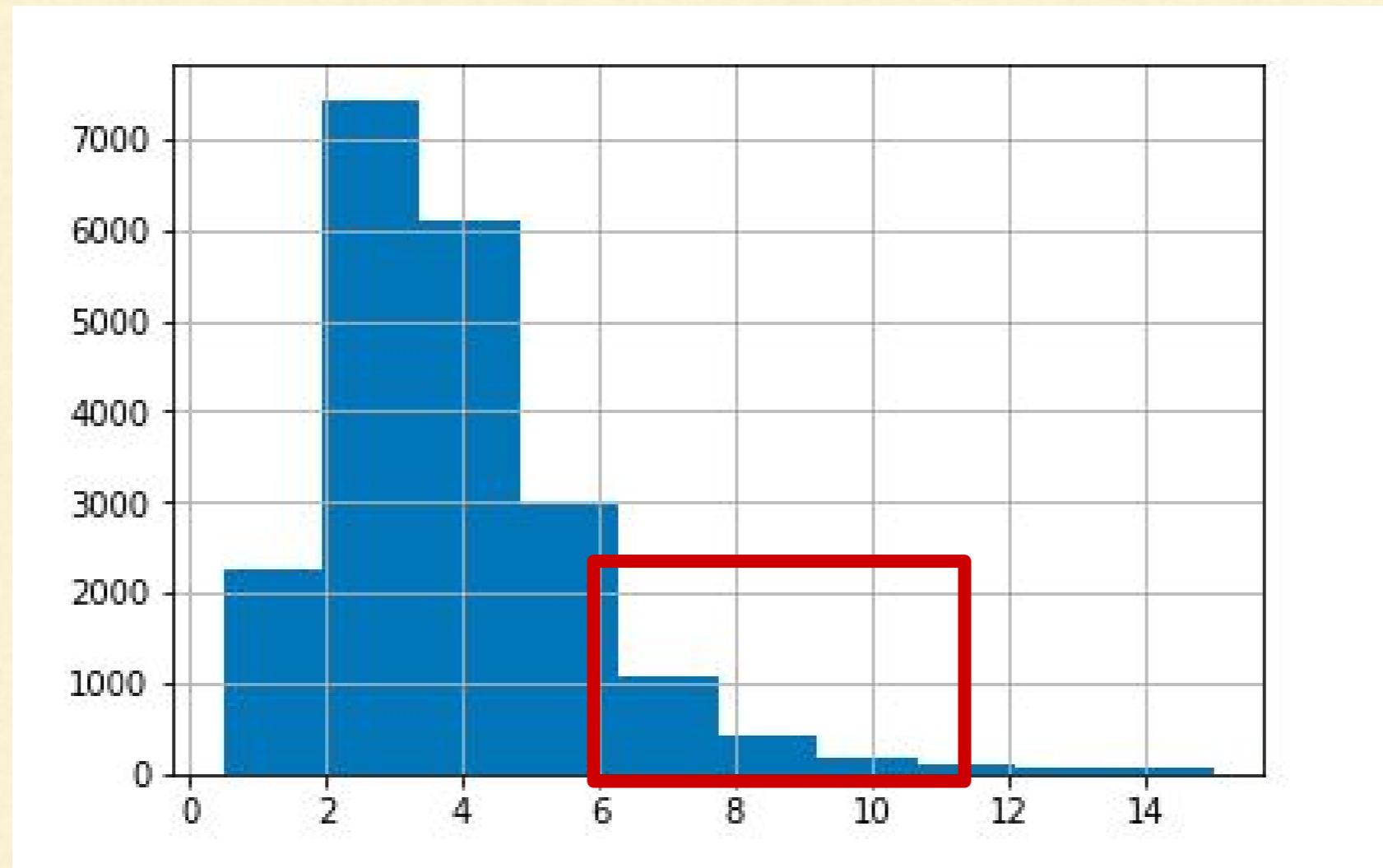
- Most values are clustered around 2-5 thousand dollars



# Create a Test Set

## Create Histogram of Median Income

- Some median income go beyond 6



# Create a Test Set

---

## Important Points

- Since median income is important attribute to predict median housing prices
- It is important to have sufficient number of instances
  - In the dataset for each **stratum**
- This means that
  - We should not have too many **strata**
  - And each **stratum** should be large enough

# Create a Test Set

---

Let's limit the number of categories of median  
income

# Create a Test Set

---

## Limit the Categories in Median Income

- Divide median income by 1.5 so that
  - We will not have too many **strata**
  - And each **stratum** will be large enough

# Create a Test Set

---

## Limit the Categories in Median Income

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

**Run it in Notebook**

# Create a Test Set

## Limit the Categories in Median Income

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

```
3.0    7236
2.0    6581
4.0    3639
5.0    2362
1.0    822
Name: income_cat, dtype: int64
```

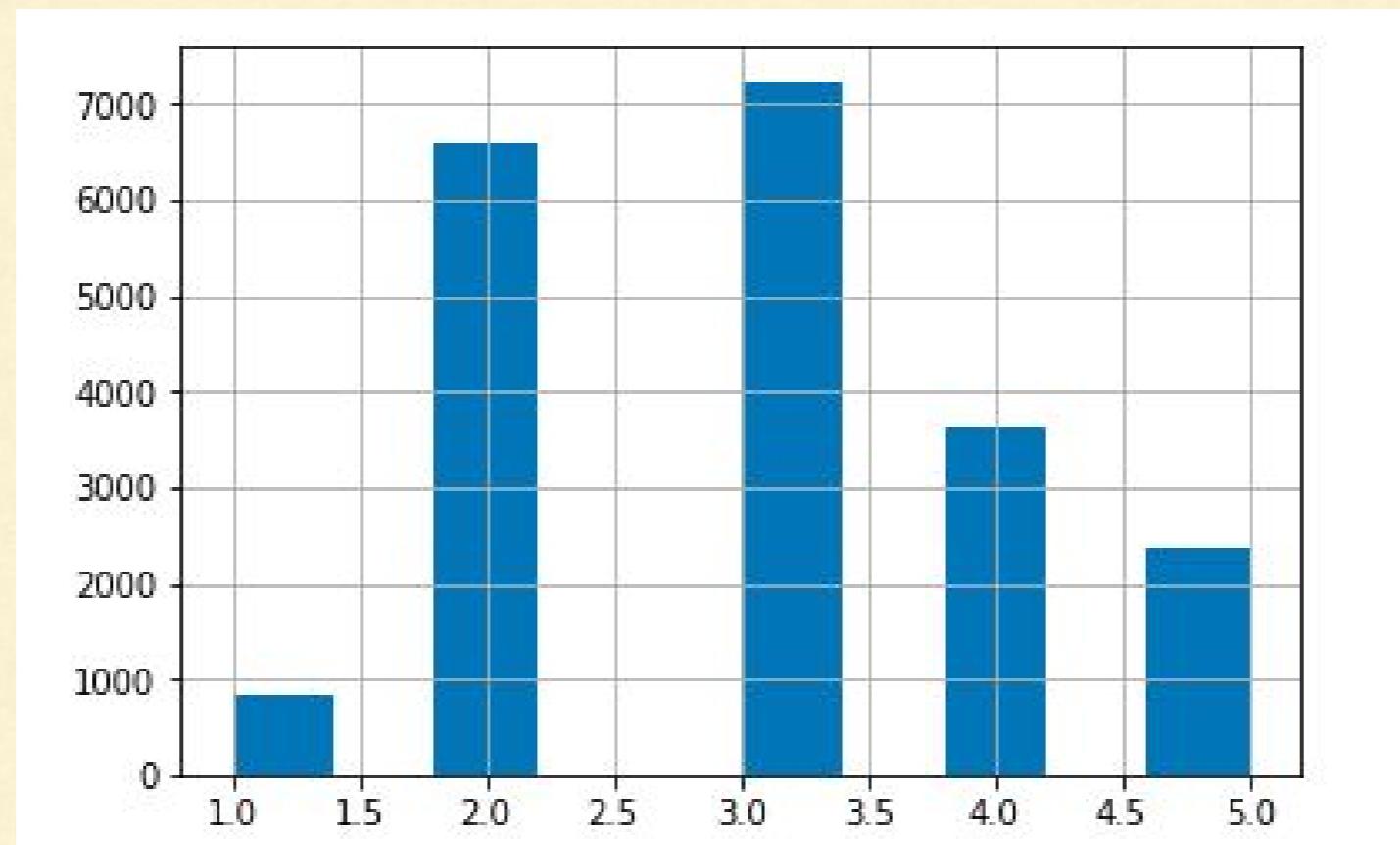
Each stratum is large enough

Not too many strata

# Create a Test Set

## Limit the Categories in Median Income

```
>>> housing["income_cat"].hist()
```



# Create a Test Set

---

## Stratified Sampling - Sklearn **StratifiedShuffleSplit** Class

- Now we are ready to do stratified sampling
- Use Scikit-learn's **StratifiedShuffleSplit** class

# Create a Test Set

## Stratified Sampling - Sklearn StratifiedShuffleSplit Class

```
>>> 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]
```

Run it in notebook

# Create a Test Set

---

Did Stratified Sampling work as expected?

# Create a Test Set

## Did Stratified Sampling Work - Stratified Sampling vs Full dataset

```
>>>  
strat_test_set["income_cat"].value  
_counts() / len(strat_test_set)
```

```
3.0    0.350533  
2.0    0.318798  
4.0    0.176357  
5.0    0.114583  
1.0    0.039729  
Name: income_cat, dtype: float64
```

Income category proportion in test set generated with stratified sampling

```
>>>  
housing["income_cat"].value_counts  
() / len(housing)
```

```
3.0    0.350581  
2.0    0.318847  
4.0    0.176308  
5.0    0.114438  
1.0    0.039826  
Name: income_cat, dtype: float64
```

Income category proportion in full dataset

# Create a Test Set

---

**Did Stratified Sampling Work?**

Yes, it worked.

Income category proportions are almost identical between stratified sampling and full dataset

# Create a Test Set

---

Let's compare Stratified Sampling and Random  
Sampling

# Create a Test Set

---

## Compare Stratified Sampling and Random Sampling

```
>>> def income_cat_proportions(data):
    return data["income_cat"].value_counts() / len(data)

>>> train_set, test_set = train_test_split(housing, test_size=0.2,
random_state=42)
>>> compare_props = pd.DataFrame({
        "Overall": income_cat_proportions(housing),
        "Stratified": income_cat_proportions(strat_test_set),
        "Random": income_cat_proportions(test_set),
    }).sort_index()
>>> compare_props["Rand. %error"] = 100 * compare_props["Random"] /
>>> compare_props["Overall"] - 100
>>> compare_props["Strat. %error"] = 100 * compare_props["Stratified"] /
compare_props["Overall"] - 100
```

# Create a Test Set

## Compare Stratified Sampling and Random Sampling

	Overall	Random	Stratified	Rand. %error	Strat. %error
1.0	0.039826	0.040213	0.039729	0.973236	-0.243309
2.0	0.318847	0.324370	0.318798	1.732260	-0.015195
3.0	0.350581	0.358527	0.350533	2.266446	-0.013820
4.0	0.176308	0.167393	0.176357	-5.056334	0.027480
5.0	0.114438	0.109496	0.114583	-4.318374	0.127011

Sampling bias comparison of stratified versus purely random sampling

# Create a Test Set

## Compare Stratified Sampling and Random Sampling

### Observations?

	Overall	Random	Stratified	Rand. %error	Strat. %error
1.0	0.039826	0.040213	0.039729	0.973236	-0.243309
2.0	0.318847	0.324370	0.318798	1.732260	-0.015195
3.0	0.350581	0.358527	0.350533	2.266446	-0.013820
4.0	0.176308	0.167393	0.176357	-5.056334	0.027480
5.0	0.114438	0.109496	0.114583	-4.318374	0.127011

Sampling bias comparison of stratified versus purely random sampling

# Create a Test Set

## Compare Stratified Sampling and Random Sampling

- Test set generated using stratified sampling has income category proportion almost identical to those in full data set

	Overall	Random	Stratified	Rand. %error	Strat. %error
1.0	0.039826	0.040213	0.039729	0.973236	-0.243309
2.0	0.318847	0.324370	0.318798	1.732260	-0.015195
3.0	0.350581	0.358527	0.350533	2.266446	-0.013820
4.0	0.176308	0.167393	0.176357	-5.056334	0.027480
5.0	0.114438	0.109496	0.114583	-4.318374	0.127011

Almost identical to  
full dataset

Income Category Proportions in Stratified  
Sampling

# Create a Test Set

## Compare Stratified Sampling and Random Sampling

- Test set generated using stratified sampling has income category proportion is quite skewed

	Overall	Random	Stratified	Rand. %error	Strat. %error
1.0	0.039826	0.040213	0.039729	0.973236	-0.243303
2.0	0.318847	0.324370	0.318798	1.732260	-0.015195
3.0	0.350581	0.358527	0.350533	2.266446	-0.013820
4.0	0.176308	0.167393	0.176357	-5.056334	0.027480
5.0	0.114438	0.109496	0.114583	-4.318374	0.127011

Quite skewed

Income Category Proportions in Purely Random Sampling

# Create a Test Set

---

**Compare Stratified Sampling and Random Sampling**

**Conclusion?**

# Create a Test Set

---

## **Compare Stratified Sampling and Random Sampling**

Stratified Sampling gives better test set than  
Random Sampling

# Create a Test Set

---

- Remove income\_cat attribute so that data is back to its original state

```
>>> for set in (strat_train_set, strat_test_set):  
    set.drop(["income_cat"], axis=1, inplace=True)
```

[Run in Notebook](#)

# Create a Test Set

---

Why did we spend so much time in test set generation?

# Create a Test Set

---

- Test set generation is often neglected
- But most important part of a Machine Learning project

# Checklist for Machine Learning Projects

---

1. Frame the problem and look at the big picture
2. Get the data
- 3. Explore the data to gain insights**
4. Prepare the data for Machine Learning algorithms
5. Explore many different models and short-list the best ones
6. Fine-tune model
7. Present the solution
8. Launch, monitor, and maintain the system

# Discover and Visualize the Data

---

- Let's understand the data in depth
- Make sure
  - Test set is kept aside
  - And explore only training set

# Discover and Visualize the Data

---

- If training set is large
  - Sample training set
  - To make manipulations easy and fast
- Since our training set is small
  - We can directly work on full set

# Discover and Visualize the Data

---

- Create copy of training set first
- So that we can play with it without harming the training set

```
>>> housing = strat_train_set.copy()
```

Run it on Notebook

# Discover and Visualize the Data

---

## Visualizing Geographical Data

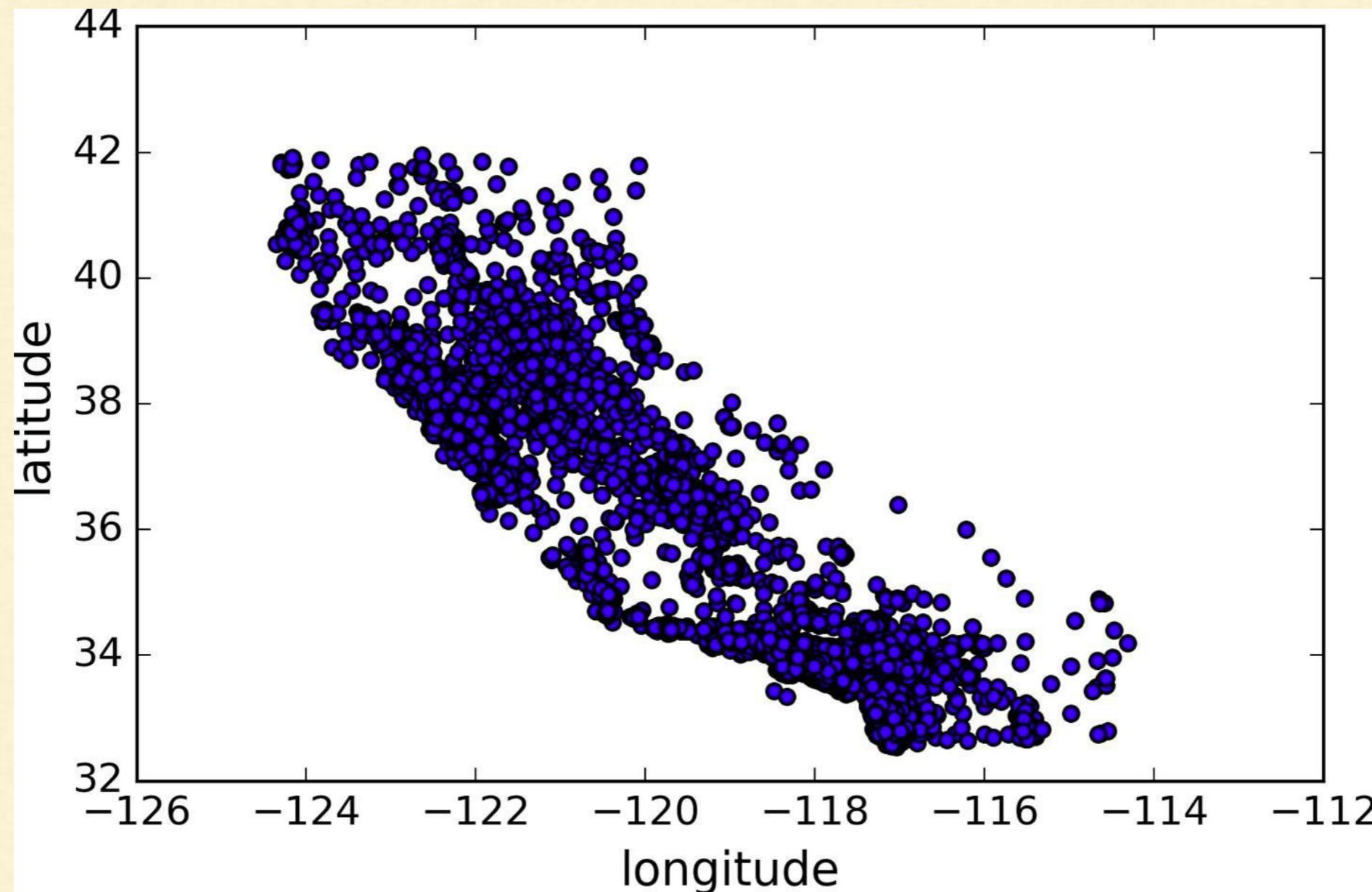
- Since there is geographical information of latitude and longitude
- Create Scatterplot of all district to visualize the data

```
>>> housing.plot(kind="scatter", x="longitude",
y="latitude")
```

Run it on Notebook

# Discover and Visualize the Data

## Visualizing Geographical Data



# Discover and Visualize the Data

---

## Problem?

- Hard to see any particular pattern
- We can not visualize the places with high density points

# Discover and Visualize the Data

---

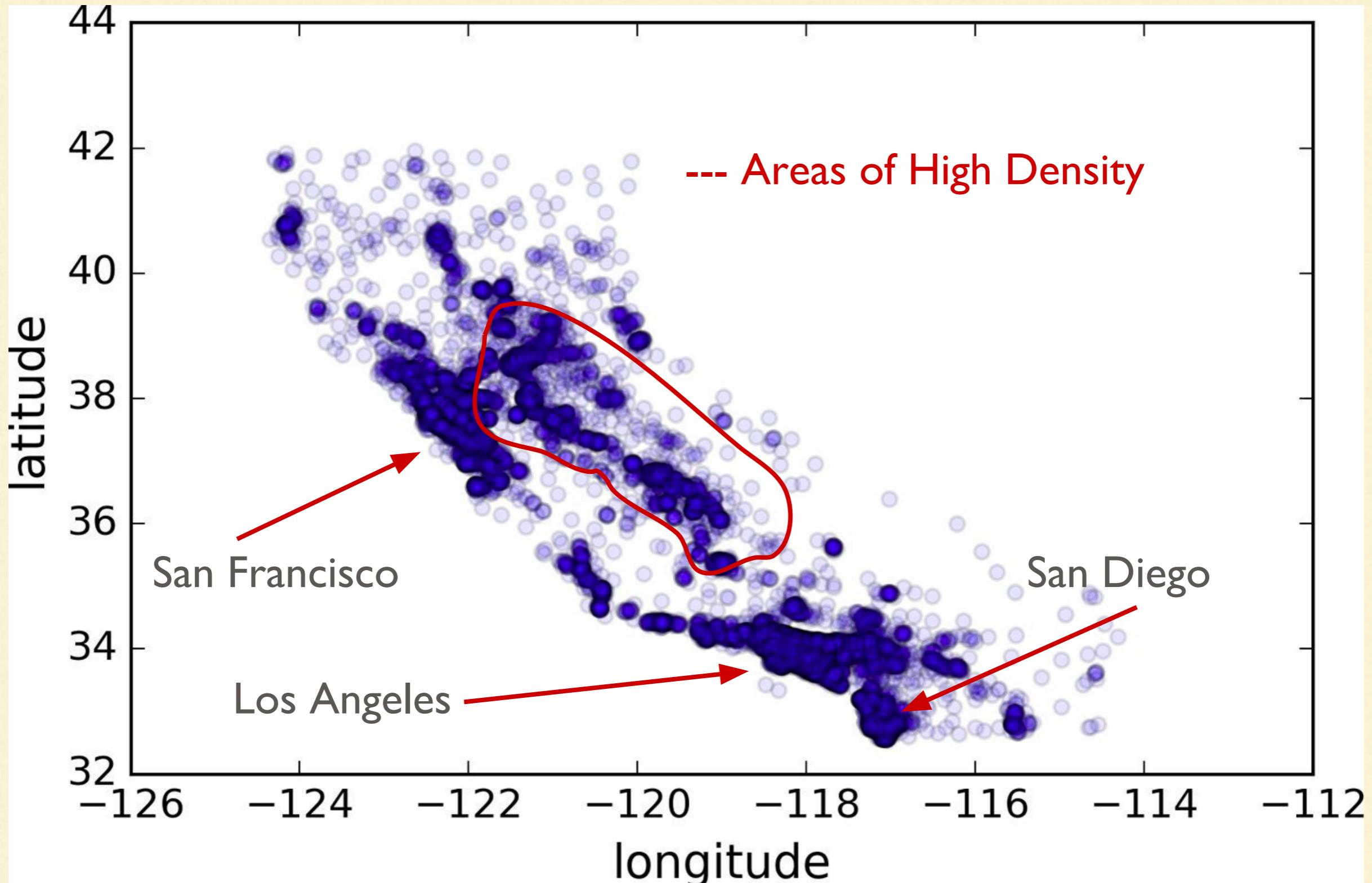
## Solution

- Setting the alpha option to 0.1
- Makes it look like heat map

```
>>> housing.plot(kind="scatter", x="longitude",
y="latitude", alpha=0.1)
```

**Run it on Notebook**

# Discover and Visualize the Data



# Discover and Visualize the Data

---

**Let's look into the patterns in depth**

```
>>> housing.plot(kind="scatter", x="longitude",
y="latitude", alpha=0.4,
    s=housing["population"]/100, label="population",
figsize=(10,7),
    c="median_house_value", cmap=plt.get_cmap("jet"),
colorbar=True,
sharex=False)

>>> plt.legend()
```

**Run it on Notebook**

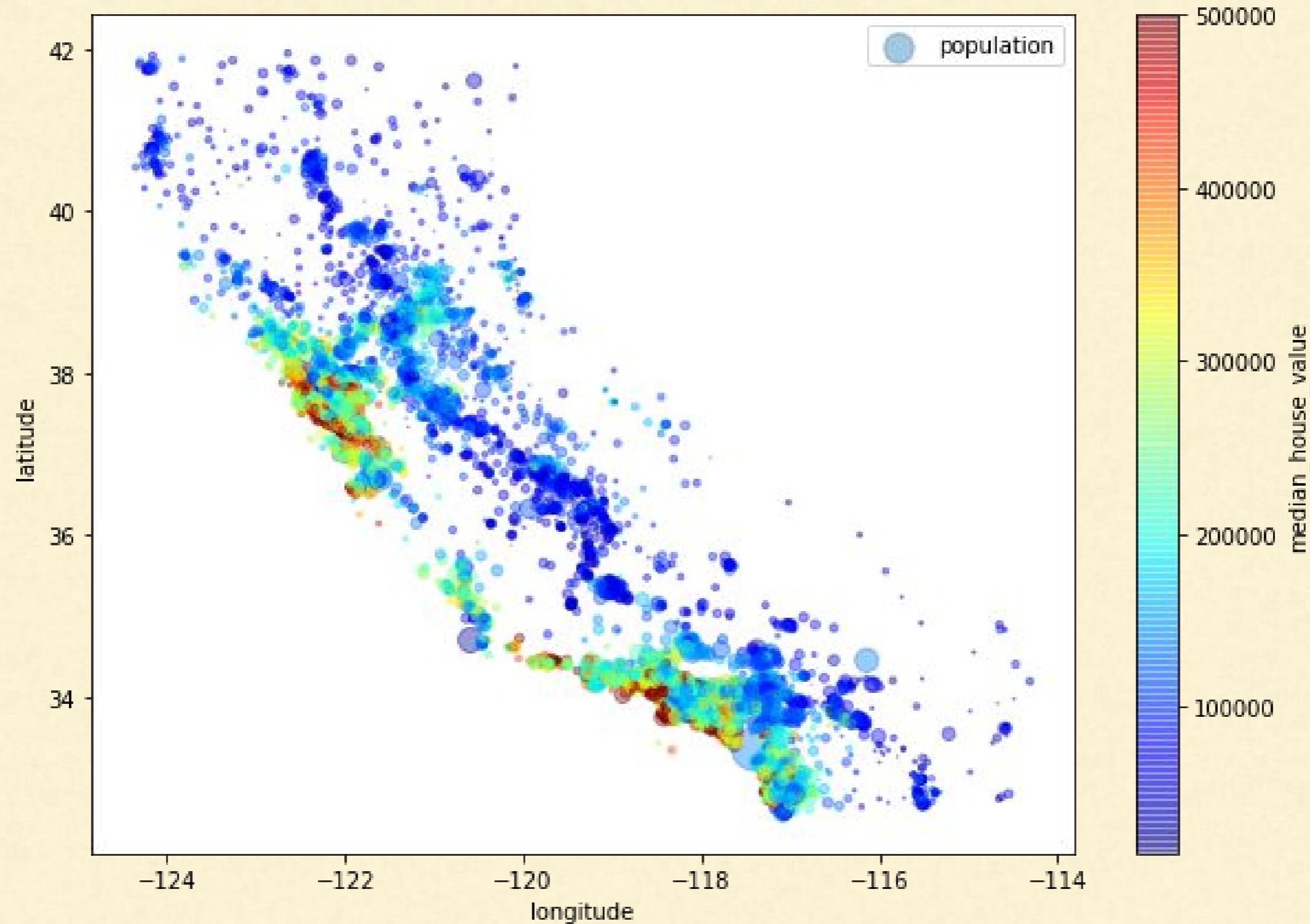
# Discover and Visualize the Data

---

## Let's look into the patterns in depth

- The radius of each circle represents the district's population (**option s**)
- The color represents the price (**option c**).
- We are using a predefined color map (**option cmap**)
  - Called **jet**
  - Which ranges from **blue** (low values)
  - To **red** (high prices)

# Discover and Visualize the Data

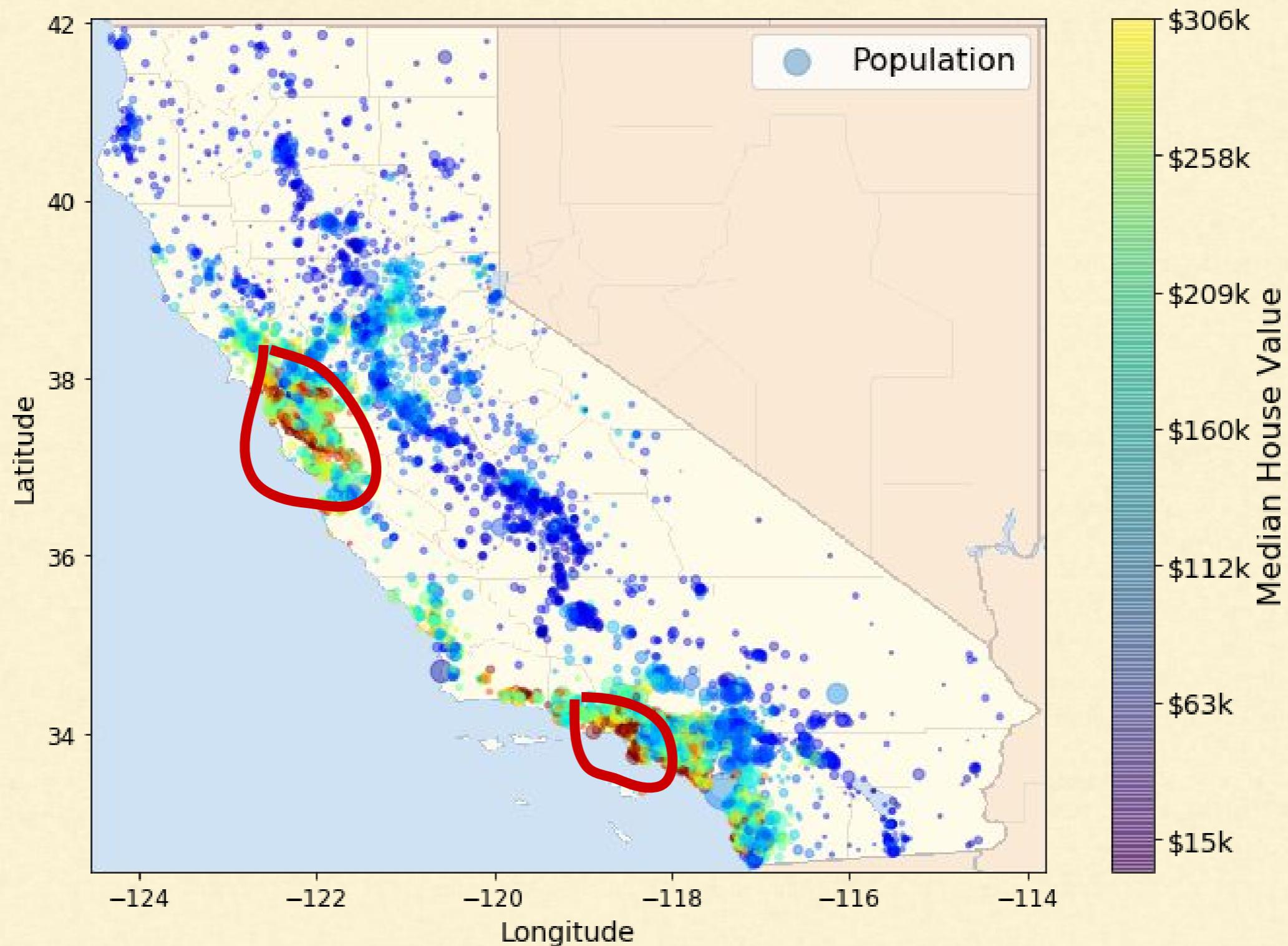


# Discover and Visualize the Data

---

**Observations?**

# Discover and Visualize the Data



# Discover and Visualize the Data

---

## Observations

- House prices are high
  - In the high density area
  - Closer to ocean

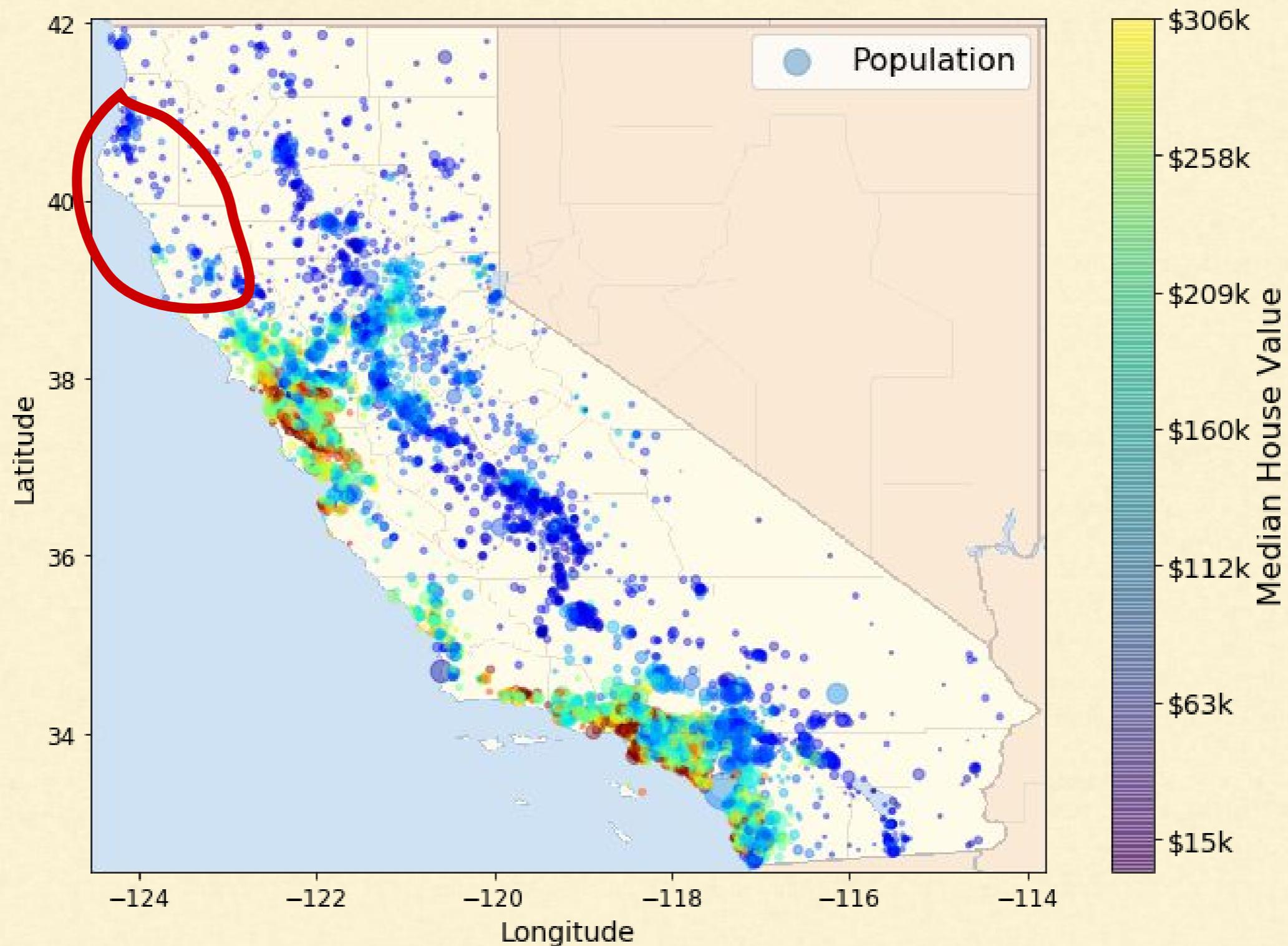
# Discover and Visualize the Data

---

## Observations

- Now look at this

# Discover and Visualize the Data



# Discover and Visualize the Data

---

## Observations

- In Northern California the housing prices in coastal districts are not too high
- So it is not a simple rule

# Discover and Visualize the Data

---

## Looking for Correlations

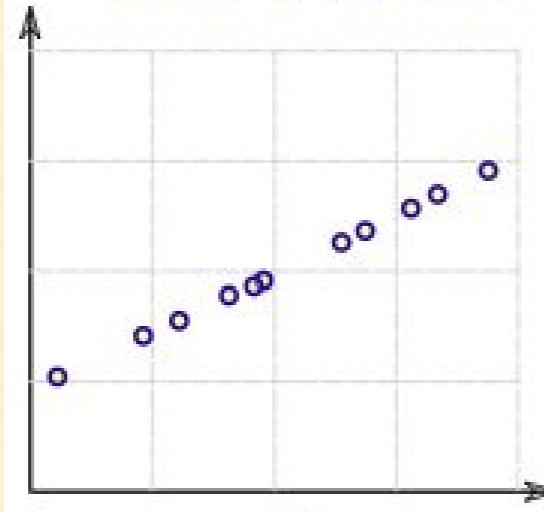
- Correlation indicates
  - The extent to which two or more variables fluctuate together

# Discover and Visualize the Data

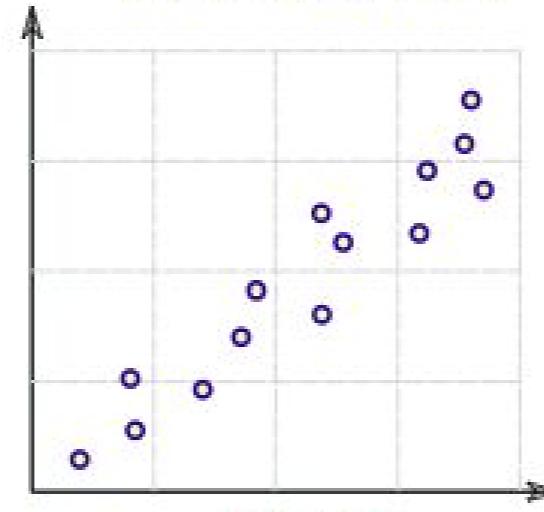
## Positive Correlation

When the values increase together

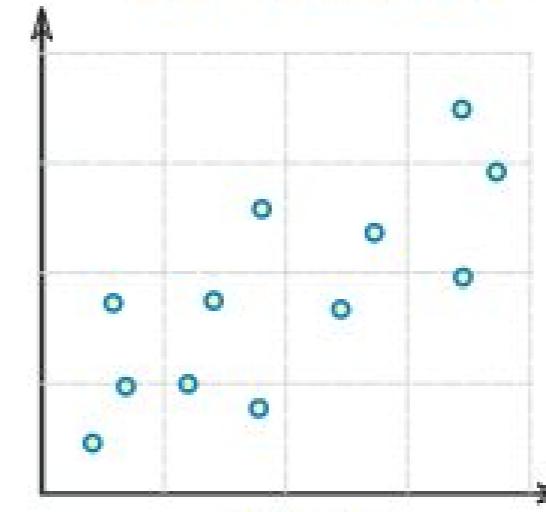
*Perfect  
Positive  
Correlation*



*High  
Positive  
Correlation*



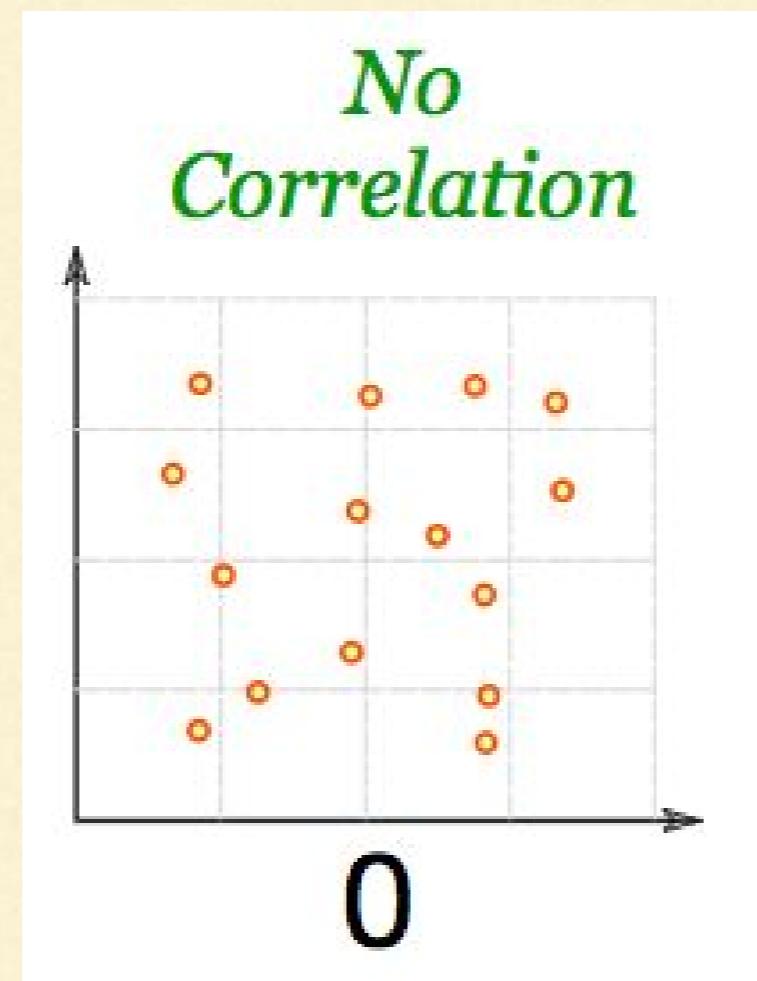
*Low  
Positive  
Correlation*



# Discover and Visualize the Data

## No Correlation

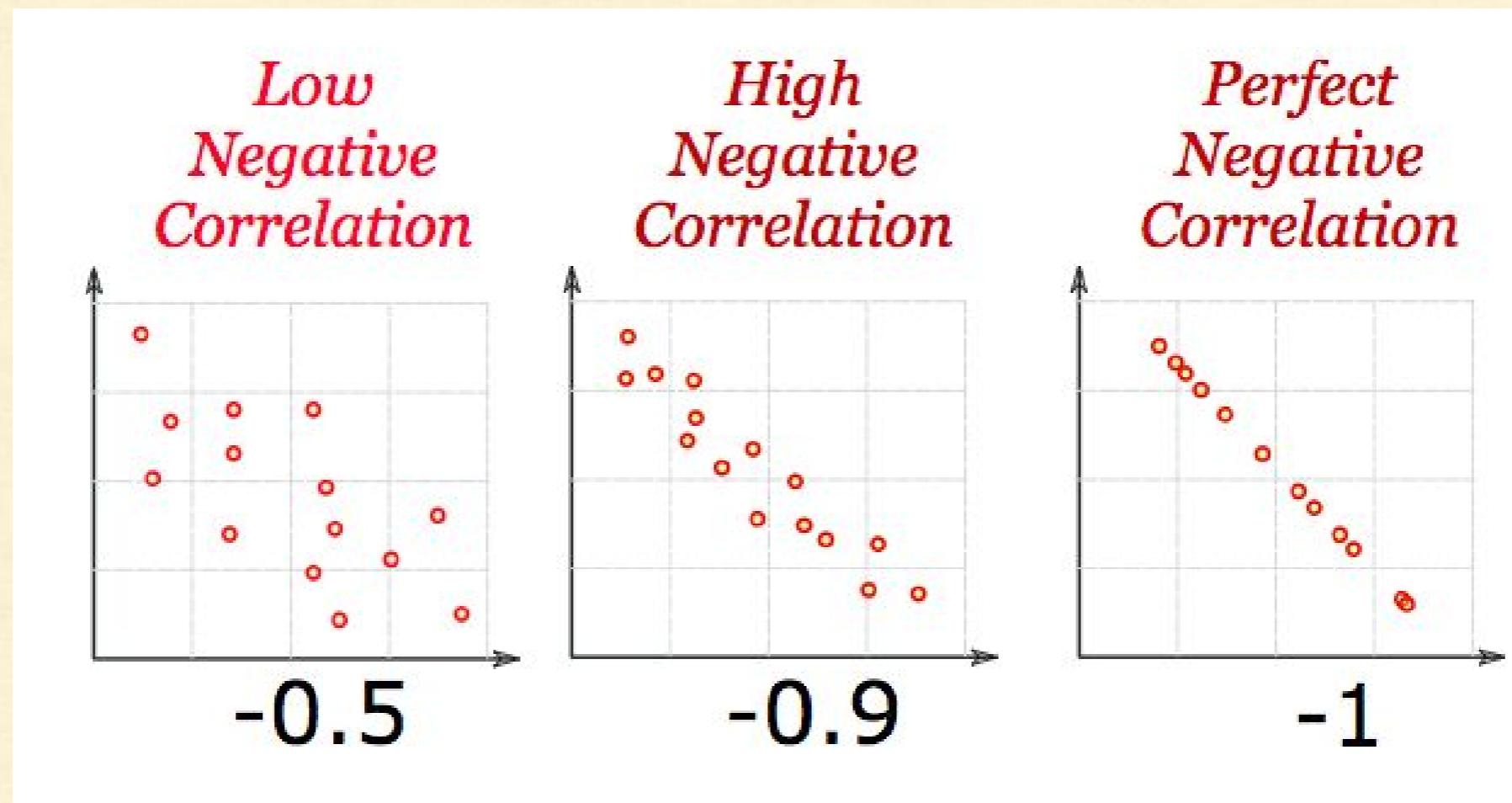
When values are not linked at all



# Discover and Visualize the Data

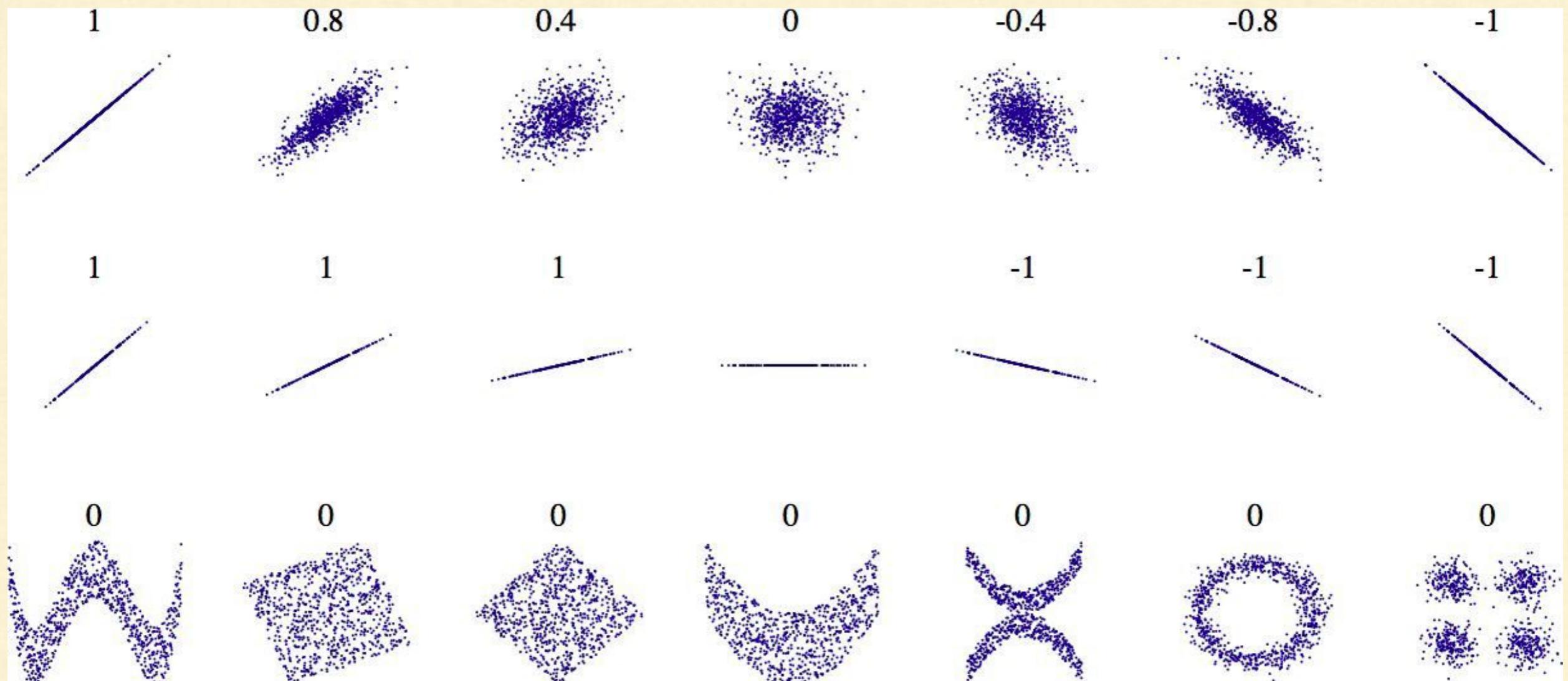
## Negative Correlation

When one value decreases as the other increases



# Discover and Visualize the Data

## Standard Correlation Coefficient of Various Datasets



# Discover and Visualize the Data

---

## Looking for Correlations

- Since the dataset is not too large
- We can compute the correlations
- Between every pair of attributes
- Using `corr()` method

# Discover and Visualize the Data

---

## Correlations with the Median House Value

```
>>> corr_matrix = housing.corr()  
>>>  
corr_matrix["median_house_value"].sort_values(ascending=  
False)
```

Run it in Notebook

# Discover and Visualize the Data

## Correlations with the Median House Value

```
median_house_value      1.000000
median_income          0.687160
total_rooms            0.135097
housing_median_age    0.114110
households             0.064506
total_bedrooms         0.047689
population             -0.026920
longitude              -0.047432
latitude               -0.142724
Name: median_house_value, dtype: float64
```

### Positive Correlation

Median house value tends to go up when the median income goes up

# Discover and Visualize the Data

## Correlations with the Median House Value

```
median_house_value      1.000000
median_income           0.687160
total_rooms              0.135097
housing_median_age       0.114110
households                0.064506
total_bedrooms            0.047689
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
Name: median_house_value, dtype: float64
```

### Negative Correlation

Between the latitude and the median house value

Prices have a slight tendency to go down when you go north

# Discover and Visualize the Data

---

## Looking for Correlations - `scatter_matrix`

- We can also look for correlations
  - Using Pandas' `scatter_matrix` function
  - Plots every numerical attribute against every other numerical attribute

# Discover and Visualize the Data

---

## Looking for Correlations - `scatter_matrix`

- Let's look for correlation using `scatter_matrix`
  - We have eleven numerical attributes
  - We'll get eleven by eleven plots (will not fit in page)

# Discover and Visualize the Data

---

## Looking for Correlations - scatter\_matrix

Focus on attributes most correlated

- median\_house\_value (target)
- median\_income
- total\_rooms
- Housing\_median\_age
- Note: *On same x & y, Pandas decides it can give you more useful information, and plots the density plot of just that column of data.*

# Discover and Visualize the Data

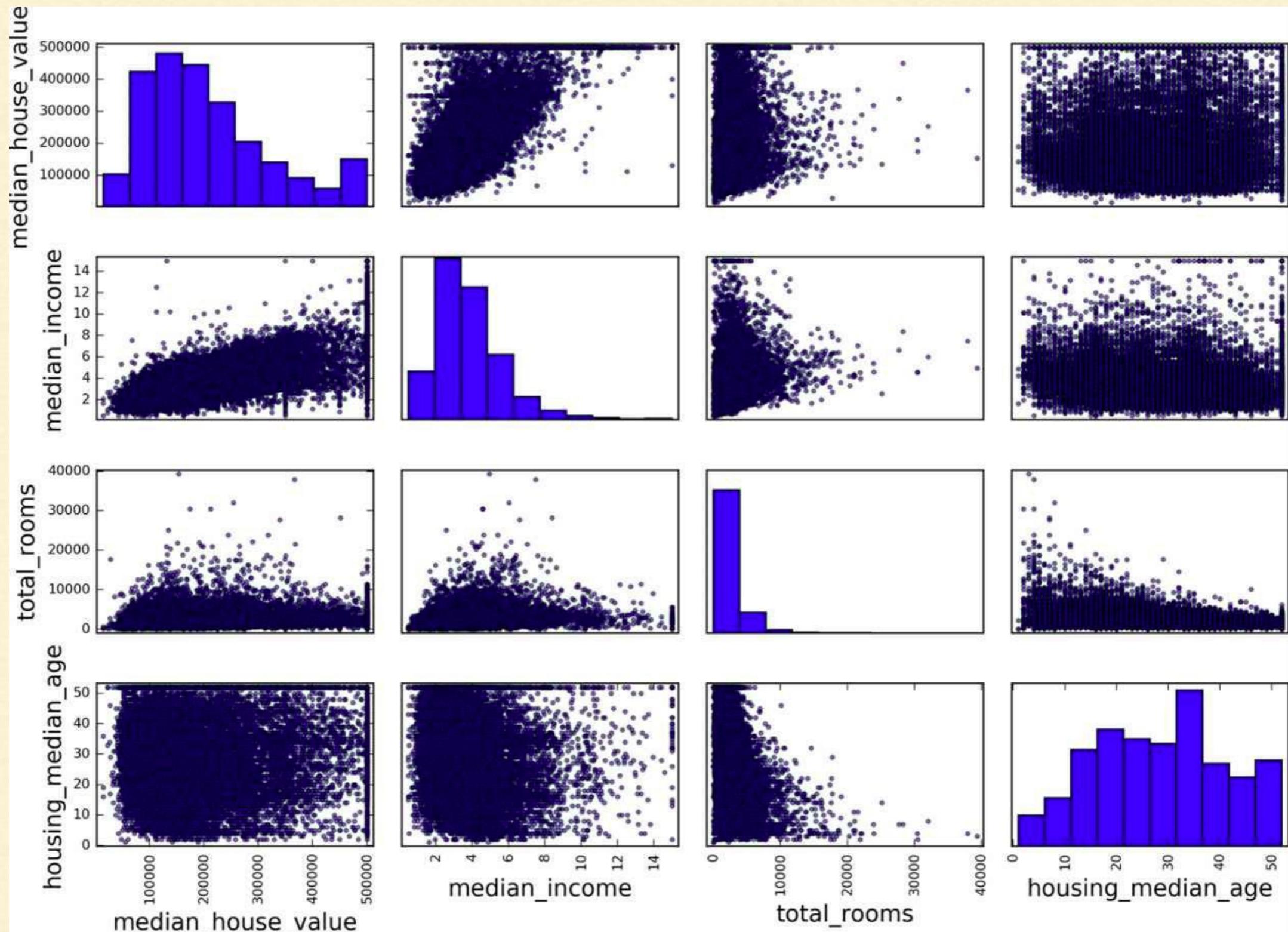
---

## Looking for Correlations - scatter\_matrix

```
>>> from pandas.tools.plotting import scatter_matrix  
>>> attributes = ["median_house_value", "median_income",  
"total_rooms", "housing_median_age"]  
>>> scatter_matrix(housing[attributes], figsize=(12, 8))
```

Run it in Notebook

# Discover and Visualize the Data



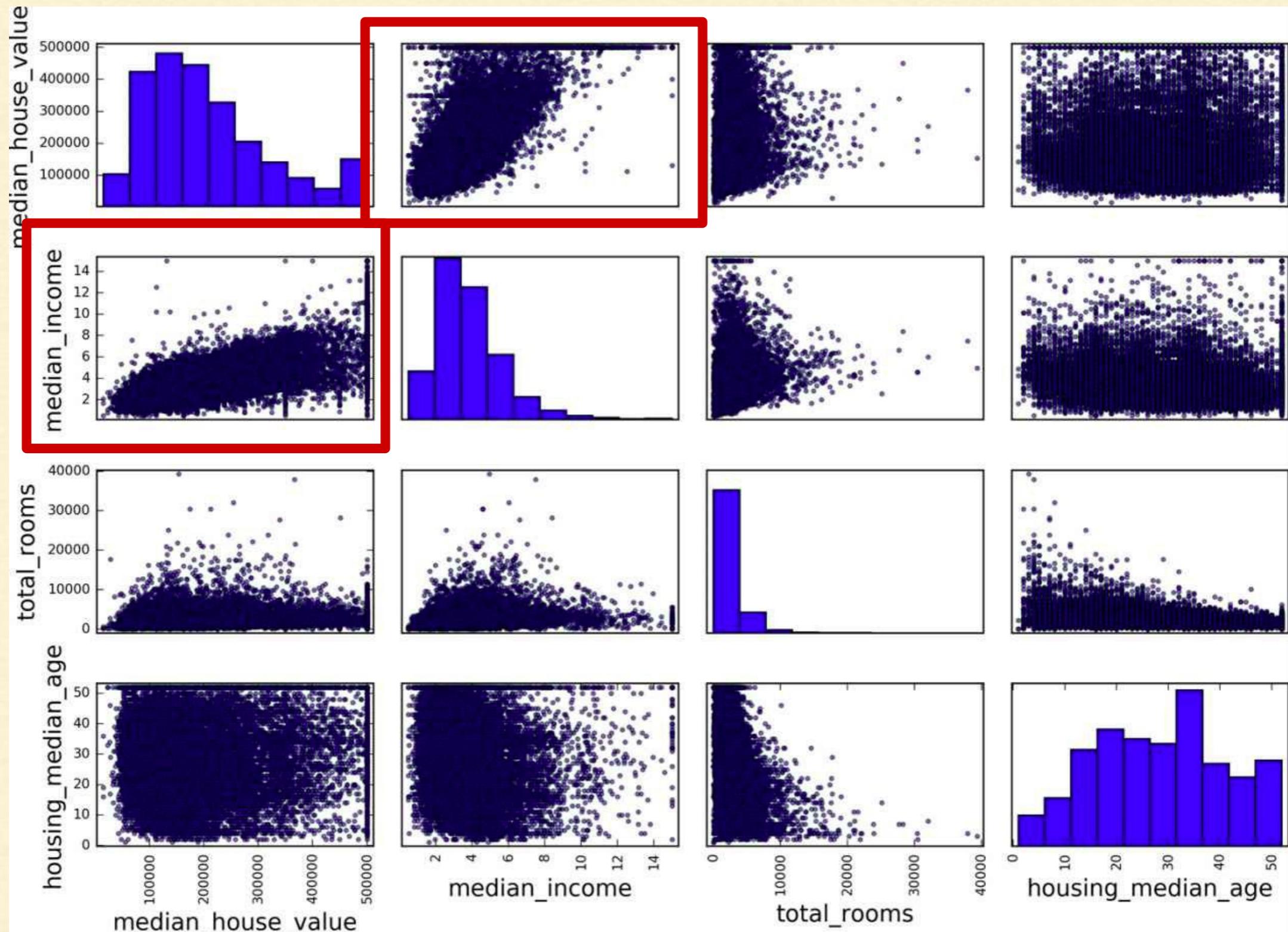
# Discover and Visualize the Data

---

## **Looking for Correlations - Most promising Attribute - Question**

Which is the most promising attribute to predict median house value from the correlation plot?

# Discover and Visualize the Data



# Discover and Visualize the Data

---

**Looking for Correlations - Most promising Attribute - Answer**

Median income

# Discover and Visualize the Data

---

## Looking for Correlations - Most promising Attribute

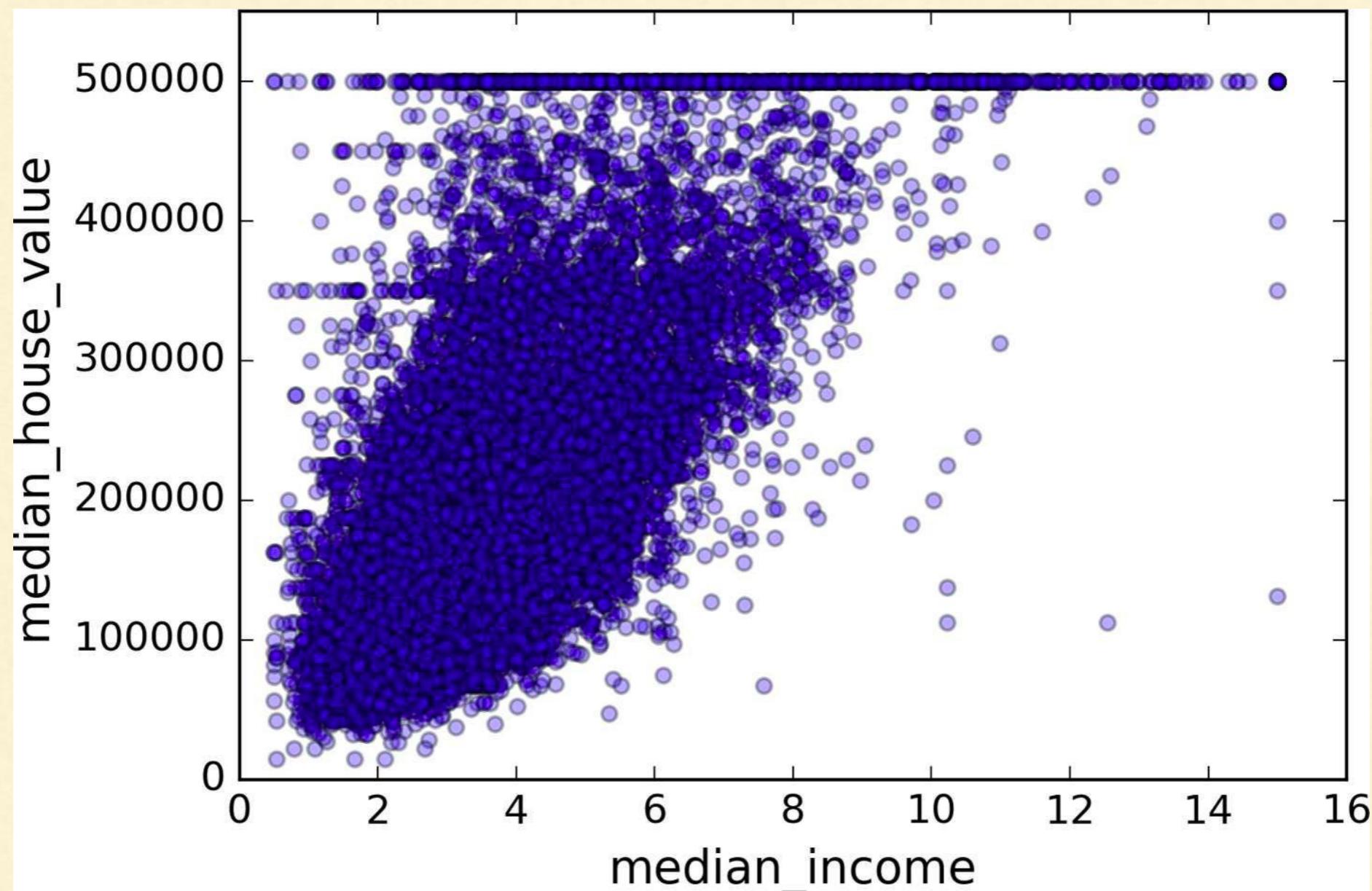
- Let's zoom in to see correlation between median house value and median income

```
>>> housing.plot(kind="scatter", x="median_income",
y="median_house_value", alpha=0.1)
>>> plt.axis([0, 16, 0, 550000])
```

Run it in Notebook

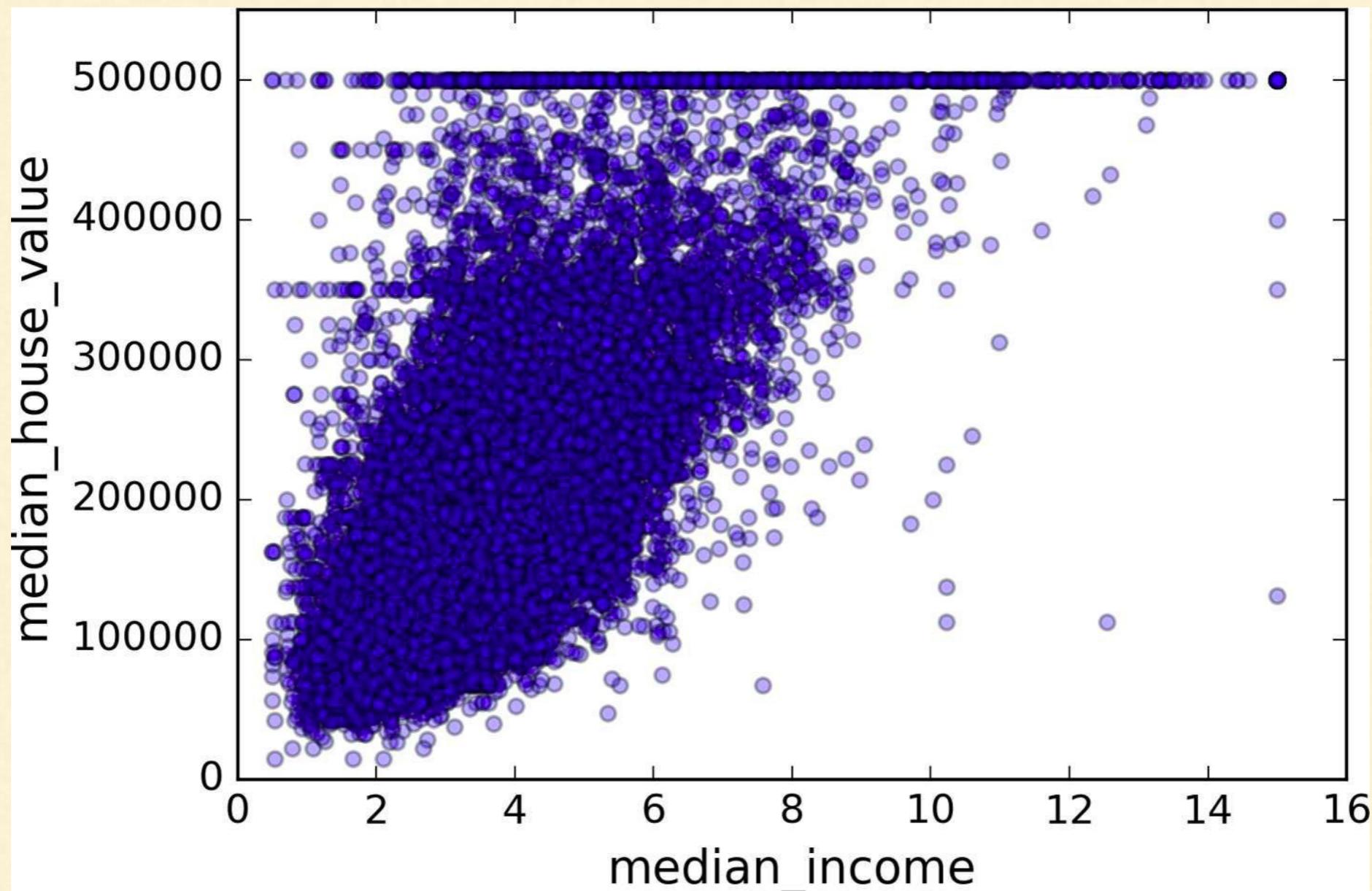
# Discover and Visualize the Data

**Looking for Correlations - Median House Value and Median Income**

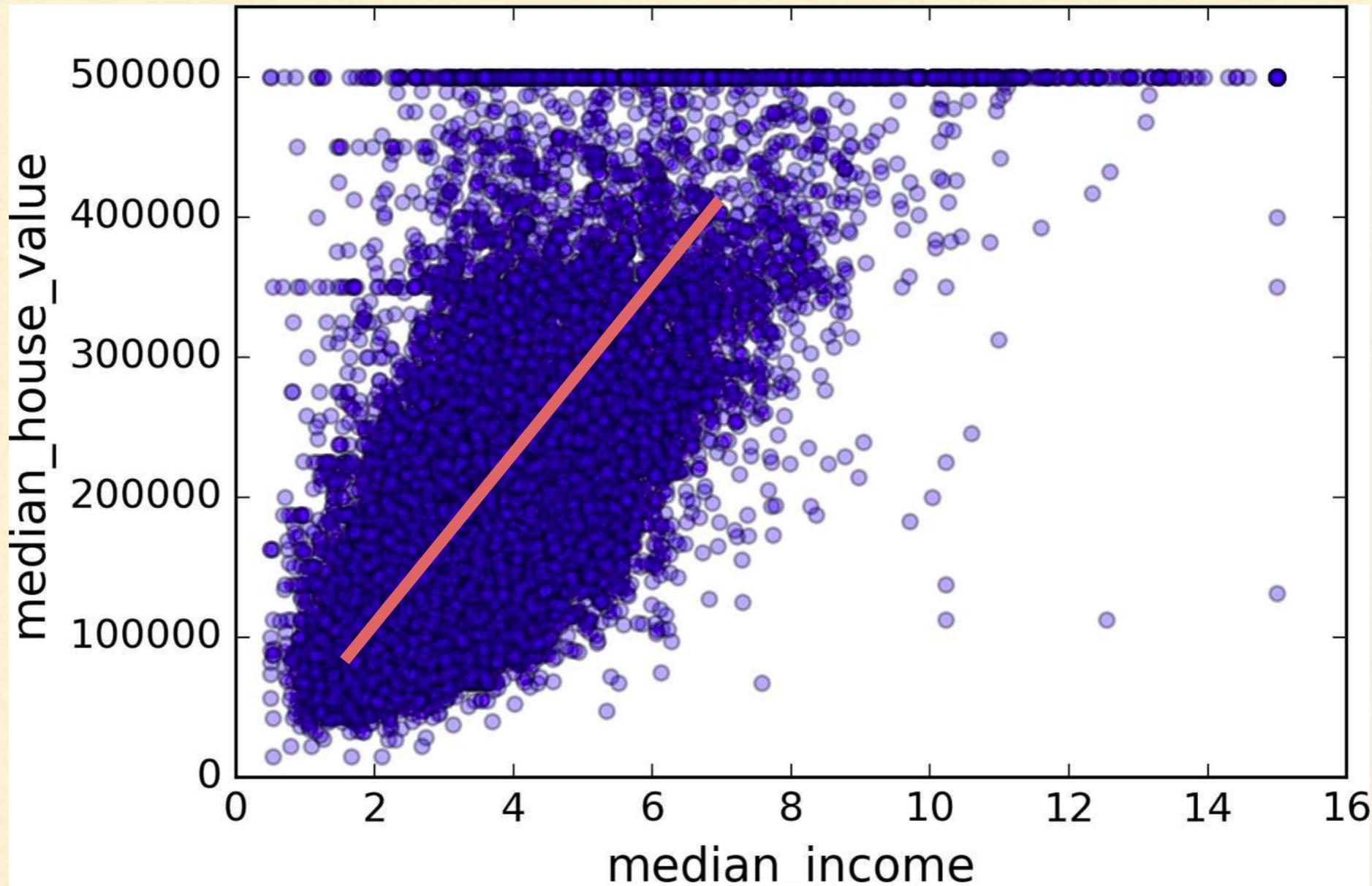


# Discover and Visualize the Data

## Observations??

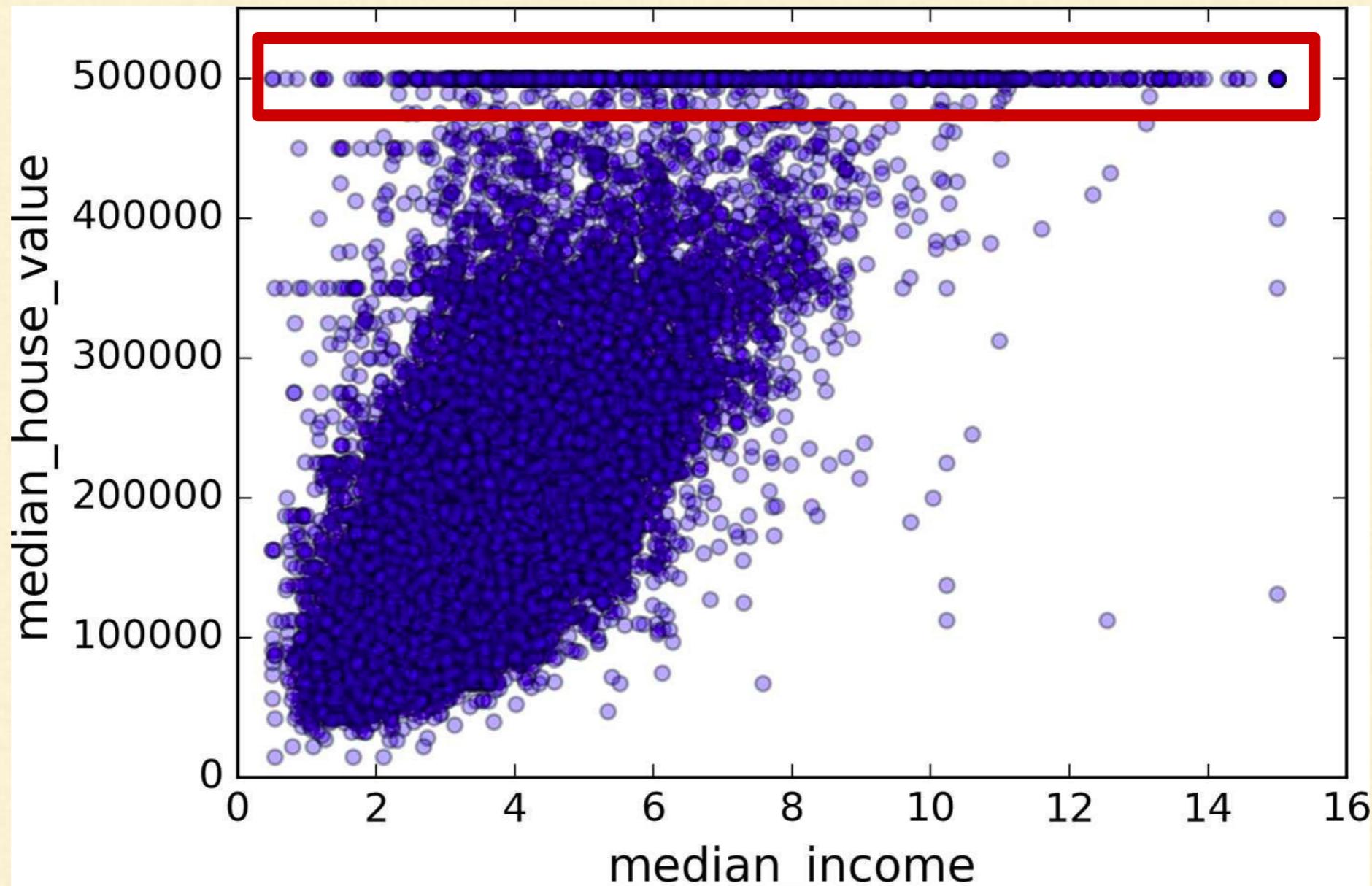


# Discover and Visualize the Data



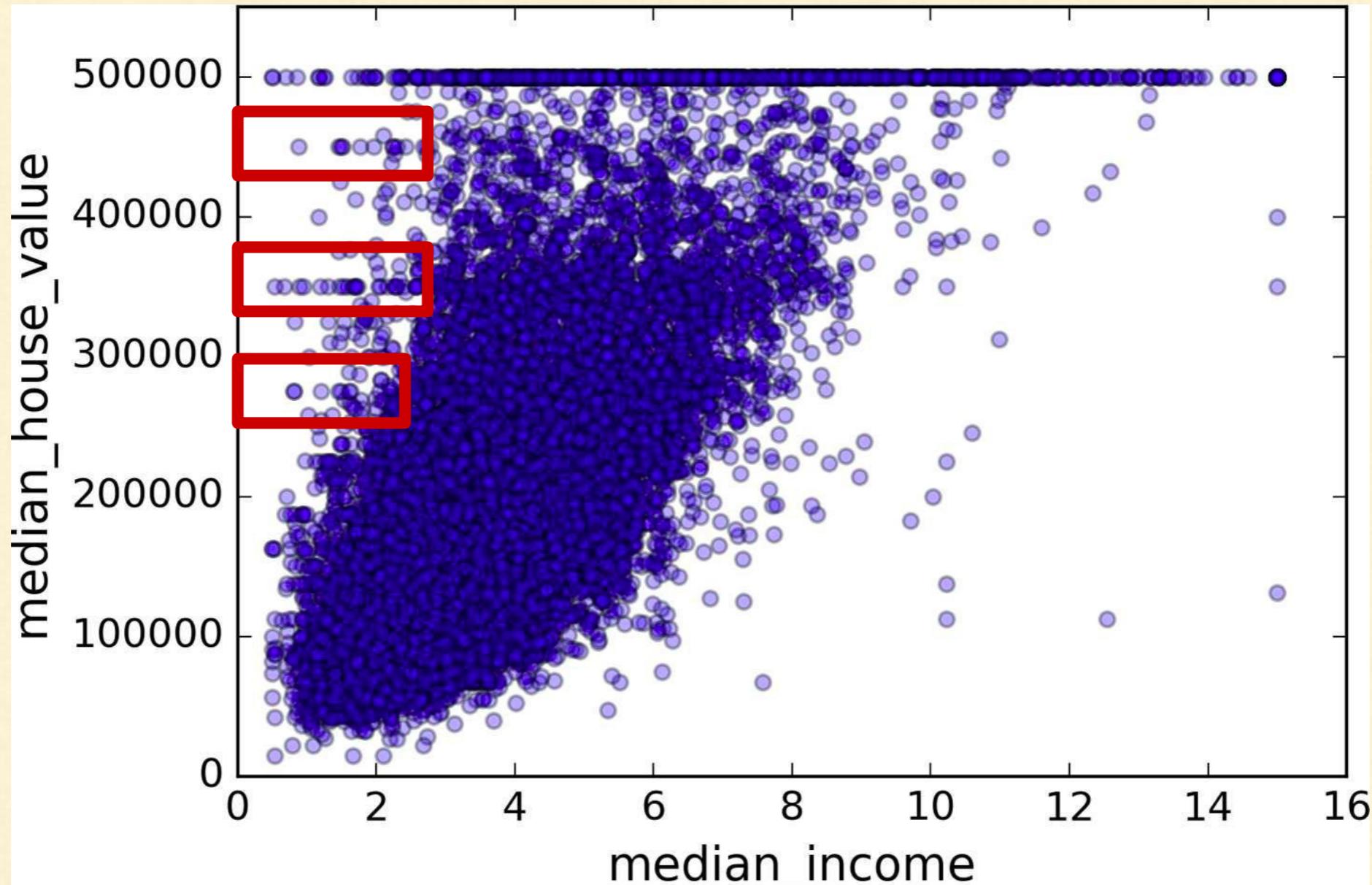
I. Correlation is very strong

# Discover and Visualize the Data



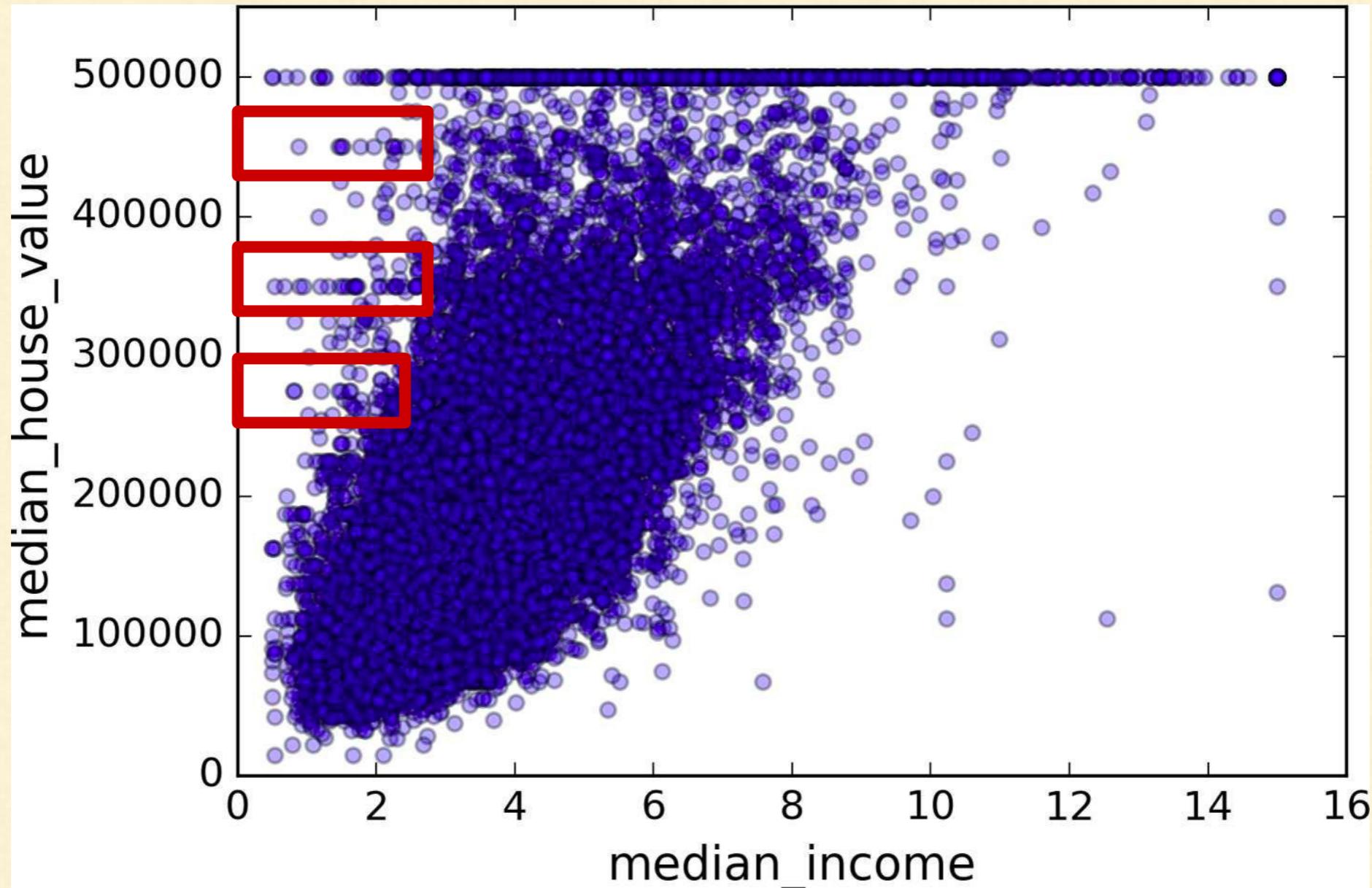
2. Price cap at \$500, 000

# Discover and Visualize the Data



3. Horizontal lines at \$450, 000 & \$350, 000 & \$280,000

# Discover and Visualize the Data



We should remove corresponding districts to prevent algorithm from learning to reproduce these data quirks

# Discover and Visualize the Data

---

## Experimenting with Attribute Combinations

- Till now we have covered
  - Ways to explore data
  - And Gain insights
- We've identified
  - Few data quirks - To be cleaned up before feeding the data to ML algorithm
  - Some attributes have a tail-heavy distribution

# Discover and Visualize the Data

---

## Experimenting with Attribute Combinations

- Our mileage will vary with each project
- But the general ideas are similar

# Discover and Visualize the Data

---

## Experimenting with Attribute Combinations

- Our mileage will vary with each project
- But the general ideas are similar
- We may want to try out various attribute combinations
  - Before preparing data for Machine Learning algorithms

# Discover and Visualize the Data

## Experimenting with Attribute Combinations - Examples

- The **total number of rooms** is not very useful
  - If we don't know how many households there are
  - What about **number of rooms per household**

	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

# Discover and Visualize the Data

## Experimenting with Attribute Combinations - Examples

- The **total number of bedrooms** is not very useful
- We want to compare it to
- **Number of bedrooms per room**

	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

# Discover and Visualize the Data

## Experimenting with Attribute Combinations - Examples

- **Population per household** also seems an interesting attribute combination
  - Number of people per household

	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

# Discover and Visualize the Data

---

## Experimenting with Attribute Combinations - Examples

- Let's create the new attributes

```
>>> 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"]
```

Run it in Notebook

# Discover and Visualize the Data

---

## Experimenting with Attribute Combinations - Examples

- Let's create correlation matrix again

```
>>> corr_matrix = housing.corr()
```

```
>>>
```

```
corr_matrix["median_house_value"].sort_values(ascending= False)
```

Run it in Notebook

# Discover and Visualize the Data

## Experimenting with Attribute Combinations - Examples

```
median_house_value          1.000000
median_income                0.687160
rooms_per_household         0.146285
total_rooms                  0.135097
housing_median_age           0.114110
households                   0.064506
total_bedrooms                0.047689
population_per_household    -0.021985
population                     -0.026920
longitude                      -0.047432
latitude                        -0.142724
bedrooms_per_room              -0.259984
Name: median_house_value, dtype: float64
```

# Discover and Visualize the Data

## Experimenting with Attribute Combinations - Examples

median_house_value	1.000000
median_income	0.687160
rooms_per_household	0.146285
total_rooms	0.135097
housing_median_age	0.114110
households	0.064506
total_bedrooms	0.047689
population_per_household	-0.021985
population	-0.026920
longitude	-0.047432
latitude	-0.142724
bedrooms_per_room	-0.259984
Name: median_house_value, dtype: float64	

# Discover and Visualize the Data

---

## Experimenting with Attribute Combinations - Examples

- **bedrooms\_per\_room** attribute is
  - Much more correlated with the median house value
  - Than the total number of rooms or bedrooms
  - Apparently houses with a lower bedroom/room ratio tend to be more expensive

# Discover and Visualize the Data

---

## Summary

- This round of exploration does not have to be thorough
- Insights gained from this step
  - Will help in building first prototype
- This step is an iterative process
  - Once prototype is up and running
  - Analyze to gain more insights
  - And repeat this step

# Checklist for Machine Learning Projects

---

1. Frame the problem and look at the big picture
2. Get the data
3. Explore the data to gain insights
- 4. Prepare the data for Machine Learning algorithms**
5. Explore many different models and short-list the best ones
6. Fine-tune model
7. Present the solution
8. Launch, monitor, and maintain the system

# Discover and Visualize the Data

---

Let's prepare data for Machine Learning  
algorithms

# Prepare the Data for ML Algorithms

---

## Automate the Process - Why?

- Write functions than manually doing it
  - Allows to reproduce these transformations easily on any dataset
  - Gradually build a library that you can reuse in future projects
  - Use these in your live system on new data
  - Easily try various transformations

# Prepare the Data for ML Algorithms

---

## Automate the Process

- Let's revert to a clean training set
  - Copy strat\_train\_set
  - Drop the target value from training set

# Prepare the Data for ML Algorithms

---

## Automate the Process

```
>>> housing = strat_train_set.drop("median_house_value",  
axis=1)
```

```
>>> housing_labels =  
strat_train_set["median_house_value"].copy()
```

**Note-** drop() creates a copy of the data and does not  
affect strat\_train\_set

**Run it in Notebook**

# Prepare the Data for ML Algorithms

---

## Data Cleaning

# Prepare the Data for ML Algorithms

---

## Data Cleaning

- ML algorithms can not work with missing features
- Create functions to take care of missing features

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values

- Already noticed that “total\_bedrooms” has missing values

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude          20640 non-null float64
latitude           20640 non-null float64
housing_median_age 20640 non-null float64
total_rooms         20640 non-null float64
total_bedrooms      20433 non-null float64
population         20640 non-null float64
households          20640 non-null float64
median_income       20640 non-null float64
median_house_value  20640 non-null float64
ocean_proximity    20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

20640 - 20433 =  
207 missing values

# Prepare the Data for ML Algorithms

---

## Data Cleaning - Missing Values - How to Fix?

- Three options to fix this
  - Get rid of corresponding districts where `total_bedroom` has missing values
  - Get rid of the whole attribute
  - Set the values to some value (zero, the mean, the median, etc.)

# Prepare the Data for ML Algorithms

---

## Data Cleaning - Missing Values - DataFrame's methods

- We can fix missing values using DataFrame's methods
  - `dropna()`
  - `drop()`
  - `fillna()`

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values - Sample Dataset

- Let's experiment with these methods on sample dataset

```
>>> sample_incomplete_rows =  
housing[housing.isnull().any(axis=1)].head()
```

Run it in Notebook

# Prepare the Data for ML Algorithms

---

## Data Cleaning - Missing Values - Option One

- `dropna()` - drops the missing values

```
>>> sample_incomplete_rows.dropna(subset=["total_bedrooms"])
```

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values - Option Two

- `drop()` - drops the attribute

```
>>> sample_incomplete_rows.drop(subset=["total_bedrooms"])
```

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	3468.0	1405.0	3.1662	<1H OCEAN

No total\_bedrooms Attribute Now

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values - Option Three

- `fillna()` - sets the missing values
- Let's fill the missing values with the median

```
>>> median = housing["total_bedrooms"].median()
```

```
>>>
```

```
sample_incomplete_rows["total_bedrooms"].fillna(median,  
inplace=True)
```

```
>>> sample_incomplete_rows
```

Run it in Notebook

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values - Option Three

- `fillna()` - sets the missing values

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	3.1662	<1H OCEAN

# Prepare the Data for ML Algorithms

---

## Data Cleaning - Missing Values

- In the previous step
  - Save the computed median value
  - Later we need the saved median value in test set
  - While evaluating the system
- Saved median value will also required
  - To replace missing values in the new data once we go live

# Prepare the Data for ML Algorithms

---

## Data Cleaning - Missing Values - Scikit-Learn Imputer Class

- Scikit-Learn provides **Imputer** class
  - To take care of missing values
- Create an instance of Imputer class
- Specify each attribute's missing values
  - Should be replaced with the median of that attribute

# Prepare the Data for ML Algorithms

---

## Data Cleaning - Missing Values - Scikit-Learn Imputer Class

- Create an instance of Imputer class
- Define strategy as median

```
>>> from sklearn.preprocessing import Imputer
```

```
>>> imputer = Imputer(strategy="median")
```

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values - Scikit-Learn Imputer Class

- Median can be computed
  - Only on numerical attributes
- Create copy of data
  - Without text attribute ocean\_proximity

```
>>> housing_num = housing.drop("ocean_proximity",
axis=1)
```

# Prepare the Data for ML Algorithms

---

## Data Cleaning - Missing Values - Fit the Imputer Instance

- Fit the imputer instance to the training data
- Using the fit() method

```
>>> imputer.fit(housing_num)
```

# Prepare the Data for ML Algorithms

---

## Data Cleaning - Missing Values - Fit the Imputer Instance

- After calling `fit()`
  - Imputer class computes the median of every attribute
  - And stores the computed medians in
  - **statistics\_** instance variable

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values - Transform Training Set

- Now let's replace missing values in the training set

```
>>> X = imputer.transform(housing_num)
```

```
>>> X
```

```
array([[ -121.89 ,  37.29 ,  38. , ...,  710. ,  339. ,  
       2.7042],  
      [ -121.93 ,  37.05 ,  14. , ...,  306. ,  113. ,  
       6.4214],  
      [ -117.2 ,  32.77 ,  31. , ...,  936. ,  462. ,  
       2.8621],  
      ...,  
      [ -116.4 ,  34.09 ,   9. , ...,  2098. ,  765. ,  
       3.2723],  
      [ -118.01 ,  33.82 ,  31. , ...,  1356. ,  356. ,  
       4.0625],  
      [ -122.45 ,  37.77 ,  52. , ...,  1269. ,  639. ,  
       3.575 ]])
```

Numpy array

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values - Transform Training Set

- Convert Numpy array to Pandas Dataframe

```
>>> housing_tr = pd.DataFrame(  
    X,  
    columns=housing_num.columns  
)  
>>> housing_tr.head()
```

# Prepare the Data for ML Algorithms

## Data Cleaning - Missing Values - Transformed Training Set

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042
1	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214
2	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621
3	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839
4	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347

Transformed Training Set

# Prepare the Data for ML Algorithms

---

Question - In the training set, only `total_bedroom` attribute was having missing values. Why did we transform the entire training set?

# Prepare the Data for ML Algorithms

---

## Answer

- To be on the safer side
- As new data may have missing values when system go live

# Prepare the Data for ML Algorithms

---

Handling Text and Categorical Attributes

# Prepare the Data for ML Algorithms

---

## Handling Categorical Attributes

- Most ML algorithms prefer to work with numbers
- Let's convert “**ocean\_proximity**” attribute to numbers

# Prepare the Data for ML Algorithms

## Handling Categorical Attributes

```
>>> housing_cat = housing['ocean_proximity']
>>> housing_cat.head(10)
```

```
17606      <1H OCEAN
18632      <1H OCEAN
14650      NEAR OCEAN
3230       INLAND
3555       <1H OCEAN
19480       INLAND
8879       <1H OCEAN
13685       INLAND
4937       <1H OCEAN
4861       <1H OCEAN
Name: ocean_proximity, dtype: object
```

# Prepare the Data for ML Algorithms

## Example - Pandas's factorize() Method

```
>>> df = pd.DataFrame({  
    'A':['type1','type3','type3', 'type2', 'type0']  
})
```

```
>>> df[ 'A'].factorize()
```

```
(array([0, 1, 1, 2, 3]),  
Index(['type1', 'type3', 'type2', 'type0'], dtype='object'))
```

type1

type3

type3

type2

type0

# Prepare the Data for ML Algorithms

---

## Handling Categorical Attributes - Convert Text Labels to Numbers

```
>>> housing_cat_encoded, housing_categories =  
housing_cat.factorize()  
>>> housing_cat_encoded[:10]
```

Output-

```
array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0])
```

# Prepare the Data for ML Algorithms

---

## Handling Categorical Attributes - Pandas's factorize() Method

Check Categories

```
>>> housing_categories
```

Output -

```
Index(['<1H OCEAN', 'NEAR OCEAN', 'INLAND', 'NEAR BAY',  
'ISLAND'], dtype='object')
```

# Prepare the Data for ML Algorithms

---

## Handling Categorical Attributes

- In the previous example,
  - Machine Learning algorithm may assume that
  - Two nearby values are more similar than
  - Two distant values
- ML algo may assume that
  - 0 and 1 are similar to each other as their distance is less

```
array([ 0 , 0 , 1 , 2 , 0 , 2 , 0 , 2 , 0 , 0 ])
```

# Prepare the Data for ML Algorithms

---

## Handling Categorical Attributes - One-Hot Encoding

- To fix this,
  - A common solution is to create one binary attribute per category
- Example
  - One attribute equal to
    - 1 when the category is “<IH OCEAN”
    - and 0 otherwise
  - Another attribute equal to
    - 1 when the category is “INLAND”
    - and 0 otherwise

# Prepare the Data for ML Algorithms

## Handling Categorical Attributes - One Hot Encoding

CompanyName	Categoricalvalue	Price
VW	1	20000
Acura	2	10011
Honda	3	50000
Honda	3	10000

Sample Dataset

VW	Acura	Honda	Price
1	0	0	20000
0	1	0	10011
0	0	1	50000
0	0	1	10000

One Hot Encoding

**Only one attribute will be equal to 1 (hot), while the others will be 0 (cold)**

# Prepare the Data for ML Algorithms

---

## Handling Categorical Attributes - **OneHotEncoder**

- Scikit-Learn provides **OneHotEncoder** encoder to convert
  - Integer categorical values
  - Into one-hot vectors
- Let's encode the categories as one-hot vectors

# Prepare the Data for ML Algorithms

## Handling Categorical Attributes - OneHotEncoder

```
>>> from sklearn.preprocessing import OneHotEncoder  
>>> encoder = OneHotEncoder()  
>>> housing_cat_1hot =  
encoder.fit_transform(housing_cat_encoded.reshape(-1,1))  
>>> housing_cat_1hot
```

Output -

```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'  
with 16512 stored elements in Compressed Sparse Row format>
```

# Prepare the Data for ML Algorithms

## Handling Categorical Attributes - OneHotEncoder

- OneHotEncoder returns a SciPy sparse array
- We can convert it to a dense array if needed

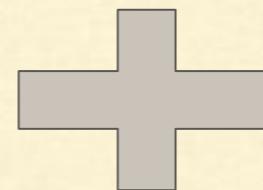
```
>>> housing_cat_1hot.toarray()
```

```
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.]])
```

# Prepare the Data for ML Algorithms

## Handling Categorical Attributes - Combining

Text Categories to  
Integer Categories

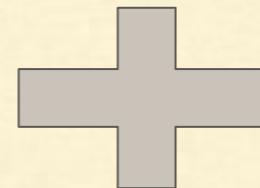


Integer Categories to  
One Hot Vectors

# Prepare the Data for ML Algorithms

## Handling Categorical Attributes - Combining

Text Categories to  
Integer Categories



Integer Categories to  
One Hot Vectors

Use CategoricalEncoder to Combine the Two  
Operations

# Prepare the Data for ML Algorithms

---

## Handling Categorical Attributes - CategoricalEncoder

- CategoricalEncoder combines the process of transforming
  - Text Categories to Integer Categories
  - Integer Categories to One Hot Vectors
- We've defined the code for CategoricalEncoder in the notebook
  - This code will be shipped in Scikit-Learn in future releases
- For now just go to Notebook and run it

# Prepare the Data for ML Algorithms

## Handling Categorical Attributes - CategoricalEncoder

```
>>> cat_encoder = CategoricalEncoder(encoding="onehot-dense")
>>> housing_cat_reshaped = housing_cat.values.reshape(-1, 1)
>>> housing_cat_1hot = cat_encoder.fit_transform(housing_cat_reshaped)
>>> housing_cat_1hot
```

```
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       ...,
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.]])
```

# Prepare the Data for ML Algorithms

## Handling Categorical Attributes - CategoricalEncoder

```
>>> cat_encoder.categories_
```

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'], dtype=object)]
```

# Prepare the Data for ML Algorithms

---

## Custom Transformers

# Prepare the Data for ML Algorithms

---

## Custom Transformers

- We can write our own transformers for tasks such as
  - Custom cleanup operations or
  - Combining specific attributes

# Prepare the Data for ML Algorithms

---

## Custom Transformers

- Scikit-Learn **Pipeline** class helps us in
  - Defining and Executing sequence of transformations
  - In the right order

# Prepare the Data for ML Algorithms

---

## How to Create Custom Transformers

### Steps

- Create a class
- Implement three methods
  - `fit()`
  - `transform()`
  - `fit_transform()`
    - Or add `TransformerMixin` as a base class instead of `fit_transform`

# Prepare the Data for ML Algorithms

---

## How to Create Custom Transformers

- Let's create custom transformer for
- Combining the attributes as we discussed earlier
  - rooms\_per\_household
  - population\_per\_household

# Prepare the Data for ML Algorithms

---

**Create Custom Transformers Class for Combining Attributes**

Please see **CombinedAttributesAdder** class in the notebook

# Prepare the Data for ML Algorithms

---

## Custom Transformers - Summary

- Now since we have a class for combining attributes
- We can easily add the class to Scikit-Learn pipeline
- And automate the addition of new attributes combinations

# Prepare the Data for ML Algorithms

---

## Feature Scaling

# Prepare the Data for ML Algorithms

---

## Feature Scaling

- Let's observe the housing dataset once more
  - The total number of rooms ranges from about 6 to 39,320
  - While median incomes only range from 0 to 15

**This is a problem**

# Prepare the Data for ML Algorithms

---

## Feature Scaling

- ML algorithms do not perform well
  - When the input numerical attributes
  - Have very different scales

# Prepare the Data for ML Algorithms

---

Solution?

# Prepare the Data for ML Algorithms

---

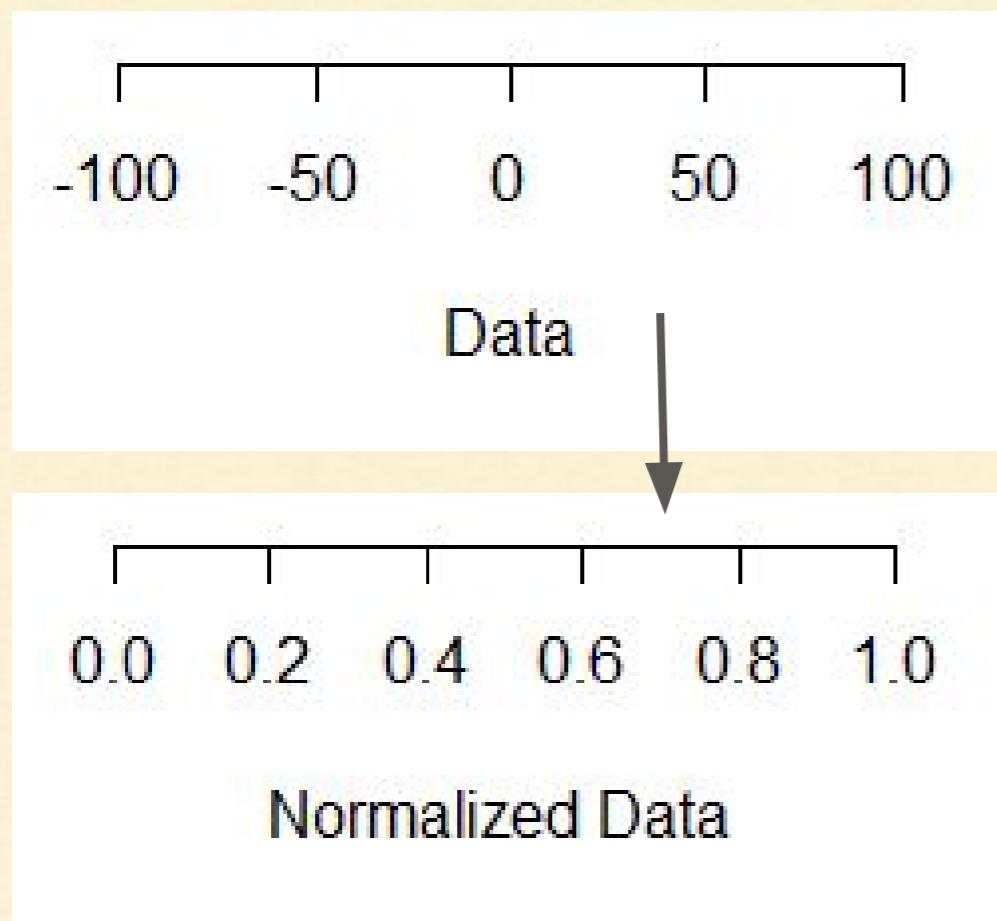
## Feature Scaling

- Feature Scaling is one of the most important
  - Transformation we need to apply to our data
- Two ways to make sure all attributes have same scale
  - **Min-max scaling**
  - **Standardization**

# Prepare the Data for ML Algorithms

## Feature Scaling - Min-max Scaling

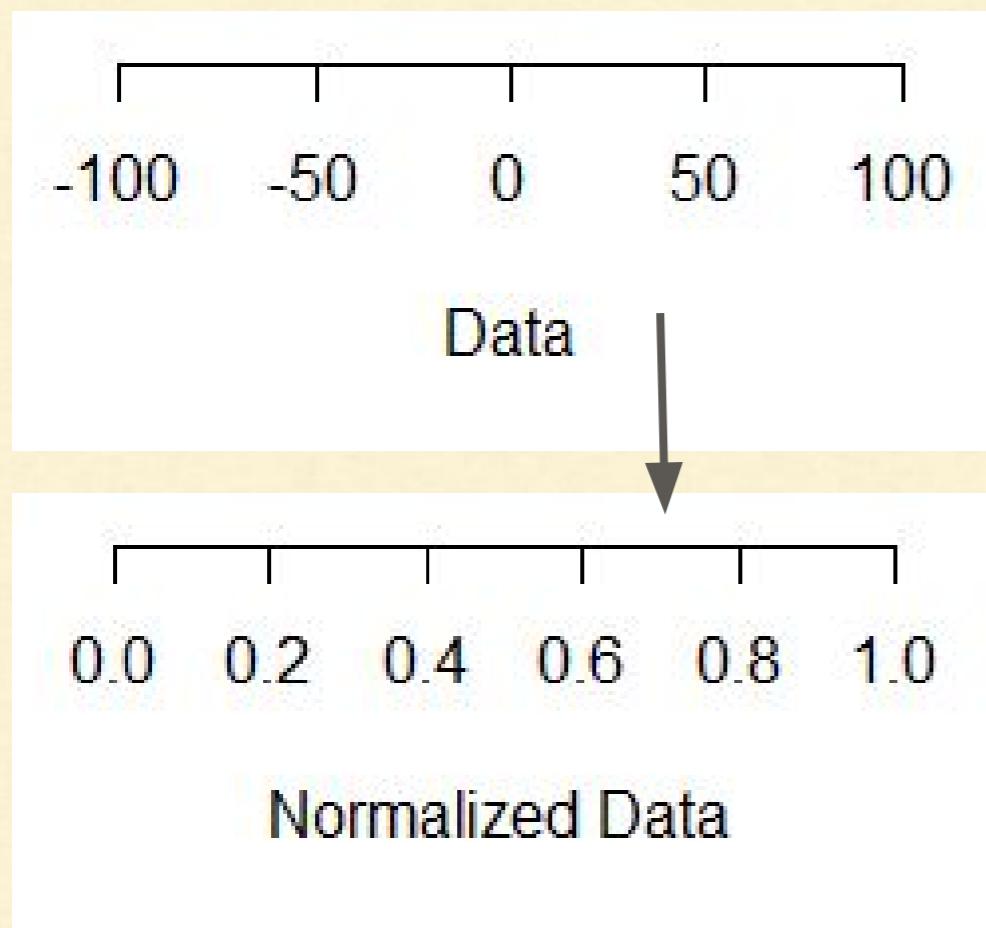
- Also known as Normalization
- Normalized values are in the range of  $[0, 1]$



# Prepare the Data for ML Algorithms

# Feature Scaling - Min-max Scaling

- Also known as Normalization
  - Normalized values are in the range of [0, 1]



$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Original Value

Normalized Value

# Prepare the Data for ML Algorithms

## Feature Scaling - Min-max Scaling - Example

```
# Creating DataFrame first
```

```
>>> import pandas as pd  
>>> s1 = pd.Series([1, 2, 3, 4, 5, 6], index=range(6))  
>>> s2 = pd.Series([10, 9, 8, 7, 6, 5], index=range(6))  
>>> df = pd.DataFrame(s1, columns=['s1'])  
>>> df['s2'] = s2  
>>> df
```

	s1	s2
0	1	10
1	2	9
2	3	8
3	4	7
4	5	6
5	6	5

# Prepare the Data for ML Algorithms

## Feature Scaling - Min-max Scaling - Example

```
# Use Scikit-Learn minmax_scaling
```

```
>>> from mlxtend.preprocessing import minmax_scaling
```

```
>>> minmax_scaling(df, columns=['s1', 's2'])
```

	s1	s2
0	1	10
1	2	9
2	3	8
3	4	7
4	5	6
5	6	5

Original

	s1	s2
0	0.0	1.0
1	0.2	0.8
2	0.4	0.6
3	0.6	0.4
4	0.8	0.2
5	1.0	0.0

Scaled (In range of 0 and 1)

# Prepare the Data for ML Algorithms

---

## Feature Scaling - Standardization

- In Machine Learning, we handle various types of data like
  - Audio signals and
  - Pixel values for image data
  - And this data can include multiple dimensions

# Prepare the Data for ML Algorithms

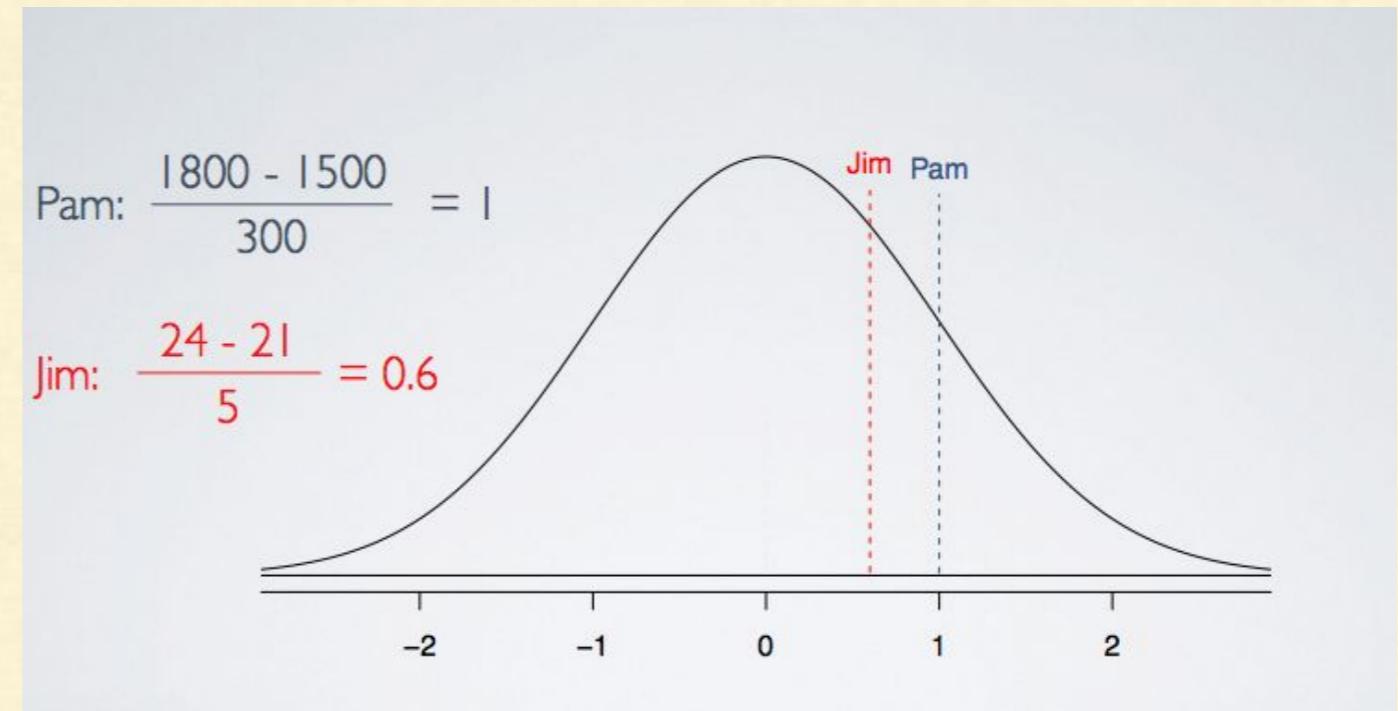
## Feature Scaling - Standardization

We scale the values by calculating

- How many standard deviation is the value away from the mean

SAT scores  $\sim N(\text{mean} = 1500, \text{SD} = 300)$

ACT scores  $\sim N(\text{mean} = 21, \text{SD} = 5)$



# Prepare the Data for ML Algorithms

## Feature Scaling - Standardization

- The general method of calculation
  - Calculate distribution mean and standard deviation for each feature
  - Subtract the mean from each feature
  - Divide the result from previous step of each feature by its standard deviation

$$x' = \frac{x - \bar{x}}{\sigma}$$

Standardized Value

# Prepare the Data for ML Algorithms

# Feature Scaling - Standardization

- In Standardization, features are rescaled
  - So that output will have the properties of
  - Standard normal distribution with
    - Zero mean and
    - Unit variance

$$\bar{x} = 0 \text{ and } \sigma = 1$$

# Prepare the Data for ML Algorithms

---

## Feature Scaling - Standardization

- Scikit-Learn provides
  - **StandardScaler** class for standardization

# Prepare the Data for ML Algorithms

---

## Feature Scaling - Which One to Use?

- Min-max scales in the range of [0,1]
- Standardization does not bound values to a specific range
  - It may be problem for some algorithms
  - Example- Neural networks expect an input value ranging from 0 to 1
- We'll learn more use cases as we proceed in the course

# Prepare the Data for ML Algorithms

---

## Transformation Pipelines

# Prepare the Data for ML Algorithms

---

## Transformation Pipelines

- As we can see that
  - We need many transformation steps to be executed in right order
  - We'll use Scikit-Learn **Pipeline** class
  - For specifying sequence of transformations

# Prepare the Data for ML Algorithms

---

## Transformation Pipelines - Pipeline for Numerical Attributes

- Let's build a pipeline for the numerical attributes
- Steps we have done so far
  - Handle missing values - **Imputer** class with strategy as median
  - Custom transformer for combining attributes -  
**CombinedAttributesAdder**
  - Feature Scaling - **StandardScaler**

# Prepare the Data for ML Algorithms

## Transformation Pipelines - Pipeline for Numerical Attributes

- Let's build a pipeline for the numerical attributes

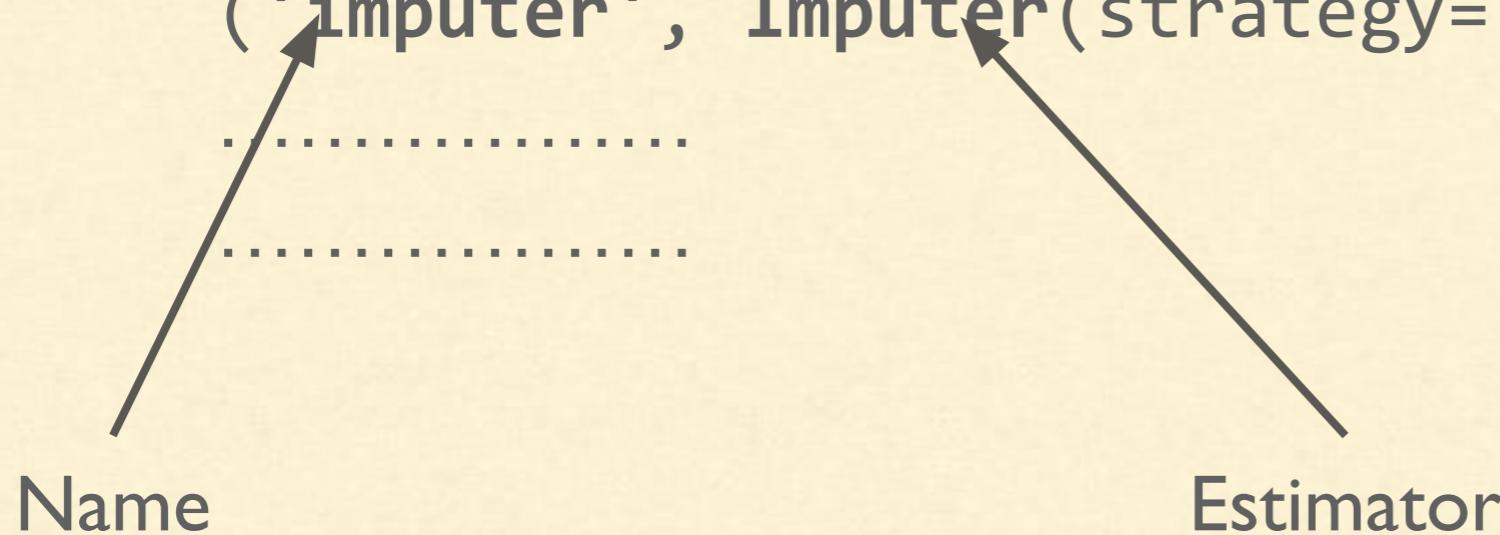
```
>>> from sklearn.pipeline import Pipeline  
>>> from sklearn.preprocessing import StandardScaler  
>>> num_pipeline = Pipeline([  
    ('imputer', Imputer(strategy="median")),  
    ('atribbs_adder', CombinedAttributesAdder()),  
    ('std_scaler', StandardScaler()),  
)  
>>> housing_num_tr = num_pipeline.fit_transform(housing_num)
```

# Prepare the Data for ML Algorithms

## Transformation Pipelines - Pipeline for Numerical Attributes

- The Pipeline constructor takes a list of name/estimator pairs defining a sequence of steps.
- All but the last estimator must be transformers

```
num_pipeline = Pipeline([  
    ('imputer', Imputer(strategy="median"))],
```



# Prepare the Data for ML Algorithms

---

## Transformation Pipelines

- When we call the pipeline's `fit()` method
  - It calls `fit_transform()` sequentially on all transformers
  - Passing the output of each call as the parameter to the next transformer
  - Until it reaches the final estimator
  - For the final estimator it just calls the `fit()` method

# Prepare the Data for ML Algorithms

---

## Transformation Pipelines - DataFrameSelector

- Scikit-Learn doesn't handle DataFrames
- How to select columns from pipeline?
- Let's create a class **DataFrameSelector**
  - To select numerical or categorical columns

# Prepare the Data for ML Algorithms

## Transformation Pipelines - DataFrameSelector

```
class DataFrameSelector(BaseEstimator, TransformerMixin):  
    def __init__(self, attribute_names):  
        self.attribute_names = attribute_names  
    def fit(self, X, y=None):  
        return self  
    def transform(self, X):  
        return X[self.attribute_names].values
```

# Prepare the Data for ML Algorithms

## Transformation Pipelines - Pipeline for Numerical Attributes

- So final pipeline for Numerical attributes looks like

```
>>> num_attribs = list(housing_num)

>>> num_pipeline = Pipeline([
    ('selector', DataFrameSelector(num_attribs)),
    ('imputer', Imputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

```

# Prepare the Data for ML Algorithms

## Transformation Pipelines - Pipeline for Categorical Attributes

- So final pipeline for categorical attributes looks like

```
>>> cat_attribs = ["ocean_proximity"]  
>>> cat_pipeline = Pipeline([  
    ('selector', DataFrameSelector(cat_attribs)),  
    ('cat_encoder', CategoricalEncoder(encoding="onehot-dense")),  
])
```

# Prepare the Data for ML Algorithms

## Transformation Pipelines - Full Pipeline

- Let's Combine the num\_pipeline and cat\_pipeline using Scikit-Learn **FeatureUnion** class

```
>>> from sklearn.pipeline import FeatureUnion  
>>> full_pipeline = FeatureUnion(transformer_list=[  
        ("num_pipeline", num_pipeline),  
        ("cat_pipeline", cat_pipeline),  
    ])
```

# Prepare the Data for ML Algorithms

---

## Transformation Pipelines - Full Pipeline

- Each sub pipeline starts with a selector transformer
- It simply transforms the data by selecting the desired attributes
- While drops the rest
- And converts the resulting DataFrame to a NumPy array

# Prepare the Data for ML Algorithms

## Transformation Pipelines - Run the Whole Pipeline

- Run the whole pipeline

```
>>> housing_prepared = full_pipeline.fit_transform(housing)  
>>> housing_prepared
```

```
array([[-1.15604281,  0.77194962,  0.74333089, ...,  0.        ,  
       0.        ,  0.        ],  
      [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.        ,  
       0.        ,  0.        ],  
      [ 1.18684903, -1.34218285,  0.18664186, ...,  0.        ,  
       0.        ,  1.        ],  
      ...,  
      [ 1.58648943, -0.72478134, -1.56295222, ...,  0.        ,  
       0.        ,  0.        ],  
      [ 0.78221312, -0.85106801,  0.18664186, ...,  0.        ,  
       0.        ,  0.        ],  
      [-1.43579109,  0.99645926,  1.85670895, ...,  0.        ,  
       1.        ,  0.        ]])
```

# Checklist for Machine Learning Projects

---

1. Frame the problem and look at the big picture
2. Get the data
3. Explore the data to gain insights
4. Prepare the data for Machine Learning algorithms
- 5. Explore many different models and short-list the best ones**
6. Fine-tune model
7. Present the solution
8. Launch, monitor, and maintain the system

# Train Models and Short-list Best Ones

---

Select and Train a Model

# Train Models and Short-list Best Ones

---

## Train a Model - Steps Done So Far

- Framed the problem
- Got data and explored it
- Created training and test set
- Transformation pipelines to
  - Clean up
  - Prepare data
- Now we are ready to train the model

# Train Models and Short-list Best Ones

---

## Train a Model - Linear Regression

- The goal of this step is to
  - Train few(two to five models) and
  - Select the best one
- Let's understand overfitting and underfitting before training a model

# Train Models and Short-list Best Ones

---

## What is Overfitting?

- Say if you are visiting a foreign country
- And the taxi driver rips you off
- You might be tempted to say that
  - All taxi drivers in that country are thieves
- Here we are overgeneralizing

# Train Models and Short-list Best Ones

---

## What is Overfitting?

- Machines can also fall into this trap of overgeneralization like humans
- This is called **overfitting**
- In overfitting
  - The model performs well on the training data
  - But does not generalize well on unknown data

# Train Models and Short-list Best Ones

---

## When does overfitting occur?

- Overfitting occurs when a machine learning algorithm captures the noise of the data.
- Intuitively, overfitting occurs when the model or the algorithm fits the data too well.
- Overfitting is often a result of an excessively complicated model,

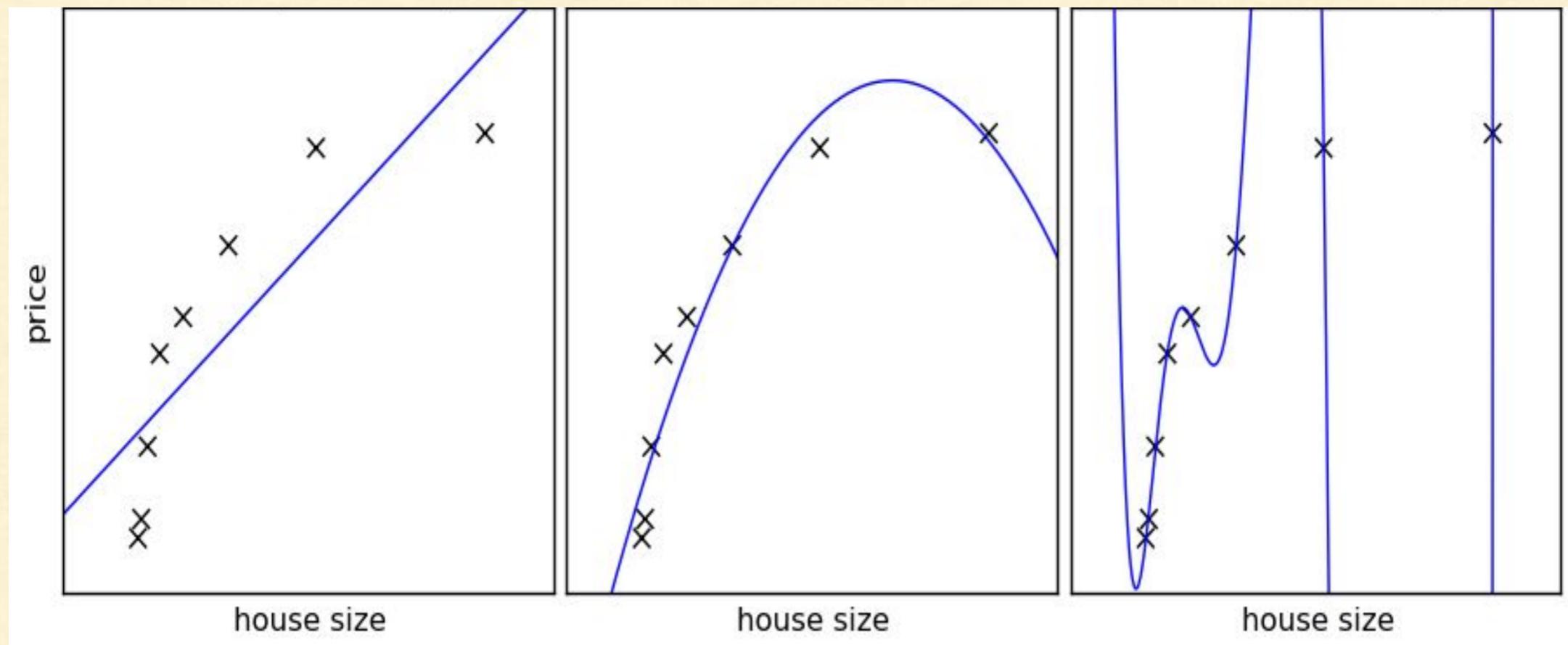
# Train Models and Short-list Best Ones

---

Let us consider an example of overfitting

# Train Models and Short-list Best Ones

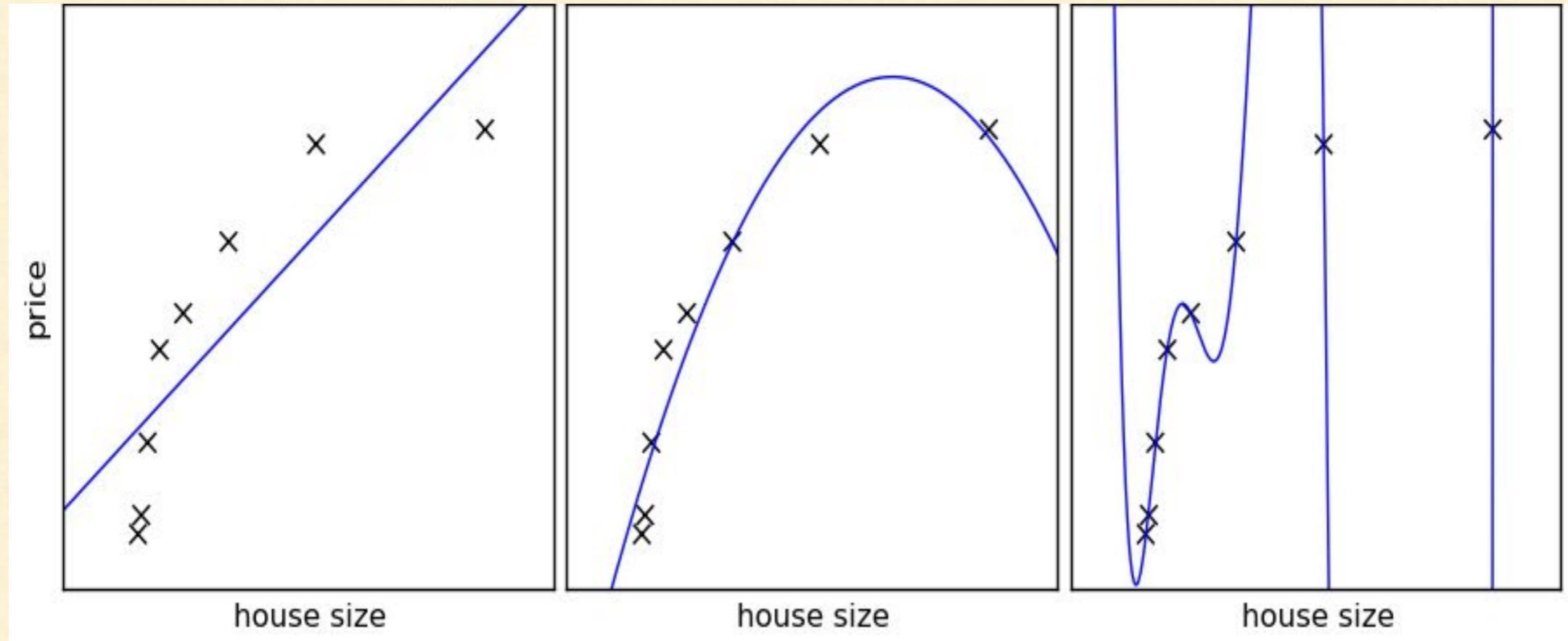
## Overfitting Example



Here a high-degree polynomial strongly overfits the housing prices data. It performs much better on the training data than the simple linear model.

# Train Models and Short-list Best Ones

## Overfitting Example



**But** if the training set is noisy, then the model is likely to detect patterns in the noise itself and these patterns will not generalize to new instances.

# Train Models and Short-list Best Ones

---

## Understanding Overfitting

A complex model may detect patterns like the fact that all countries in the training data with a W in their name have a life satisfaction greater than 7:

- New Zealand (7.3)
- Norway (7.4)
- Sweden (7.2)
- Switzerland (7.5)

# Train Models and Short-list Best Ones

---

## Understanding Overfitting

If our model is trained on such countries than it will not generalize well for the following countries.

How confident are you that the W-satisfaction rule generalizes to

- Rwanda ???
- Zimbabwe ???

# Train Models and Short-list Best Ones

---

## Regularization - Tackling overfitting

Regularization can be one way in which we can tackle the problem of overfitting. In overfitting we -

- Put constrain the model to make it simpler
- The amount of regularization to apply during learning can be controlled by a hyperparameter

# Train Models and Short-list Best Ones

---

## Regularization - Tackling overfitting

Considering a linear model. It has two parameters,  $\theta_0$  and  $\theta_1$ . It gives the learning algorithm two degrees of freedom to adapt the model to the training data:

- It can tweak the height ( $\theta_0$ )
- Or the slope ( $\theta_1$ ) of the line

# Train Models and Short-list Best Ones

---

## Regularization - Tackling overfitting

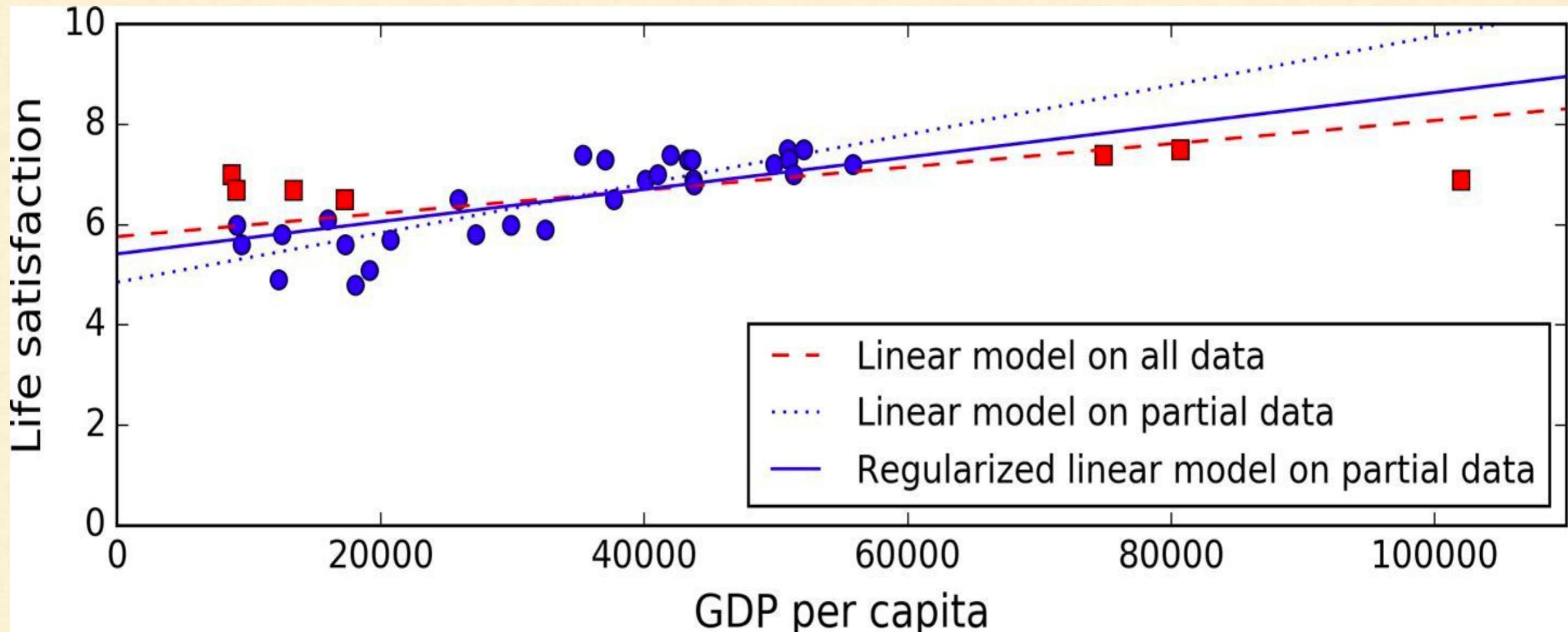
If we forced  $\theta_1 = 0$ , the algorithm would have only one degree of freedom

If we allow the algorithm to modify  $\theta_1$  then :

- It will produce a simpler model than with two degrees of freedom
- But it will be more complex than with just one degree of freedom

# Train Models and Short-list Best Ones

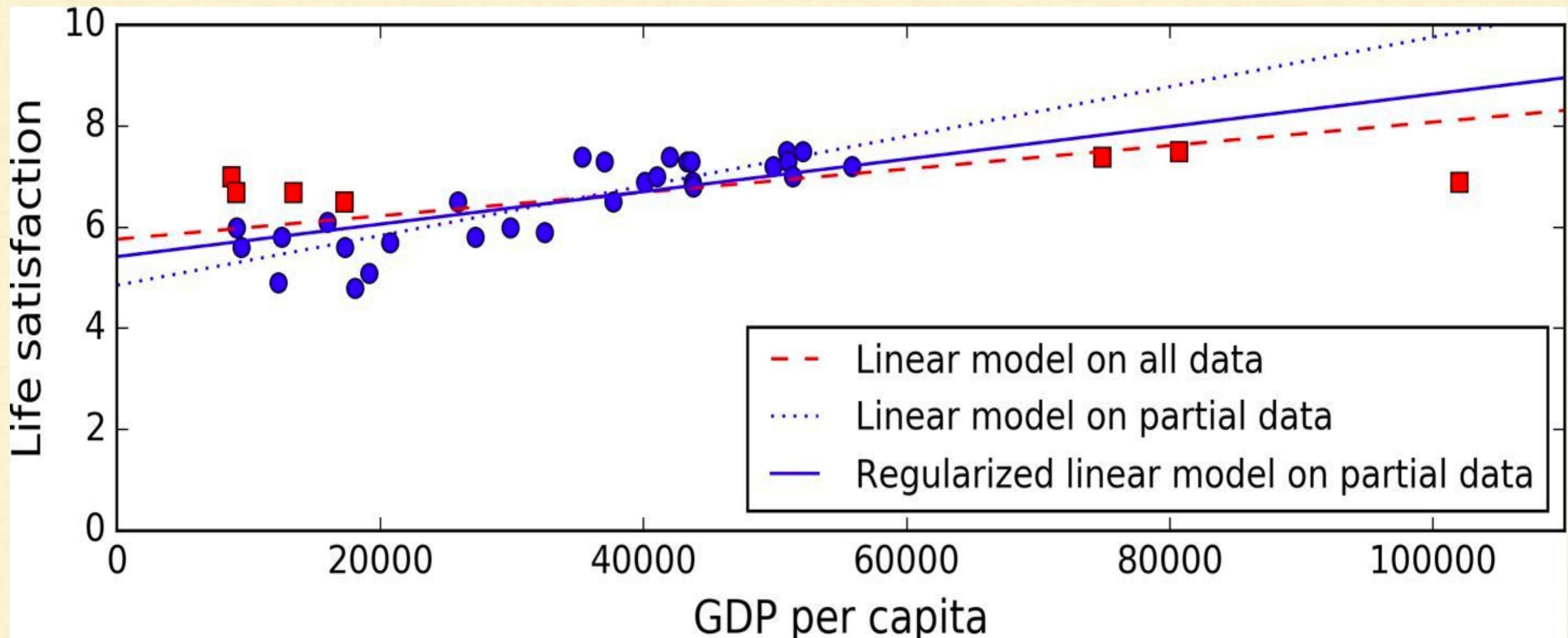
## Regularization - Tackling overfitting



- The dotted line represents the original model that was trained with a few countries missing.

# Train Models and Short-list Best Ones

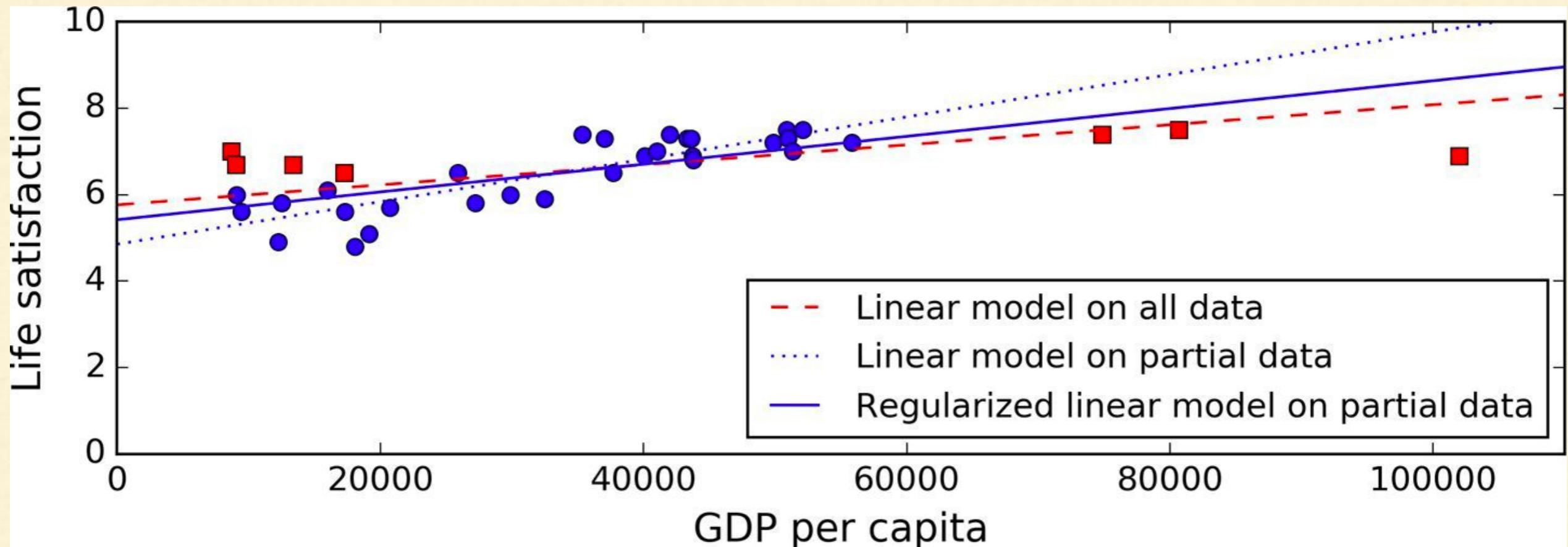
## Regularization - Tackling overfitting



- The dashed line is our second model trained with all countries

# Train Models and Short-list Best Ones

## Regularization - Tackling overfitting



- The solid line is a linear model trained with the same data as the first model but with a regularization constraint.

# Train Models and Short-list Best Ones

---

## Regularization - Tackling overfitting

Observations from the above example:

- Regularization forced the model to have a smaller slope
- Fits a bit less on the training data that the model was trained on
- But actually allows it to generalize better to new examples.

# Train Models and Short-list Best Ones

---

## Regularization - Tackling overfitting

How do we choose the best model ???

When finding the best model, we'll have to find the right balance between

- Fitting the data perfectly
- And keeping the model simple enough to ensure that it will generalize well

# Train Models and Short-list Best Ones

---

## What is Underfitting?

It occurs when your model is too simple to learn the underlying structure of the data. Underfitting is often a result of an excessively simple model.

For example, a linear model of life satisfaction is prone to underfit because reality is just more complex than the model, so its predictions are bound to be inaccurate, even on the training examples.

# Train Models and Short-list Best Ones

---

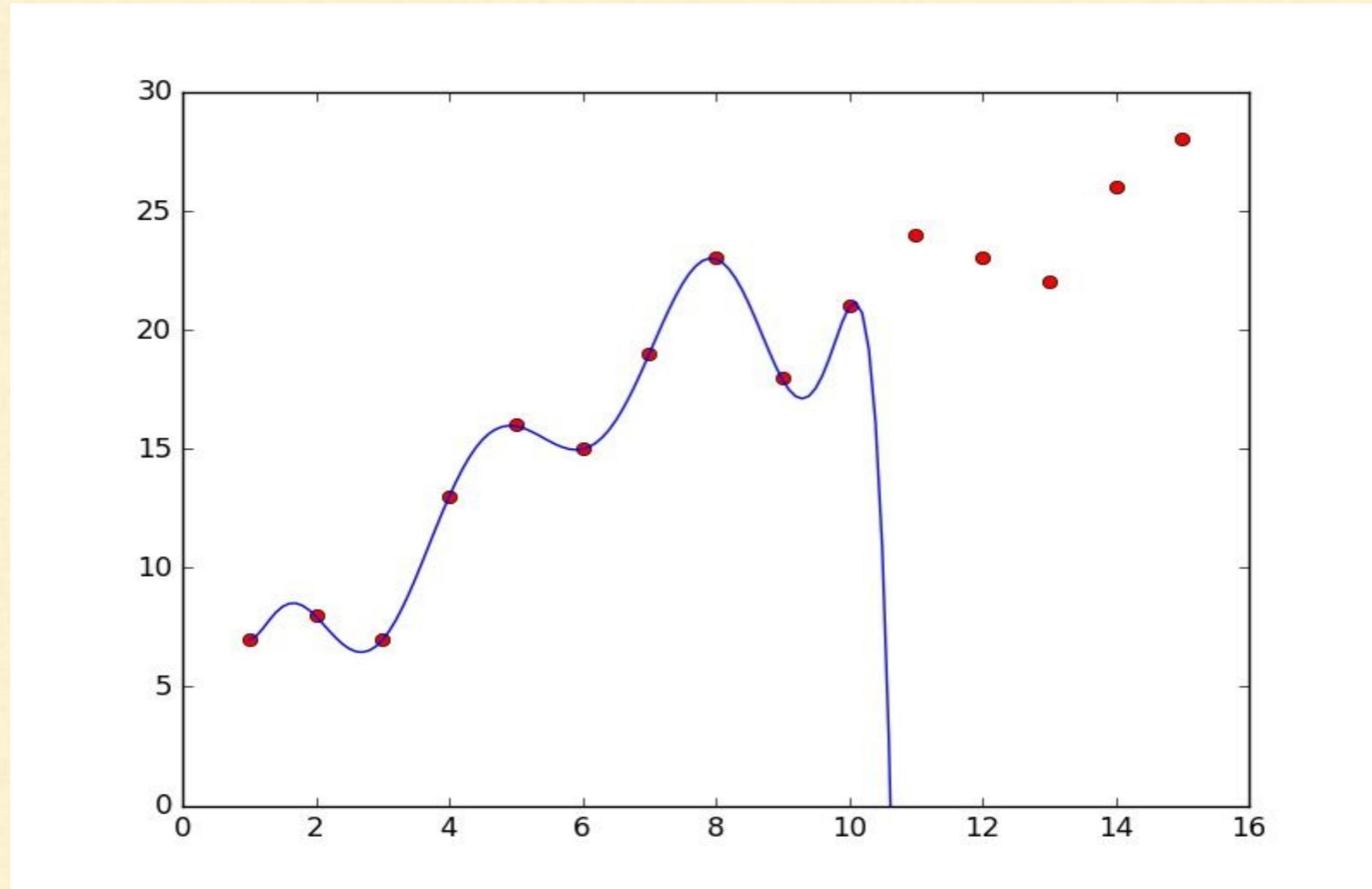
## When does Underfitting occur ?

It happens when :

- Features do not provide enough information to make good predictions
- When the model or the algorithm does not fit the data well enough
- Model is too simple

# Train Models and Short-list Best Ones

## Example of Underfitting



In the above plot the model doesn't fit perfectly on the training data, hence it is a case of underfitting.

# Train Models and Short-list Best Ones

---

## How to solve the problem of Underfitting?

- Select a more powerful model
- Feed better features to training algorithm
- Reduce the constraints on the model (e.g., reduce the regularization hyperparameter)

# Train Models and Short-list Best Ones

---

## Train a Model - Linear Regression

Let's train the model now

# Train Models and Short-list Best Ones

## Train a Model - Linear Regression

```
>>> from sklearn.linear_model import LinearRegression  
>>> lin_reg = LinearRegression()  
>>> lin_reg.fit(housing_prepared, housing_labels)
```



# Train Models and Short-list Best Ones

## Train a Model - Linear Regression

```
>>> from sklearn.linear_model import LinearRegression  
>>> lin_reg = LinearRegression()  
>>> lin_reg.fit(housing_prepared, housing_labels)
```



**Congrats! We've trained the first model :)**

# Train Models and Short-list Best Ones

## Train a Model - Linear Regression

- Let's try the full pipeline on a few training instances

```
>>> some_data = housing.iloc[:5]
>>> some_labels = housing_labels.iloc[:5]
>>> some_data_prepared = full_pipeline.transform(some_data)

>>> print("Predictions:", lin_reg.predict(some_data_prepared))
>>> print("Actual values", list(some_labels))
```

# Train Models and Short-list Best Ones

## Train a Model - Linear Regression - Results

Actual	286,600.0	340,600.0	196,900.0	46,300.0	254,500.0
Predicted	210,644.6	317,768.8	210,956.4	59,218.9	189,747.5
% Error	-26.5	-6.7	7.13	27.90	-25.44

# Train Models and Short-list Best Ones

---

## Train a Model - Linear Regression - Observations

- The predictions are not exactly accurate
- The fourth prediction is off by more than 27%!

# Train Models and Short-list Best Ones

---

## Train a Model - Linear Regression - RMSE

- Let's find the regression model's RMSE on the whole training set
- Using Scikit-Learn's `mean_squared_error` function

# Train Models and Short-list Best Ones

---

## Train a Model - Linear Regression - RMSE

```
>>> from sklearn.metrics import mean_squared_error  
>>> housing_predictions = lin_reg.predict(housing_prepared)  
>>> lin_mse = mean_squared_error(housing_labels,  
housing_predictions)  
>>> lin_rmse = np.sqrt(lin_mse)  
>>> lin_rmse  
>>> 68628.413493824875
```

# Train Models and Short-list Best Ones

## Train a Model - Underfitting the Training Data

```
>>> lin_rmse  
>>> 68628.413493824875
```

- This is clearly not a great RMSE score
  - Most districts' median\_housing\_values range
  - Between \$120,000 and \$265,000
  - So a typical prediction error of \$68,628 is not very satisfying
- This is example of
  - Model underfitting the training data

# Train Models and Short-list Best Ones

---

## Train a Model - Underfitting the Training Data

Solutions of Underfitting?

- **Select a more powerful model**
- Feed better features to training algorithm

**Let's train a DecisionTreeRegressor - Powerful model**

# Train Models and Short-list Best Ones

## Train a Model - DecisionTreeRegressor

- Capable of finding complex nonlinear relationships in the data
- We will cover Decision Trees in details later in the course

# Train Models and Short-list Best Ones

---

## Train a Model - DecisionTreeRegressor

```
>>> from sklearn.tree import DecisionTreeRegressor  
>>> tree_reg = DecisionTreeRegressor()  
>>> tree_reg.fit(housing_prepared, housing_labels)
```

# Train Models and Short-list Best Ones

## Train a Model - DecisionTreeRegressor

### Evaluate on the training set

```
>>> housing_predictions = tree_reg.predict(housing_prepared)  
>>> tree_mse = mean_squared_error(housing_labels,  
housing_predictions)  
>>> tree_rmse = np.sqrt(tree_mse)  
>>> tree_rmse
```

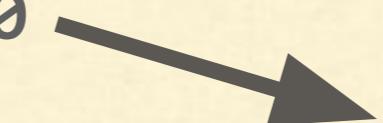
Output - 0.0

# Train Models and Short-list Best Ones

## Train a Model - DecisionTreeRegressor

### Evaluate on the training set

```
>>> housing_predictions = tree_reg.predict(housing_prepared)  
>>> tree_mse = mean_squared_error(housing_labels,  
housing_predictions)  
>>> tree_rmse = np.sqrt(tree_mse)  
>>> tree_rmse
```

Output - 0.0   
**Really???**

# Train Models and Short-list Best Ones

---

## Train a Model - Overfitting the Training Data

- Could this model really be absolutely perfect?
- This is called **overfitting the training data**
- Let's evaluate the previous DecisionTreeRegressor model
  - Using Cross-Validation

# Train Models and Short-list Best Ones

---

Better Evaluation Using Cross-Validation

# Train Models and Short-list Best Ones

---

- We should not touch the test set until
  - We are confident about the model on training set

# Train Models and Short-list Best Ones

---

- We should not touch the test set until
  - We are confident about the model on training set

Then how should we validate the model?

# Train Models and Short-list Best Ones

---

- We should not touch the test set until
  - We are confident about the model on training set

Then how should we validate the model?

Answer - Using Cross-Validation

# Train Models and Short-list Best Ones

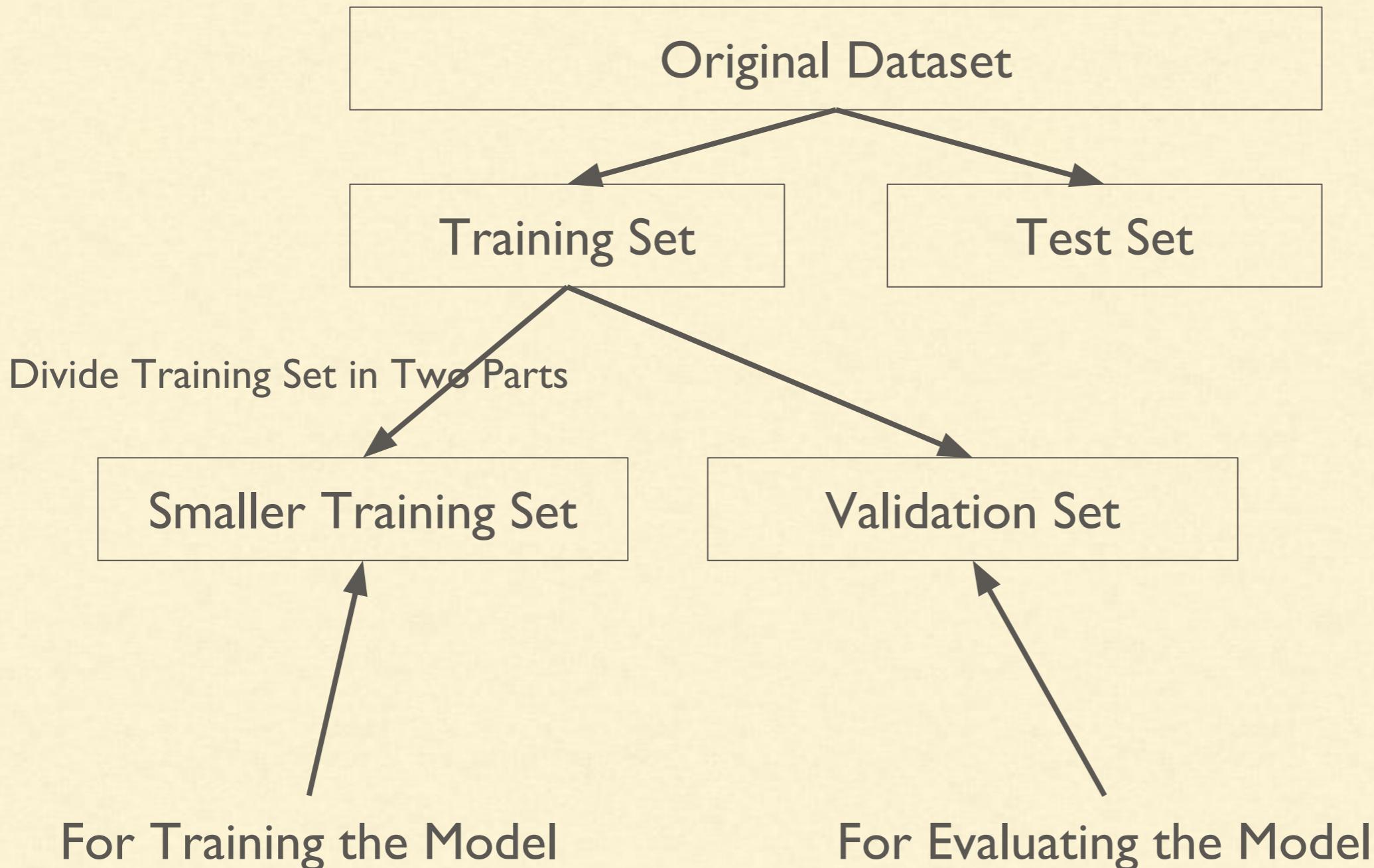
---

## Cross-Validation

- In cross-validation we use
  - Part of the training set for training
  - And part for model validation

# Train Models and Short-list Best Ones

## Cross-Validation



# Performance measure - Cross Validation



# Train Models and Short-list Best Ones

---

## Cross-Validation

- Use the Scikit-Learn's **train\_test\_split** function to split
  - Training set into a smaller
    - Training set and
    - Validation set
  - Train models against the
    - Smaller training set and
    - Evaluate them against validation set

# Train Models and Short-list Best Ones

---

## Cross-Validation

- We can also use Scikit-Learn's **cross-validation** feature

# Train Models and Short-list Best Ones

## Cross-Validation - K-fold - DecisionTreeRegressor

- We can also use Scikit-Learn's **cross-validation** feature

```
>>> from sklearn.model_selection import cross_val_score  
>>> scores = cross_val_score(tree_reg, housing_prepared,  
housing_labels, scoring="neg_mean_squared_error", cv=10)  
>>> tree_rmse_scores = np.sqrt(-scores)
```

K-fold Cross-Validation

10 folds

# Train Models and Short-list Best Ones

---

## Cross-Validation - K-fold

- Previous code performs **K-fold cross-validation**
  - It randomly splits the training set into
    - 10 distinct subsets called folds
  - Then it trains and evaluates the Decision Tree model 10 times
    - Picking a different fold for evaluation every time
    - And training on the other 9 folds
  - The result is an array containing the 10 evaluation scores

# Train Models and Short-list Best Ones

---

## Cross-Validation - K-fold - DecisionTreeRegressor - Output

```
>>> def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
>>> display_scores(tree_rmse_scores)

Scores: [ 70232.0136482   66828.46839892   72444.08721003
 70761.50186201  71125.52697653   75581.29319857   70169.59286164
 70055.37863456  75370.49116773   71222.39081244]

Mean: 71379.0744771

Standard deviation: 2458.31882043
```

# Train Models and Short-list Best Ones

## Cross-Validation - K-fold - Linear Regression

- Now compute the same score for Linear Regression

```
>>> lin_scores = cross_val_score(lin_reg, housing_prepared,  
housing_labels, scoring="neg_mean_squared_error", cv=10)  
>>> lin_rmse_scores = np.sqrt(-lin_scores)  
>>> display_scores(lin_rmse_scores)
```

**Scores:** [ 66782.73843989 66960.118071 70347.95244419  
74739.57052552 68031.13388938 71193.84183426 64969.63056405  
68281.61137997 71552.91566558 67665.10082067]  
**Mean:** 69052.4613635  
**Standard deviation:** 2731.6740018

# Train Models and Short-list Best Ones

## Cross-Validation - K-fold - Linear Regression Vs Decision Tree

### Linear Regression

Mean:

69, 052.4613635

Standard deviation:

2, 731.6740018

### DecisionTreeRegressor

Mean:

71, 379.0744771

Standard deviation:

2, 458.31882043

# Train Models and Short-list Best Ones

## Cross-Validation - K-fold - Linear Regression Vs Decision Tree

### Linear Regression

Mean:

69, 052.4613635

Standard deviation:

2, 731.6740018

### DecisionTreeRegressor

Mean:

71, 379.0744771

Standard deviation:

2, 458.31882043

Decision Tree model performed worse than the Linear Regression model.  
Decision Tree model is overfitting when RMSE came to 0.0

# Train Models and Short-list Best Ones

---

## Cross-Validation - K-fold - Important points

- Cross-validation gives
  - Estimate of the performance of your model and
  - Also a measure of how precise this estimate
- The Decision Tree has a score of
  - Approximately 71, 379
  - With precision of  $\pm 2, 458$  (Standard Deviation)

# Train Models and Short-list Best Ones

---

## **RandomForestRegressor**

- Let's train our last model
  - **RandomForestRegressor**
- Random Forests work by training many Decision Trees
  - On random subsets of the features
  - Then averaging out their predictions
- We will cover Random Forests in details later in the course

# Train Models and Short-list Best Ones

## RandomForestRegressor - RMSE

- Let's train our last model using **Random Forests**

```
>>> from sklearn.ensemble import RandomForestRegressor  
>>> forest_reg = RandomForestRegressor(random_state=42)  
>>> forest_reg.fit(housing_prepared, housing_labels)  
>>> housing_predictions = forest_reg.predict(housing_prepared)  
>>> forest_mse = mean_squared_error(housing_labels,  
housing_predictions)  
>>> forest_rmse = np.sqrt(forest_mse)
```

Output - 21941.911027380233

# Train Models and Short-list Best Ones

## RandomForestRegressor - Using Cross-Validation

- Let's train **Random Forests** using cross-validation

```
>>> from sklearn.model_selection import cross_val_score  
>>> forest_scores = cross_val_score(forest_reg, housing_prepared,  
housing_labels, scoring="neg_mean_squared_error", cv=10)  
>>> forest_rmse_scores = np.sqrt(-forest_scores)
```

# Train Models and Short-list Best Ones

## RandomForestRegressor - Using Cross-Validation

- Let's train **Random Forests** using cross-validation

```
>> display_scores(forest_rmse_scores)
```

**Scores:** [ 51650.94405471 48920.80645498 52979.16096752  
 54412.74042021 50861.29381163 56488.55699727 51866.90120786  
 49752.24599537 55399.50713191 53309.74548294 ]

**Mean:** 52564.1902524

**Standard deviation:** 2301.87380392

# Train Models and Short-list Best Ones

## Training Models Comparison

**Linear  
Regression**  
Mean - 69, 052  
SD - 2, 731

**Decision  
Tree**  
Mean - 71, 379  
SD - 2, 458

**Random  
Forest**  
Mean - 52, 564  
SD - 2, 301

SD - Standard Deviation

Random Forests perform lot better

# Train Models and Short-list Best Ones

---

## Explore More Models

- Till now we have explored few models
- Now the goal is to shortlist
  - A few (two to five) promising models

# Checklist for Machine Learning Projects

---

1. Frame the problem and look at the big picture
2. Get the data
3. Explore the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Explore many different models and short-list the best ones
6. **Fine-tune model**
7. Present the solution
8. Launch, monitor, and maintain the system

# Fine-tune Model

---

Fine-Tune Your Model

# Fine-tune Model

---

## Fine-Tune Model

- This section is to help you understand various fine-tuning methods
- It's okay
  - If you do not understand the code, terms and concepts at this moment
  - We will cover the concepts and code in details as we progress in the course

# Fine-tune Model

---

## Fine-Tune Model

- Now we have two to five promising models
- Let's fine-tune them

# Fine-tune Model

---

## Fine-Tune Model

- Now we have two to five promising models
- Let's fine-tune them

How do we fine-tune?

# Fine-tune Model

---

## Fine-Tune Model - Hyperparameters

- Before learning about how to fine-tune the models
- Let's learn about Hyperparameters first

# Fine-tune Model

---

## What are Hyperparameters?

- A machine learning model is a mathematical formula
  - With a number of parameters that need to be learned from the data
- The soul of machine learning is
  - Fitting a model to the data
- By training a model with existing data
  - We fit the model parameters

# Fine-tune Model

---

## What are Hyperparameters?

- There is another kind of parameters
  - That cannot be directly learned from the model training process
- These parameters express
  - Higher-level properties of the model such as
  - Its complexity or
  - How fast it should learn

# Fine-tune Model

---

## What are Hyperparameters?

- These parameters are called **Hyperparameters**
- Hyperparameters are usually decided before the actual training begins
- Can be decided by
  - Setting different values
  - Training different models and
  - Choosing the values that work best

# Fine-tune Model

---

## Some examples of Hyperparameters?

- Number of leaves or depth of a tree
- Learning rate
  - How fast a model should learn
- Number of hidden layers in a deep neural network
- Number of clusters in a k-means clustering

# Fine-tune Model

---

## Fine-Tune Model - Solutions

- Now let's learn how to fine-tune models

# Fine-tune Model

---

## Fine-Tune Model - Solutions

- One solution is to fiddle with the hyperparameters manually
- Until we find a great combination of hyperparameter values
- This is very tedious work
- And we may not have time to explore many combinations

# Fine-tune Model

---

## Grid Search

# Fine-tune Model

---

## Fine-Tune Model - Grid Search

- Evaluates all possible combinations of hyperparameters values
  - Using cross-validation
- All we need to tell
  - Which hyperparameters we want it to experiment with and
  - What values to try out
- We will cover Grid Search in details later in the course

# Fine-tune Model

---

## Fine-Tune Model - Grid Search - Example

- Following code searches for the best possible combination of hyperparameter values for **RandomForestRegressor**

# Fine-tune Model

## Fine-Tune Model - Grid Search - Example

```
>>> from sklearn.model_selection import GridSearchCV  
>>> param_grid = [  
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},  
    {'bootstrap': [False], 'n_estimators': [3, 10],  
     'max_features':[2, 3, 4]}  
,  
]  
>>> forest_reg = RandomForestRegressor()  
>>> grid_search = GridSearchCV(forest_reg, param_grid, cv=5,  
scoring='neg_mean_squared_error')  
>>> grid_search.fit(housing_prepared, housing_labels)
```

**Run it in Notebook**

# Fine-tune Model

---

## Fine-Tune Model - Grid Search - Example

- Get the best combination of parameters (We will cover this later in the course)

```
>>> grid_search.best_params_
```

Output-

```
{'max_features': 8, 'n_estimators': 30}
```

# Fine-tune Model

---

## Fine-Tune Model - Grid Search - Example

- Get the best estimator (We will cover this later in the course)

```
>>> grid_search.best_estimator_
```

Output-

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features=8, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=30,
                      n_jobs=1, oob_score=False, random_state=42,
                      verbose=0, warm_start=False
)
```

# Fine-tune Model

---

## Fine-Tune Model - Grid Search - Example

- Get the score of each hyperparameter combination tested during the grid search (We will cover this later in the course)

```
>>> cvres = grid_search.cv_results_
>>> for mean_score, params in zip(cvres["mean_test_score"],
cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

# Fine-tune Model

---

Randomized Search

# Fine-tune Model

---

## Fine-Tune Model - Randomized Search

- The grid search approach is fine
  - When you are exploring relatively few combinations
- When the hyperparameter search space is large, use

### **RandomizedSearchCV**

- We will cover Randomized Search in details later in the course

# Fine-tune Model

---

## Fine-Tune Model - Randomized Search

- Randomized search
  - Instead of trying out all possible combinations
  - Evaluates a given number of random combinations
  - By selecting a random value for each hyperparameter
  - At every iteration

# Fine-tune Model

---

## Fine-Tune Model - Randomized Search

- Has two main benefits
- One
  - Say our randomized search runs for 1000 iterations
  - It will explore 1,000 different values for each hyperparameter
  - While Grid Search explores few values per hyperparameter

# Fine-tune Model

---

## Fine-Tune Model - Randomized Search

- Has two main benefits
- Two
  - We have more control over the computing budget we want to allocate to hyperparameter search
  - Simply by setting the number of iterations

# Fine-tune Model

---

## Fine-Tune Model - Randomized Search

Check the RandomizedSearchCV code in Notebook

# Fine-tune Model

---

## Ensemble Methods

# Fine-tune Model

---

## Fine-Tune Model - Ensemble Methods

- Another way of fine-tuning models is to
  - Combine best performing models
- The ensemble(group)
  - Oftens perform better than the
  - Best individual model
- Just like in previous example
  - Random forests performed better than the
  - Individual decision trees

# Fine-tune Model

---

Analyze the Best Models and Their Errors

# Fine-tune Model

---

## Analyze the Best Models

- Let's see the importance score of each attribute

```
>>> feature_importances =  
grid_search.best_estimator_.feature_importances_  
>>> extra_attribs = ["rooms_per_hhold", "pop_per_hhold",  
"bedrooms_per_room"]  
>>> cat_encoder = cat_pipeline.named_steps["cat_encoder"]  
>>> cat_one_hot_attribs = list(cat_encoder.categories_[0])  
>>> attributes = num_attribs + extra_attribs + cat_one_hot_attribs  
>>> sorted(zip(feature_importances, attributes), reverse=True)
```

# Fine-tune Model

## Analyze the Best Models

### Observations??

```
[ (0.36615898061813418, 'median_income'),
  (0.16478099356159051, 'INLAND'),
  (0.10879295677551573, 'pop_per_hhold'),
  (0.073344235516012421, 'longitude'),
  (0.062909070482620302, 'latitude'),
  (0.056419179181954007, 'rooms_per_hhold'),
  (0.053351077347675809, 'bedrooms_per_room'),
  (0.041143798478729635, 'housing_median_age'),
  (0.014874280890402767, 'population'),
  (0.014672685420543237, 'total_rooms'),
  (0.014257599323407807, 'households'),
  (0.014106483453584102, 'total_bedrooms'),
  (0.010311488326303787, '<1H OCEAN'),
  (0.0028564746373201579, 'NEAR OCEAN'),
  (0.0019604155994780701, 'NEAR BAY'),
  (6.0280386727365991e-05, 'ISLAND') ]
```

# Fine-tune Model

---

Evaluate model on the Test Set

# Fine-tune Model

---

## Evaluate Model on Test Set

- After tweaking the models for a while
- When we are confident about the model
- Then evaluate the model on the test set

# Fine-tune Model

---

## Evaluate Model on Test Set - Process

### I. Get the final model

# Fine-tune Model

---

## Evaluate Model on Test Set - Process

### I. Get the final model

```
>>> final_model = grid_search.best_estimator_
```

# Fine-tune Model

---

## Evaluate Model on Test Set - Process

### 2. Get predictors and labels from test set

```
# Predictors  
  
">>>> X_test = strat_test_set.drop("median_house_value",  
axis=1)  
  
# Labels  
  
>>> y_test = strat_test_set["median_house_value"].copy()
```

# Fine-tune Model

---

## Evaluate Model on Test Set - Process

3. Run full\_pipeline to transform the data

```
>>> X_test_prepared = full_pipeline.transform(X_test)
```

# Fine-tune Model

---

## Evaluate Model on Test Set - Process

### 4. Evaluate the final model on the test set

```
# Predictions on Test set
```

```
>>> final_predictions = final_model.predict(X_test_prepared)
```

```
# Calculate final RMSE
```

```
>>> final_mse = mean_squared_error(y_test,  
final_predictions)
```

```
>>> final_rmse = np.sqrt(final_mse)
```

# Fine-tune Model

---

## Evaluate Model on Test Set - Process

4. Evaluate the final model on the test set

```
>>> final_rmse = final_model.predict(X_test_prepared)
```

Output-

47,766.0039

# Fine-tune Model

---

Congratulation! We have trained and tested  
our first model :)

# Fine-tune Model

---

## Evaluate Model on Test Set

- This model is not perfect though
- We'll learn more ways to build better models as we progress in the course

# Fine-tune Model

---

## Evaluate Model on Test Set - Summary

- If we do lot of hyperparameters tuning than the
  - Performance of the model will be worse than the
  - Performance measured in cross-validation
- This happens because
  - The model is fine-tuned to perform well on the validation data
  - And performs badly on unknown datasets

# Fine-tune Model

---

## Evaluate Model on Test Set - Summary

- We must resist the temptation to tweak hyperparameters to look good on test set
- As model will not generalize to new data

# Checklist for Machine Learning Projects

---

1. Frame the problem and look at the big picture
2. Get the data
3. Explore the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Explore many different models and short-list the best ones
6. Fine-tune model
- 7. Present the solution**
8. Launch, monitor, and maintain the system

# Present the Solution

---

## Pre Launch Phase

# Present the Solution

---

## Pre Launch Phase

- Now we need to present the solution
  - What have we learned
  - What worked and what did not
  - What assumptions were made
  - What are model's limitations

# Present the Solution

---

## Pre Launch Phase

- Document everything
- Create nice presentations
  - With clear visualizations

# Checklist for Machine Learning Projects

---

1. Frame the problem and look at the big picture
2. Get the data
3. Explore the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Explore many different models and short-list the best ones
6. Fine-tune model
7. Present the solution
- 8. Launch, monitor, and maintain the system**

# Launch, Monitor and Maintain the system

---

## Launch the System

# Launch, Monitor and Maintain the system

---

- Prepare the solution for production
  - Plugging the production input data sources

# Launch, Monitor and Maintain the system

---

## Monitor the System

# Launch, Monitor and Maintain the system

---

## Monitor the System

- Write monitoring code to check
  - Model's live performance at regular intervals
  - Trigger alerts when it drops
- Important to catch performance degradation
  - As models tends to **rot**
  - If not trained on fresh data

# Launch, Monitor and Maintain the system

---

## Plug Human Evaluation Pipeline into System

- Sample the model's predictions and evaluate them from time to time
- This generally requires a human analysis
- These analysts may be
  - Field experts or
  - Workers on a crowdsourcing platform such as
    - Amazon Mechanical Turk
    - CrowdFlower

# Launch, Monitor and Maintain the system

---

## Evaluate the Input Data Quality

- Evaluate the model's input data quality from time to time
- Performance may degrade if the input data quality is not good
  - For example, malfunction sensors sending random values
- Monitoring input data quality is
  - Especially important for online learning systems

# Launch, Monitor and Maintain the system

---

## Maintain the System

# Launch, Monitor and Maintain the system

---

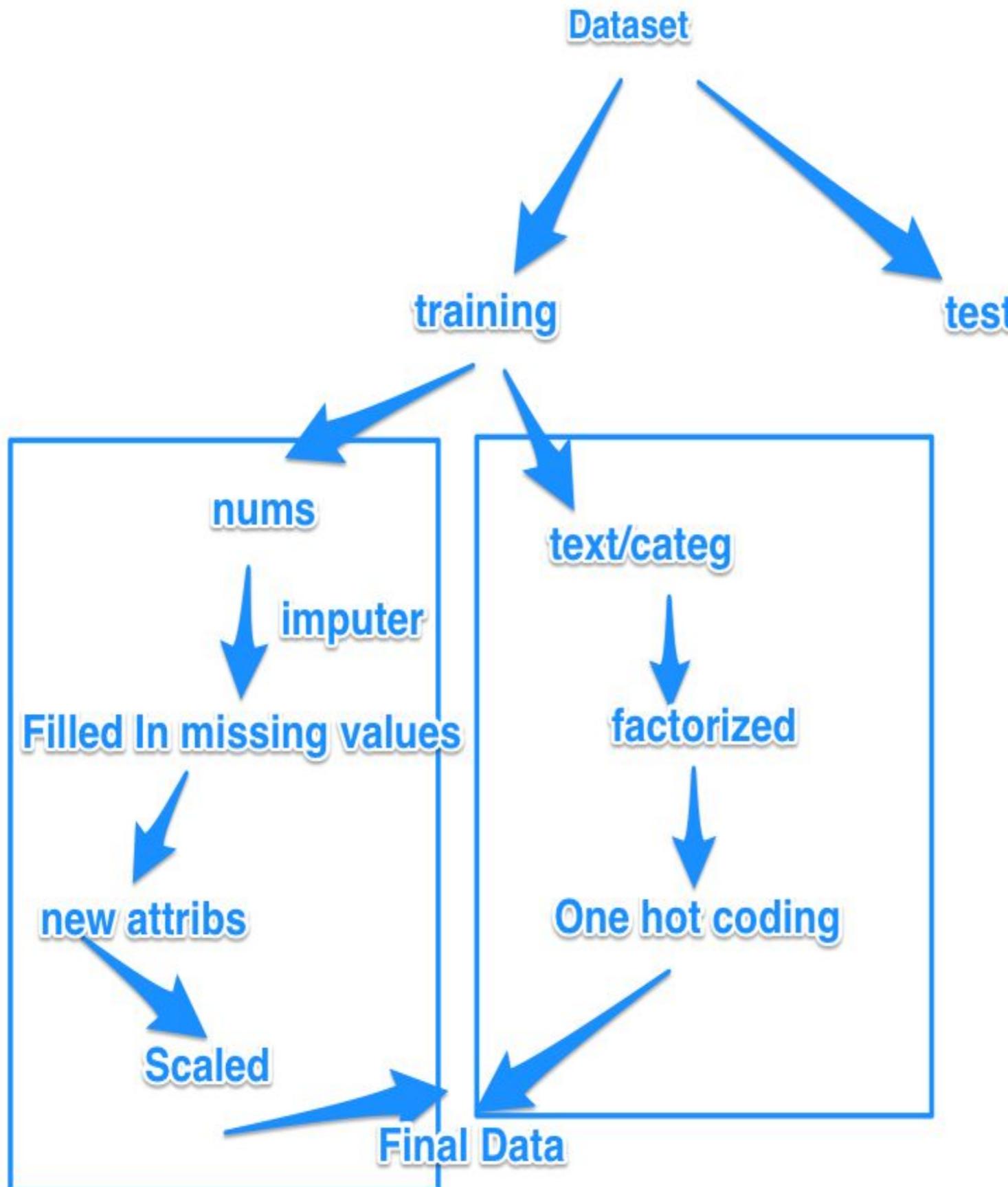
## Maintain the System

- Train model on a regular basis using fresh data
  - Automate the process of regularly updating of training model with fresh data
  - Else system's performance may fluctuate severely over time
- For an online learning system
  - Make sure to save snapshots of its state at regular intervals
  - So that we can easily roll back to a previously working state

---

---

# Summary



# Summary

---

- As we can see, much of the work is in the
  - Data preparation step
  - Deploying
  - Building monitoring tools
  - Setting up human evaluation pipelines and
  - Automating regular model training

# Summary

---

- Be comfortable with the overall process
- Know three or four algorithms
  - Rather than to spend all your time exploring advanced algorithms
  - And not spending enough time on the overall process
- Spend enough time on the overall process

# Summary

---

## In this chapter we learnt about

1. Statistical Inference, Probability and Measures of Central Tendency
2. Creating a Machine Learning Model
  - a. Getting the data
  - b. Exploring the data
  - c. Splitting our data into Train and Test set
  - d. Exploring different models and choosing the best one
  - e. Fine Tuning our model
  - f. Presenting the solution

# Summary

---

**In this course we learnt about**

3. Stratified Sampling using Scikit learn

4. Visualizing our dataset

5. Looking for Correlations in the data

6. Data Cleaning

7. Training a model using Scikit learn

# Questions?

---

<https://discuss.cloudxlab.com>

[reachus@cloudxlab.com](mailto:reachus@cloudxlab.com)

