

Machine Learning : 06048203

$$f = G \frac{m_1 m_2}{d^2}$$



# Linear Regression

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

$$E = mc^2$$

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

$$\frac{df}{dt} = \lim_{h \rightarrow 0} \frac{f(t+h) - f(t)}{h}$$

# Linear Regression

- Linear Regression
  - Theory of Linear Regression
  - Simple Implementation with Python
  - Scikit-Learn Overview
  - Linear Regression with Scikit-learn
  - Polynomial Regression
  - Regularization

# Linear Regression

- This will include understanding:
  - Brief History
  - Linear Relationships
  - Ordinary Least Squares
  - Cost Functions
  - Gradient Descent
  - Vectorization

# Linear Regression

- 1809 - Carl Friedrich Gauss publishes his methods of calculating orbits of celestial bodies.
- Claiming to have invented least-squares back in 1795!



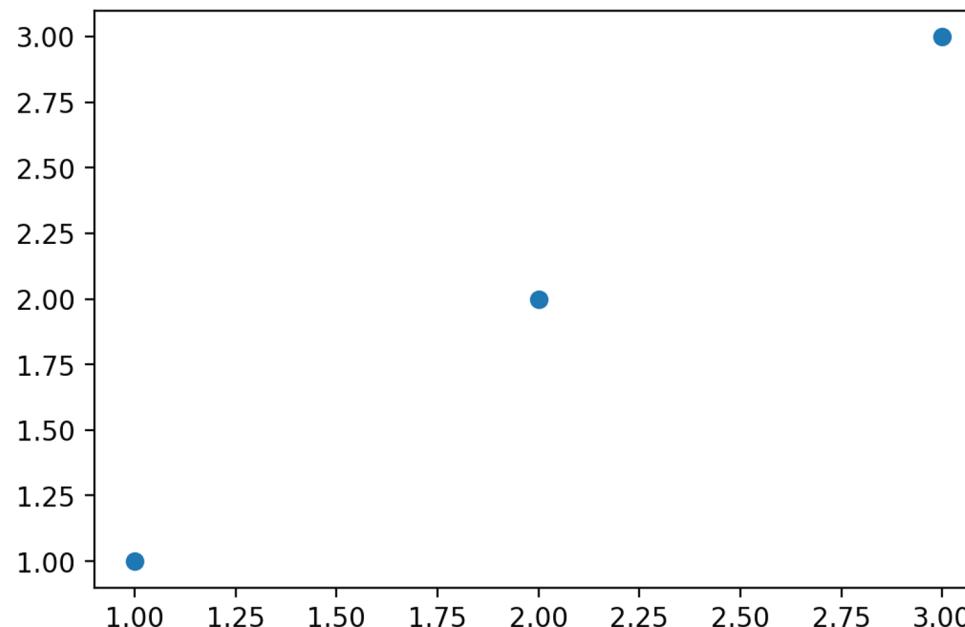
# Linear Regression

- 1808 - Robert Adrain published his formulation of least squares (a year before publication by Gauss).



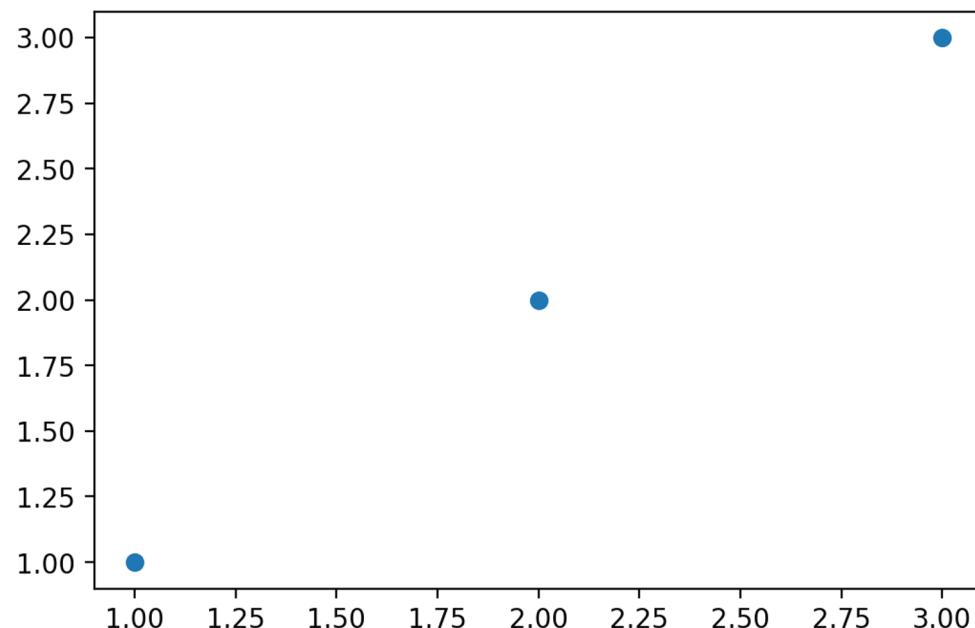
# Linear Regression

- Put simply, a linear relationship implies some constant straight line relationship.
- The simplest possible being  $y = x$ .



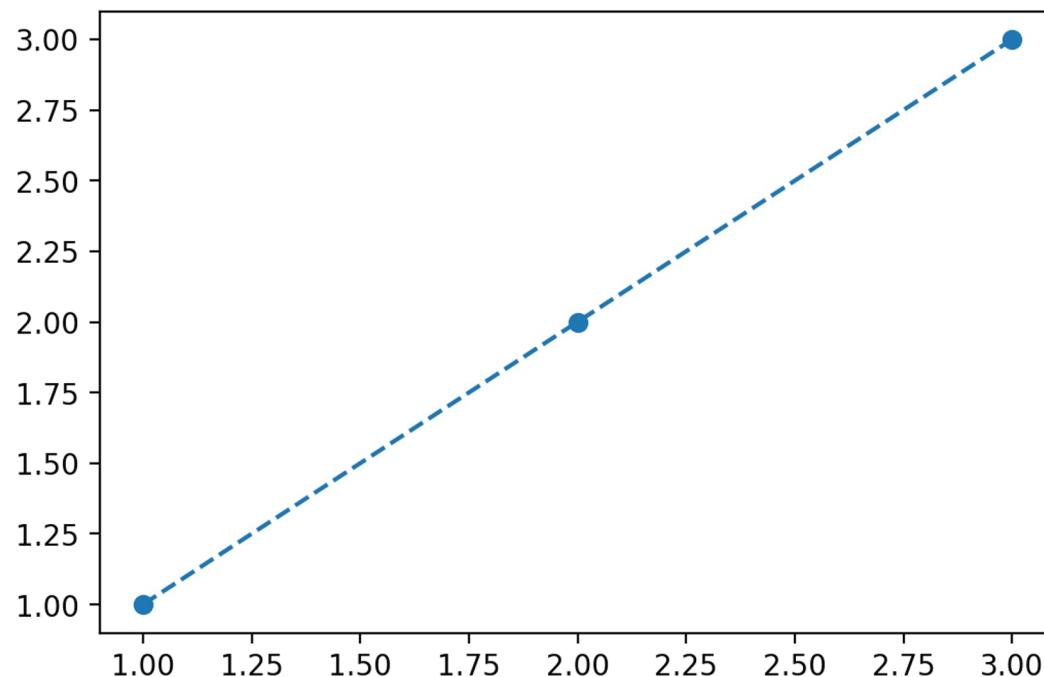
# Linear Regression

- Here we see  $x = [1,2,3]$  and  $y = [1,2,3]$



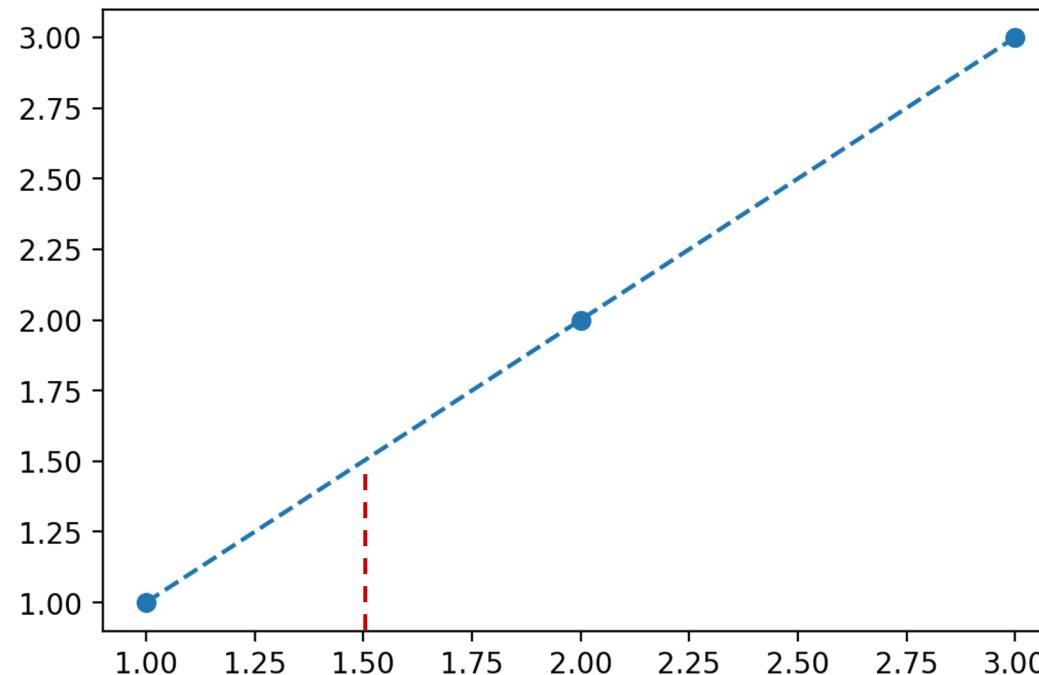
# Linear Regression

- We could then (based on the three real data points) build out the relationship  $y=x$  as our “fitted” line.



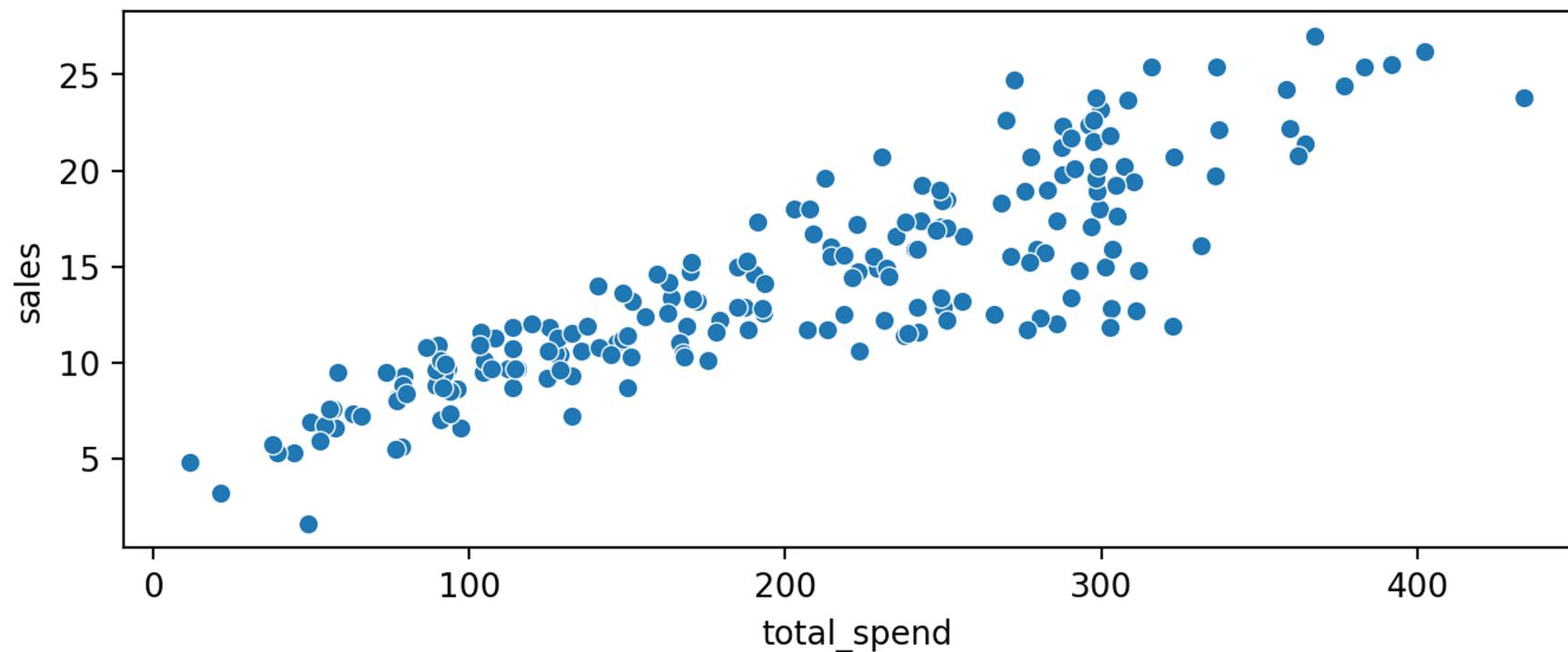
# Linear Regression

- This implies for some new x value I can predict its related y.



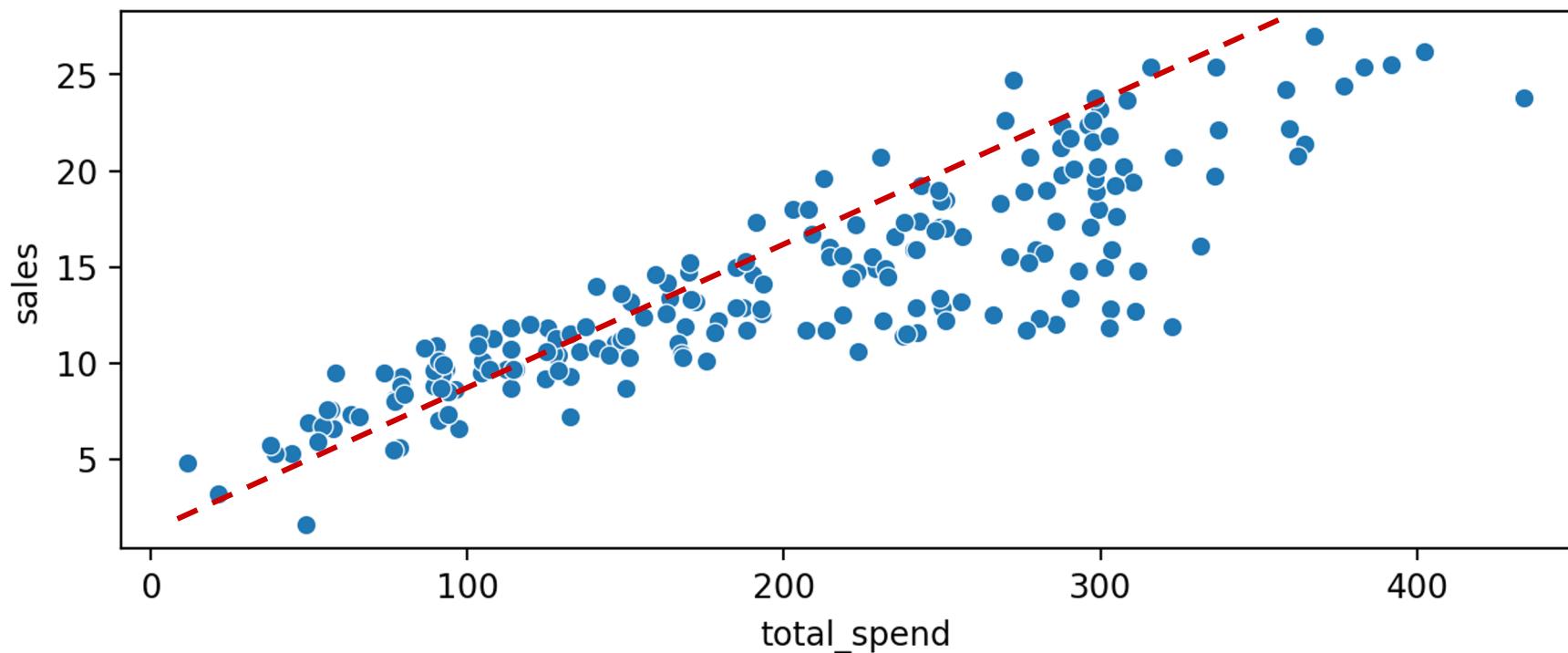
# Linear Regression

- But what happens with real data? Where do we draw this line?



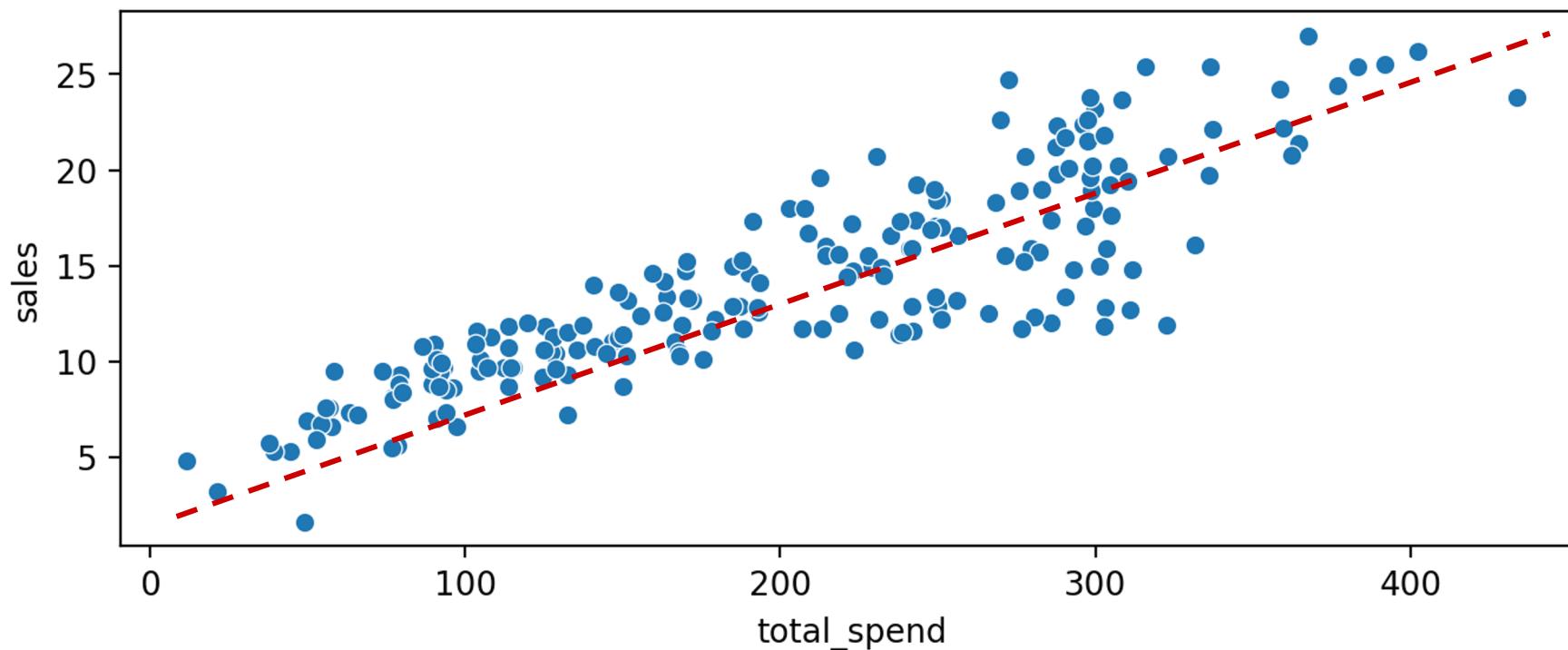
# Linear Regression

- But what happens with real data? Where do we draw this line?



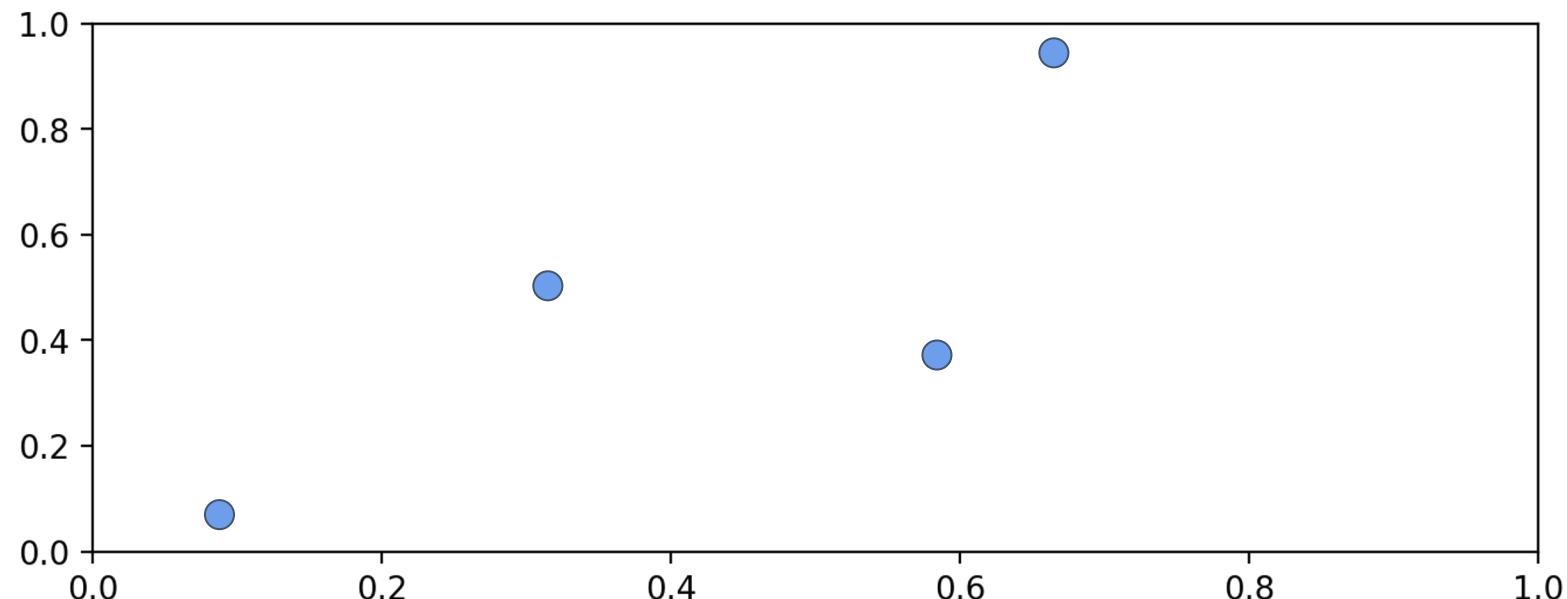
# Linear Regression

- But what happens with real data? Where do we draw this line?



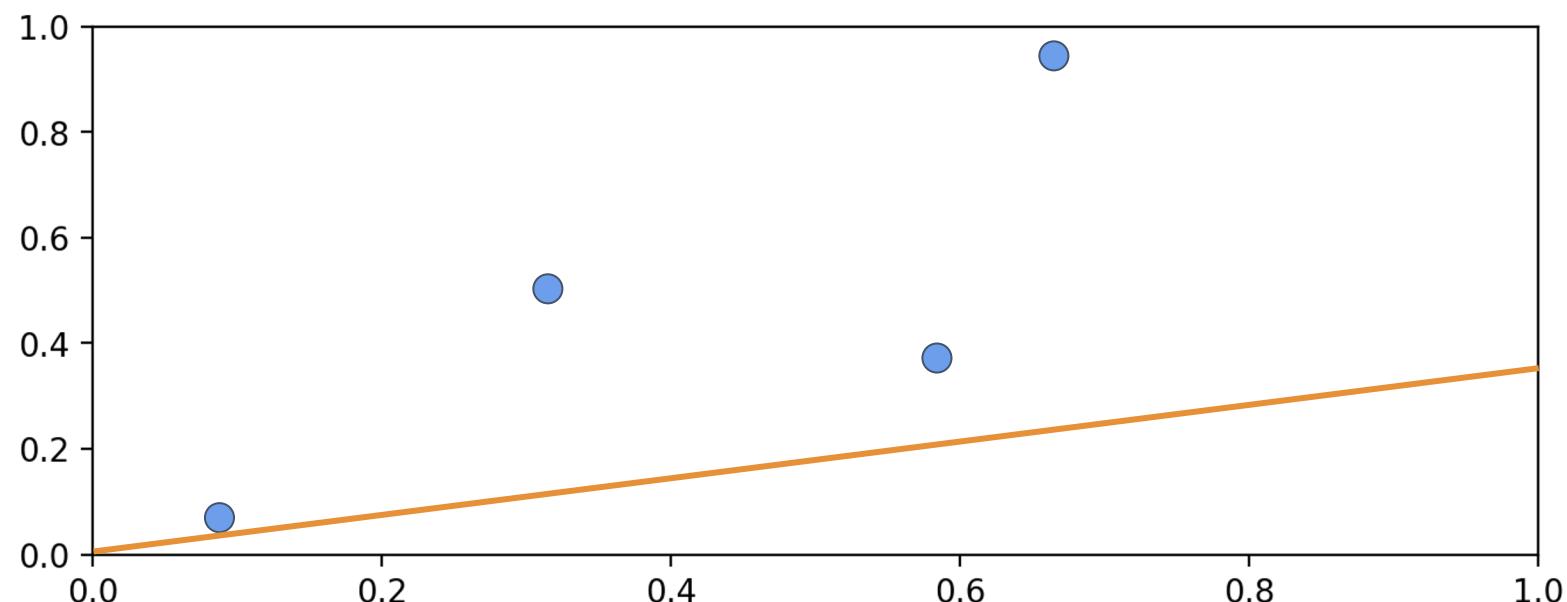
# Linear Regression

- Fundamentally, we understand we want to minimize the overall distance from the points to the line.



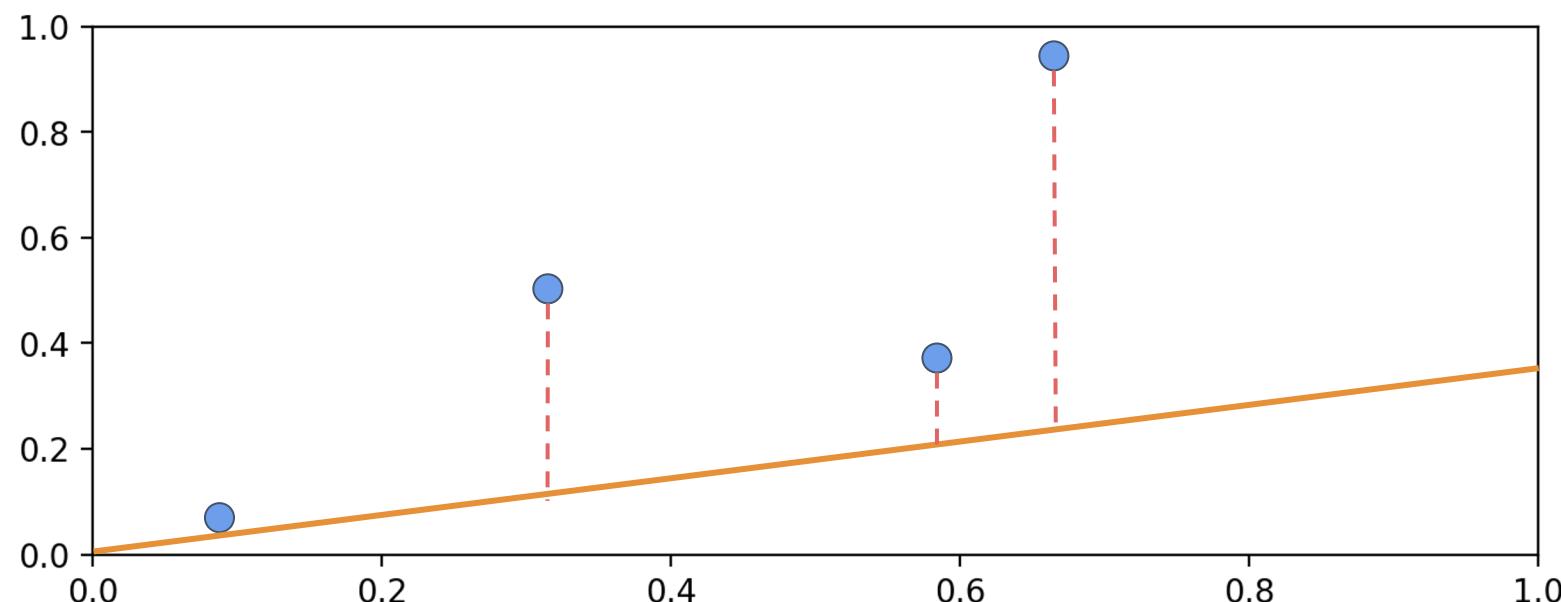
# Linear Regression

- Fundamentally, we understand we want to minimize the overall distance from the points to the line.



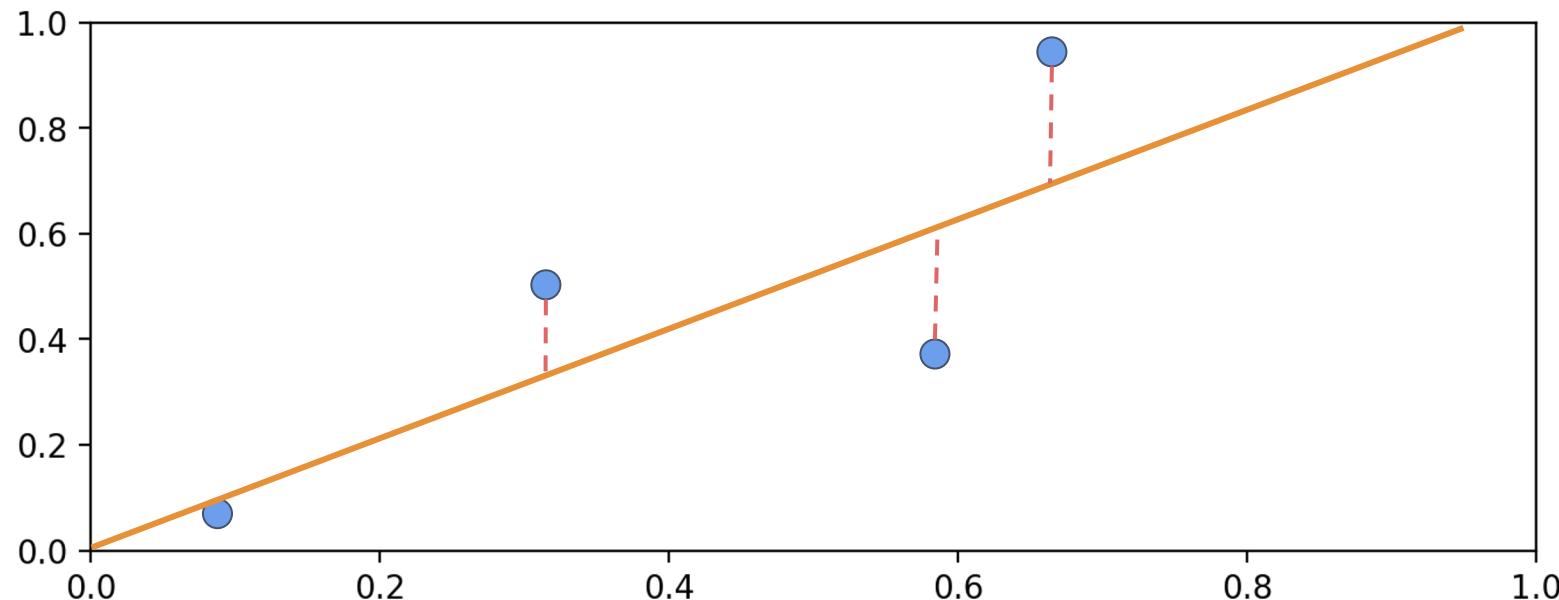
# Linear Regression

- We also know we can measure this error from the real data points to the line, known as the **residual error**.



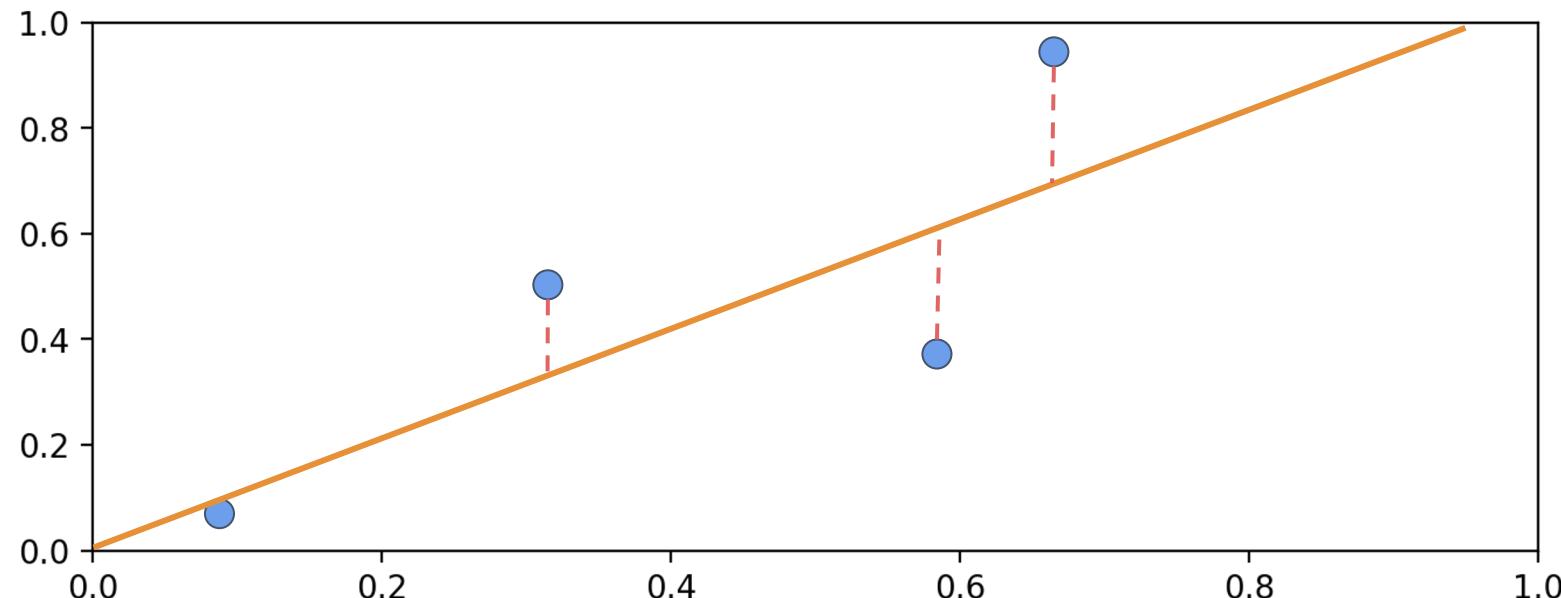
# Linear Regression

- Some lines will clearly be better fits than others.



# Linear Regression

- We can also see the **residuals** can be both positive and negative.

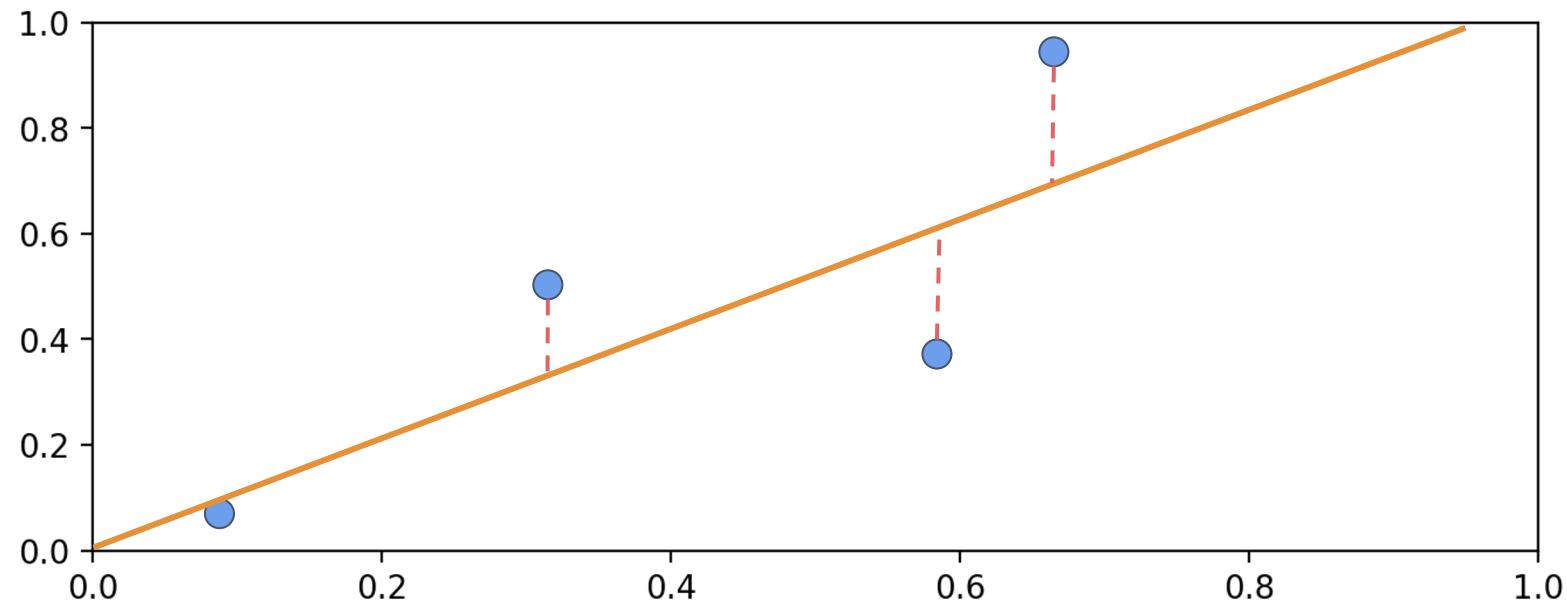


# Linear Regression

- **Ordinary Least Squares** works by minimizing the sum of the squares of the differences between the observed dependent variable (values of the variable being observed) in the given dataset and those predicted by the linear function.

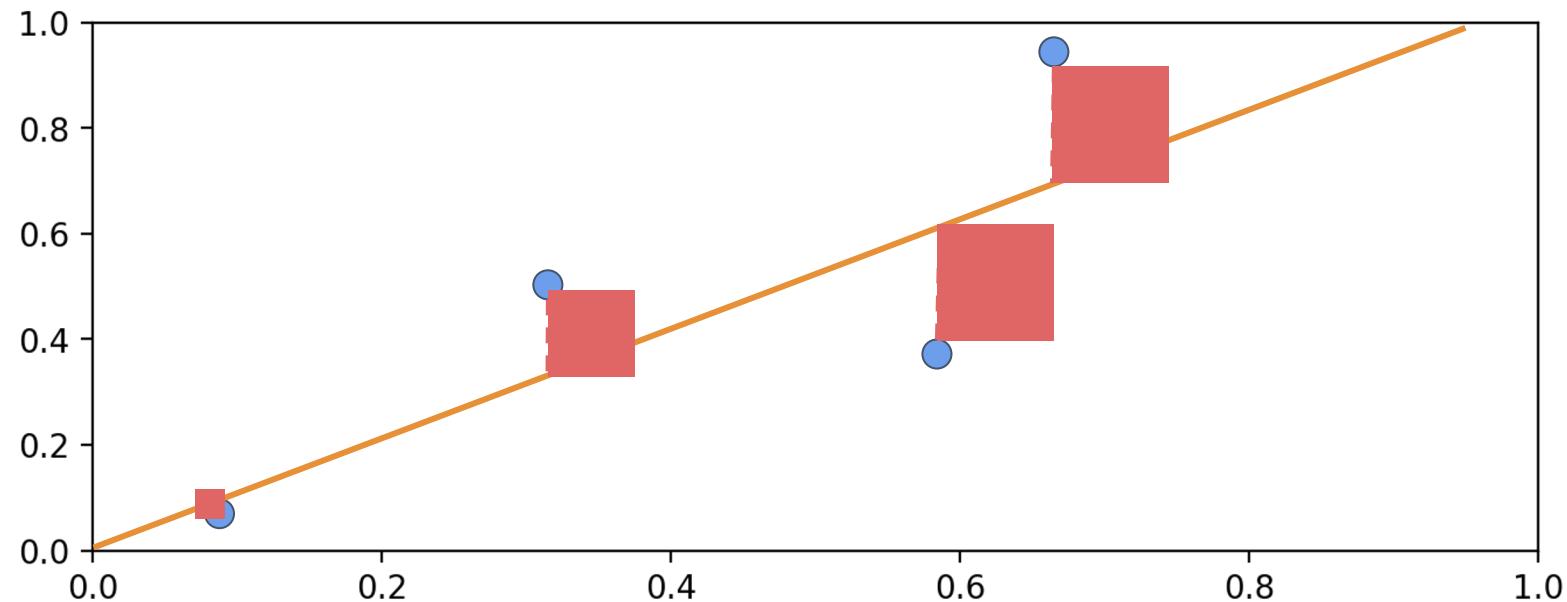
# Linear Regression

- We can visualize squared error to minimize:



# Linear Regression

- We can visualize squared error to minimize:



# Linear Regression

- Having a squared error will help us simplify our calculations later on when setting up a derivative.
- Let's continue exploring OLS by converting a real data set into mathematical notation, then working to solve a linear relationship between features and a variable!

# **Introduction to Linear Regression**

**Algorithm Theory - Part Two  
OLS Equations**

# Linear Regression

- Linear Regression OLS Theory
  - We know the equation of a simple straight line:
    - $y = mx + b$ 
      - m is slope
      - b is intercept with y-axis

# Linear Regression

- Linear Regression OLS Theory
  - We can see for  $y=mx+b$  there is only room for one possible feature  $x$ .
  - OLS will allow us to directly solve for the slope **m** and intercept **b**.
  - We will later see we'll need tools like gradient descent to scale this to multiple features.

# Linear Regression

- Linear Regression allows us to build a relationship between multiple **features** to estimate a **target output**.

Area m <sup>2</sup>	Bedrooms	Bathroom s	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

# Linear Regression

- We can translate this data into generalized mathematical notation...

	<b>x</b>		<b>y</b>
<b>Area m<sup>2</sup></b>	<b>Bedrooms</b>	<b>Bathroom s</b>	<b>Price</b>
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

# Linear Regression

- We can translate this data into generalized mathematical notation...

$x_1$	$x_2$	$x_3$	$y$
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

# Linear Regression

- We can translate this data into generalized mathematical notation...

x			y
$x_1$	$x_2$	$x_3$	
$x_1^1$	3	2	\$500,000
$x_1^2$	2	1	\$450,000
$x_1^3$	3	3	\$650,000
$x_1^4$	1	1	\$400,000
$x_1^5$	2	2	\$550,000

# Linear Regression

- Now let's build out a linear relationship between the features  $X$  and label  $y$ .

X			y
$x_1$	$x_2$	$x_3$	y
$x^1_1$	$x^1_1$	$x^1_1$	$y_1$
$x^2_1$	$x^2_1$	$x^2_1$	$y_2$
$x^3_1$	$x^3_1$	$x^3_1$	$y_3$
$x^4_1$	$x^4_1$	$x^4_1$	$y_4$
$x^5_1$	$x^5_1$	$x^5_1$	$y_5$

# Linear Regression

- Now let's build out a linear relationship between the features  $X$  and label  $y$ .

$x$	$y$
$x_1$	$x_2$

# Linear Regression

- Reformat for  $y = x$  equation

$y$	$x$
$y$	$x_1$ $x_2$ $x_3$

# Linear Regression

- Each feature should have some Beta coefficient associated with it.



$$\hat{y} = \beta_0 x_0 + \cdots + \beta_n x_n$$

---

# Linear Regression

- This is the same as the common notation for a simple line: **y=mx+b**



$$\hat{y} = \beta_0 x_0 + \cdots + \beta_n x_n$$

# Linear Regression

- This is stating there is some Beta coefficient for each feature to minimize error.



$$\hat{y} = \beta_0 x_0 + \cdots + \beta_n x_n$$

# Linear Regression

- We can also express this equation as a sum:

y	x
y	x <sub>1</sub>   x <sub>2</sub>   x <sub>3</sub>

$$\hat{y} = \beta_0 x_0 + \cdots + \beta_n x_n$$

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$

# Linear Regression

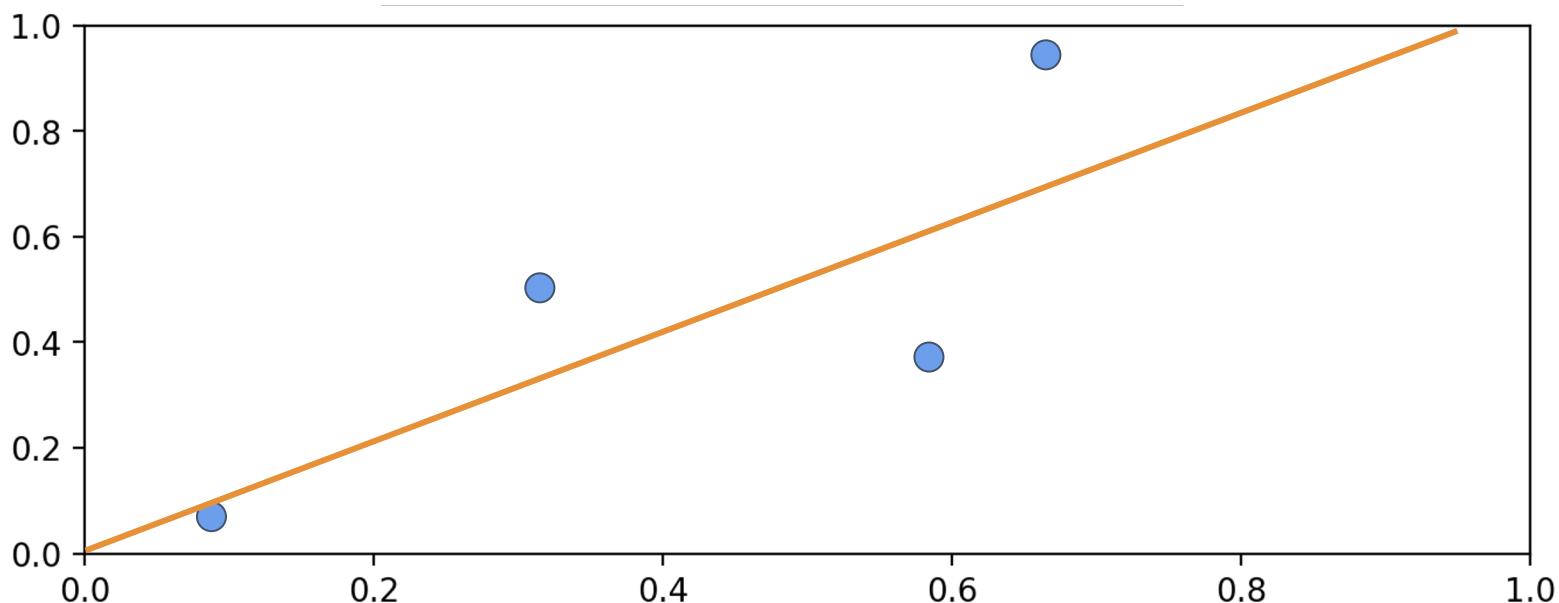
- Note the  $\hat{y}$  hat symbol displays a prediction. There is usually no set of Betas to create a perfect fit to  $y$ !

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$

# Linear Regression

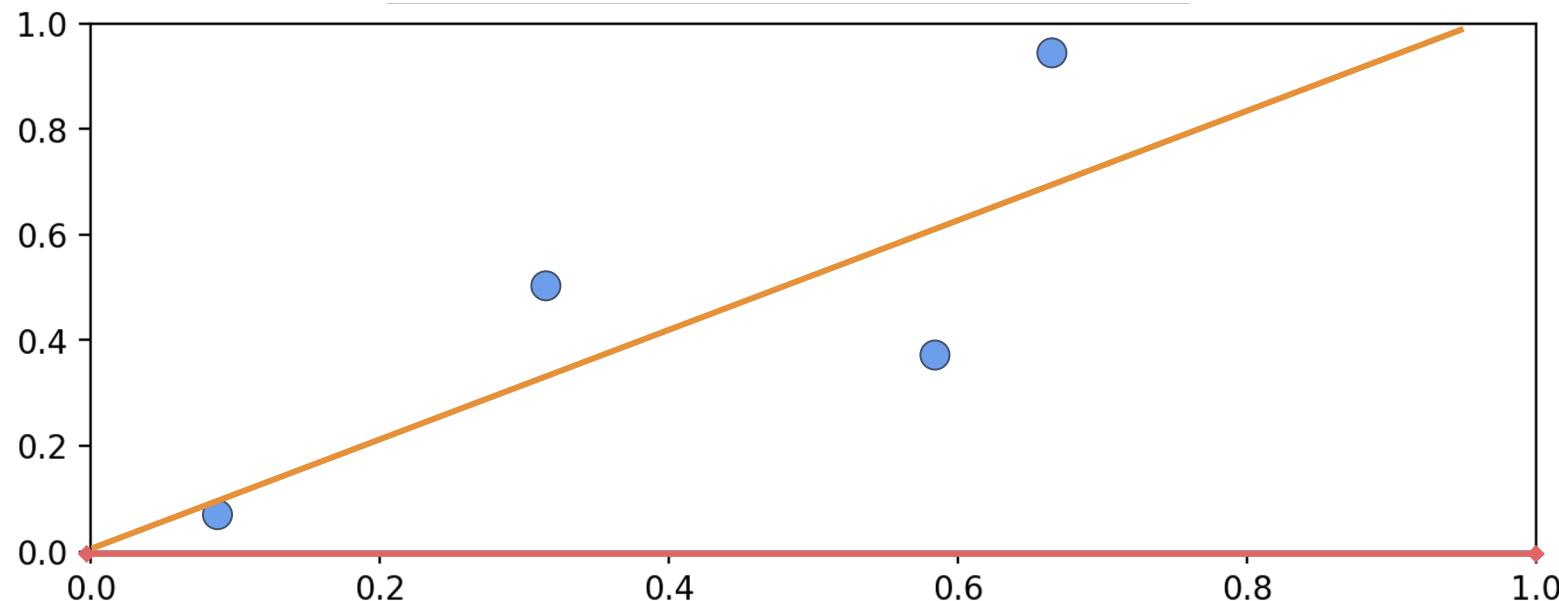
- Line equation:

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$



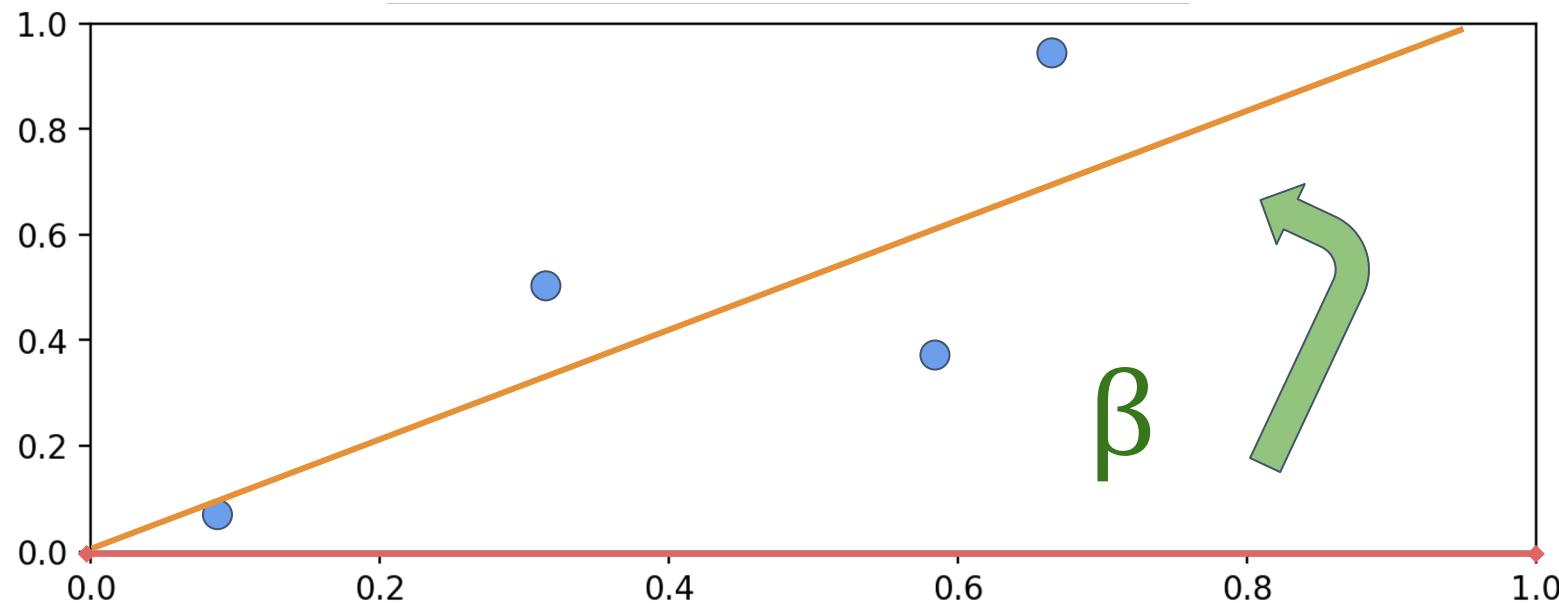
# Linear Regression

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$



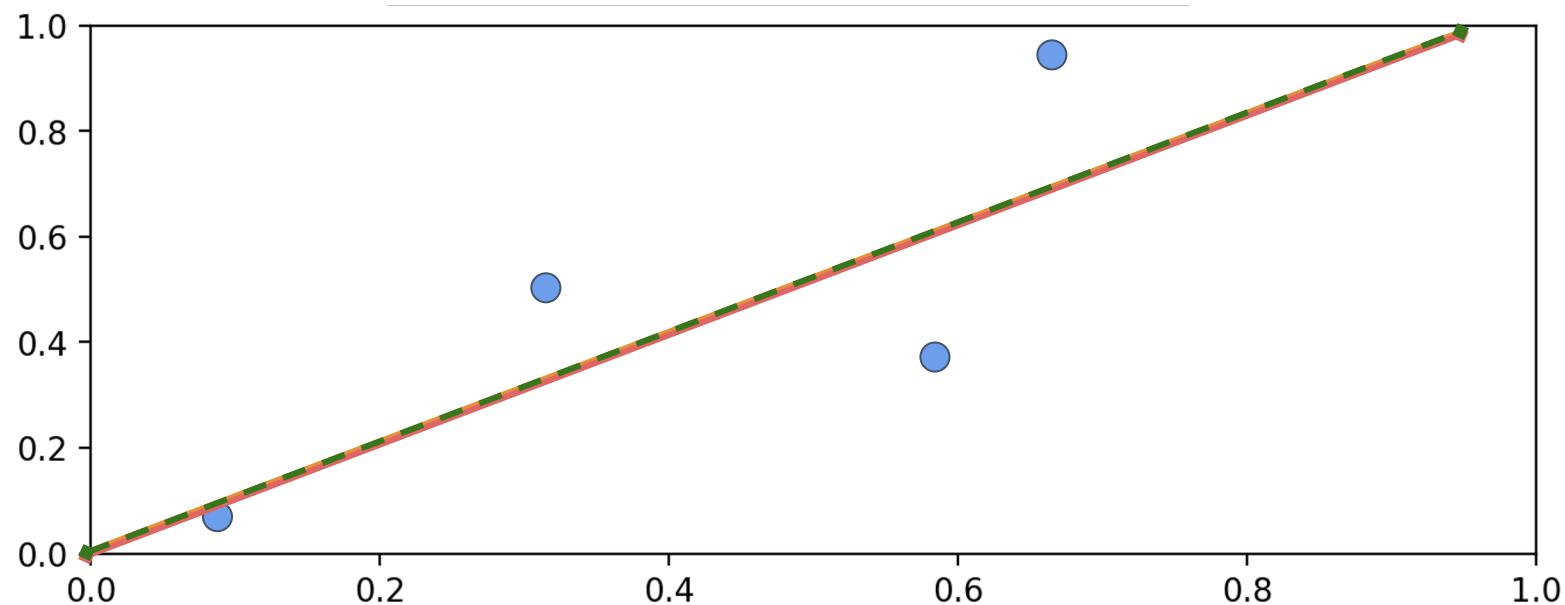
# Linear Regression

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$



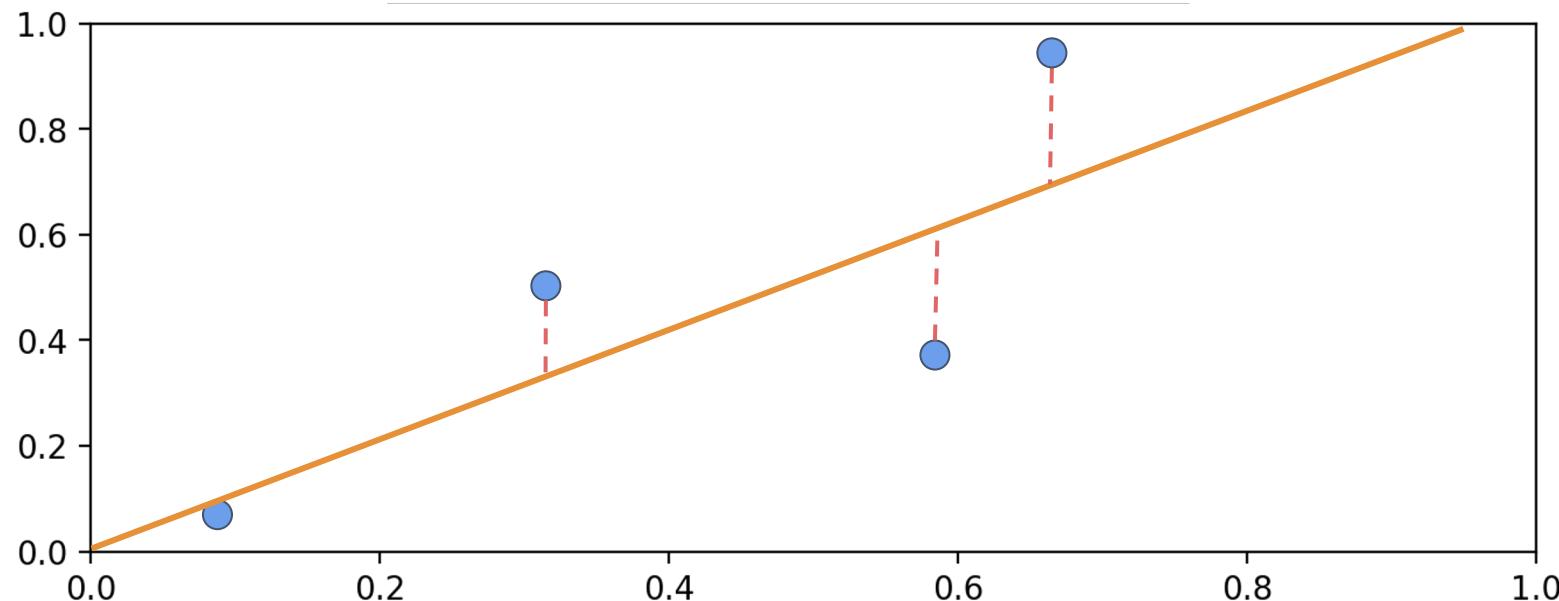
# Linear Regression

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$



# Linear Regression

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$



# Linear Regression

- For simple problems with one X feature we can easily solve for Betas values with an analytical solution.
- Let's quickly solve a simple example problem, then later we will see that for multiple features we will need gradient descent.

# Linear Regression

- As we expand to more than a single feature however, an analytical solution quickly becomes unscalable.
- Instead we shift focus on **minimizing a cost function** with **gradient descent**.

# Linear Regression

- We can use **gradient descent** to solve a **cost function** to calculate Beta values!

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$

# Introduction to Linear Regression

---

Cost Function

# Linear Regression

- What we know so far:
  - Linear Relationships
    - $y = mx+b$
  - OLS
    - Solve simple linear regression
  - Not scalable for multiple features
  - Translating real data to Matrix Notation
  - Generalized formula for Beta coefficients

# Linear Regression

- Recall we are searching for Beta values for a best-fit line.

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$

# Linear Regression

- The equation below simply defines our line, but how to choose beta coefficients?

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$

# Linear Regression

- We've decided to define a “best-fit” as **minimizing the squared error.**

$$\hat{y} = \sum_{i=0}^n \beta_i x_i$$

# Linear Regression

- The residual error for some row  $j$  is:

$$|y^j - \hat{y}^j|$$

# Linear Regression

- Squared Error for some row  $j$  is then:

$$(y^j - \hat{y}^j)^2$$

# Linear Regression

- Sum of squared errors for  $m$  rows is then:

$$\sum_{j=1}^m (y^j - \hat{y}^j)^2$$

# Linear Regression

- Average squared error for  $m$  rows is then:

$$\frac{1}{m} \sum_{j=1}^m (y^j - \hat{y}^j)^2$$

# Linear Regression

- Our cost function can be defined by the squared error:

$$J(\beta) = \frac{1}{2m} \sum_{j=1}^m (y^j - \hat{y}^j)^2$$

# Linear Regression

- It will be a function of **Betas** and **Features!**

$$\begin{aligned} J(\beta) &= \frac{1}{2m} \sum_{j=1}^m (y^j - \hat{y}^j)^2 \\ &= \frac{1}{2m} \sum_{j=1}^m \left( y^j - \sum_{i=0}^n \beta_i x_i^j \right)^2 \end{aligned}$$

# Linear Regression

- Recall from calculus to minimize a function we can take its derivative and set it equal to zero.

$$\begin{aligned}\frac{\partial J}{\partial \beta_k}(\boldsymbol{\beta}) &= \frac{\partial}{\partial \beta_k} \left( \frac{1}{2m} \sum_{j=1}^m \left( y^j - \sum_{i=0}^n \beta_i x_i^j \right)^2 \right) \\ &= \frac{1}{m} \sum_{j=1}^m \left( y^j - \sum_{i=0}^n \beta_i x_i^j \right) (-x_k^j)\end{aligned}$$

# Linear Regression

- Recall from calculus to minimize a function we can take its derivative and set it equal to zero.

$$\begin{aligned}\frac{\partial J}{\partial \beta_k}(\boldsymbol{\beta}) &= \frac{\partial}{\partial \beta_k} \left( \frac{1}{2m} \sum_{j=1}^m \left( y^j - \sum_{i=0}^n \beta_i x_i^j \right)^2 \right) \\ &= \frac{1}{m} \sum_{j=1}^m \left( y^j - \sum_{i=0}^n \beta_i x_i^j \right) (-x_k^j)\end{aligned}$$

# Linear Regression

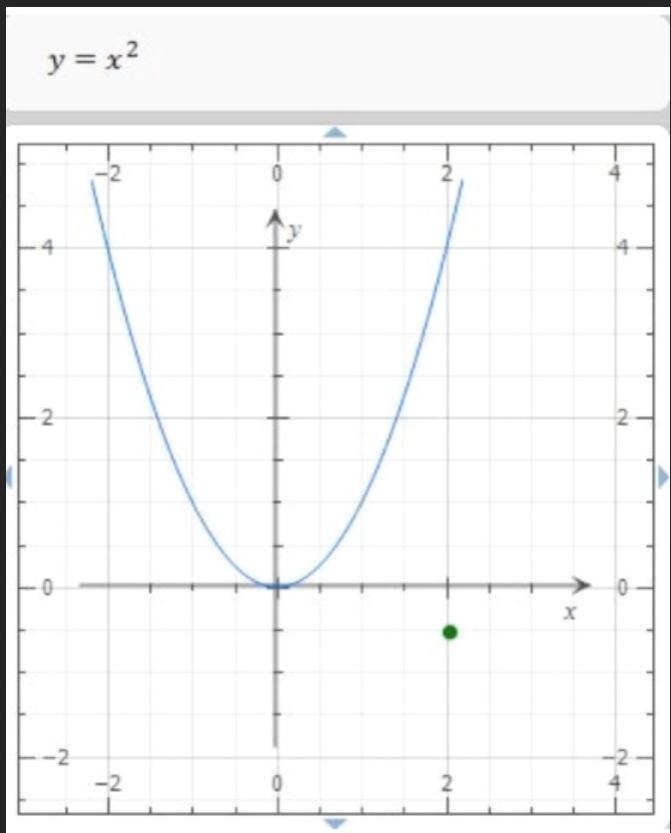
- Unfortunately, it is not scalable to try to get an analytical solution to minimize this cost function.
- In the next lecture we will learn to use **gradient descent** to minimize this **cost function**.

# Introduction to Linear Regression

---

Gradient Descent

# Gradient Descent



Minimize  $f(x) = x^2$

- Use calculus!
- $df / dx = 2x = 0$
- Solve for  $x$ :  $x = 0$

# Gradient descent Linear Regression

- **Title:** Linear Regression with Gradient Descent
- **Content:**
  - Linear regression aims to fit a line to a set of data points.
  - Equation:  $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$
  - Objective: Minimize the squared error between predicted values (from the linear model) and actual data points.

- **Title:** The Cost Function,  $J(\beta)$
- **Content:**
  - Defined as the Mean Squared Error (MSE):
$$J(\beta) = \frac{1}{2m} \sum_{i=1}^m (h_\beta(x^{(i)}) - y^{(i)})^2$$
where:
$$h_\beta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$
  - Objective: Minimize  $J(\beta)$ .

## Slide 3: Gradient Descent

- **Title:** Minimizing  $J(\beta)$  using Gradient Descent
- **Content:**
  - Iteratively update each parameter:
$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$
  - Where:
    - $\alpha$  is the learning rate.
    - The partial derivative represents the gradient of the cost function with respect to the parameter  $\beta_j$ .

# Linear Regression

- We just figured out a **cost function** to minimize!
- Taking the cost function derivative and then solving for zero to get the set of Beta coefficients will be too difficult to solve directly through an analytical solution.

# Linear Regression

- Instead we can describe this cost function through vectorized matrix notation and use **gradient descent** to have a computer figure out the set of Beta coefficient values that minimize the **cost/loss** function.

# Linear Regression

- Recall we now have the derivative of the cost function:
- 

$$\frac{\partial J}{\partial \beta_k}(\boldsymbol{\beta}) = \frac{1}{m} \sum_{j=1}^m \left( y^j - \sum_{i=0}^n \beta_i x_i^j \right) (-x_k^j)$$

# Linear Regression

- Use a **gradient** to express the derivative of the cost function with respect to each  $\beta$

$$\nabla_{\beta} J = \begin{bmatrix} \frac{\partial J}{\partial \beta_0} \\ \vdots \\ \frac{\partial J}{\partial \beta_n} \end{bmatrix}$$

# Linear Regression

- We also already know what this cost function derivative is equal to:
- 

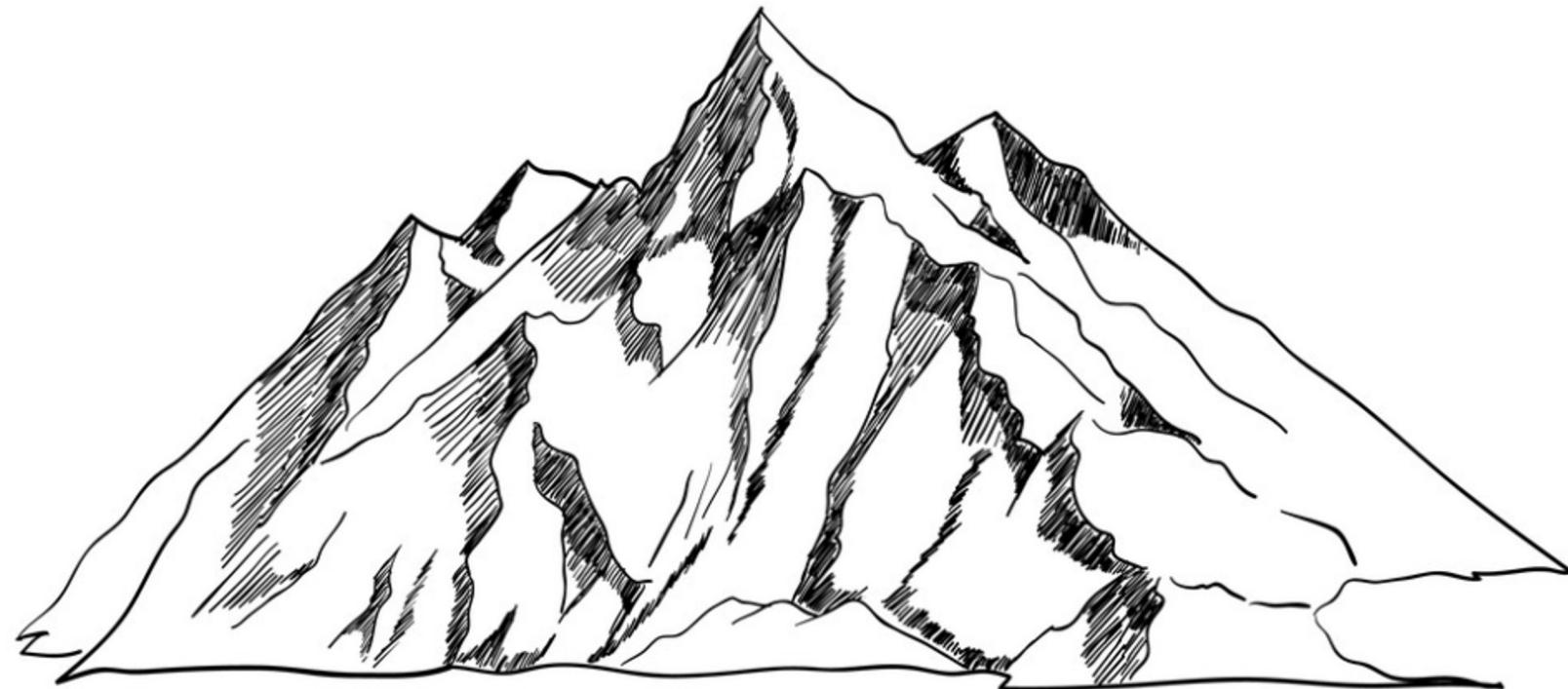
$$\nabla_{\beta} J = \begin{bmatrix} -\frac{1}{m} \sum_{j=1}^m \left( y^j - \sum_{i=0}^n \beta_i x_i^j \right) x_0^j \\ \vdots \\ -\frac{1}{m} \sum_{j=1}^m \left( y^j - \sum_{i=0}^n \beta_i x_i^j \right) x_n^j \end{bmatrix} = \begin{bmatrix} \frac{\partial J}{\partial \beta_0} \\ \vdots \\ \frac{\partial J}{\partial \beta_n} \end{bmatrix}$$

# Linear Regression

- We can use gradient descent to computationally search for the coefficients that minimize this gradient.
- Let's visually explore what this looks like in the case of a single Beta value.

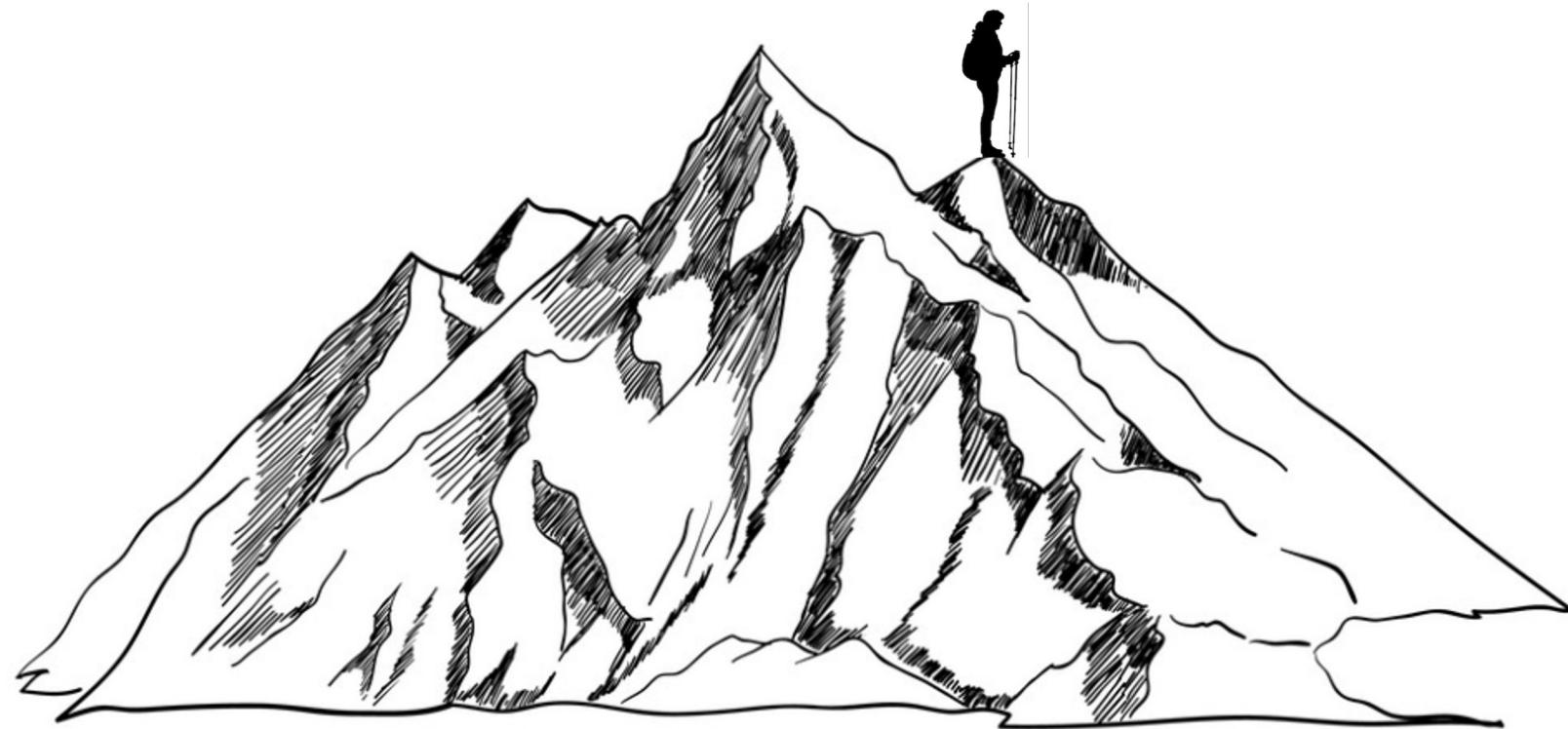
# Linear Regression

- Common mountain analogy



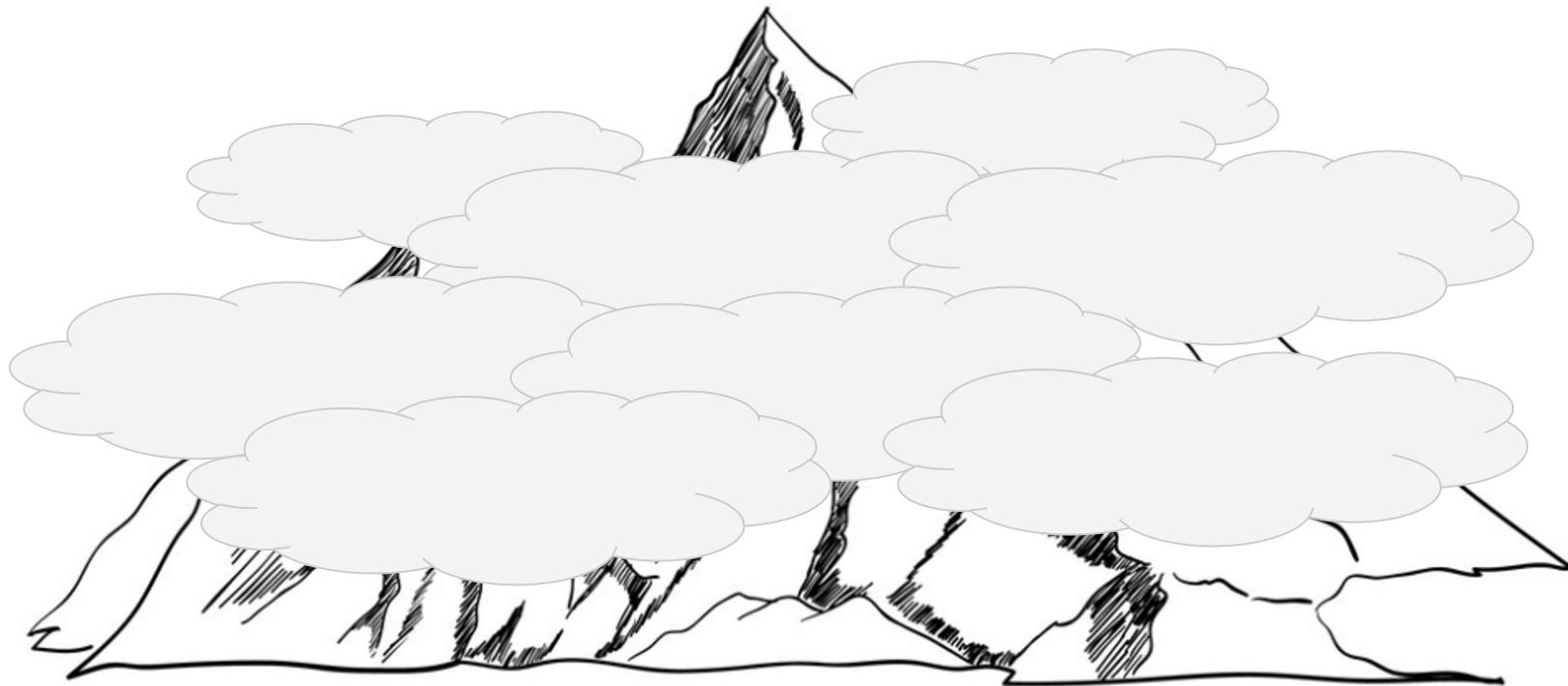
# Linear Regression

- Common mountain analogy



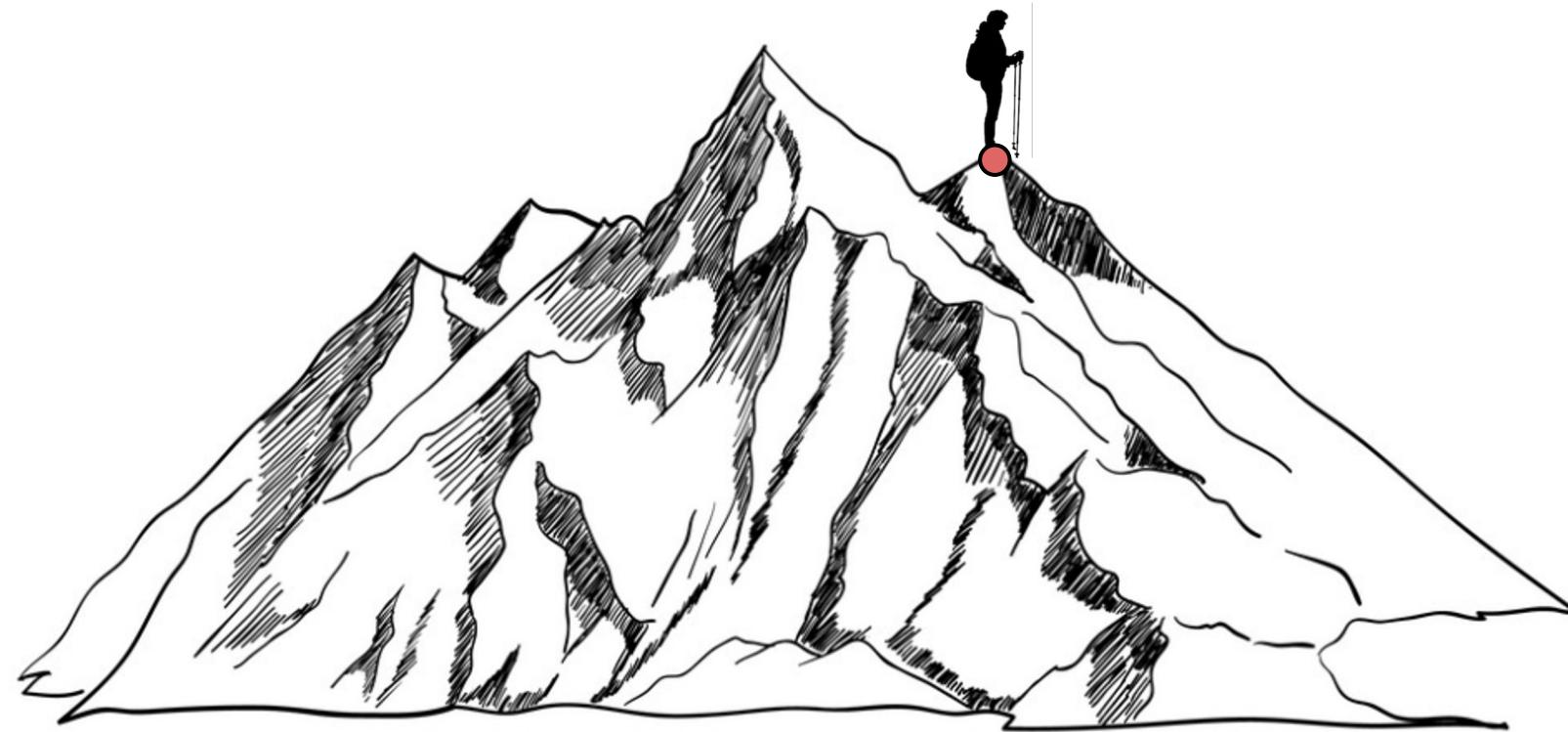
# Linear Regression

- Common mountain analogy



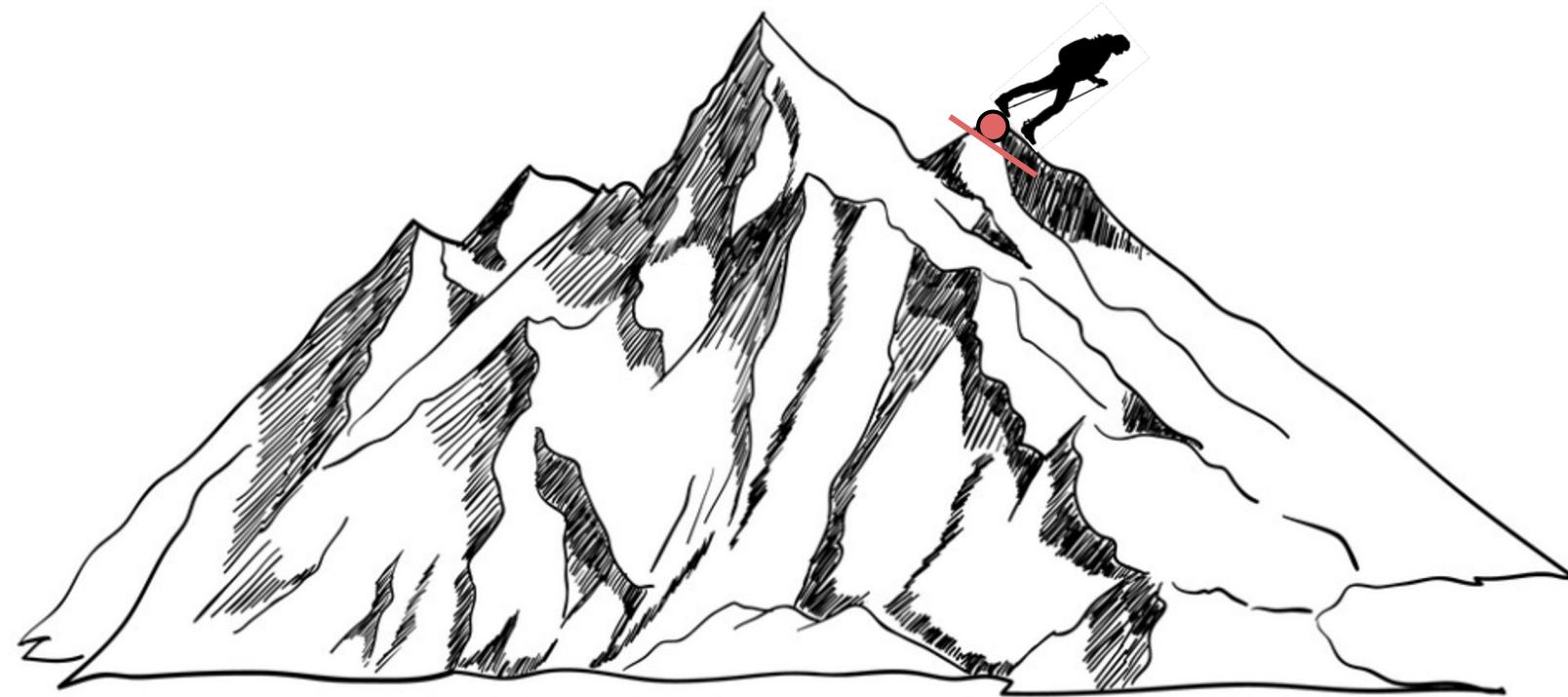
# Linear Regression

- Common mountain analogy



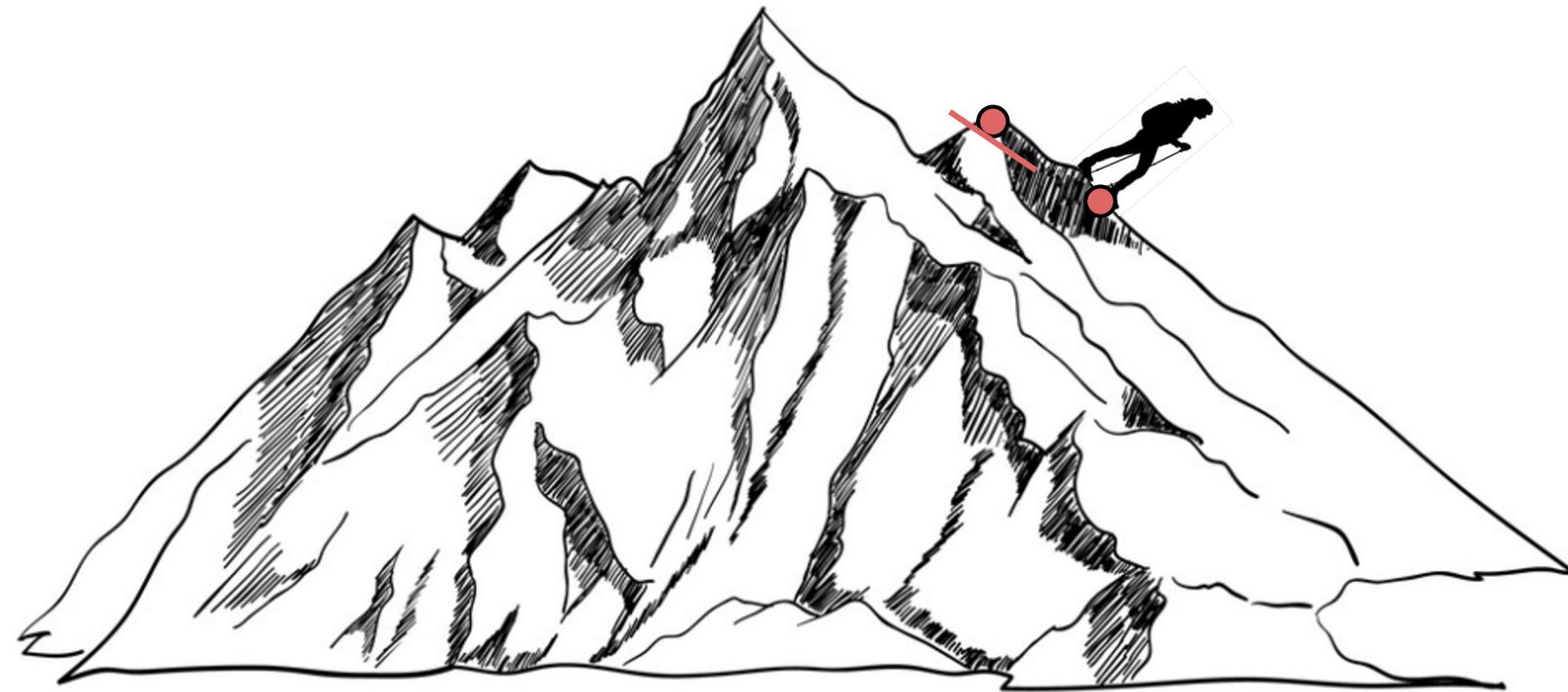
# Linear Regression

- Common mountain analogy



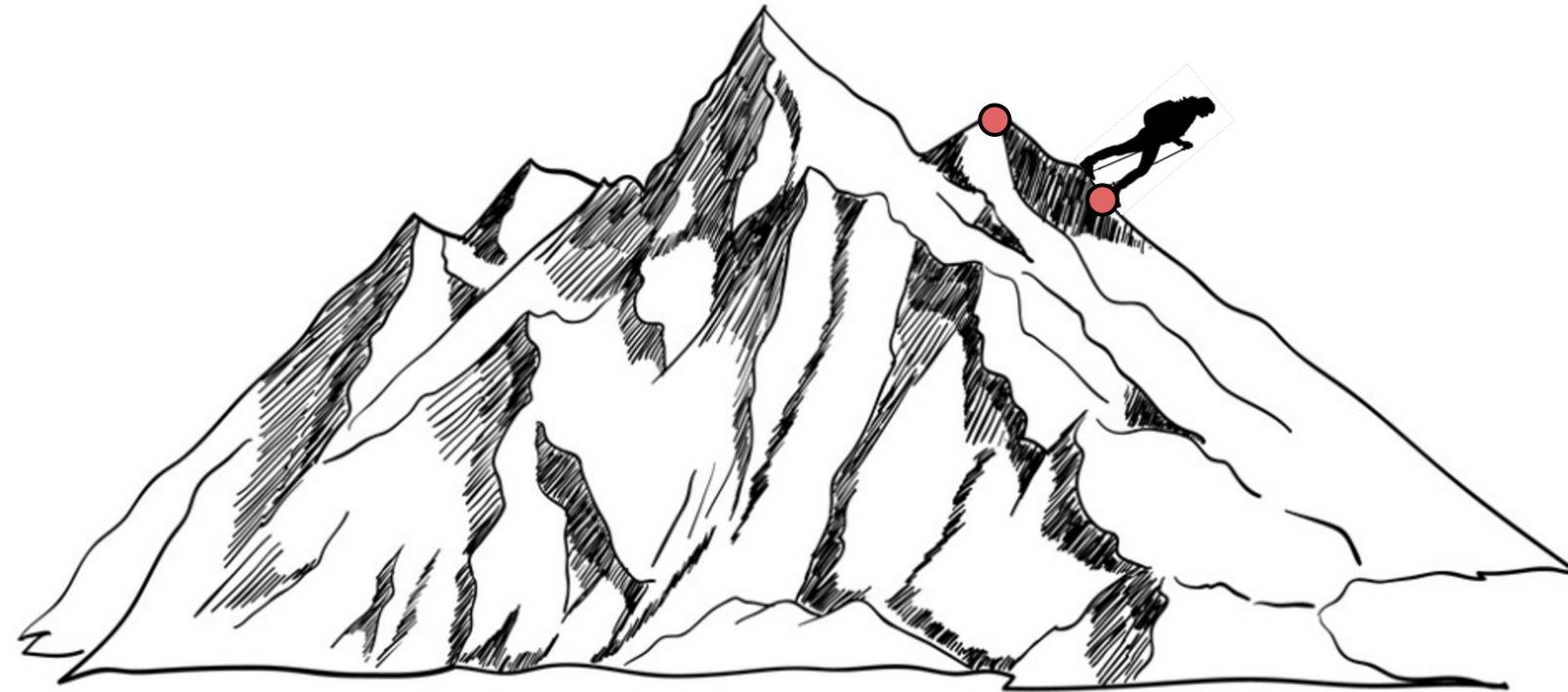
# Linear Regression

- Common mountain analogy



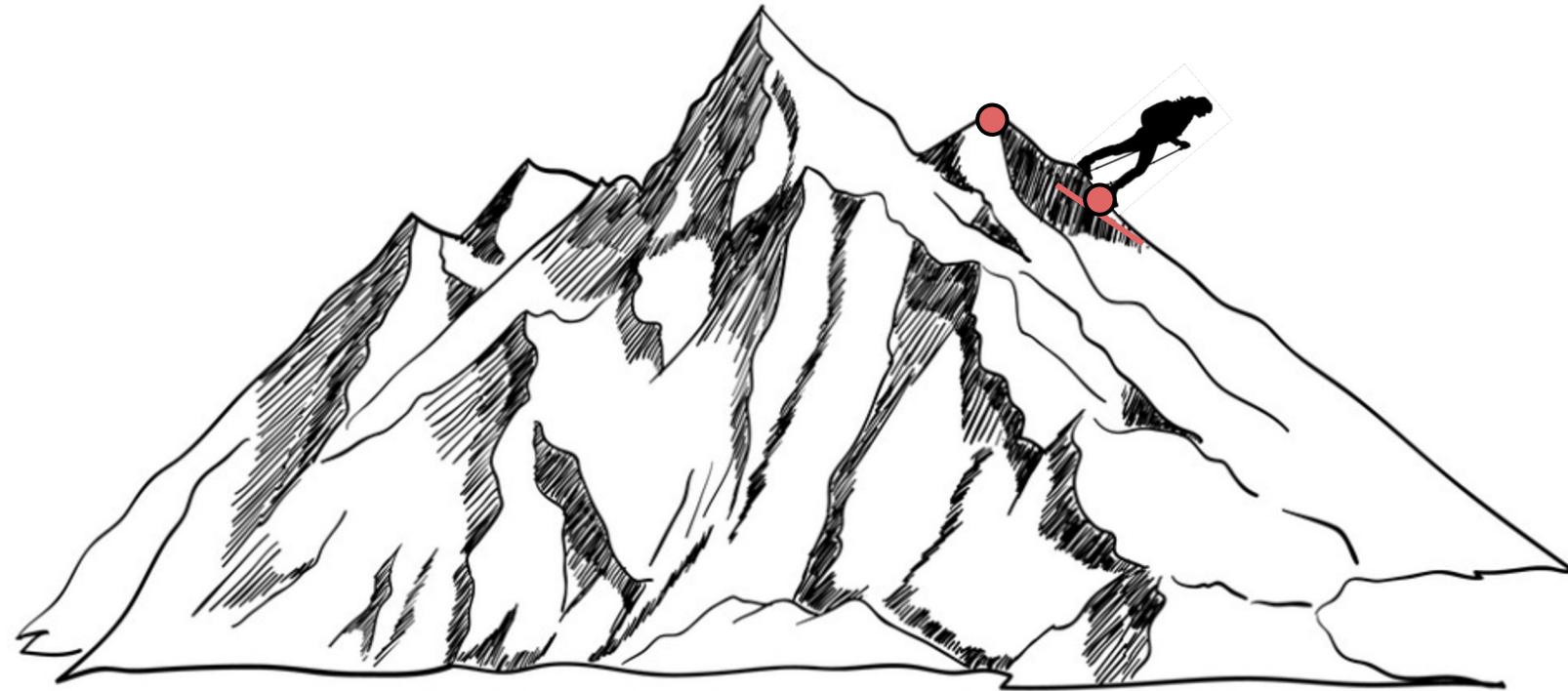
# Linear Regression

- Common mountain analogy



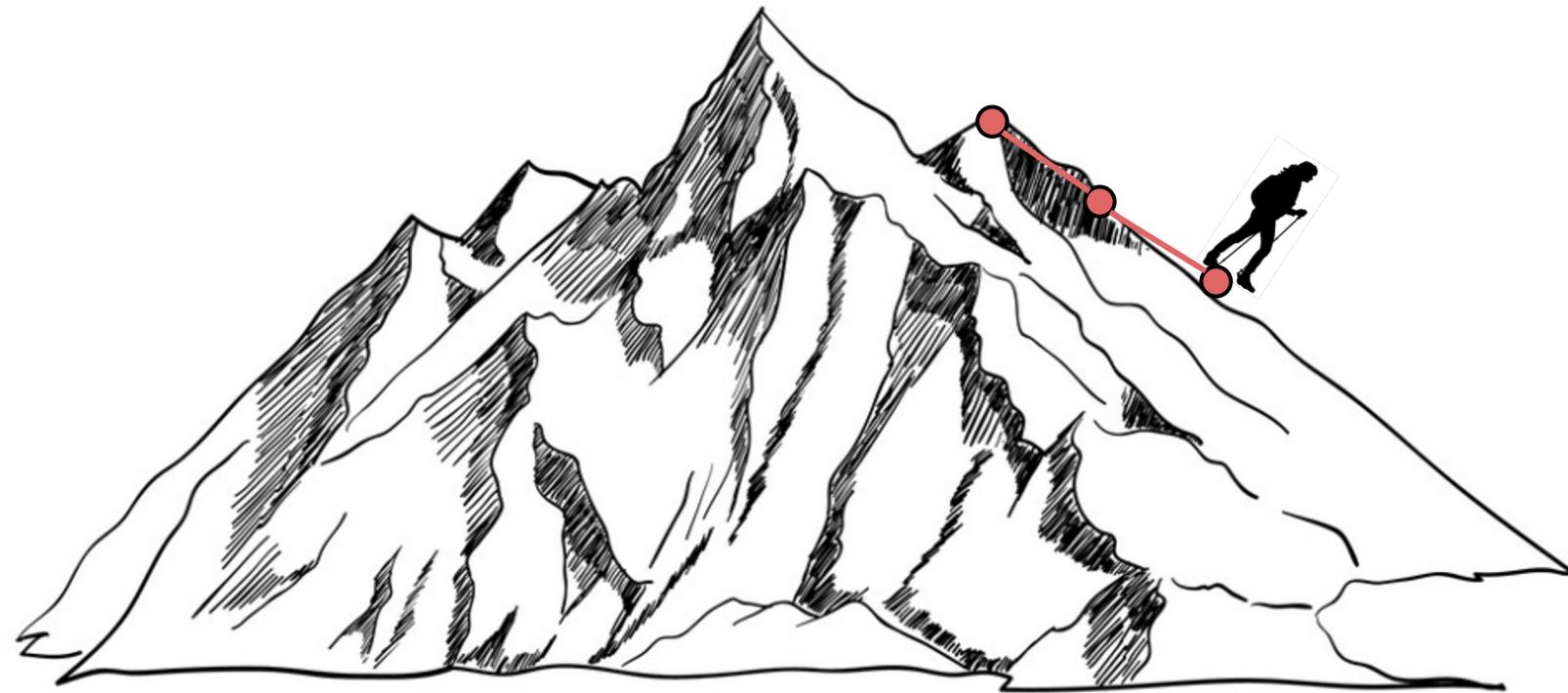
# Linear Regression

- Common mountain analogy



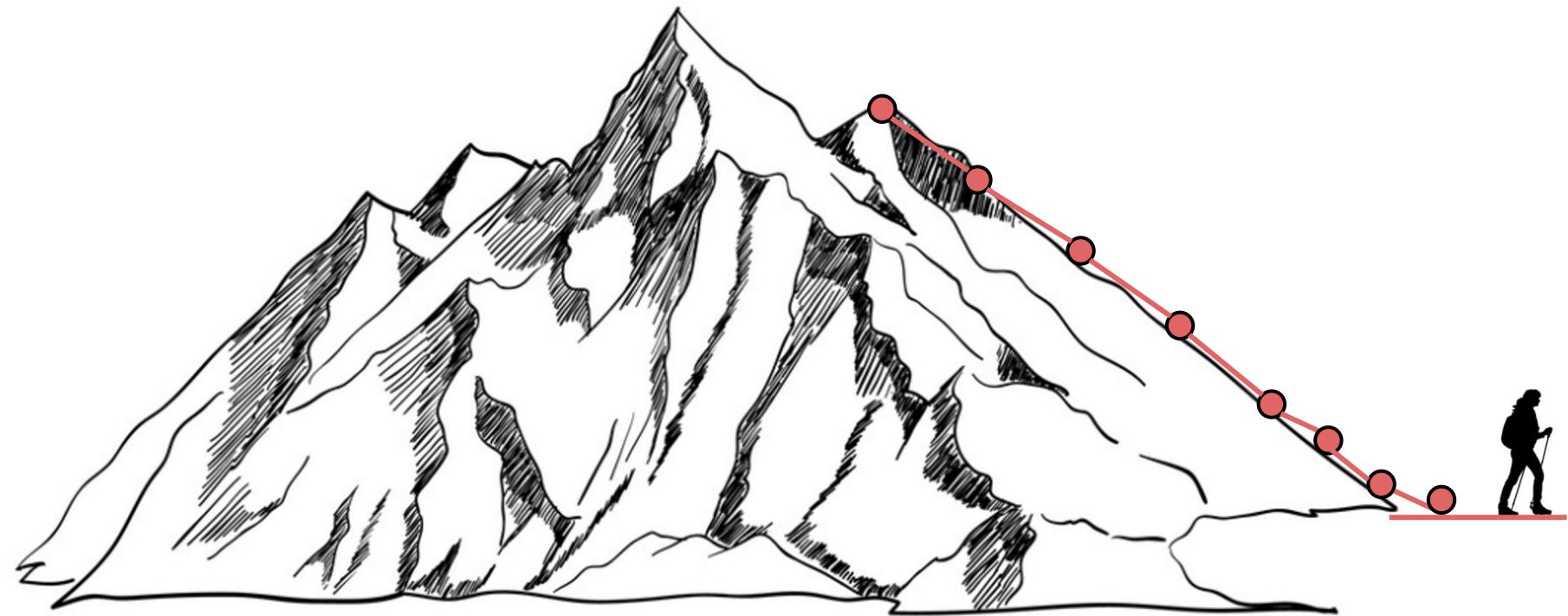
# Linear Regression

- Common mountain analogy



# Linear Regression

- Common mountain analogy

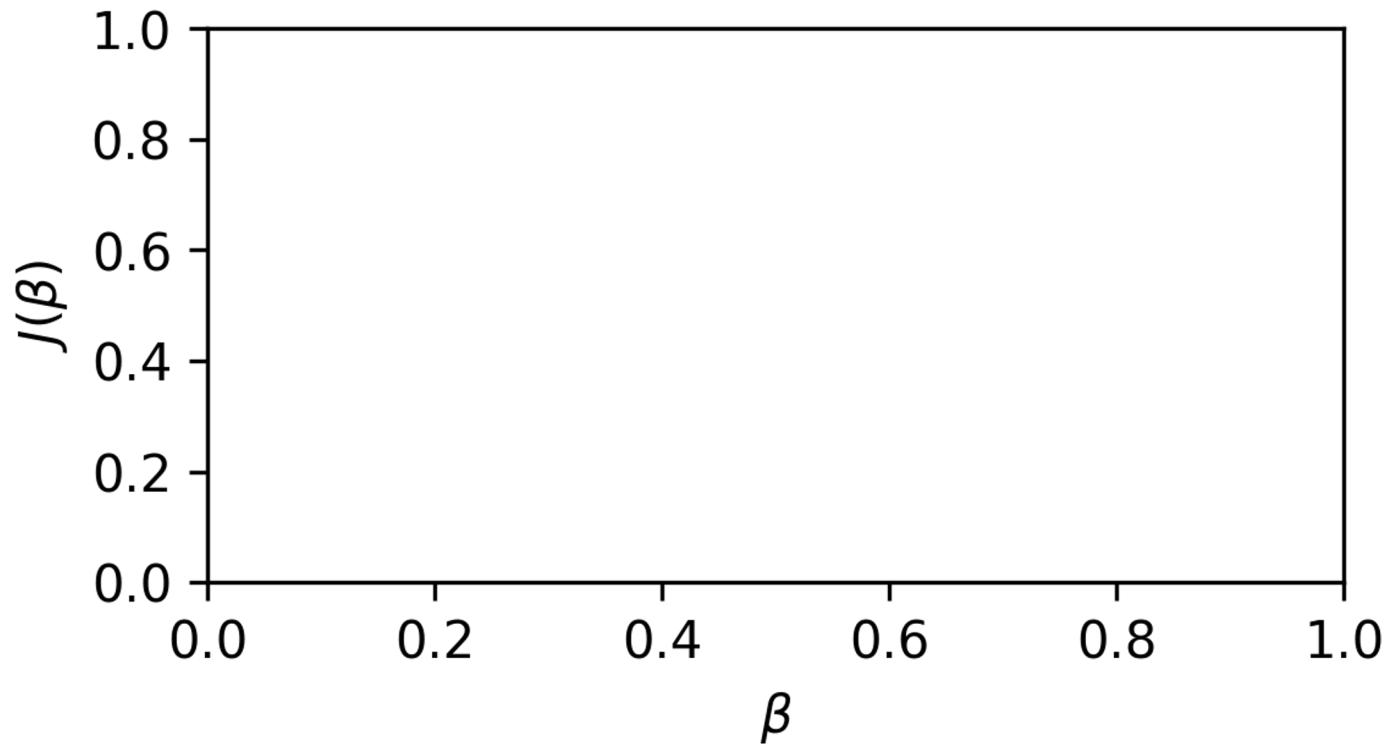


# Linear Regression

- This is exactly what gradient descent does!
- It even looks similar for the case of a single coefficient search.

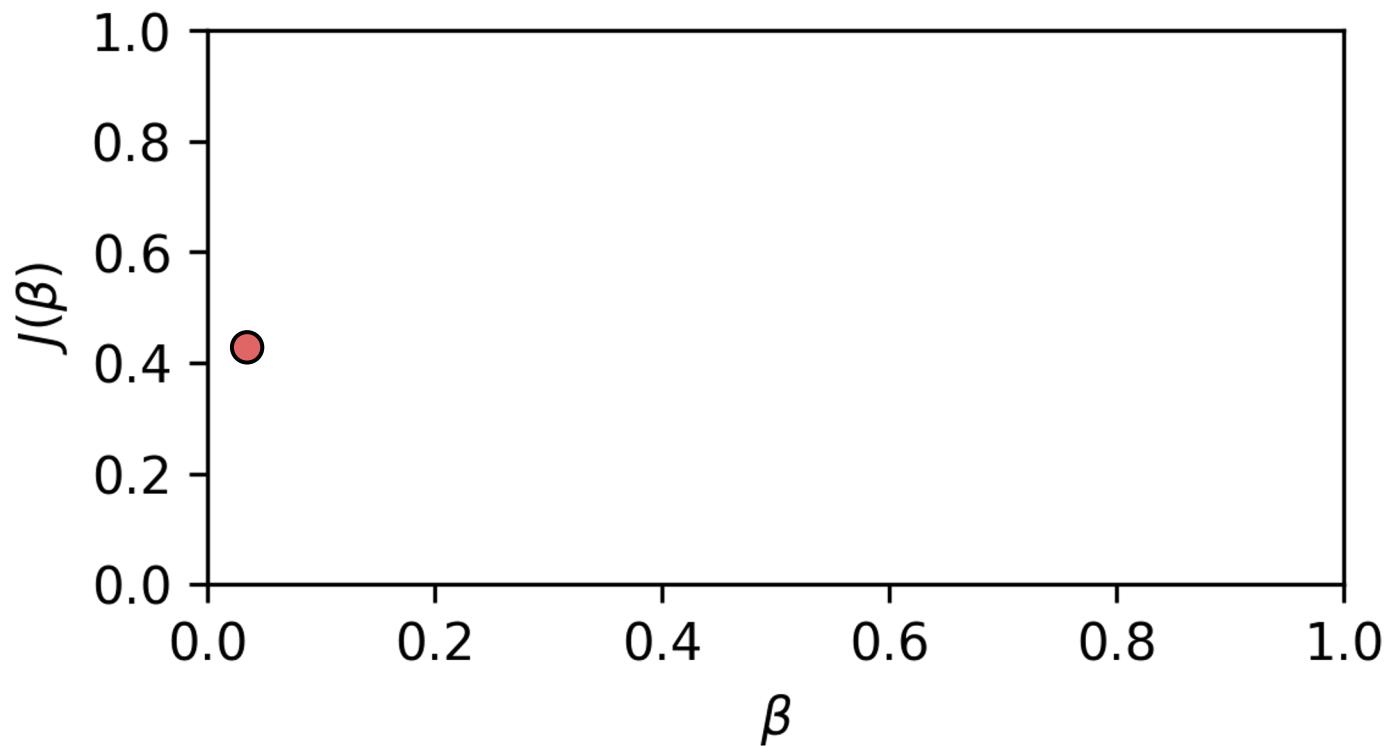
# Linear Regression

- 1 dimensional cost function (single Beta)



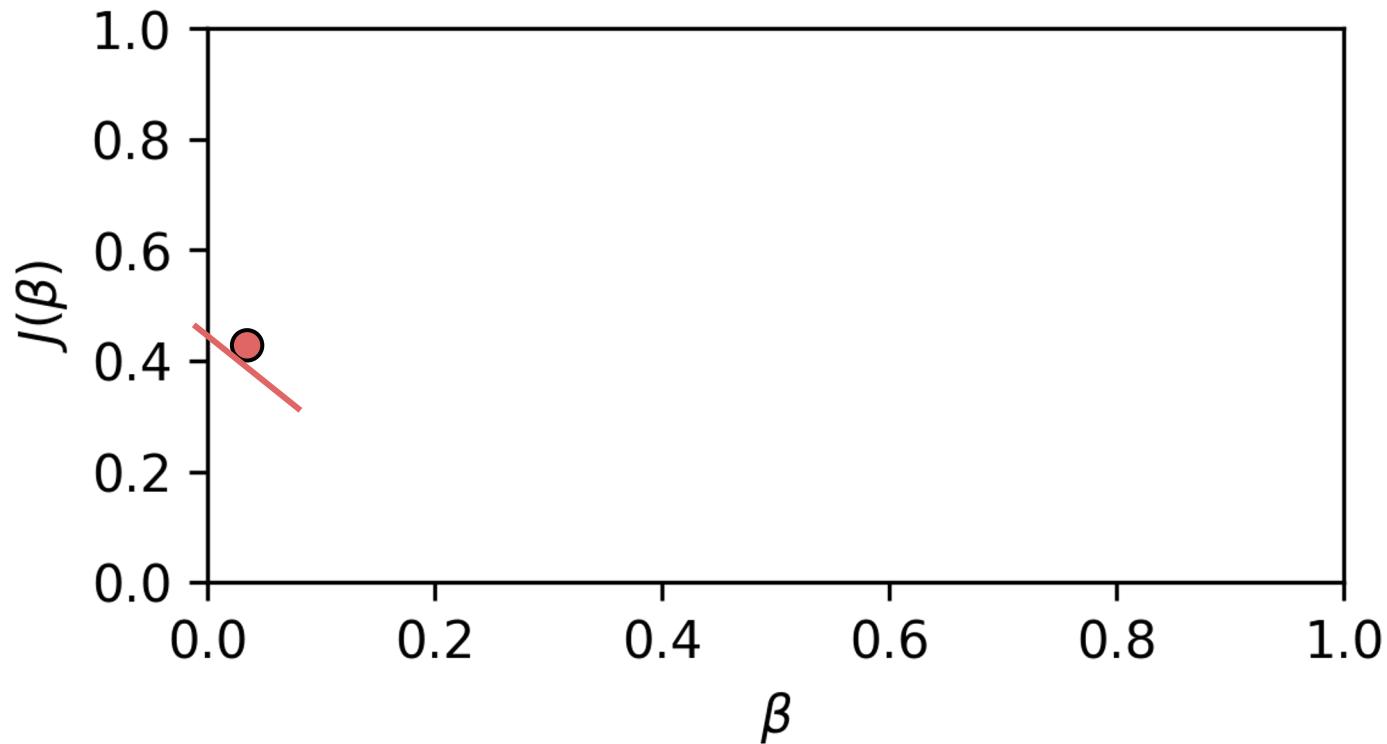
# Linear Regression

- Choose a starting point



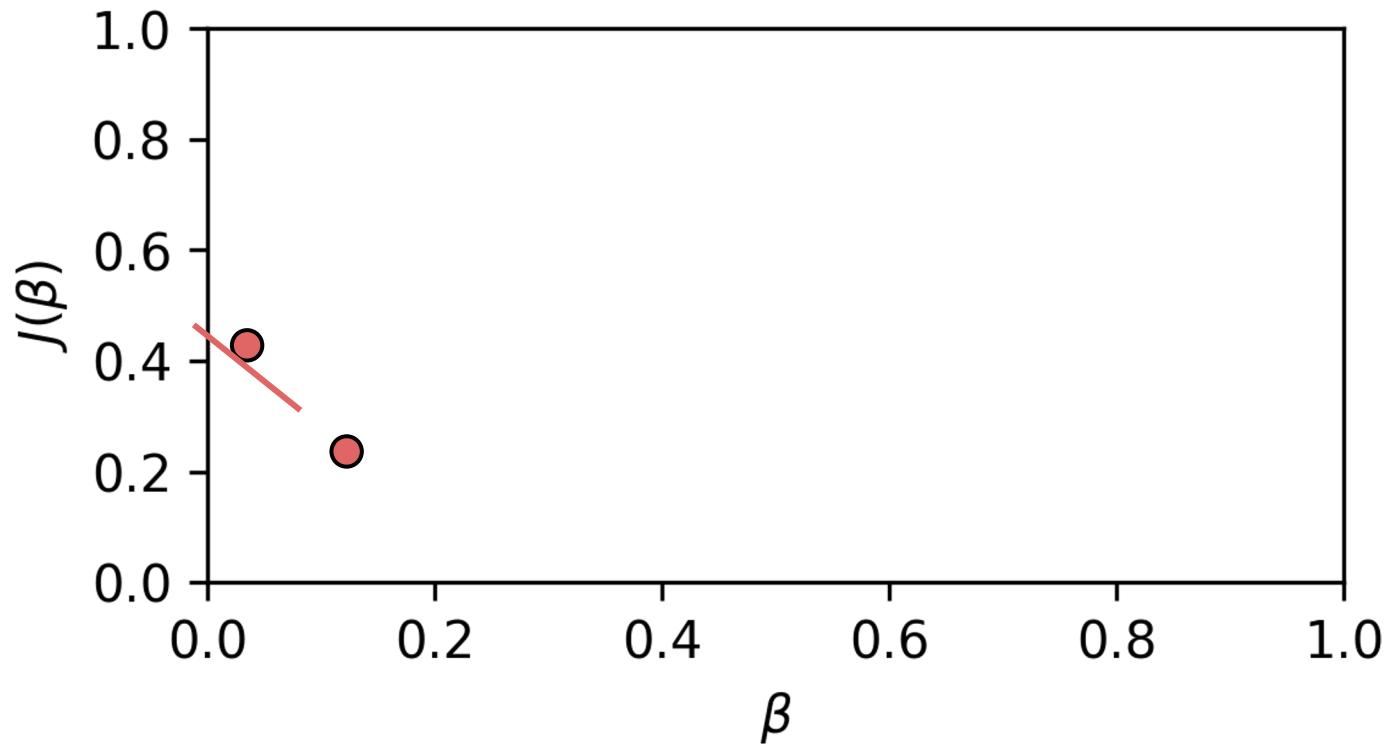
# Linear Regression

- Calculate gradient at that point



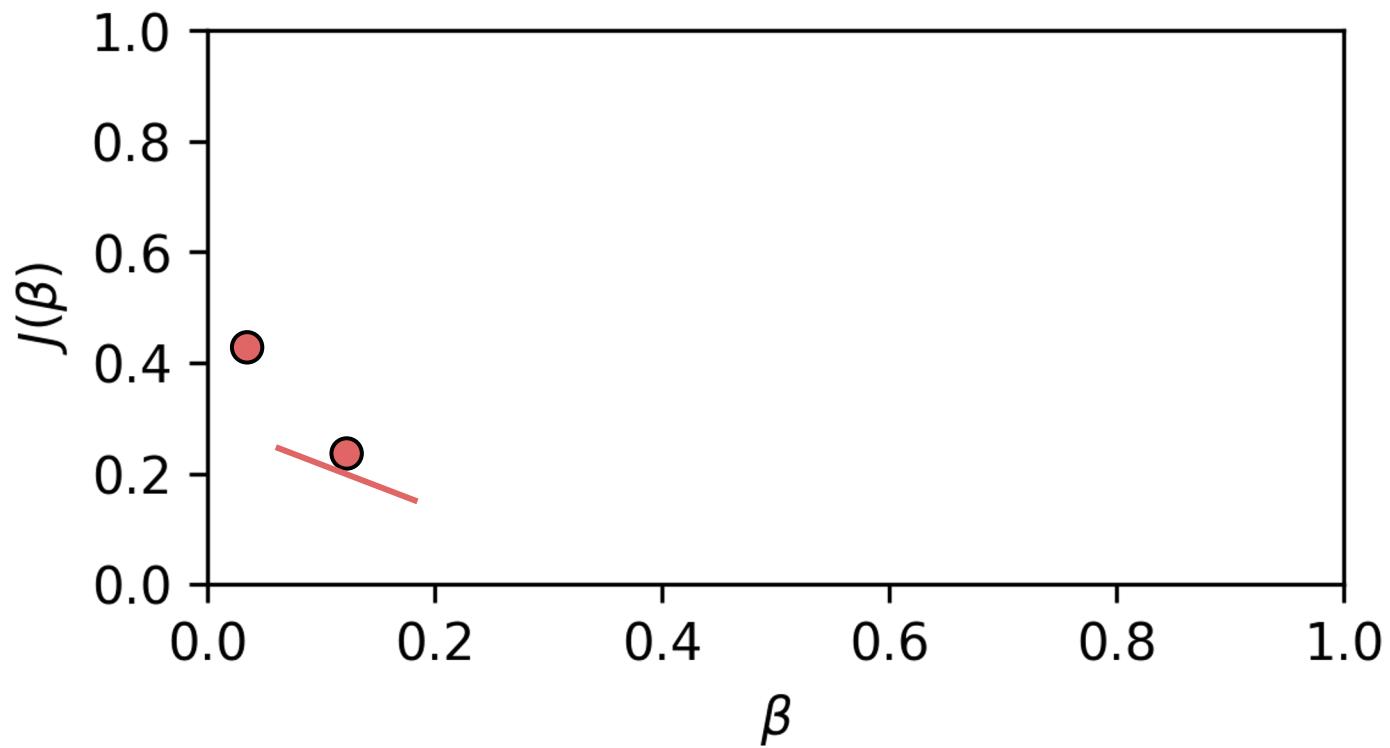
# Linear Regression

- Step forward proportional to **negative** gradient



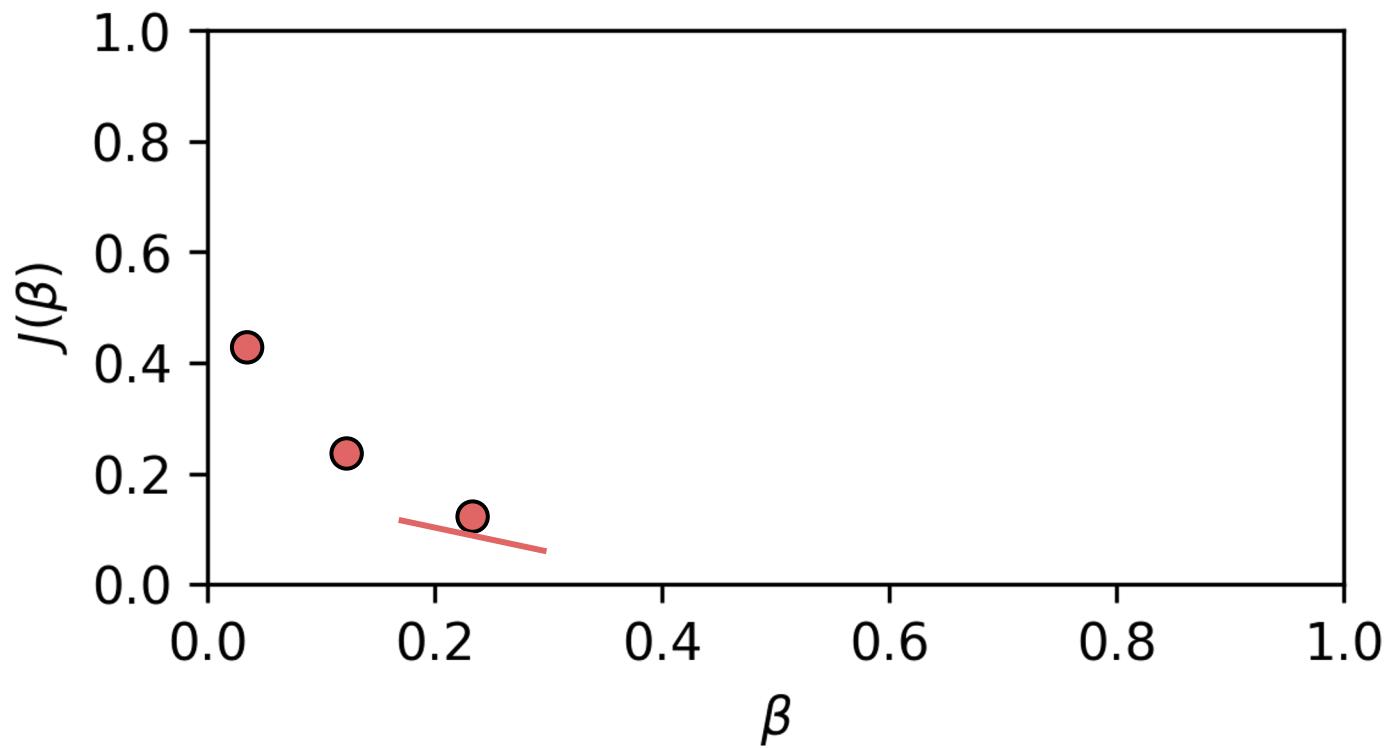
# Linear Regression

- Repeat the steps



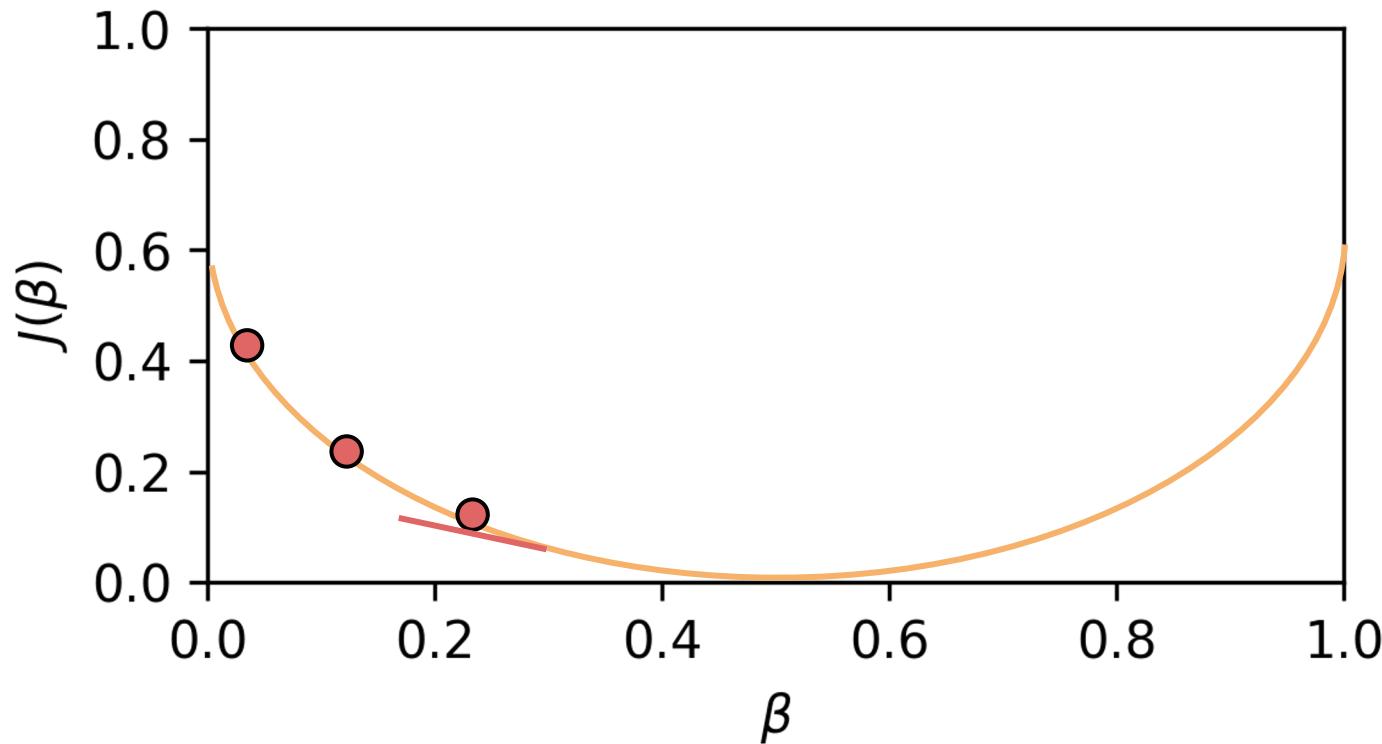
# Linear Regression

- Repeat the steps



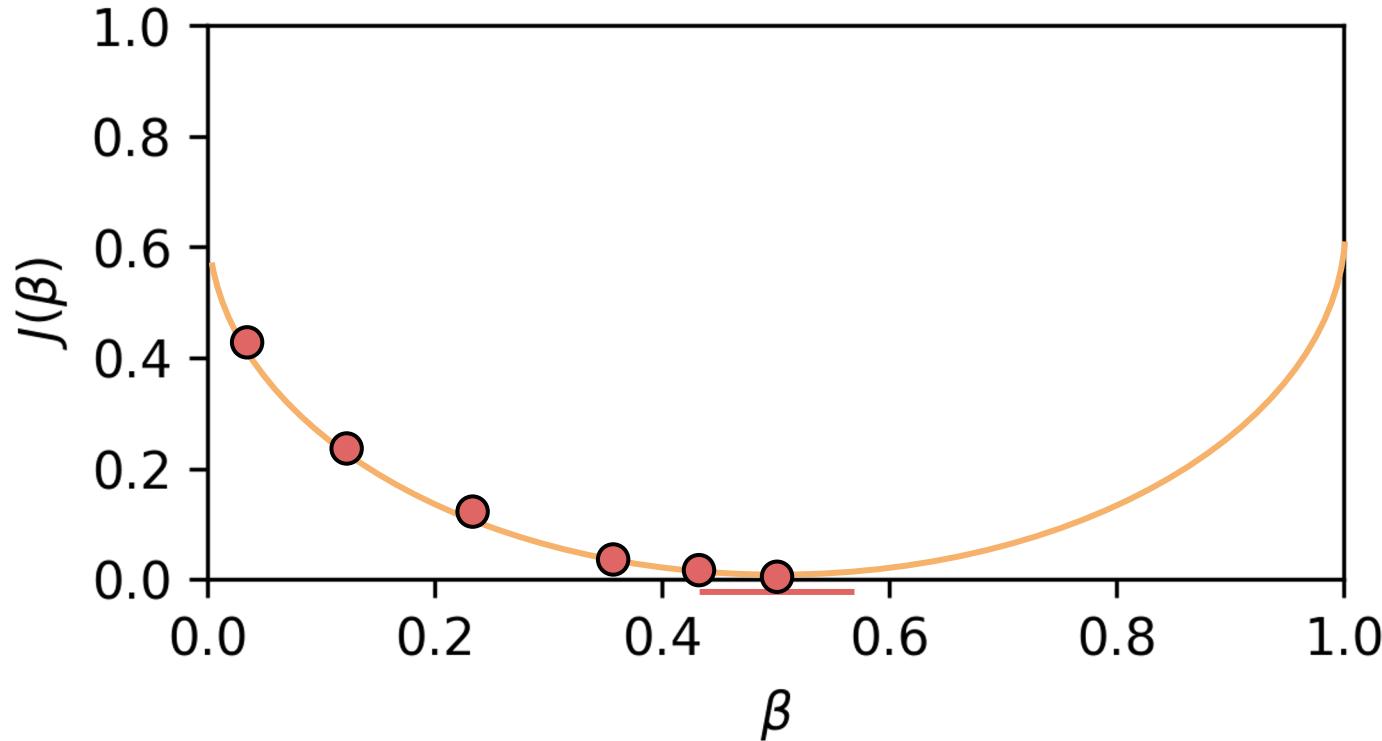
# Linear Regression

- Note how we are essentially mapping the gradient!



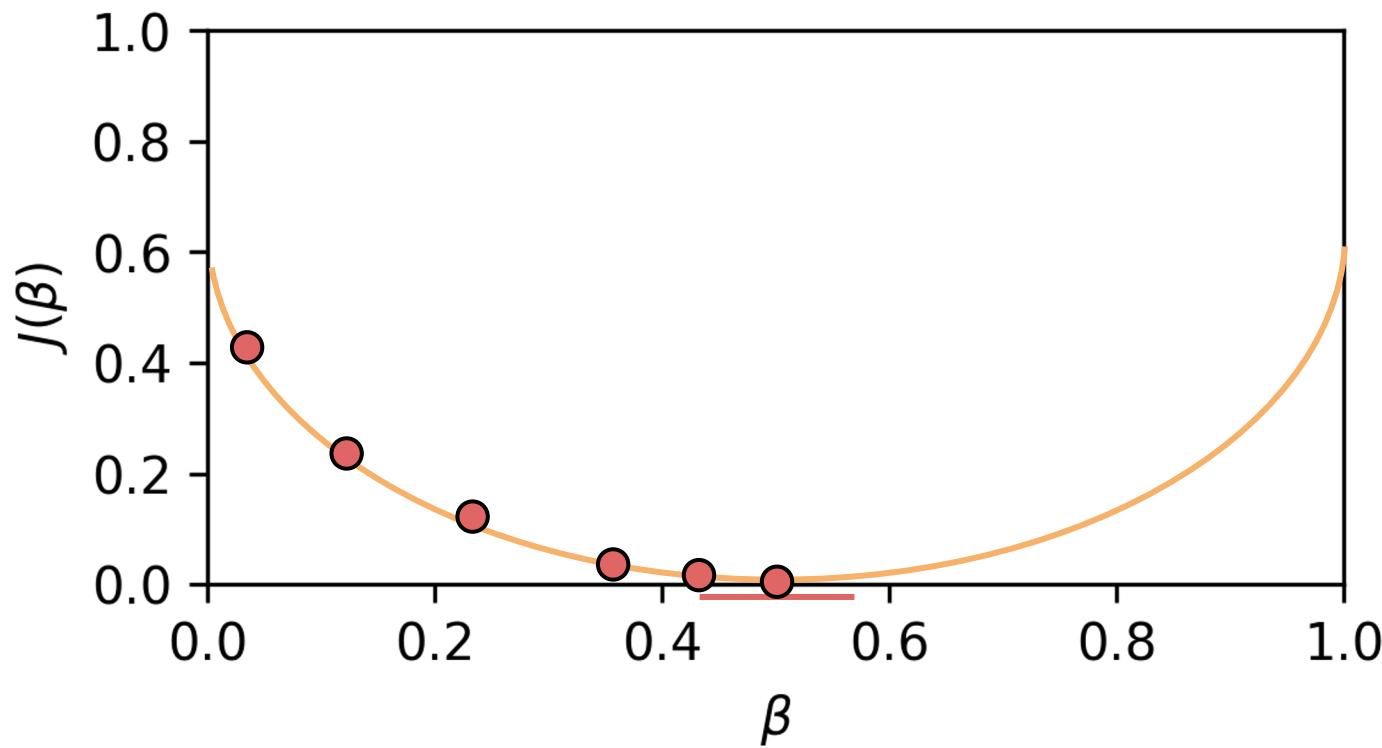
# Linear Regression

- Eventually we will find the Beta that minimizes the cost function!



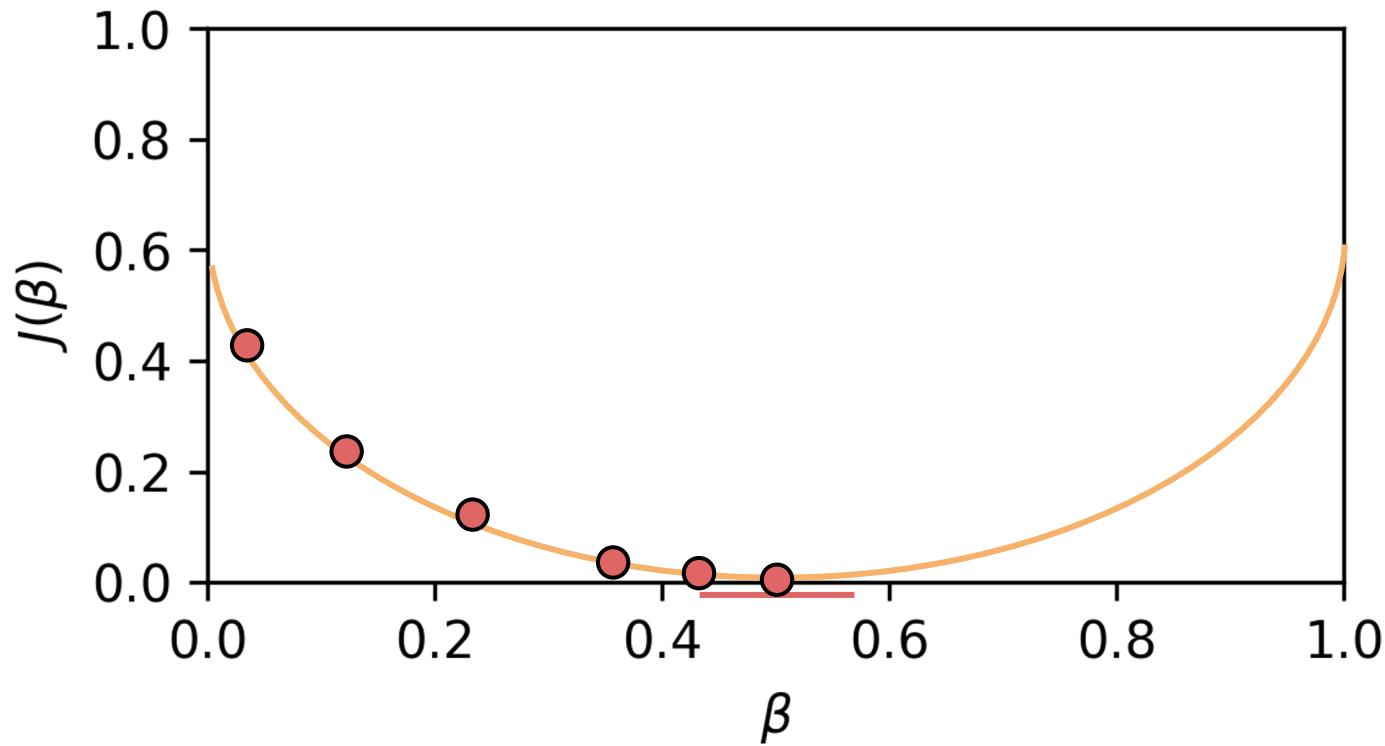
# Linear Regression

- Steps are proportional to negative gradient!



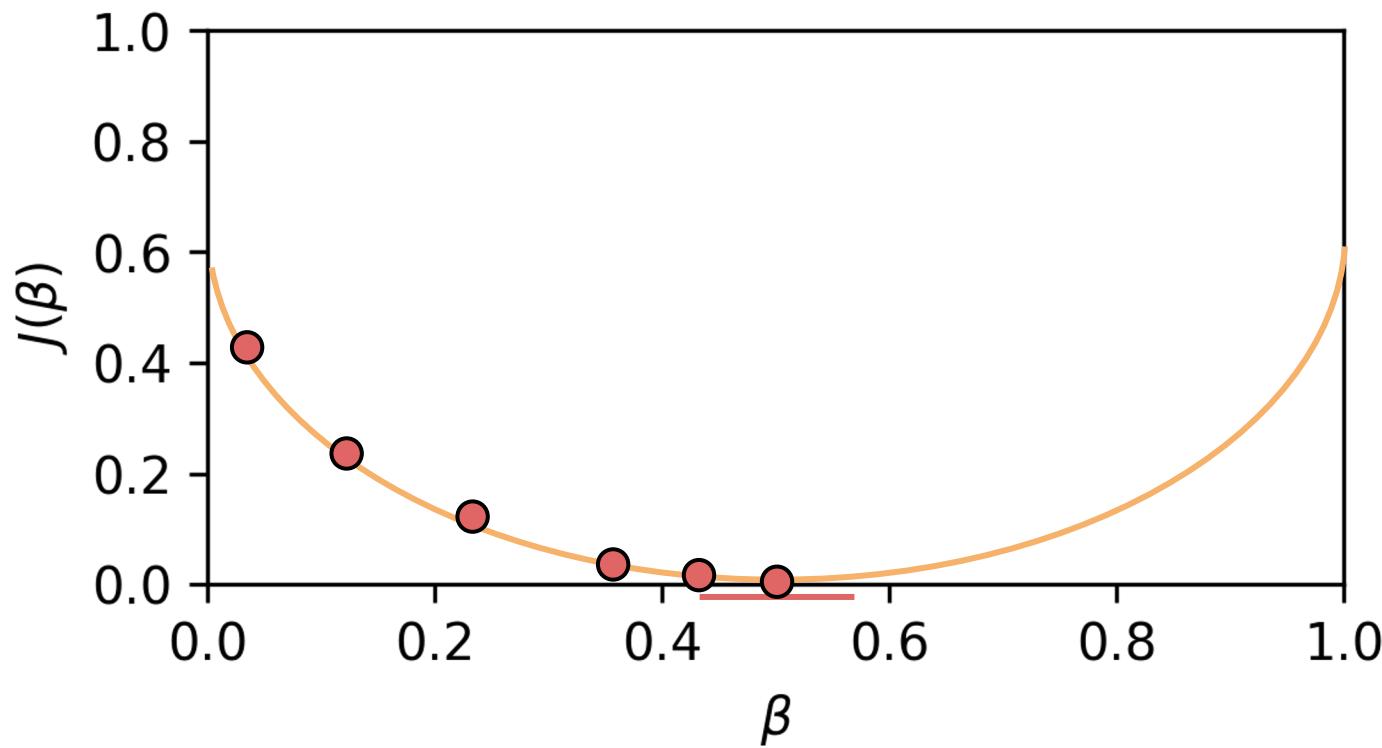
# Linear Regression

- Steeper gradient at start gives larger steps.



# Linear Regression

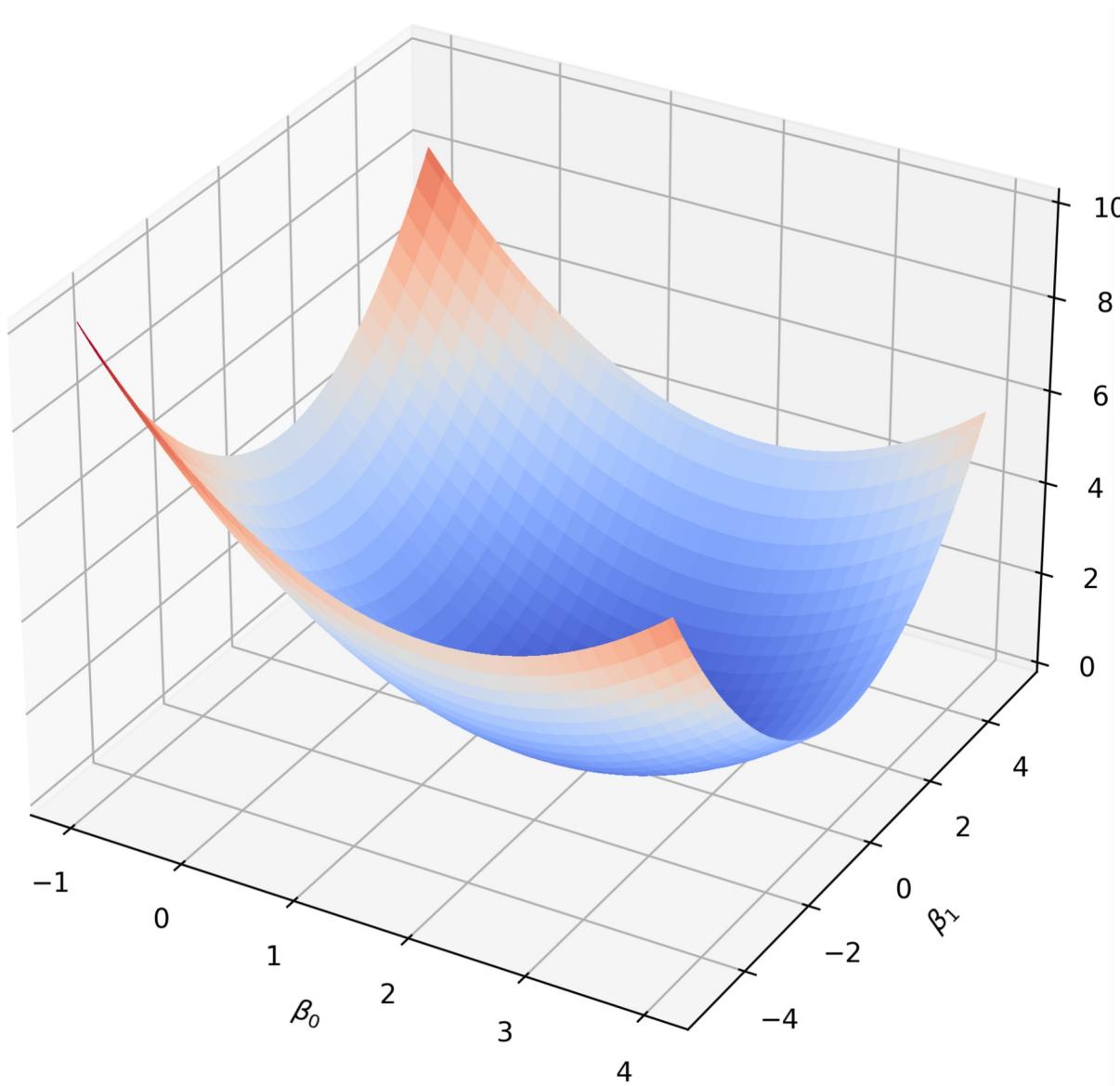
- Smaller gradient at end gives smaller steps.



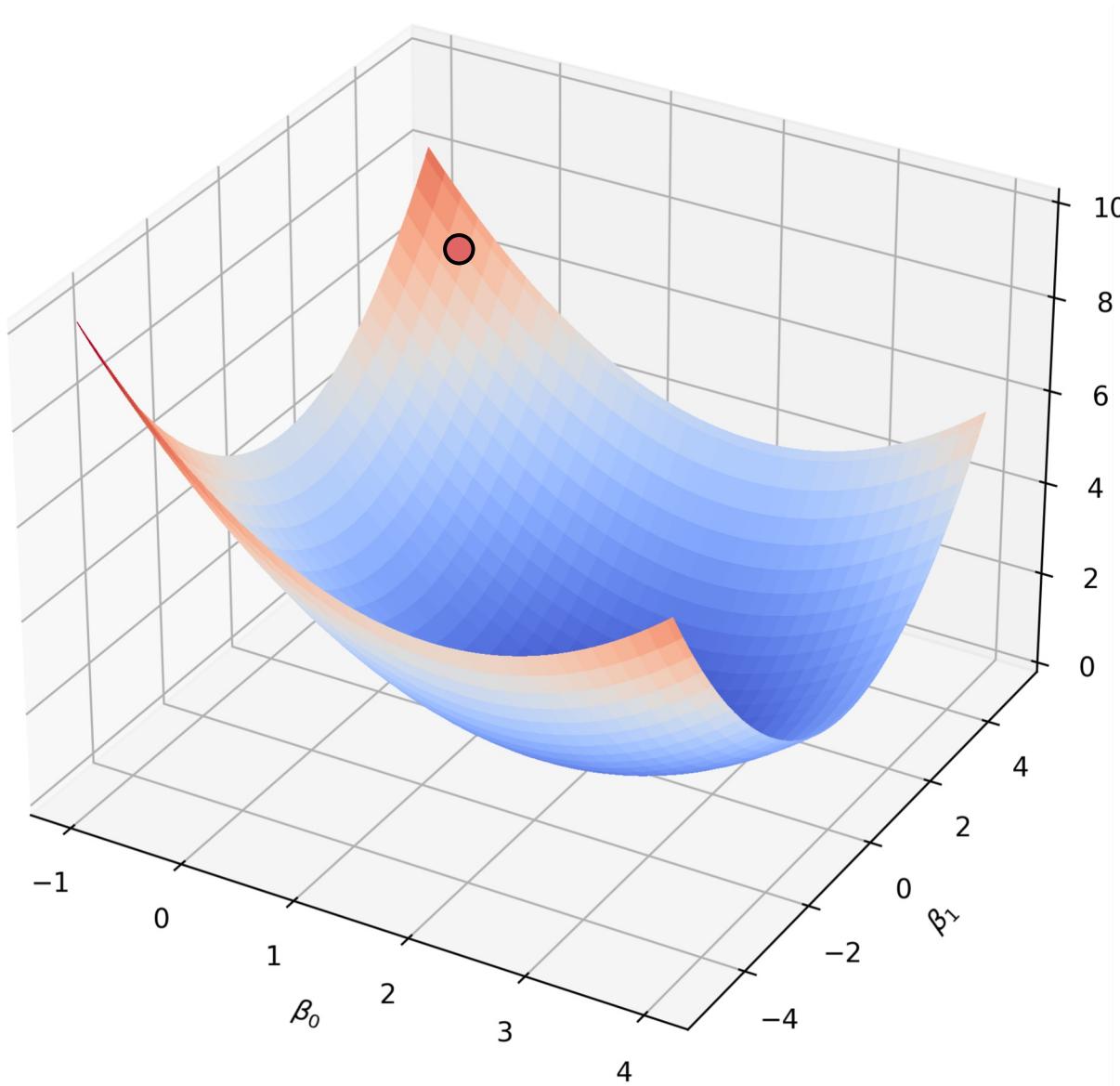
# Linear Regression

- To further understand this, let's visualize this gradient descent search for two Beta values.
- Process is still the same:
  - Calculate gradient at point.
  - Move in a step size proportional to negative gradient.
  - Repeat until minimum is found.

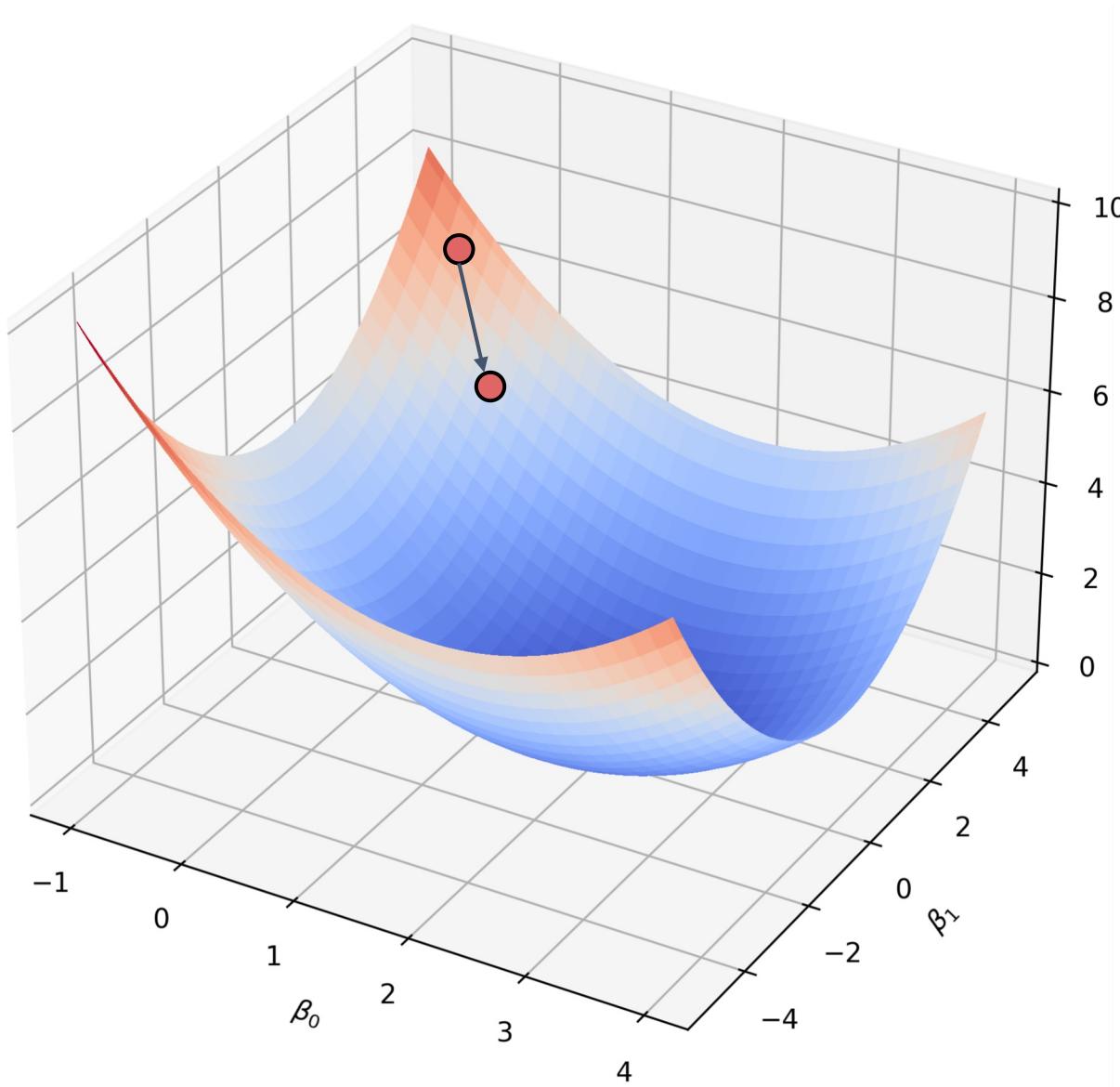
# Linear Regression



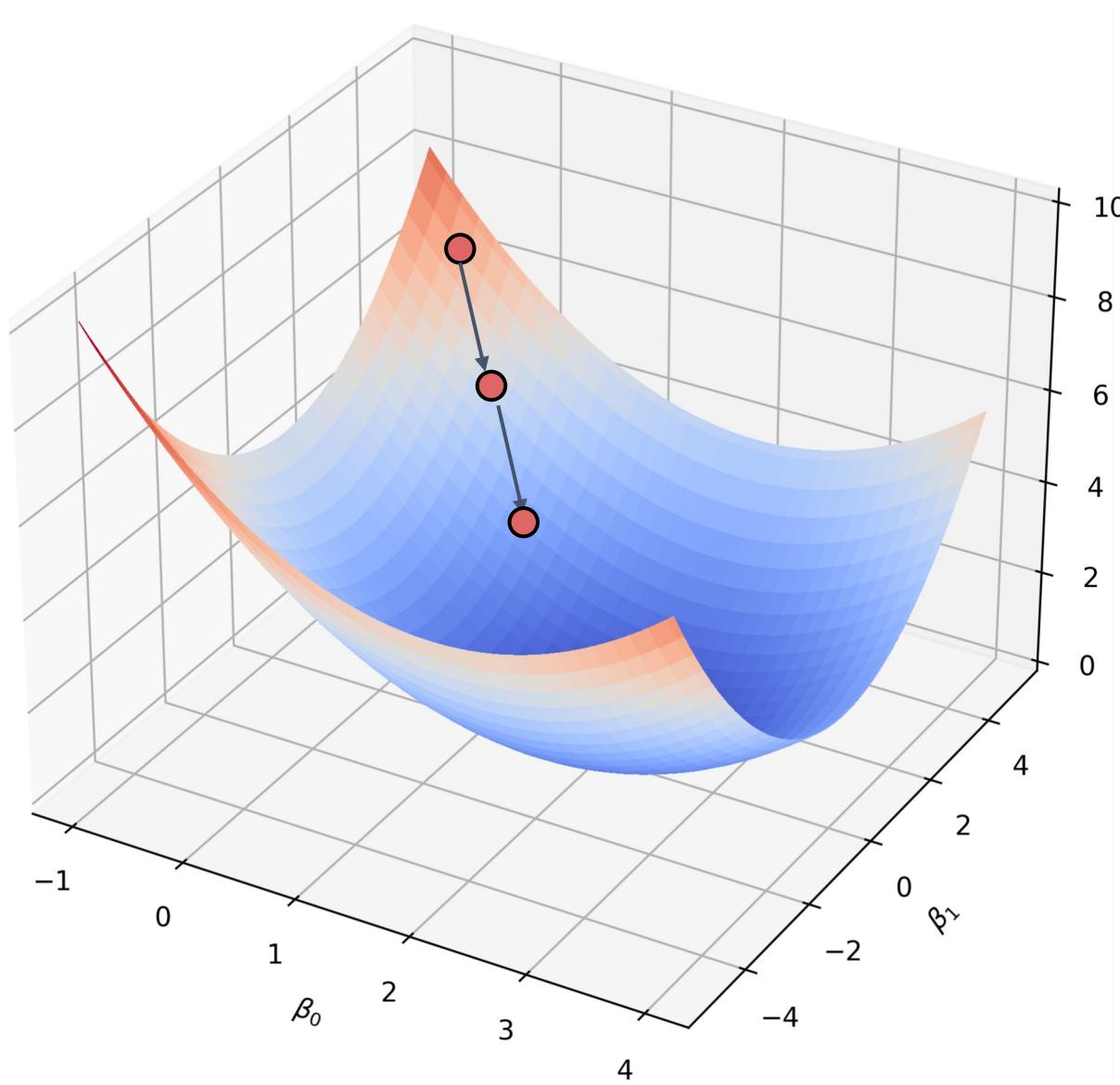
# Linear Regression



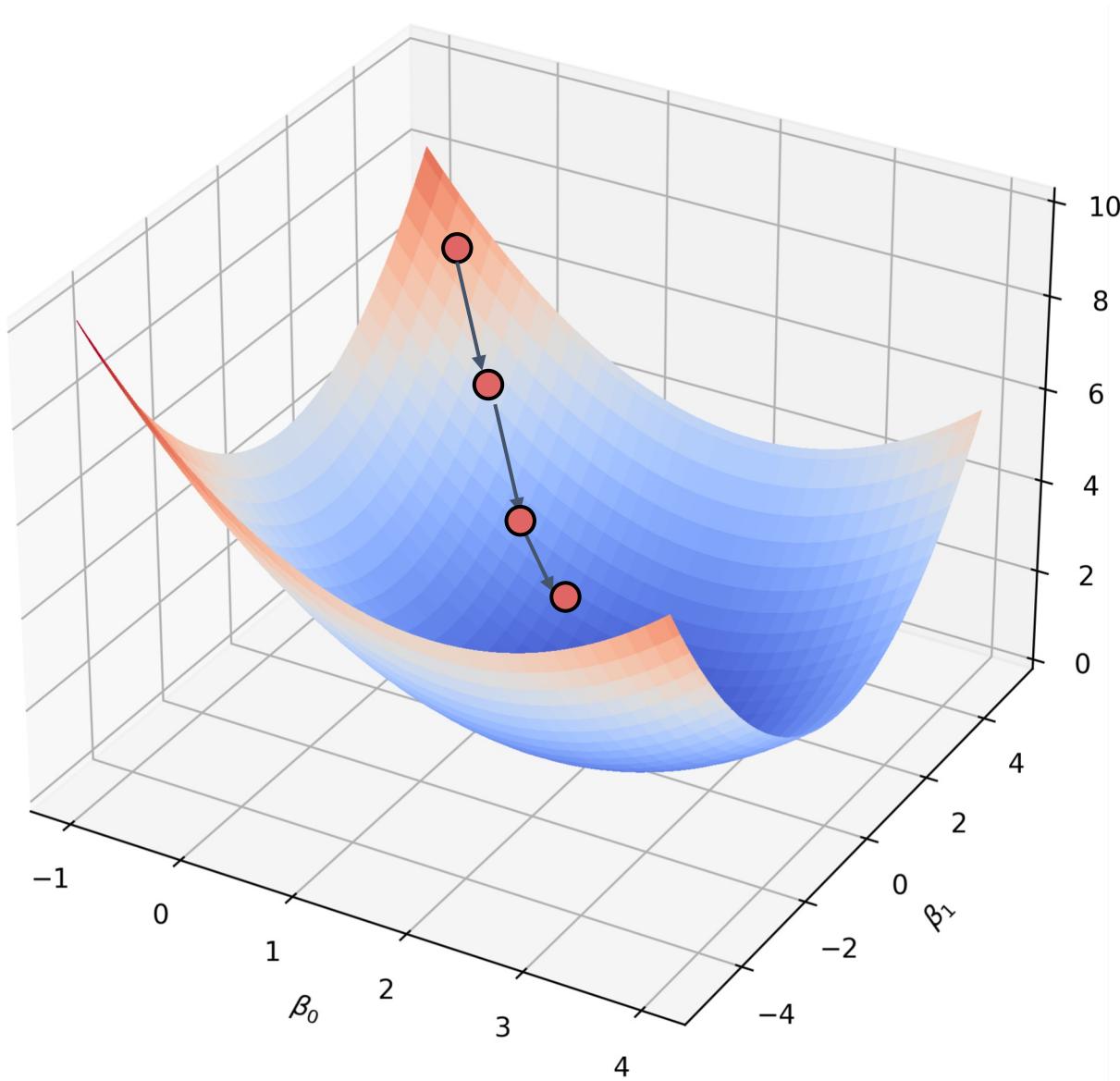
# Linear Regression



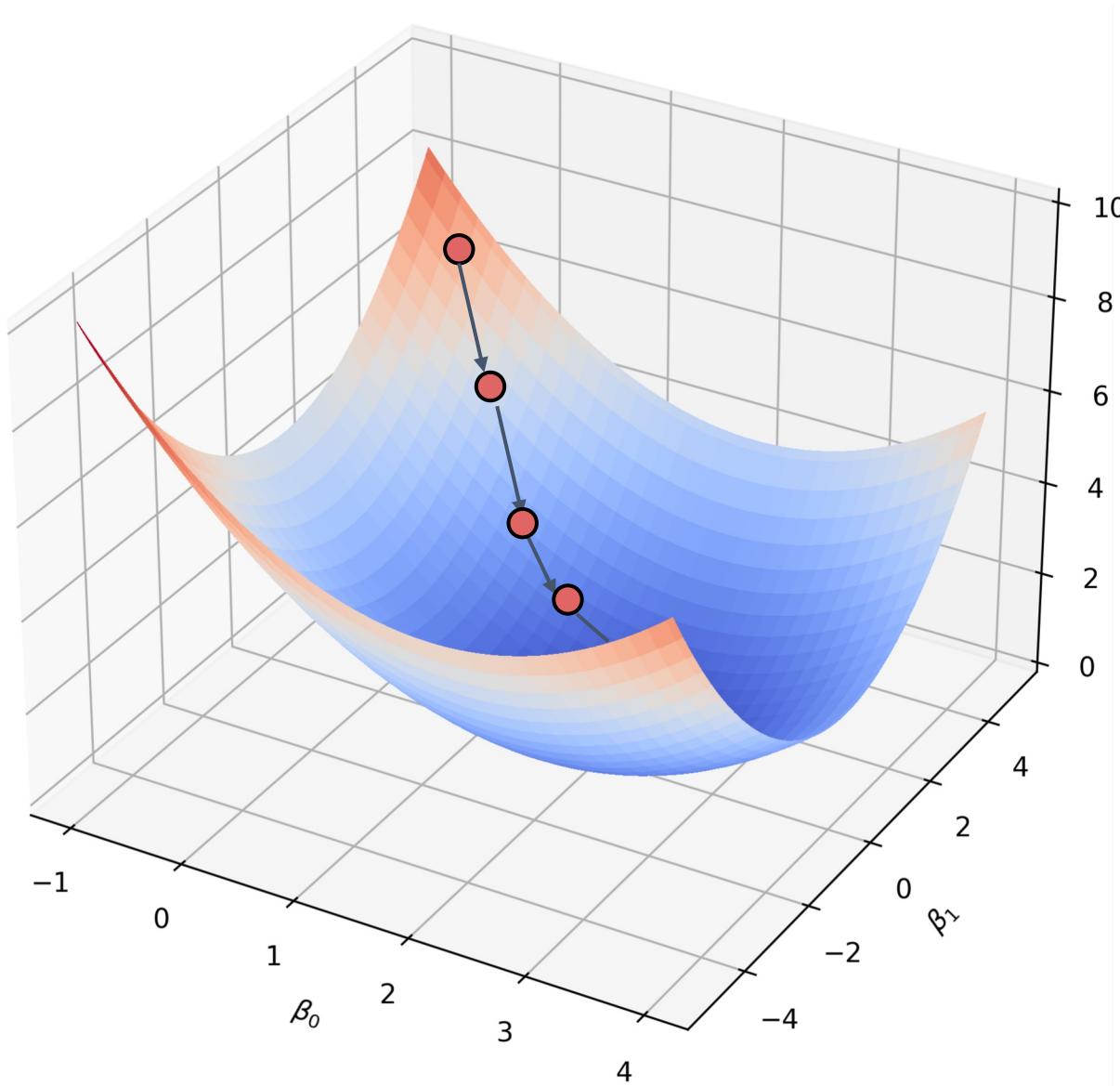
# Linear Regression



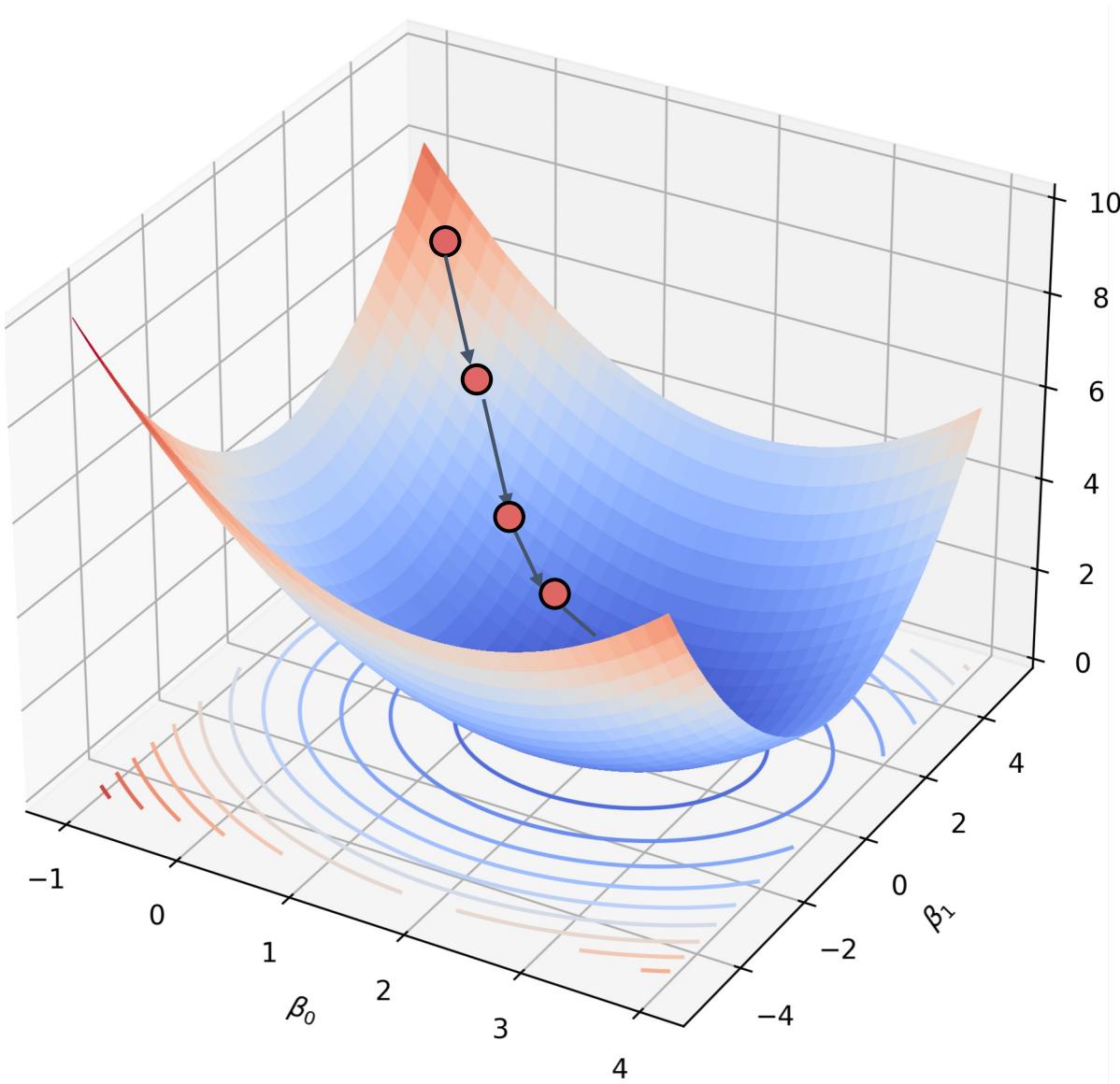
# Linear Regression



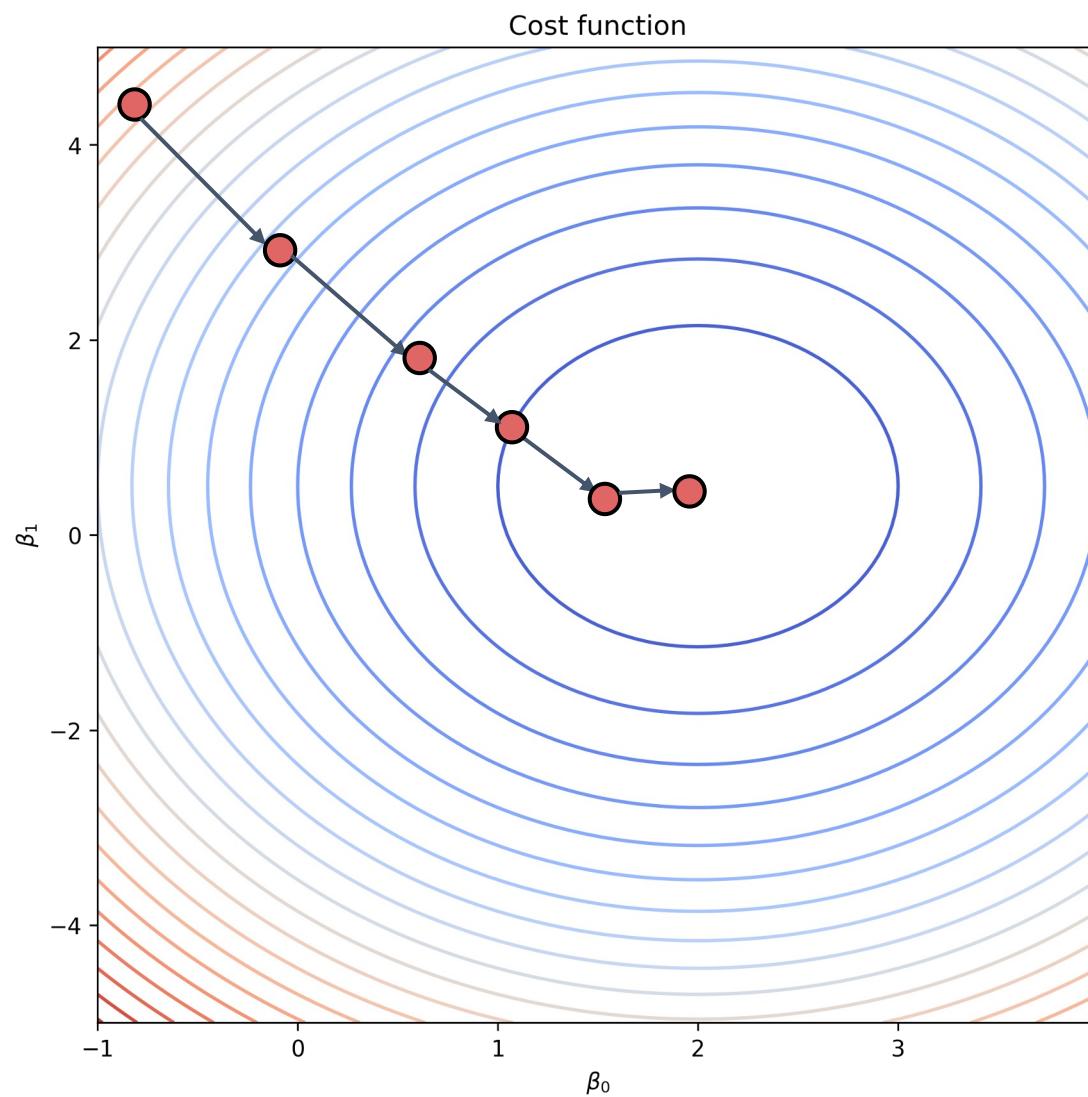
# Linear Regression



# Linear Regression



# Linear Regression



# Linear Regression

- Finally! We can now leverage all our computational power to find optimal Beta coefficients that minimize the cost function producing the line of best fit!
- We are now ready to code out Linear Regression!



# Scikit-Learn Overview

---

# Supervised Machine Learning Process

- Recall that we will perform a Train | Test split for supervised learning.



**TRAIN**

**TEST**

Area m <sup>2</sup>	Bedrooms	Bathroom	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

# Supervised Machine Learning Process

- Scikit-Learn easily does this split (as well as more advanced cross-validation)



**TRAIN**

**TEST**

Area m <sup>2</sup>	Bedrooms	Bathroom	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

# Scikit-Learn

```
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

# Supervised Machine Learning Process

- Also recall that we want to compare predictions to the y test labels.



Prediction s	Area m <sup>2</sup>	Bedrooms	Bathroom s	Price
\$410,000	180	1	1	\$400,000
\$540,000	210	2	2	\$550,000

# Scikit-Learn

```
from sklearn.model_family import ModelAlgo
```

# Scikit-Learn

```
from sklearn.model_family import ModelAlgo  
mymodel = ModelAlgo(param1, param2)
```

# Scikit-Learn

```
from sklearn.model_family import ModelAlgo  
mymodel = ModelAlgo(param1,param2)  
mymodel.fit(X_train,y_train)
```

# Scikit-Learn

```
from sklearn.model_family import ModelAlgo  
mymodel = ModelAlgo(param1,param2)  
mymodel.fit(X_train,y_train)  
predictions = mymodel.predict(X_test)
```

# Scikit-Learn

```
from sklearn.model_family import ModelAlgo  
mymodel = ModelAlgo(param1,param2)  
mymodel.fit(X_train,y_train)  
predictions = mymodel.predict(X_test)
```

```
from sklearn.metrics import error_metric
```

# Scikit-Learn

```
from sklearn.model_family import ModelAlgo  
mymodel = ModelAlgo(param1,param2)  
mymodel.fit(X_train,y_train)  
predictions = mymodel.predict(X_test)
```

```
from sklearn.metrics import error_metric  
performance = error_metric(y_test,predictions)
```

# Performance Evaluation

---

Regression Metrics

# Evaluating Regression

- Let's discuss some of the most common evaluation metrics for regression:
  - Mean Absolute Error
  - Mean Squared Error
  - Root Mean Square Error

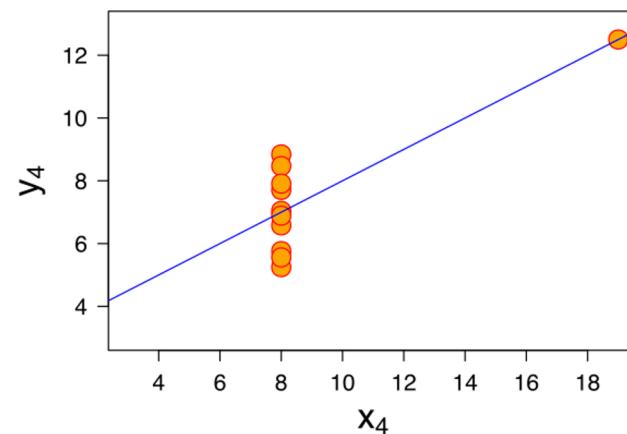
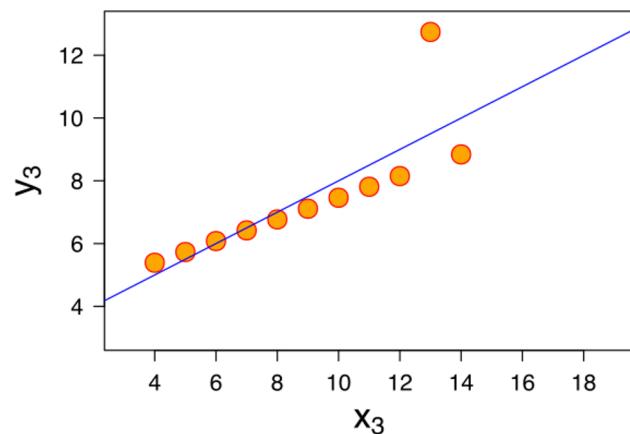
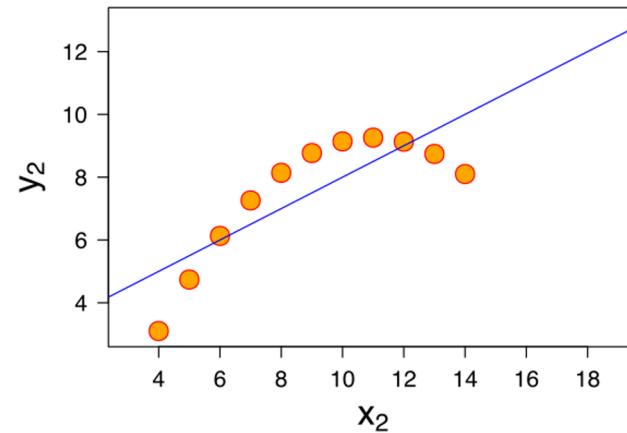
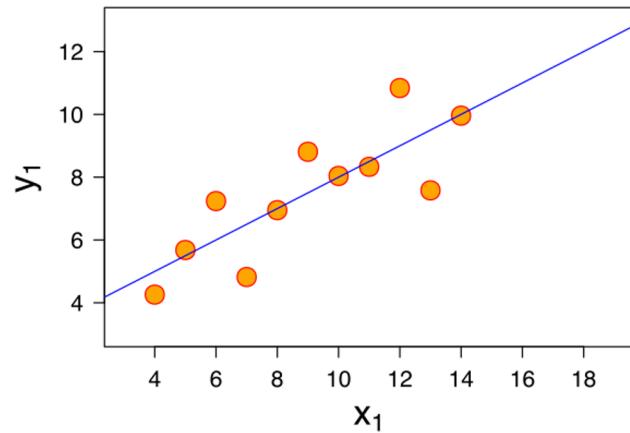
# Evaluating Regression

- Mean Absolute Error (MAE)
  - This is the mean of the absolute value of errors.
  - Easy to understand

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

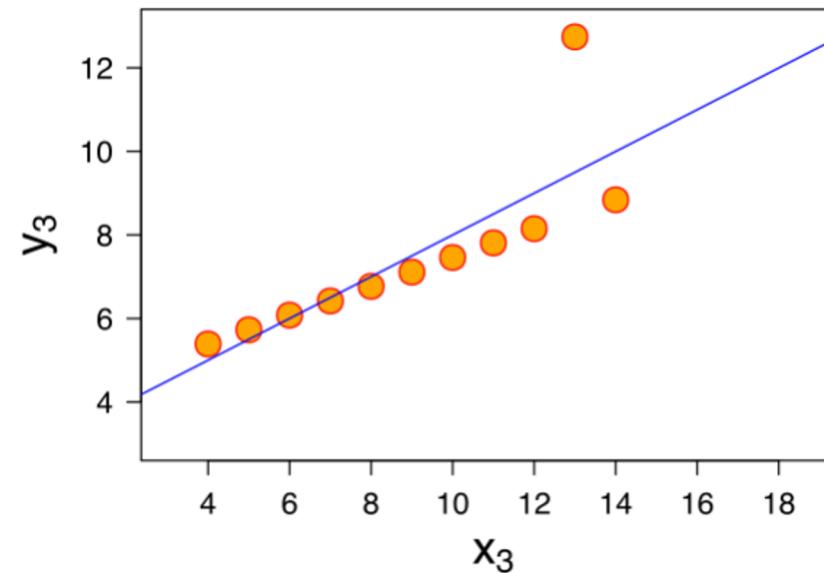
# Evaluating Regression

- MAE won't punish large errors however.



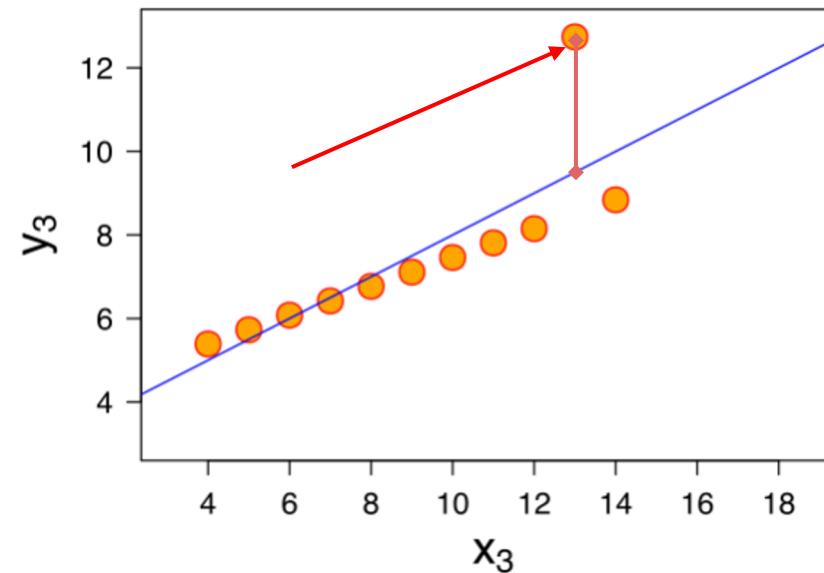
# Evaluating Regression

- MAE won't punish large errors however.



# Evaluating Regression

- We want our error metrics to account for these!



# Evaluating Regression

- Mean Squared Error (MSE)
  - Issue with MSE:
    - Different units than  $y$ .
    - It reports units of  $y$  squared!

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

# Evaluating Regression

- Root Mean Square Error (RMSE)
  - This is the root of the mean of the squared errors.
  - Most popular (has same units as  $y$ )

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

# Machine Learning

- Most common question from students:
  - “*What is a good value for RMSE?*”
  - Context is everything!
  - A RMSE of \$10 is fantastic for predicting the price of a house, but horrible for predicting the price of a candy bar!

# Machine Learning

- Compare your error metric to the average value of the label in your data set to try to get an intuition of its overall performance.
- Domain knowledge also plays an important role here!

# Machine Learning

- Context of importance is also necessary to consider.
  - We may create a model to predict how much medication to give, in which case small fluctuations in RMSE may actually be very significant.

# Machine Learning

- Context of importance is also necessary to consider.
  - If we create a model to try to improve on existing human performance, we would need some baseline RMSE to compare to.

# Evaluating Regression

- Let's quickly jump back to the notebook and calculate these metrics with SciKit-Learn!

# Evaluating Residuals

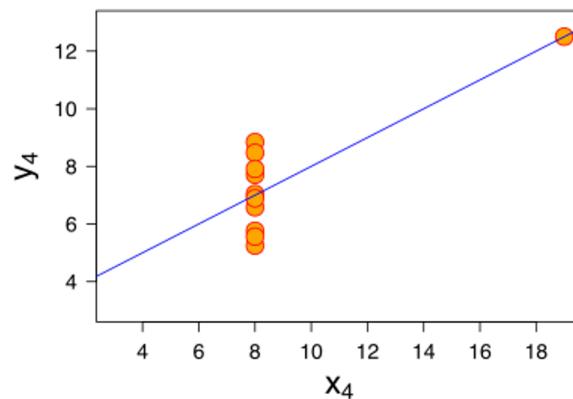
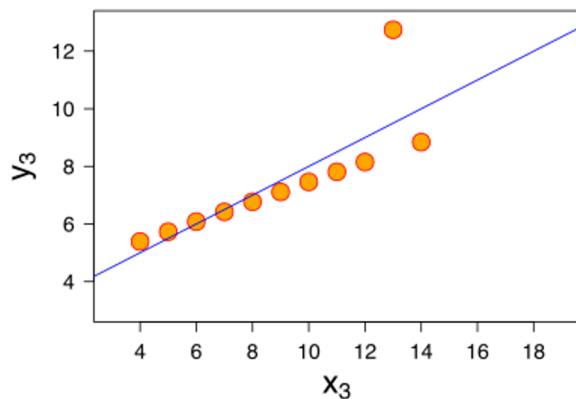
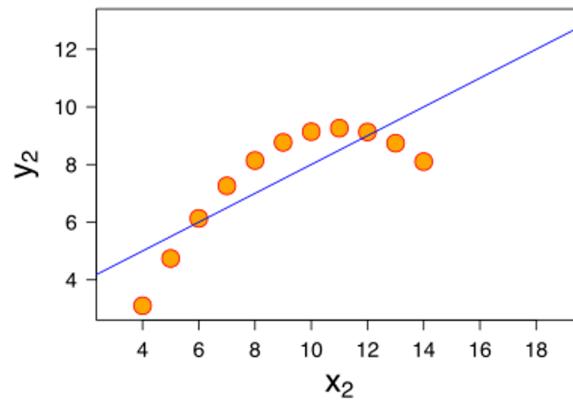
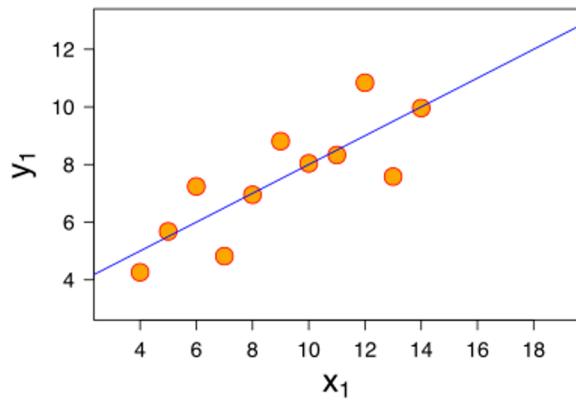
---

# Linear Regression

- Often for Linear Regression it is a good idea to separately evaluate residuals ( $y - \hat{y}$ ) and not just calculate performance metrics (e.g. RMSE).
- Let's explore why this is important...

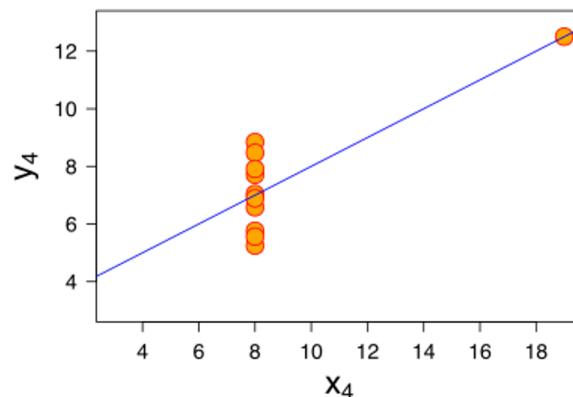
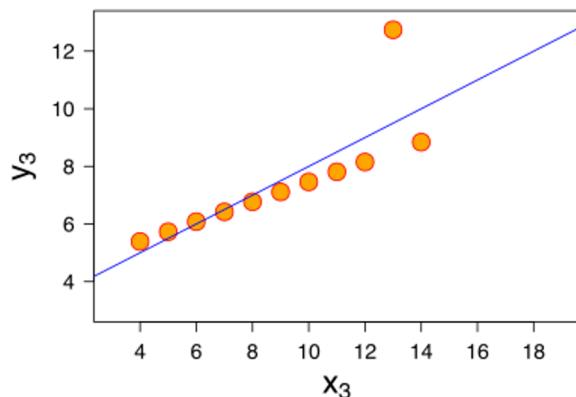
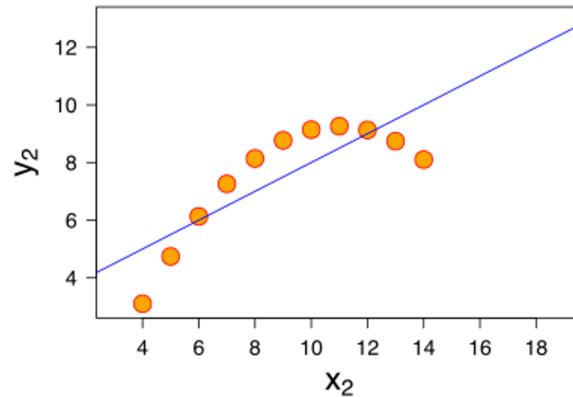
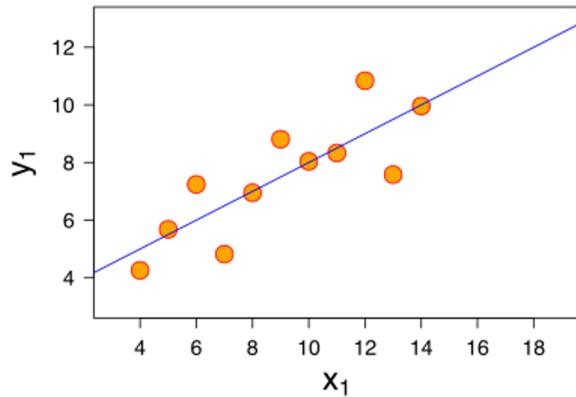
# Linear Regression

- Recall Anscombe's quartet:



# Linear Regression

- Clearly Linear Regression is not suitable!

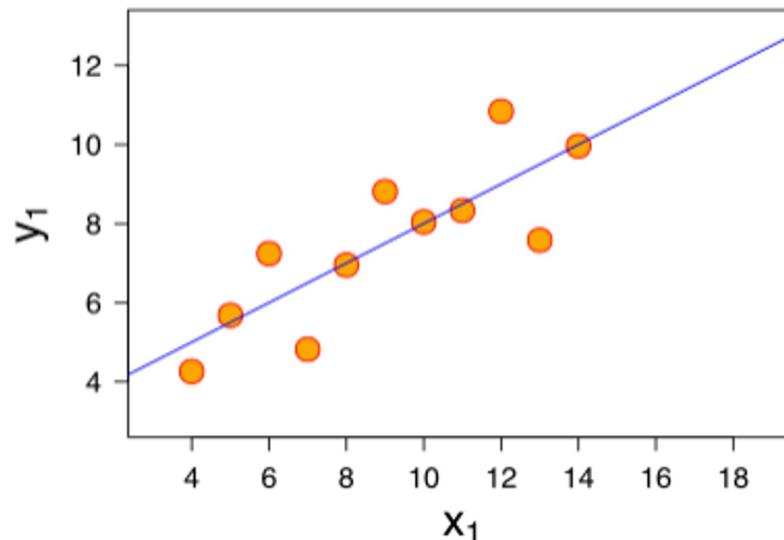


# Linear Regression

- But how can we tell if we're dealing with more than one x feature?
- We can not see this discrepancy of fit visually if we have multiple features!

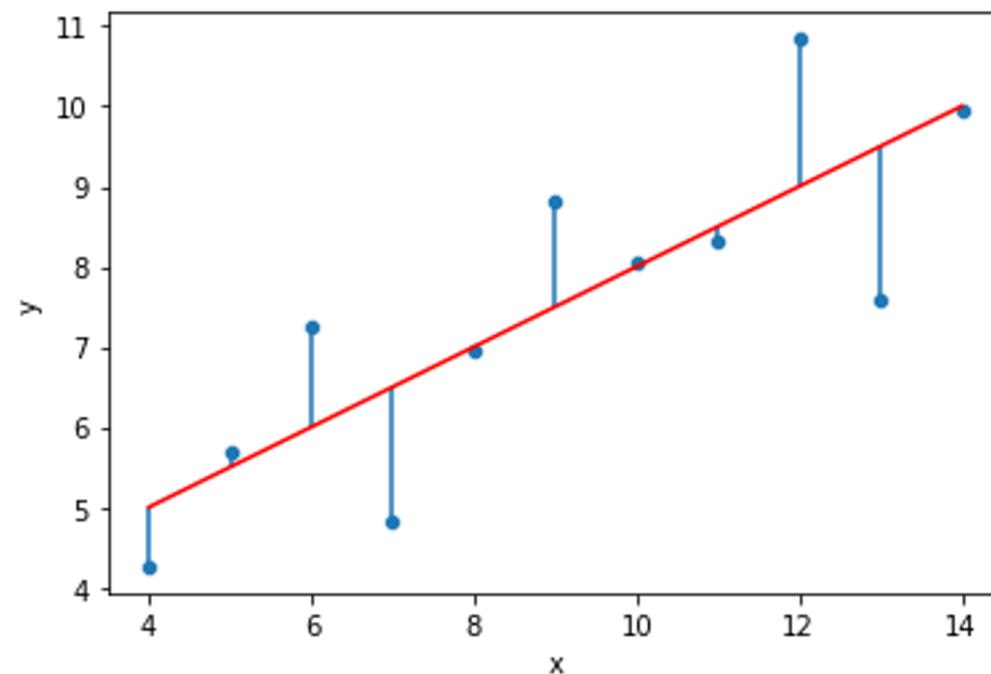
# Linear Regression

- What we could do is plot residual error against true y values.
- Consider an appropriate data set:



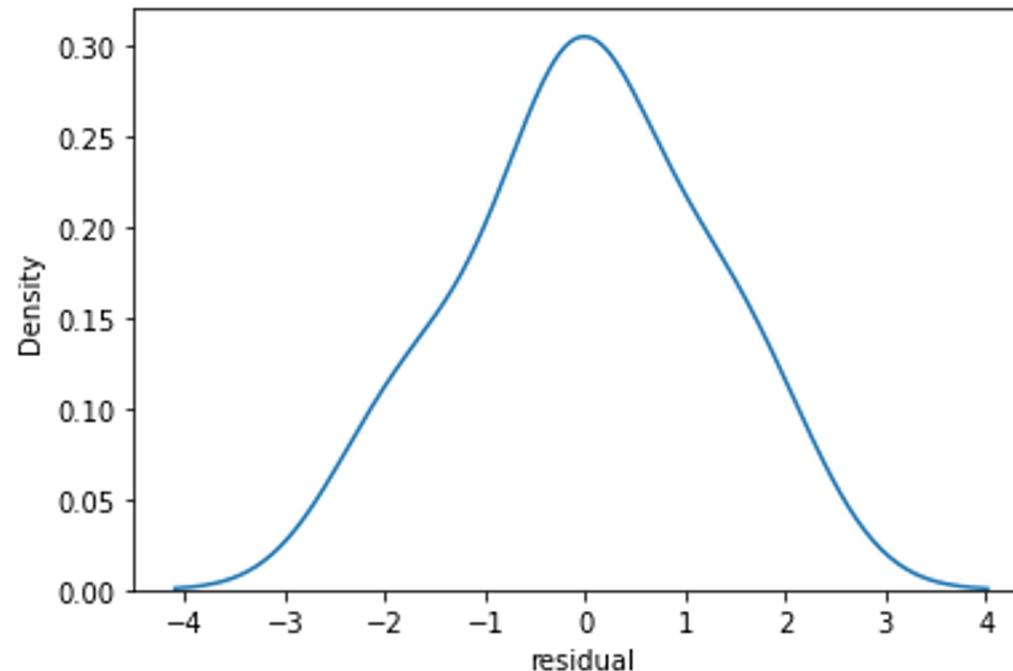
# Linear Regression

- The residual errors should be random and close to a normal distribution.



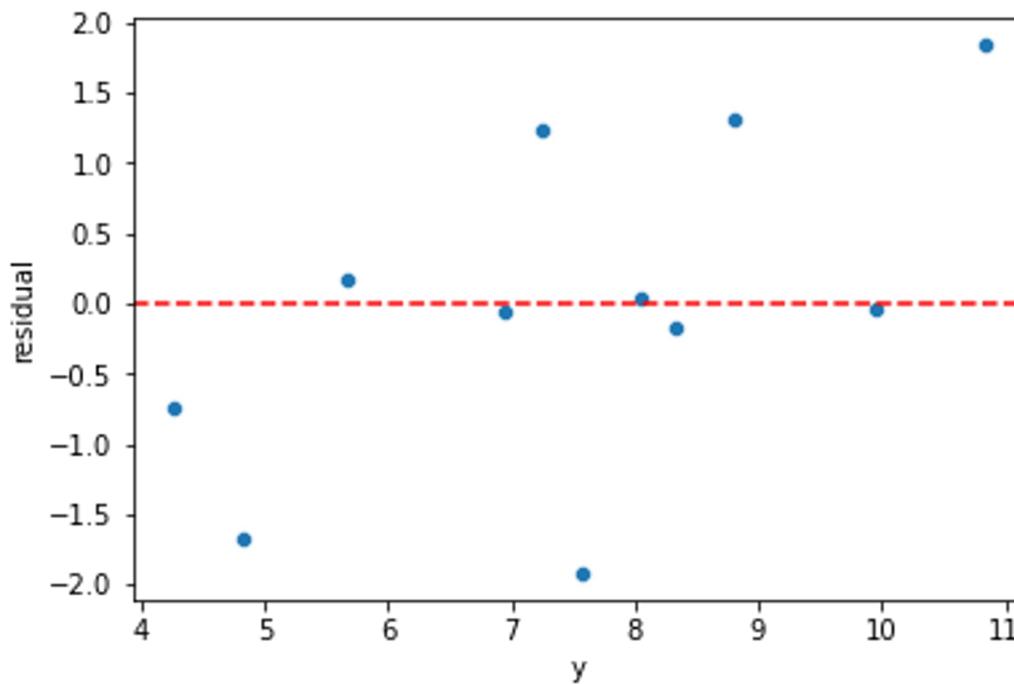
# Linear Regression

- The residual errors should be random and close to a normal distribution.



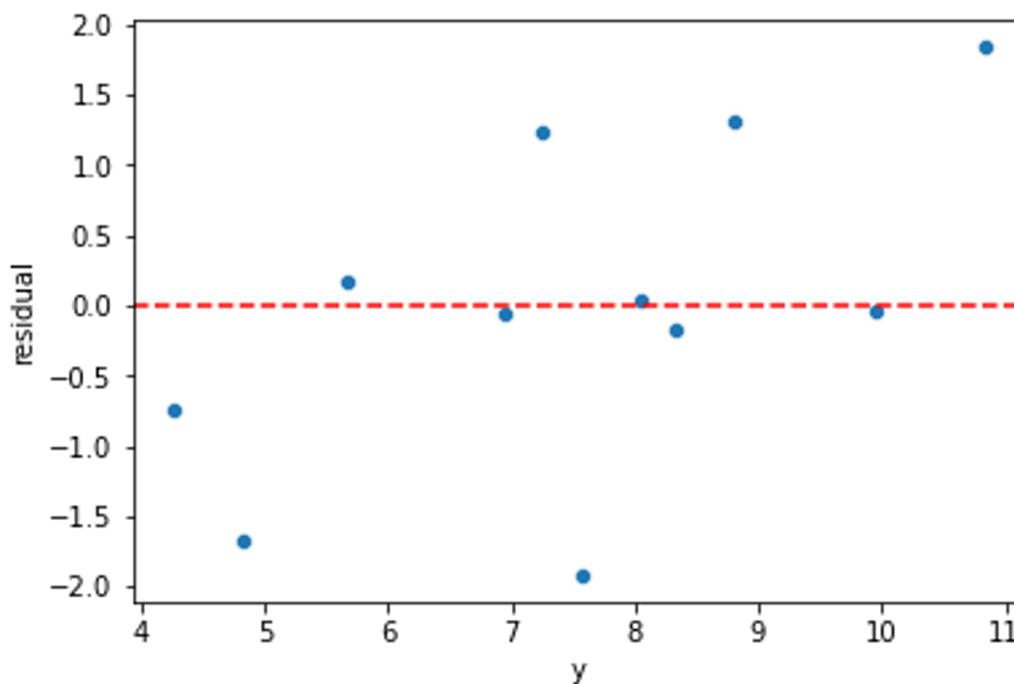
# Linear Regression

- Residual plot shows residual error vs. true y value.



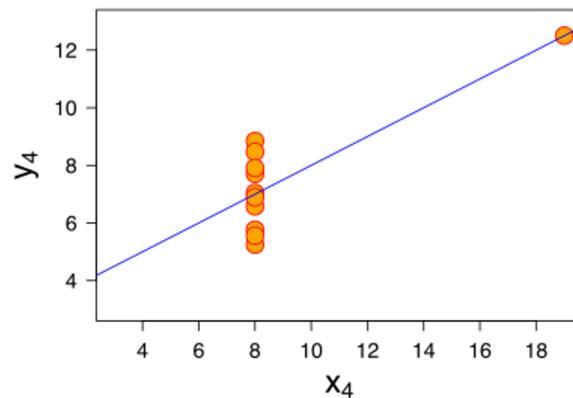
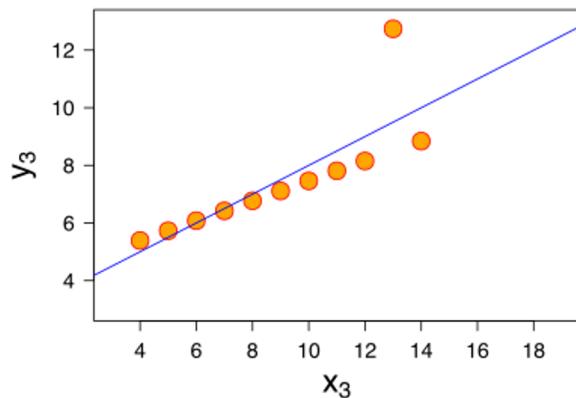
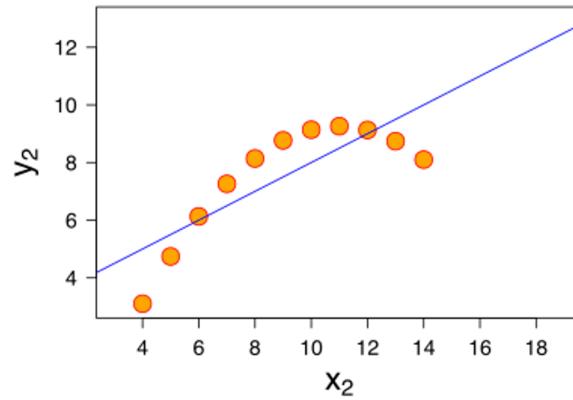
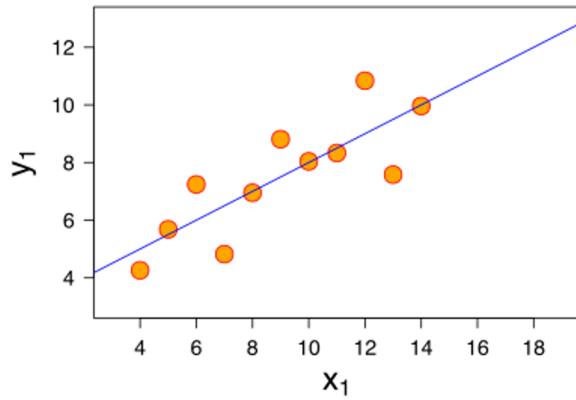
# Linear Regression

- There should be no clear line or curve.



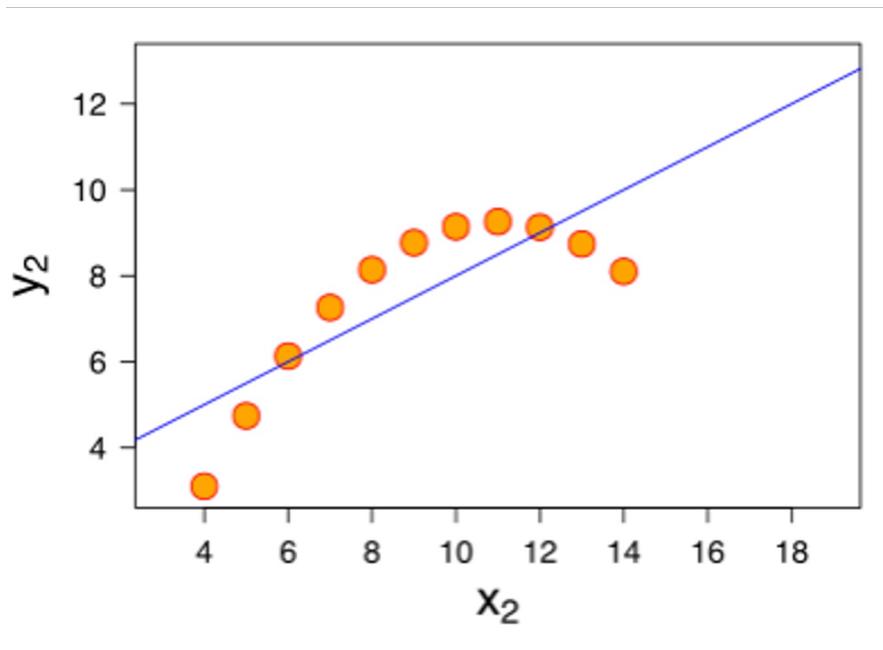
# Linear Regression

- What about non valid datasets?



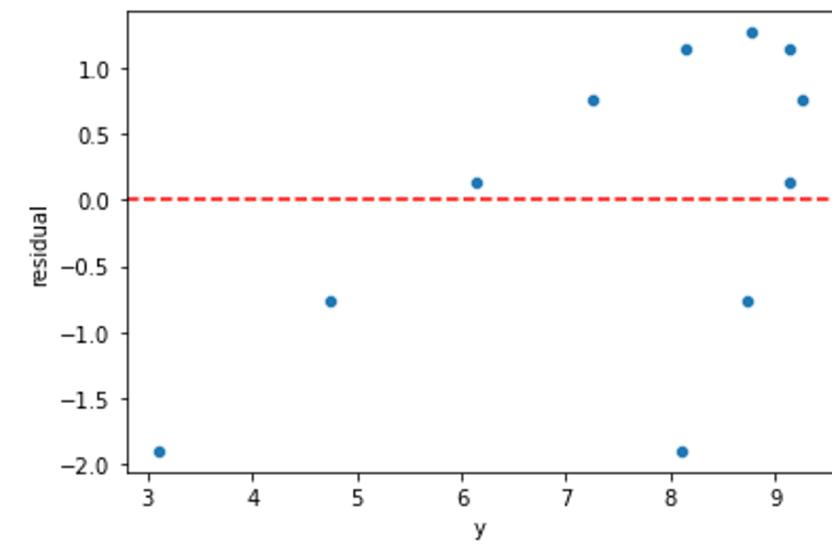
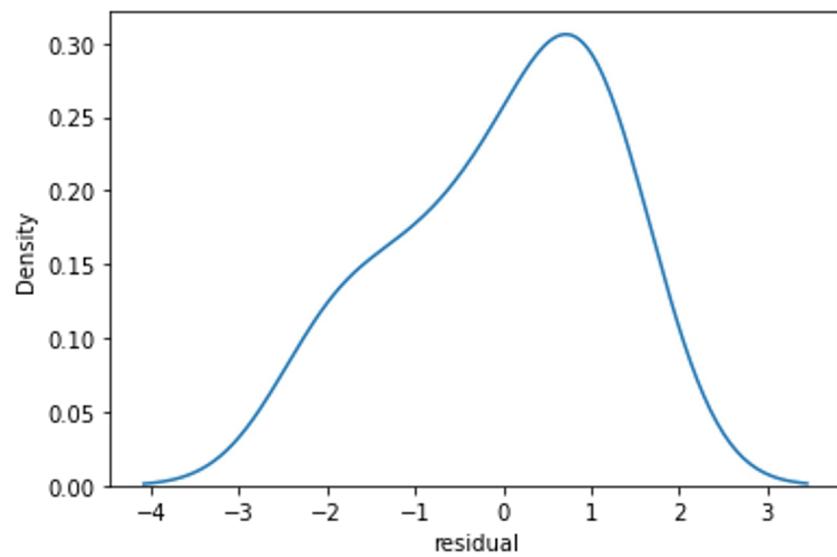
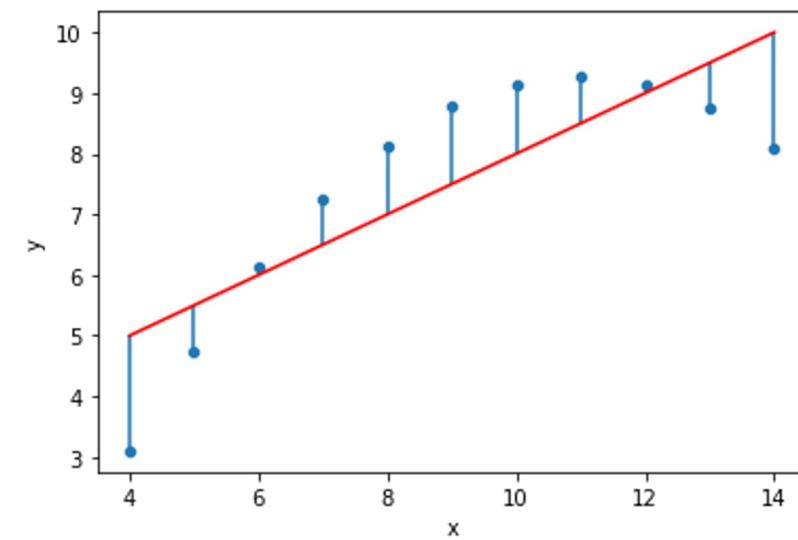
# Linear Regression

- What about non valid datasets?



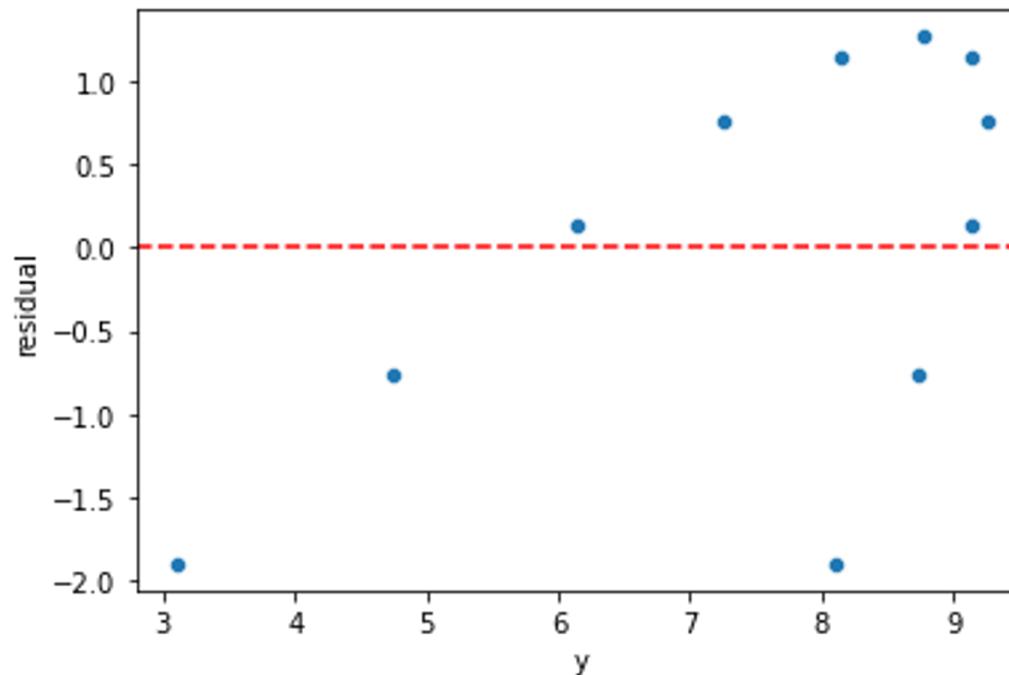
# Linear Regression

- Residual plot showing a clear pattern, indicating Linear Regression no valid!



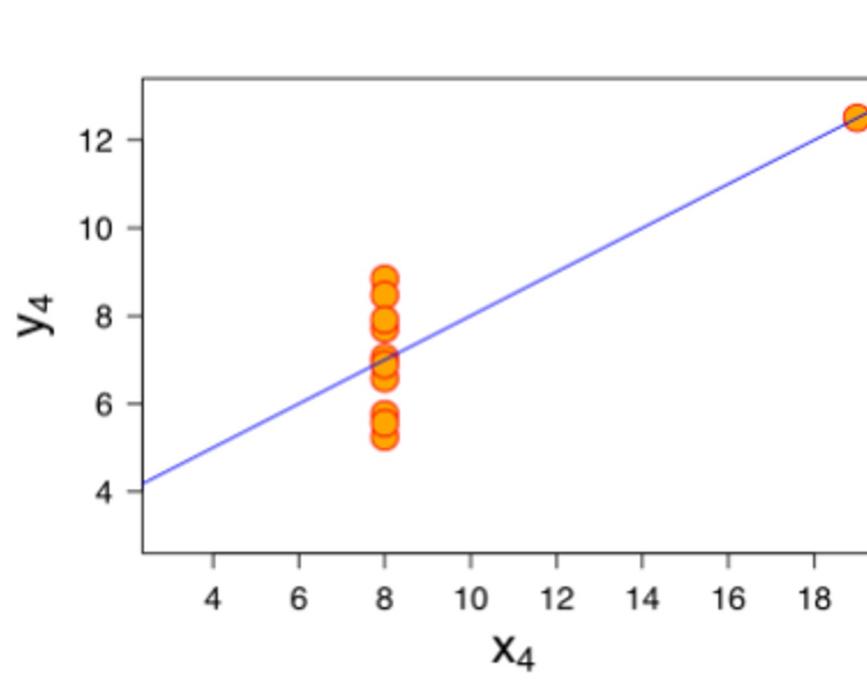
# Linear Regression

- Residual plot showing a clear pattern, indicating Linear Regression no valid!



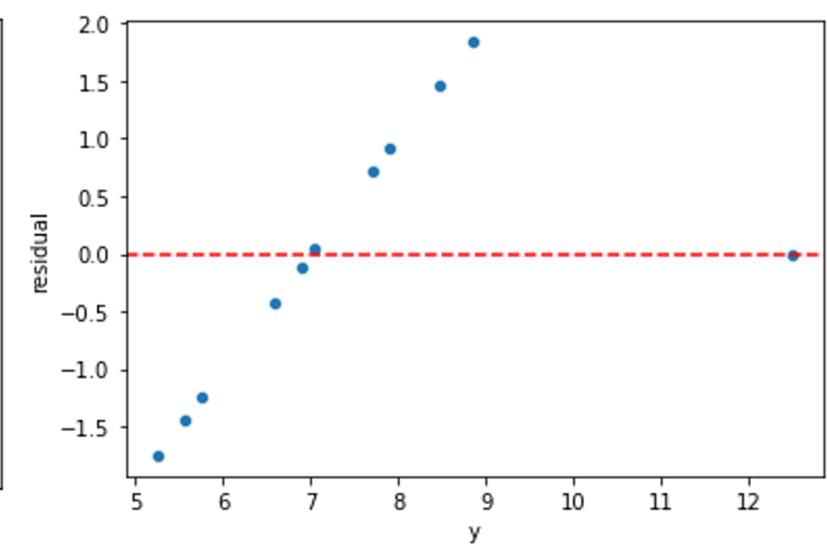
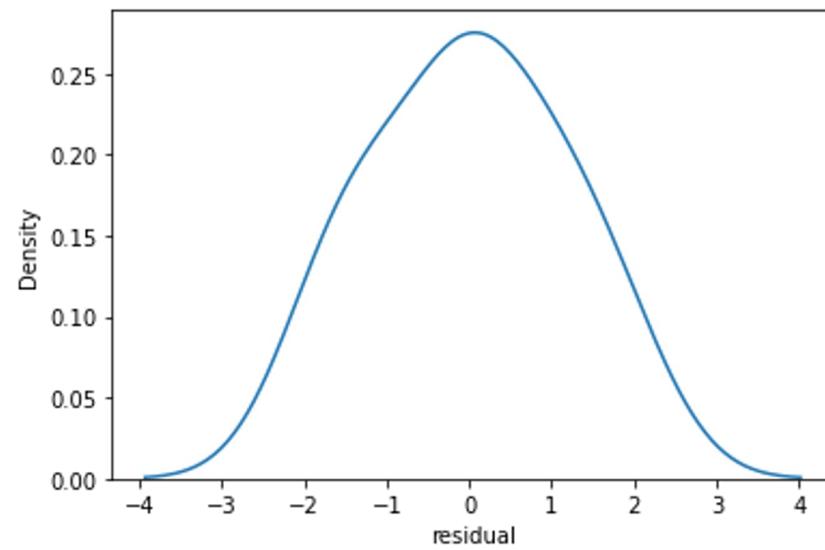
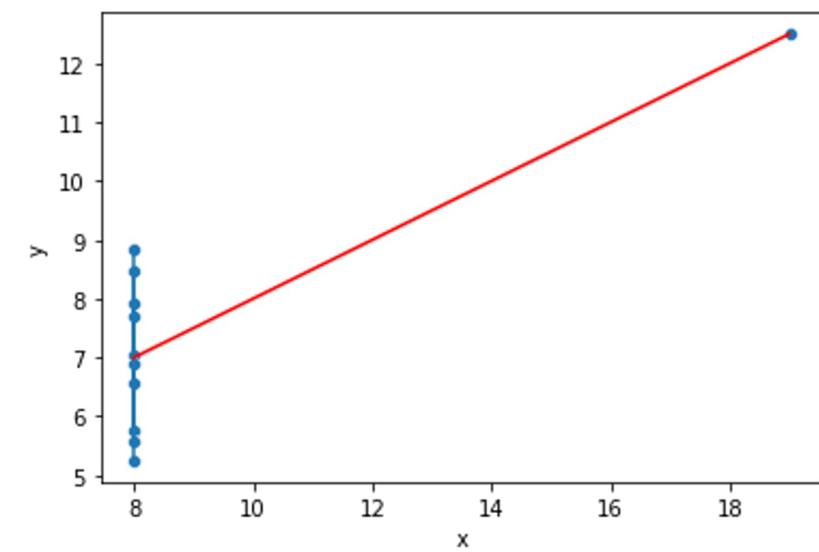
# Linear Regression

- What about non valid datasets?



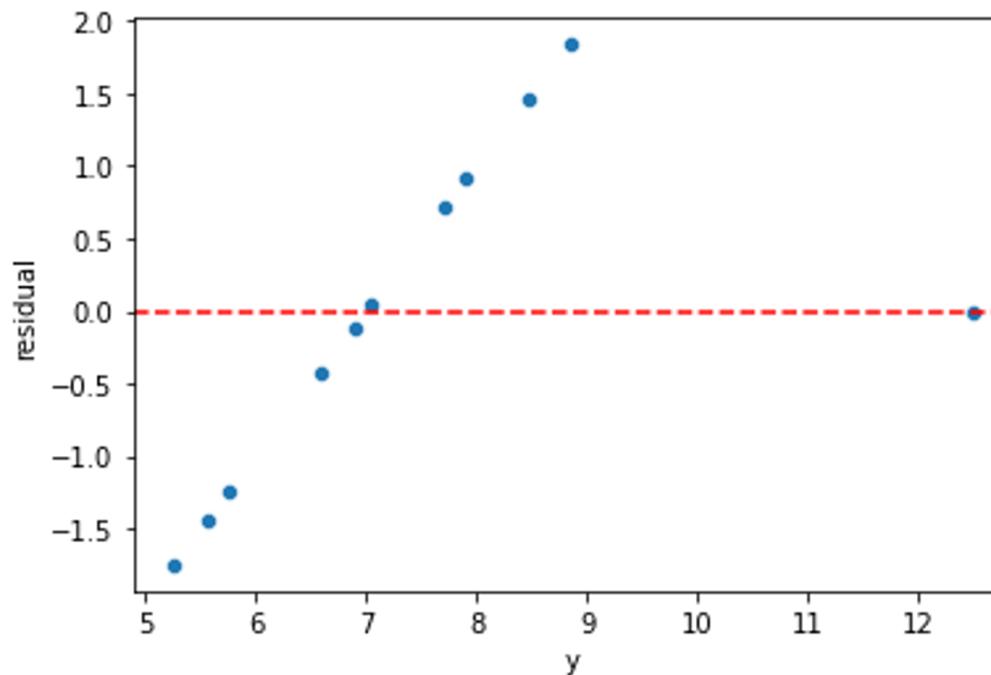
# Linear Regression

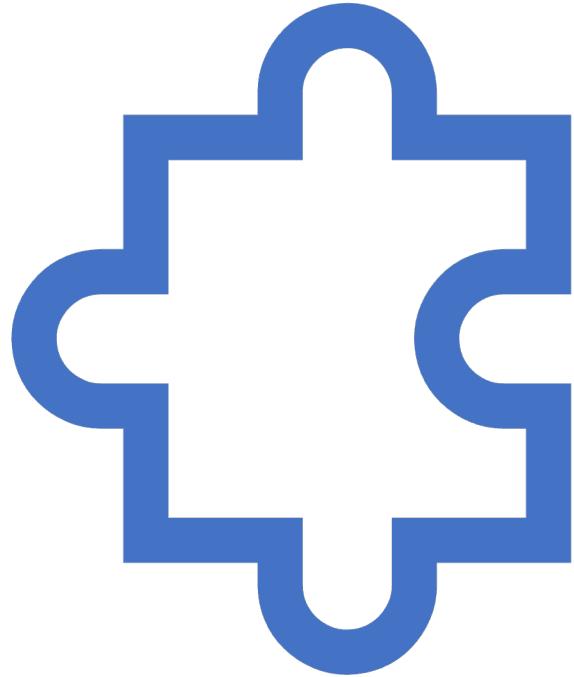
- Residual plot showing a clear pattern, indicating Linear Regression no valid!



# Linear Regression

- Residual plot showing a clear pattern, indicating Linear Regression no valid!





# Regularization

---

# Regularization

- Regularization seeks to solve a few common model issues by:
  - Minimizing model complexity
  - Penalizing the loss function
  - Reducing model overfitting (add more bias to reduce model variance)

# Regularization

- In general, we can think of regularization as a way to reduce model overfitting and variance.
  - Requires some additional bias
  - Requires a search for optimal penalty hyperparameter.

# Regularization

- Three main types of Regularization:
  - L1 Regularization
    - LASSO Regression
  - L2 Regularization
    - Ridge Regression
  - Combining L1 and L2
    - Elastic Net

# Regularization

- L1 regularization adds a penalty equal to the **absolute value** of the magnitude of coefficients.
  - Limits the size of the coefficients.
  - Can yield sparse models where some coefficients can become zero.

# Regularization

- L1 regularization adds a penalty equal to the **absolute value** of the magnitude of coefficients.

$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| = \text{RSS} + \boxed{\lambda \sum_{j=1}^p |\beta_j|}$$

# Regularization

- L2 regularization adds a penalty equal to the **square** of the magnitude of coefficients.
  - All coefficients are shrunk by the same factor.
  - Does not necessarily eliminate coefficients.

# Regularization

- L2 regularization adds a penalty equal to the **square** of the magnitude of coefficients.

$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 = \text{RSS} + \boxed{\lambda \sum_{j=1}^p \beta_j^2}$$

# Regularization

- Elastic net combines L1 and L2 with the addition of an alpha parameter deciding the ratio between them:

$$\frac{\sum_{i=1}^n (y_i - x_i^J \hat{\beta})^2}{2n} + \lambda \left( \frac{1-\alpha}{2} \sum_{j=1}^m \hat{\beta}_j^2 + \alpha \sum_{j=1}^m |\hat{\beta}_j| \right)$$

# Regularization

- These regularization methods do have a cost:
  - Introduce an additional hyperparameter that needs to be tuned.
  - A multiplier to the penalty to decide the “strength” of the penalty.

# Regularization

- Later on, we will actually cover L2 regularization (Ridge Regression) first, due to the intuition behind the squared term being easier to understand.

# Regularization

- Before we dive straight into coding regularization with Scikit-Learn, we need to discuss a few more relevant topics:
  - Feature Scaling
  - Cross Validation

# **Feature Scaling**

# Feature Scaling

- Feature scaling provides many benefits to our machine learning process!
- Some machine learning models that rely on distance metrics (e.g. KNN) **require** scaling to perform well.
- Let's discuss the main ideas behind feature scaling...

# Feature Scaling

- Feature scaling improves the convergence of steepest descent algorithms, which do not possess the property of scale invariance.
- If features are on different scales, certain weights may update faster than others since the feature values  $x_j$  play a role in the weight updates.

# Feature Scaling

- Critical benefit of feature scaling related to gradient descent.
- There are some ML Algos where scaling won't have an effect (e.g. CART based methods).

# Feature Scaling

- Scaling the features so that their respective ranges are uniform is important in comparing measurements that have different units.
- Allows us directly compare model coefficients to each other.

# Feature Scaling

- Feature scaling caveats:
  - Must always scale new unseen data before feeding to model.
  - Effects direct interpretability of feature coefficients
    - Easier to compare coefficients to one another, harder to relate back to original unscaled feature.

# Feature Scaling

- Feature scaling benefits:
  - Can lead to great increases in performance.
  - Absolutely necessary for some models.
  - Virtually no “real” downside to scaling features.

# Feature Scaling

- Two main ways to scale features:
  - Standardization
    - Rescales data to have a mean ( $\mu$ ) of 0 and standard deviation ( $\sigma$ ) of 1.
  - Normalization
    - Rescales all data values to be between 0-1.

# Feature Scaling

- Standardization:
  - Rescales data to have a mean ( $\mu$ ) of 0 and standard deviation ( $\sigma$ ) of 1 (unit variance).

$$X_{changed} = \frac{X - \mu}{\sigma}$$

# Feature Scaling

- Standardization:
  - Namesake can be confusing since this is also referred to as “Z-score normalization”.

$$X_{\text{changed}} = \frac{X - \mu}{\sigma}$$

# Feature Scaling

- Normalization:
  - Scales all data values to be between 0 and 1.

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

# Feature Scaling

- Normalization:
  - Simple and easy to understand.

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

# Feature Scaling

- There are many more methods of scaling features and Scikit-Learn provides easy to use classes that “fit” and “transform” feature data for scaling.
- Let’s quickly discuss the fit and transform calls in more detail when it comes to scaling.

# Feature Scaling

- A `.fit()` method call simply calculates the necessary statistics ( $X_{\min}$ ,  $X_{\max}$ , mean, standard deviation).
- A `.transform()` call actually scales data and returns the new scaled version of data.
- Previously saw a similar process for polynomial feature conversion.

# Feature Scaling

- Very important consideration for fit and transform:
  - We only **fit** to training data.
  - Calculating statistical information should only come from training data.
  - Don't want to assume prior knowledge of the test set!

# Feature Scaling

- Using the full data set would cause **data leakage**:
  - Calculating statistics from full data leads to some information of the test set leaking into the training process upon `transform()` conversion.

# Feature Scaling

- Feature scaling process:
  - Perform train test split
  - Fit to training feature data
  - Transform training feature data
  - Transform test feature data

# Feature Scaling

- Do we need to scale the label?
  - In general it is not necessary nor advised.
  - Normalising the output distribution is altering the definition of the target.
  - Predicting a distribution that doesn't mirror your real-world target.

# Feature Scaling

- Do we need to scale the label?
  - Can negatively impact stochastic gradient descent.
- **[stats.stackexchange.com/questions/111467](https://stats.stackexchange.com/questions/111467)**

# Feature Scaling

- Now that we understand the benefits of feature scaling, let's move on to understanding the benefits of cross-validation!

# Cross Validation

# Cross Validation

- Cross validation is a more advanced set of methods for splitting data into training and testing sets.
- Cross Validation Relevant Reading:
  - Section 5.1 of ISLR

# Cross Validation

- We understand the intuition behind performing a train test split, we want to fairly evaluate our model's performance on unseen data.
- Unfortunately this means we are not able to tune hyperparameters to the **entire** dataset.

# Cross Validation

- Is there a way we can achieve the following:
  - Train on all the data
  - Evaluate on all the data
- While it sounds impossible, we can achieve this with cross validation!
- Let's have an overview of the concept...

# Cross Validation

- Imagine our data set:

	<b>x</b>		<b>y</b>
<b>Area m<sup>2</sup></b>	<b>Bedrooms</b>	<b>Bathroom s</b>	<b>Price</b>
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

# Cross Validation

- Let's convert this data into colored blocks for cross-validation

	x		y
Area m <sup>2</sup>	Bedrooms	Bathroom s	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

# Cross Validation

- Convert to generalized form

<b>x</b>			<b>y</b>
<b>x<sub>1</sub></b>	<b>x<sub>2</sub></b>	<b>x<sub>3</sub></b>	<b>y</b>
$x^1_1$	$x^1_1$	$x^1_1$	$y_1$
$x^2_1$	$x^2_1$	$x^2_1$	$y_2$
$x^3_1$	$x^3_1$	$x^3_1$	$y_3$
$x^4_1$	$x^4_1$	$x^4_1$	$y_4$
$x^5_1$	$x^5_1$	$x^5_1$	$y_5$

# Cross Validation

- Color based off train vs. test set.

<b>x</b>	<b>y</b>
<b>x<sub>1</sub></b>	<b>y</b>
$x^1_1$	$y_1$
$x^2_1$	$y_2$
$x^3_1$	$y_3$
$x^4_1$	$y_4$
$x^5_1$	$y_5$

# Cross Validation

- Color based off train vs. test set.

	$x_1$	$x_2$	$x_3$	$y$
TRAIN	$x^1_1$	$x^1_1$	$x^1_1$	$y_1$
	$x^2_1$	$x^2_1$	$x^2_1$	$y_2$
	$x^3_1$	$x^3_1$	$x^3_1$	$y_3$
TEST	$x^4_1$	$x^4_1$	$x^4_1$	$y_4$
	$x^5_1$	$x^5_1$	$x^5_1$	$y_5$

# Cross Validation

- Color based off train vs. test set.

	<b>X</b>		<b>y</b>	
	$x_1$	$x_2$	$x_3$	<b>y</b>
TRAIN	$x^1_1$	$x^1_1$	$x^1_1$	$y_1$
	$x^2_1$	$x^2_1$	$x^2_1$	$y_2$
	$x^3_1$	$x^3_1$	$x^3_1$	$y_3$
TEST	$x^4_1$	$x^4_1$	$x^4_1$	$y_4$
	$x^5_1$	$x^5_1$	$x^5_1$	$y_5$

# Cross Validation

- For now just consider training vs testing:

	<b>X</b>		<b>y</b>	
	$x_1$	$x_2$	$x_3$	<b>y</b>
TRAIN	$x^1_1$	$x^1_1$	$x^1_1$	$y_1$
	$x^2_1$	$x^2_1$	$x^2_1$	$y_2$
	$x^3_1$	$x^3_1$	$x^3_1$	$y_3$
TEST	$x^4_1$	$x^4_1$	$x^4_1$	$y_4$
	$x^5_1$	$x^5_1$	$x^5_1$	$y_5$

# Cross Validation

- For now just consider training vs testing:

**TRAIN**

$x^1_1$	$x^1_1$	$x^1_1$	$y_1$
$x^2_1$	$x^2_1$	$x^2_1$	$y_2$
$x^3_1$	$x^3_1$	$x^3_1$	$y_3$
$x^4_1$	$x^4_1$	$x^4_1$	$y_4$
$x^5_1$	$x^5_1$	$x^5_1$	$y_5$

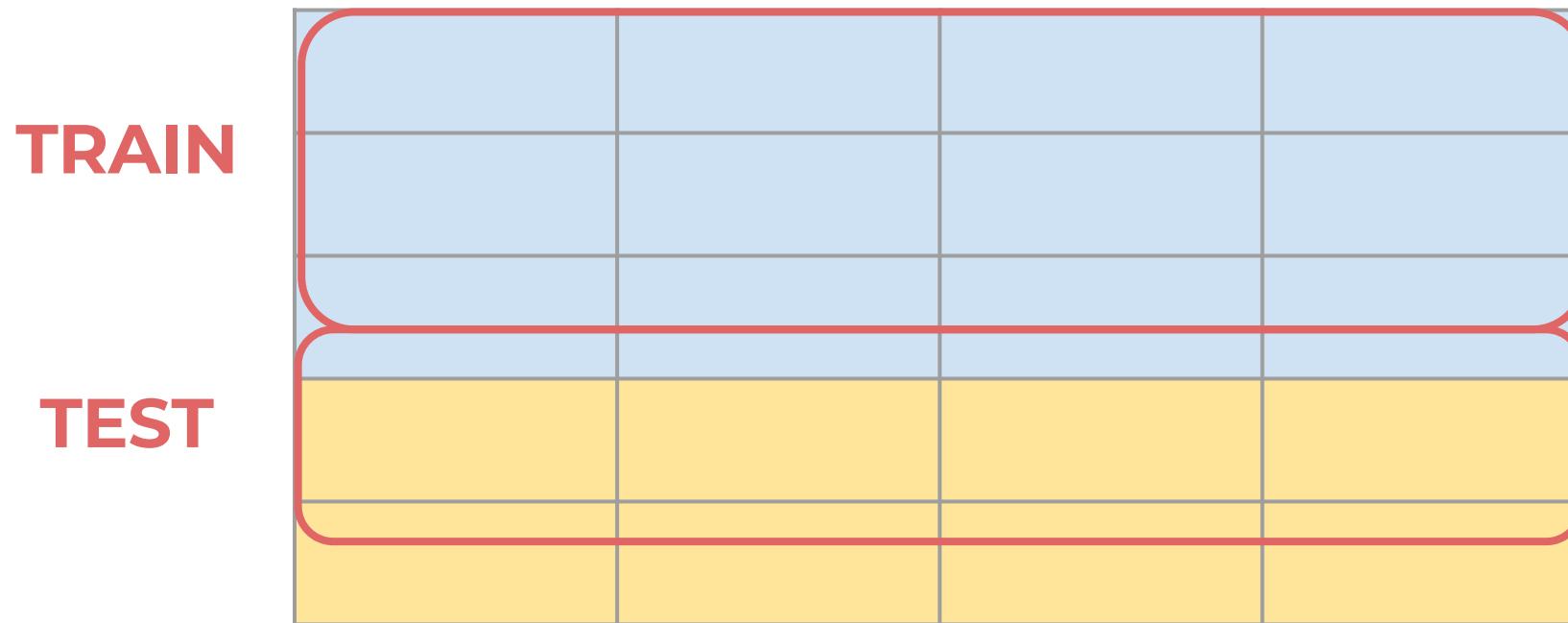
**TEST**

The diagram illustrates a dataset structure for cross-validation. It consists of a 5x4 grid of data points. The first four rows are labeled "TRAIN" and the last row is labeled "TEST". Red brackets on the left side group the first three rows under "TRAIN" and the last two rows under "TEST". Red brackets at the bottom group the first three columns under "TRAIN" and the last two columns under "TEST". The columns represent input features  $x_1$  and output labels  $y$ . The data points are as follows:

	TRAIN	TRAIN	TRAIN	TEST
TRAIN	$x^1_1$	$x^1_1$	$x^1_1$	$y_1$
TRAIN	$x^2_1$	$x^2_1$	$x^2_1$	$y_2$
TEST	$x^3_1$	$x^3_1$	$x^3_1$	$y_3$
TEST	$x^4_1$	$x^4_1$	$x^4_1$	$y_4$
	$x^5_1$	$x^5_1$	$x^5_1$	$y_5$

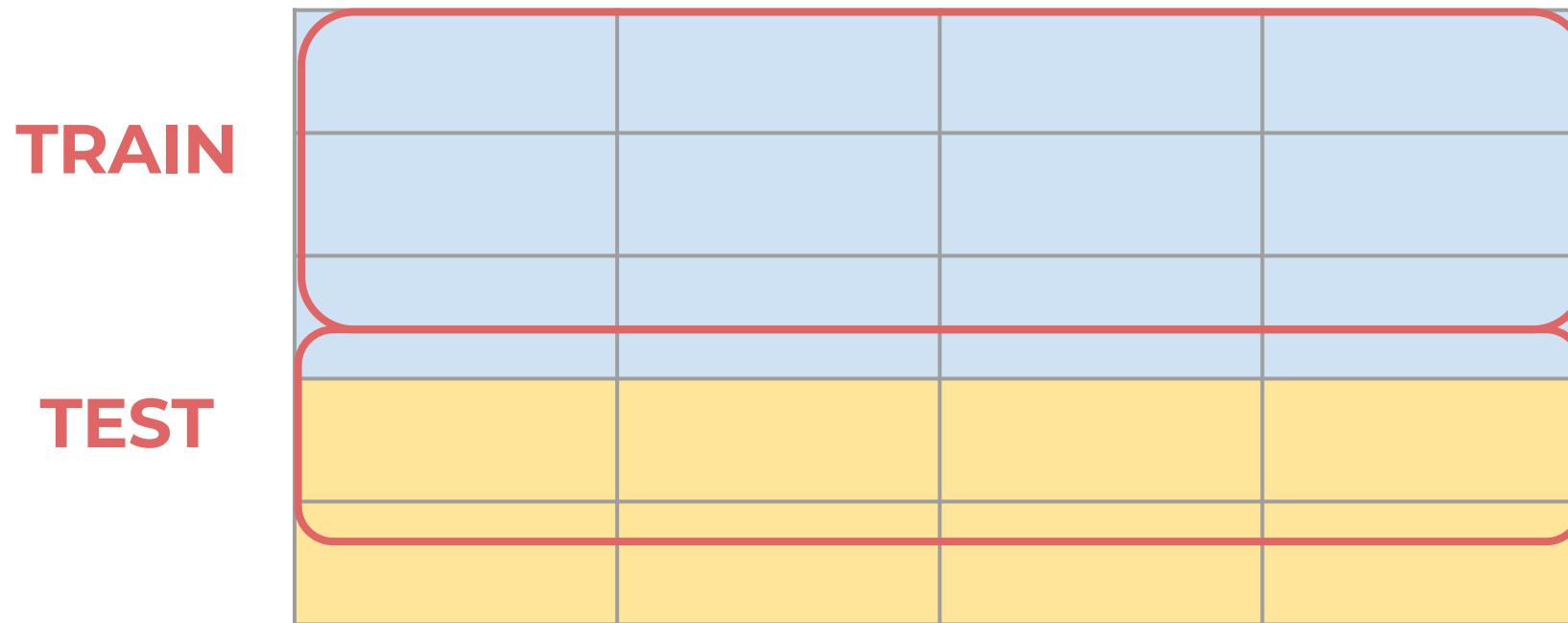
# Cross Validation

- Now we have all data, colored by training set versus test set.



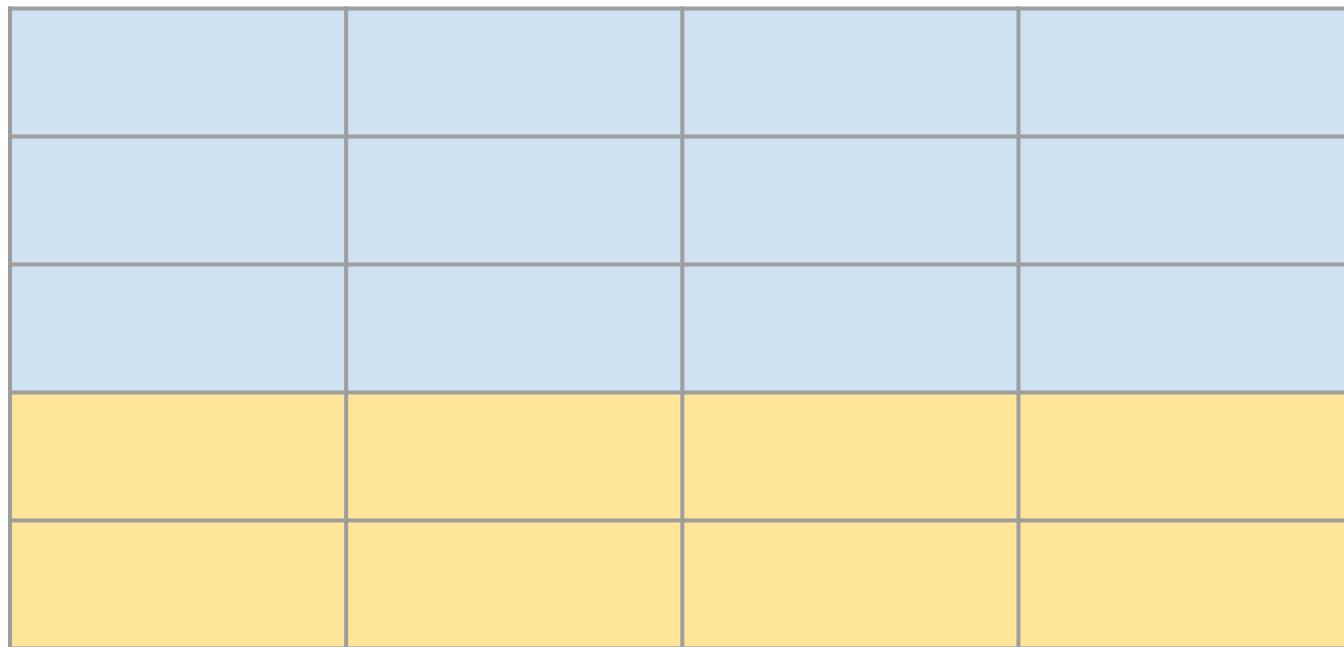
# Cross Validation

- Rotate and resize:



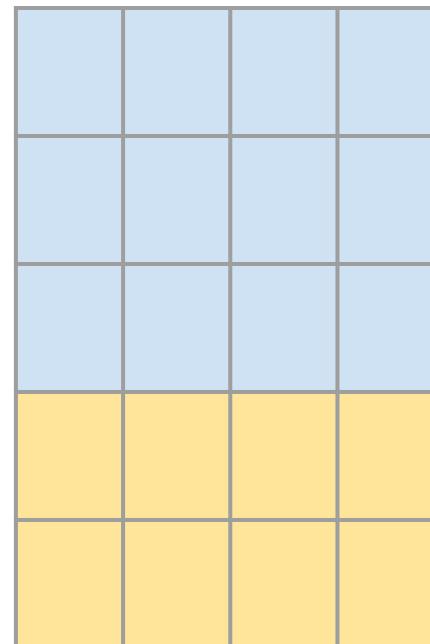
# Cross Validation

- Rotate and resize:



# Cross Validation

- Rotate and resize:



# Cross Validation

- Rotate and resize:



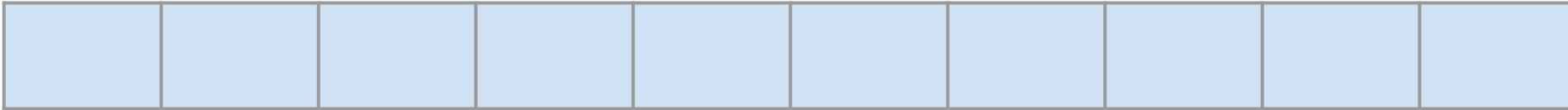
# Cross Validation

- Now we can represent full data and splits:



# Cross Validation

- Let's start with the entire original data:



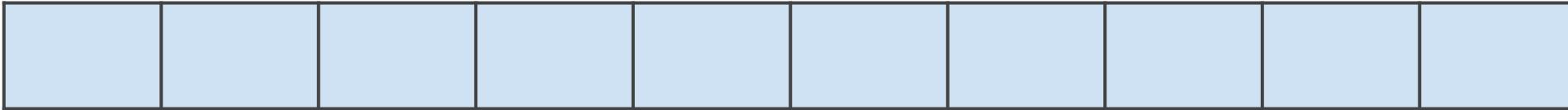
# Cross Validation

- How does cross validation work?



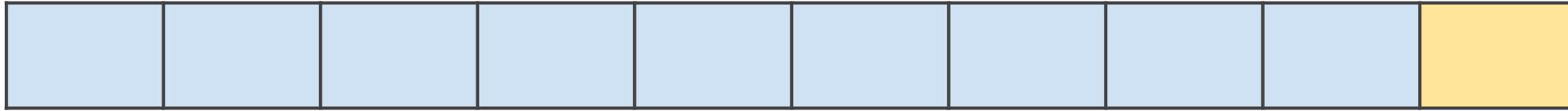
# Cross Validation

- Split data into K equal parts:



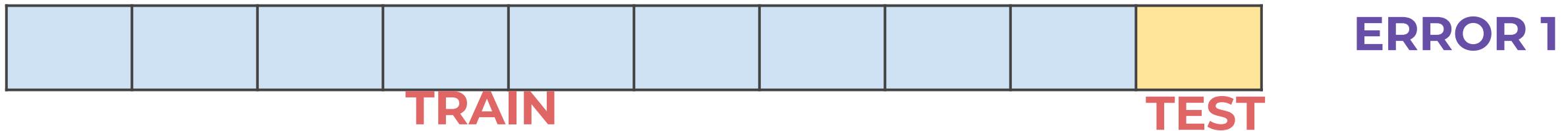
# Cross Validation

- $1/K$  left as test set



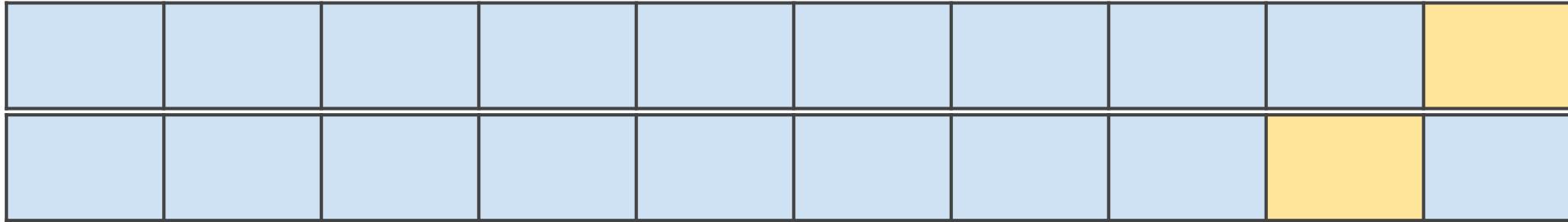
# Cross Validation

- Train model and get error metric for split:



# Cross Validation

- Repeat for another  $1/K$  split

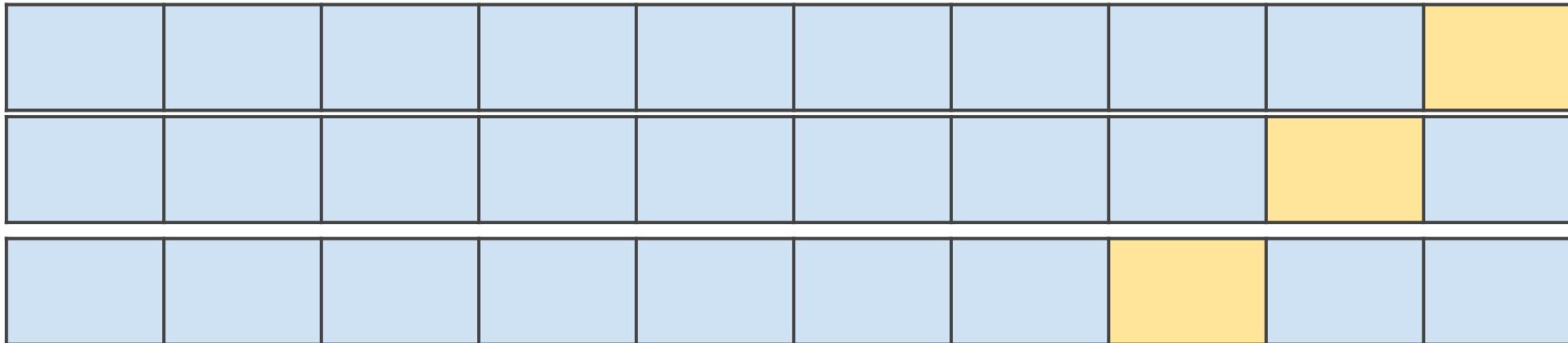


**ERROR 1**

**ERROR 2**

# Cross Validation

- Keep repeating for all possible splits



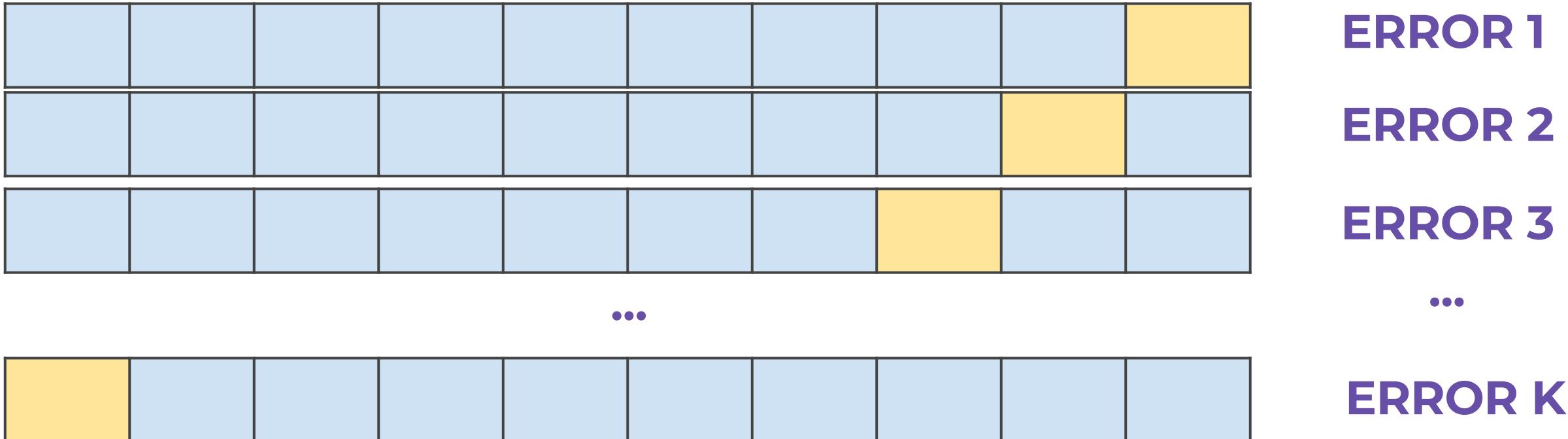
**ERROR 1**

**ERROR 2**

**ERROR 3**

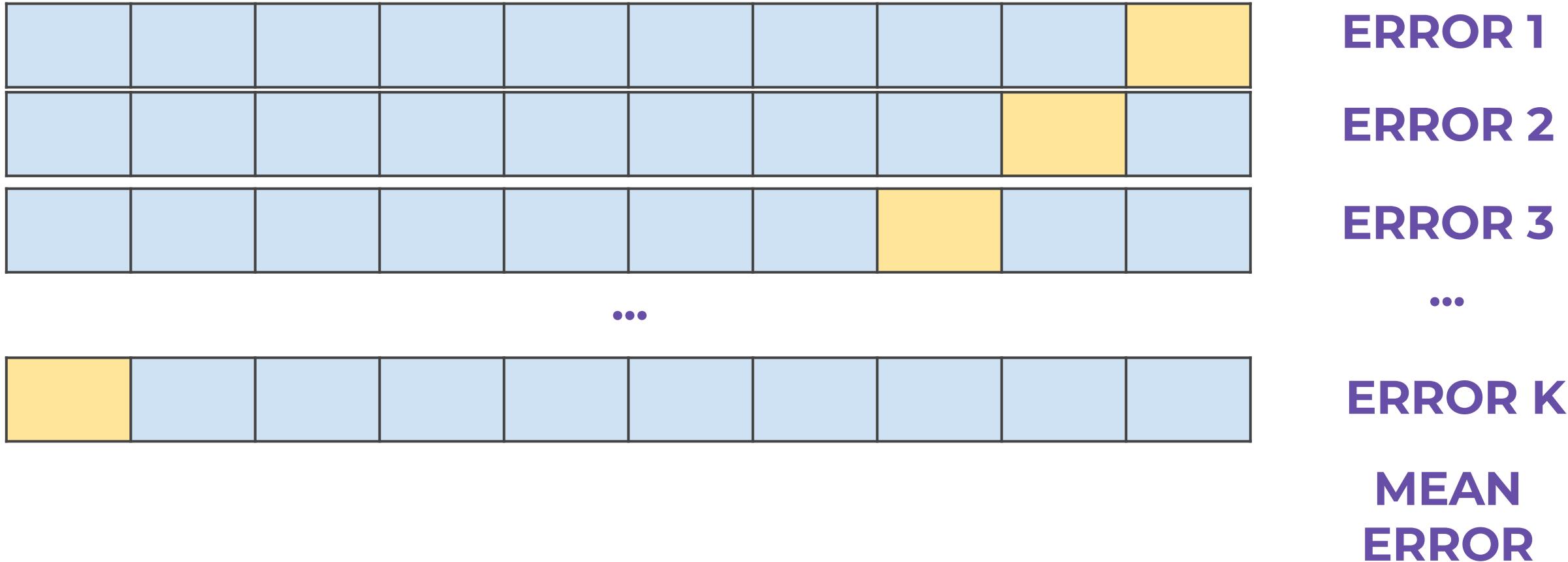
# Cross Validation

- Keep repeating for all possible splits



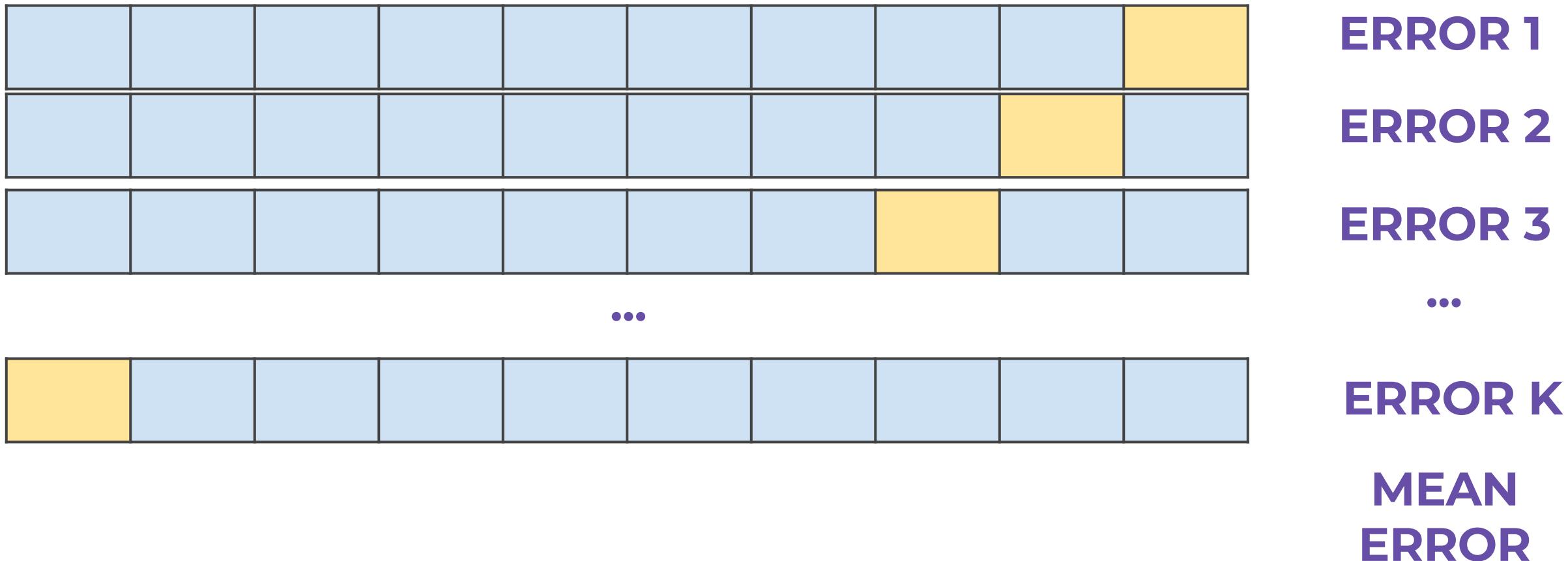
# Cross Validation

- Get average error



# Cross Validation

- Average error is the expected performance



# Cross Validation

- We were able to train on all data **and** evaluate on all data!
- We get a better sense of true performance across multiple potential splits.
- What is the cost of this?
  - We have to repeat computations  $K$  number of times!

# Cross Validation

- This is known as K-fold cross-validation.
- Common choice for K is 10 so each test set is 10% of your total data.
- Largest K possible would be K equal to the number of number of rows.
  - This is known as **leave one out** cross validation.
  - Computationally expensive!

# Cross Validation

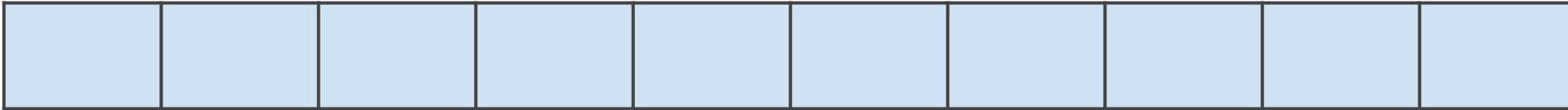
- One consideration to note with K-fold cross validation and a standard train test split is fairly tuning hyperparameters.
- If we tune hyperparameters to test data performance, are we ever fairly getting performance metrics?

# Cross Validation

- How can we understand how the model behaves for data that is has not seen **and** not been influenced by for hyperparameter tuning?
- For this we can use a **hold out** test set.
- Let's explore what this looks like...

# Cross Validation

- Start with entire data set:



# Cross Validation

- Remove a hold out test set



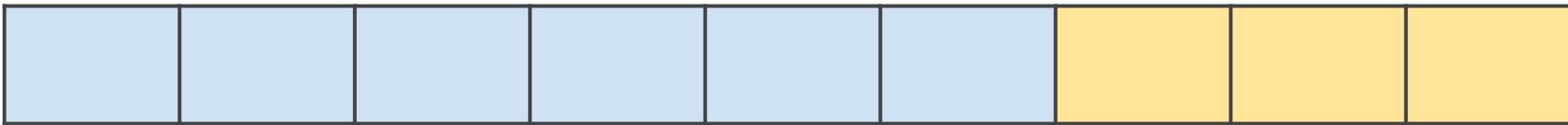
# Cross Validation

- Perform “classic” train test split:



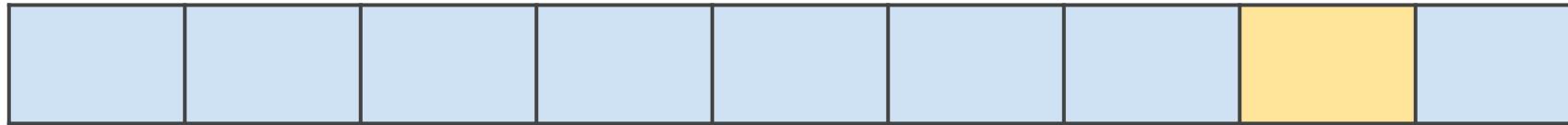
# Cross Validation

- Train and tune on this data:



# Cross Validation

- Or K-Fold cross validation

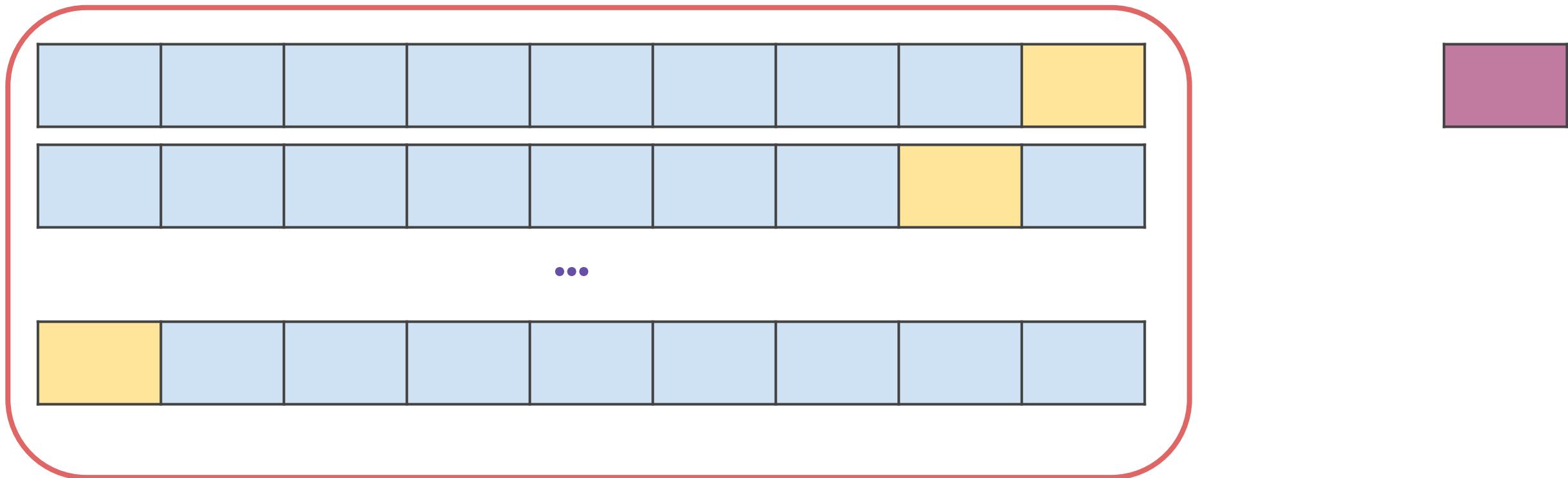


...



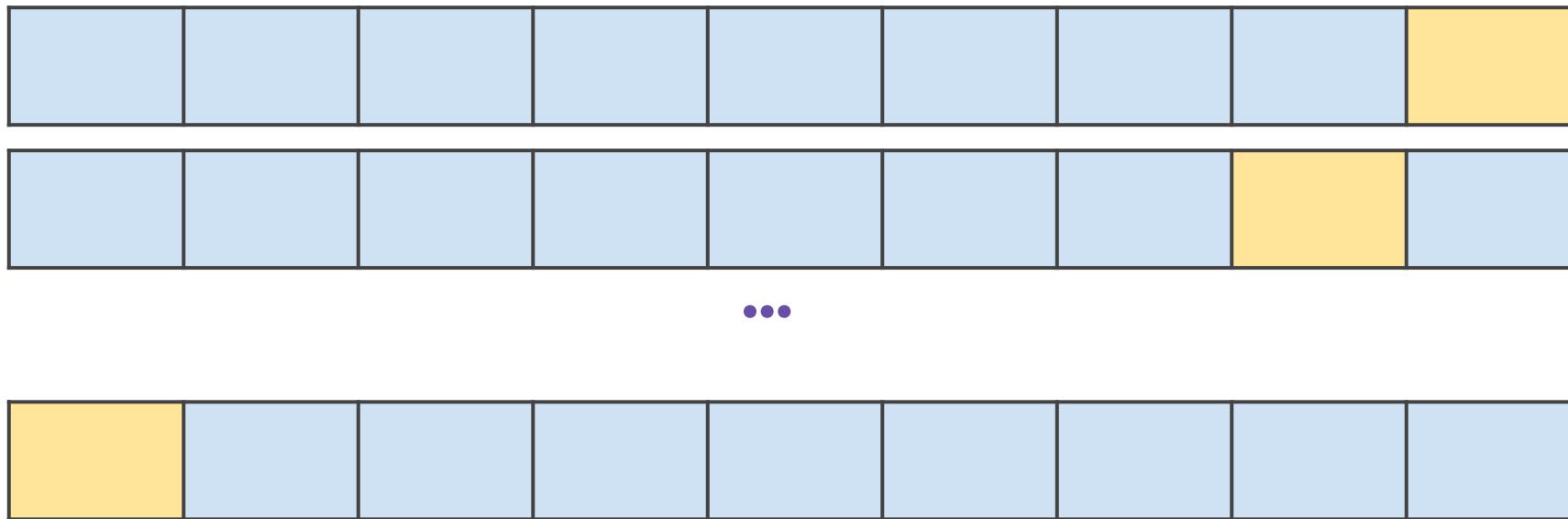
# Cross Validation

- Train **and** tune on this data:



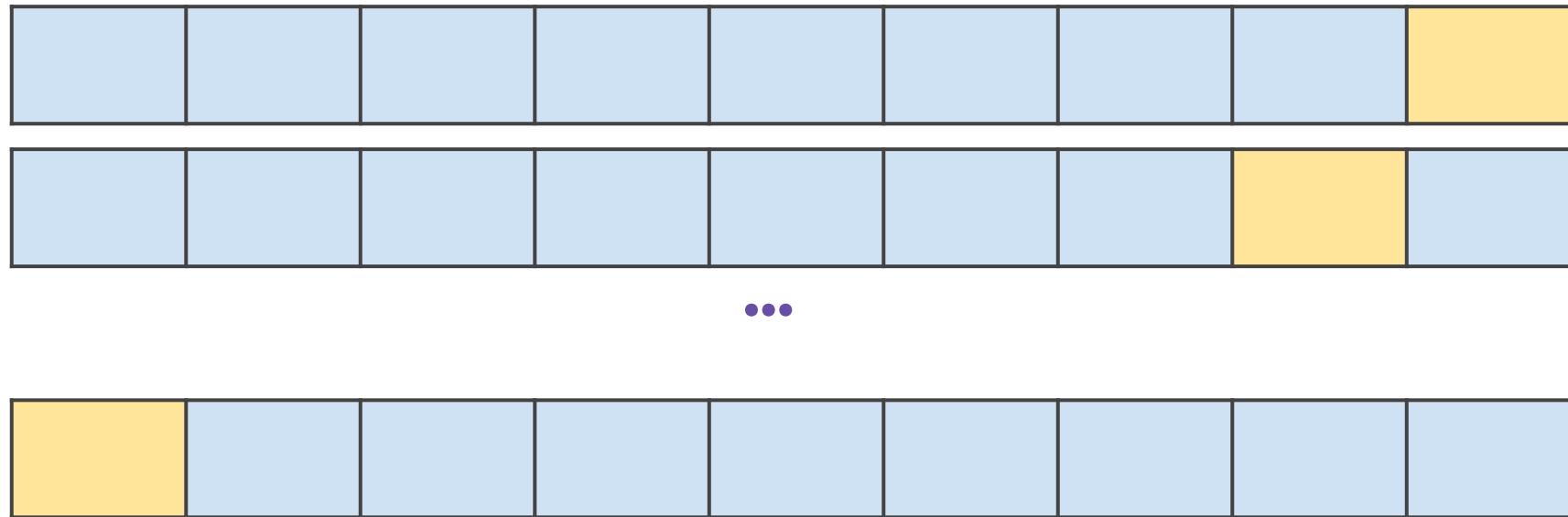
# Cross Validation

- **After** training and tuning perform **final evaluation** hold out test set.



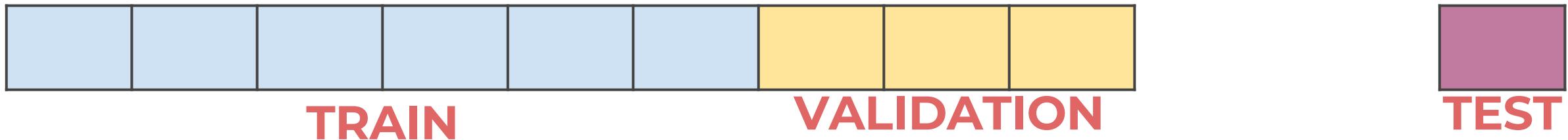
# Cross Validation

- Can **not** tune after this **final** test evaluation!



# Cross Validation

- Train | Validation | Test Split



- Allows us to get a true final performance metric to report.
- No editing model after this!

# Cross Validation

- All these approaches are valid, each situation is unique!
- Keep in mind:
  - Previous modeling work
  - Reporting requirements
  - Fairness of evaluation
  - Context of data and model

# Cross Validation

- Many regularization methods have tunable parameters we can adjust based on cross-validation techniques.
- For simplicity, there are times in the course we will opt for a simple two part train test split.

# Regularization for Linear Regression

Data Set Up

# Ridge Regression

Theory and Intuition

# Ridge Regression

- Ridge Regression is a regularization technique that works by helping reduce the potential for overfitting to the training data.
- It does this by adding in a penalty term to the error that is based on the squared value of the coefficients.

# Ridge Regression

- Ridge Regression is a regularization method for Linear Regression.
- Relevant Reading in ISLR:
  - Section 6.2.1
- Let's explore the main concepts behind how Ridge Regression works...

# Ridge Regression

- Recall the general formula for the regression line:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p$$

# Ridge Regression

- These Beta coefficients were solved by minimizing the residual sum of squares (RSS).

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p$$

# Ridge Regression

- These Beta coefficients were solved by minimizing the residual sum of squares (RSS).

---

$$\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

# Ridge Regression

- We could substitute our regression equation for  $\hat{\mathbf{y}}$ :

---

$$\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

# Ridge Regression

- We could substitute our regression equation for  $\hat{\mathbf{y}}$ :

$$\begin{aligned}\text{RSS} &= \sum_{i=1}^n (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip})^2\end{aligned}$$

# Ridge Regression

- We can then summarize RSS as:

$$\text{RSS} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$

# Ridge Regression

- The goal of Ridge Regression is to help prevent overfitting by adding an additional penalty term.

$$\text{RSS} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$

# Ridge Regression

- Ridge Regression adds a **shrinkage penalty**:

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

- Ridge Regression seeks to minimize this entire error term **RSS + Penalty**.

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

- **Shrinkage penalty** based off the squared coefficient:

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \boxed{\beta_j^2}$$

# Ridge Regression

- **Shrinkage penalty** has a **tunable lambda parameter!**

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \boxed{\lambda} \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

- Lambda determines how severe the penalty is.

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \boxed{\lambda} \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

- In theory it can be any value from 0 to positive infinity.

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \boxed{\lambda} \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

- If it is zero, then it is simply back to RSS.

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \boxed{\lambda} \sum_{j=1}^p \beta_j^2$$

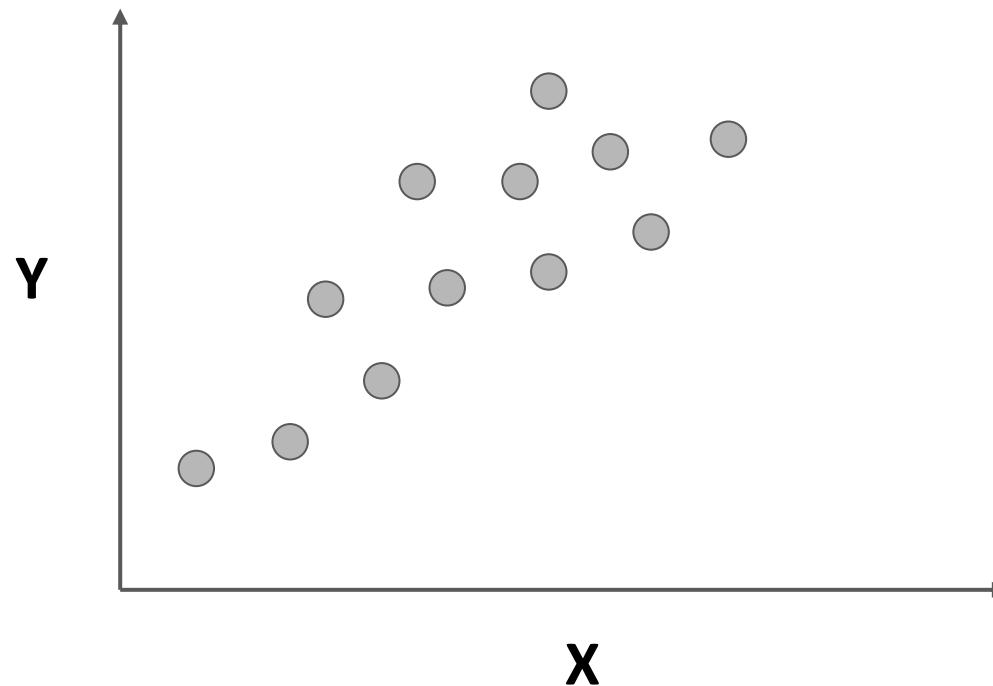
# Ridge Regression

- Let's explore a simple thought experiment to get an intuition behind Ridge Regression...

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

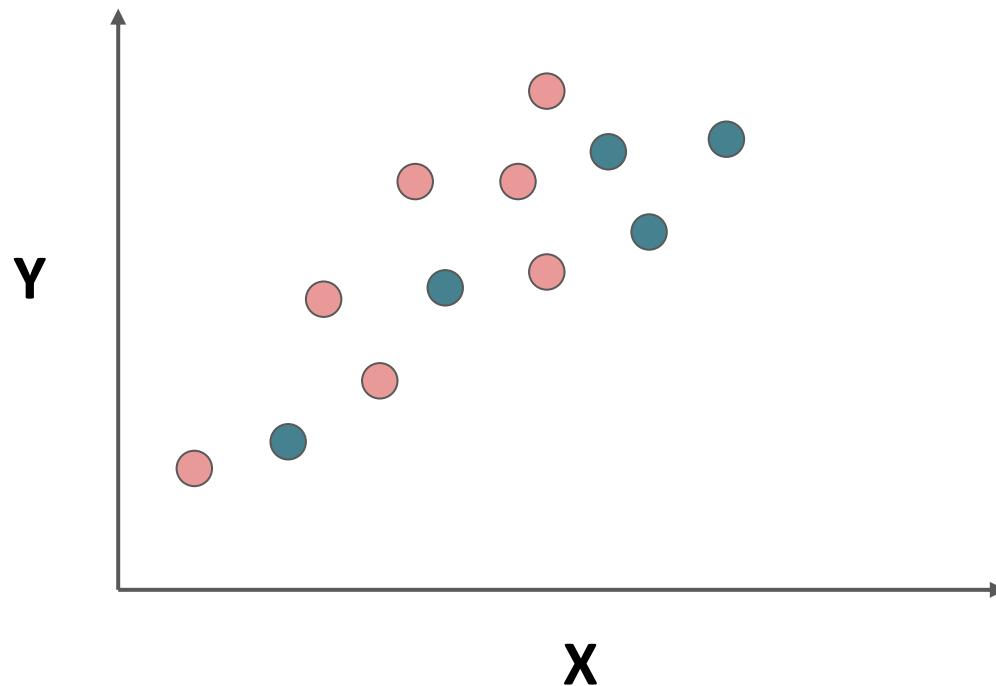
# Ridge Regression

- Imagine the following data set.



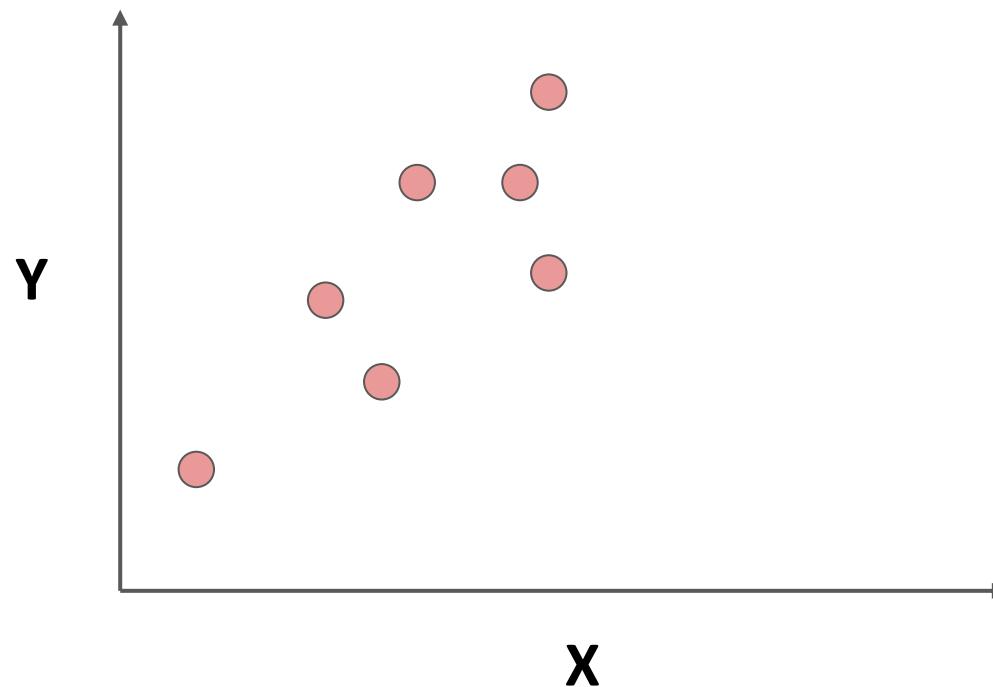
# Ridge Regression

- We can split it into a training set and test set:



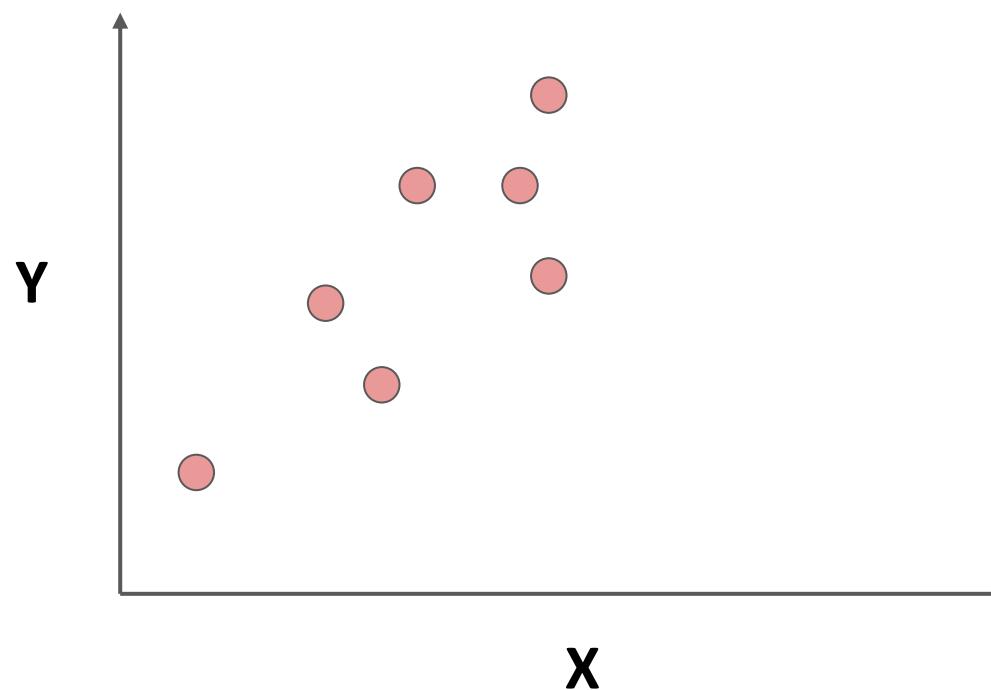
# Ridge Regression

- Now we can fit on the training data to produce the line:  $\hat{y} = \beta_1 x + \beta_0$



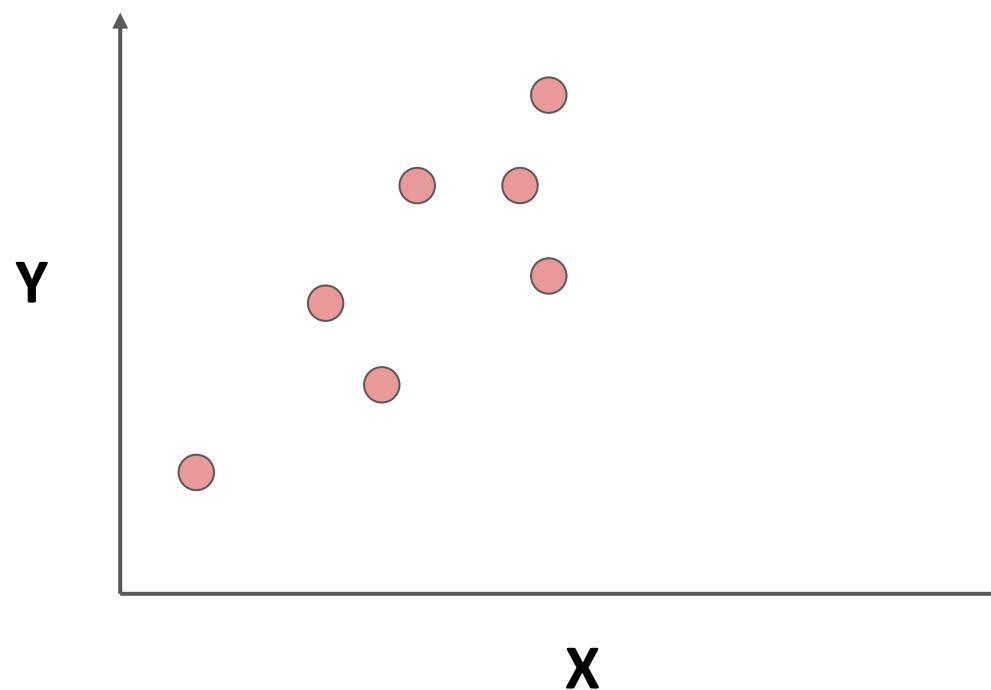
# Ridge Regression

- Regardless of RSS or Ridge error, we're still trying to create a line:  $\hat{y} = \beta_1 x + \beta_0$



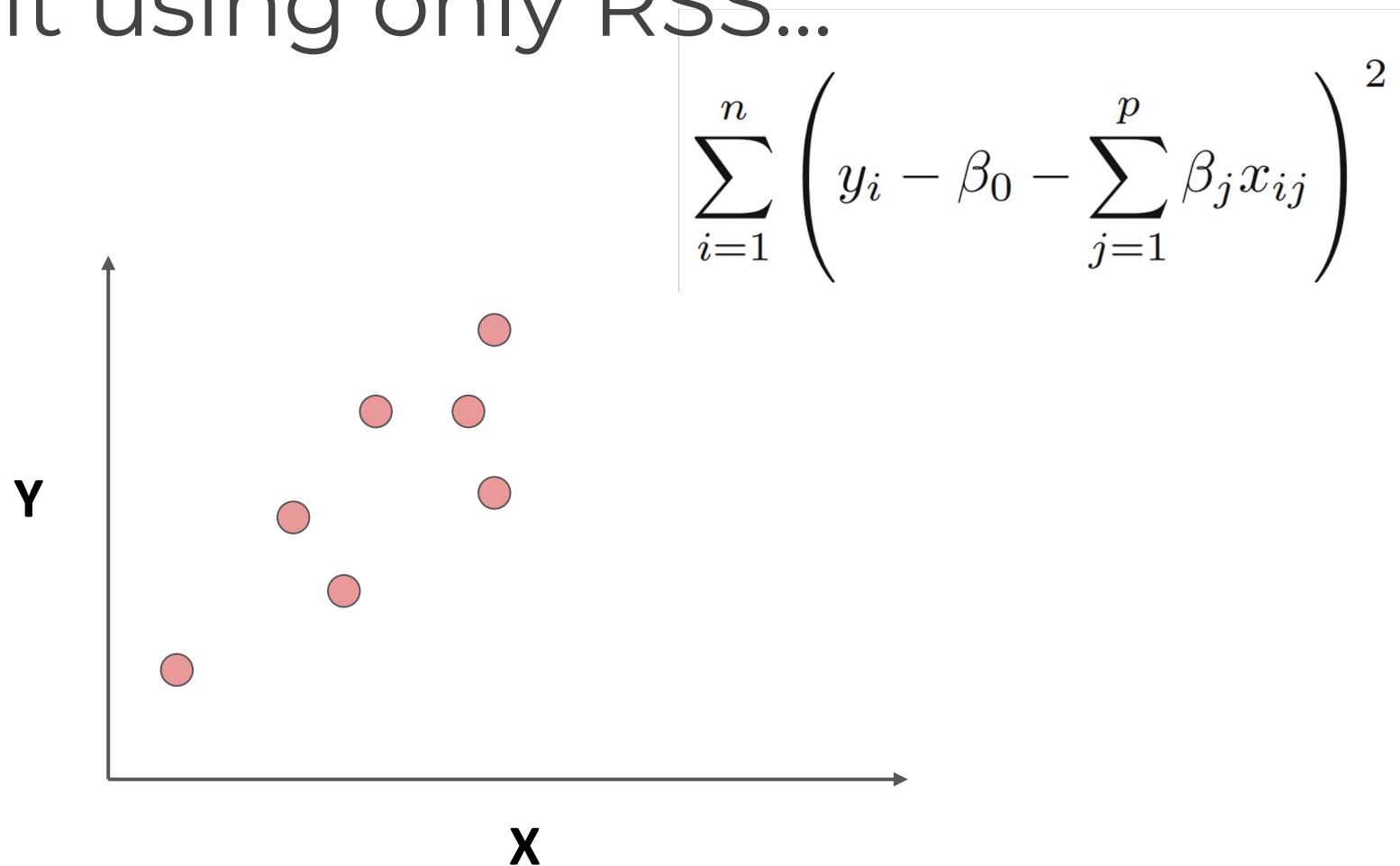
# Ridge Regression

- The only difference would be the coefficients found.



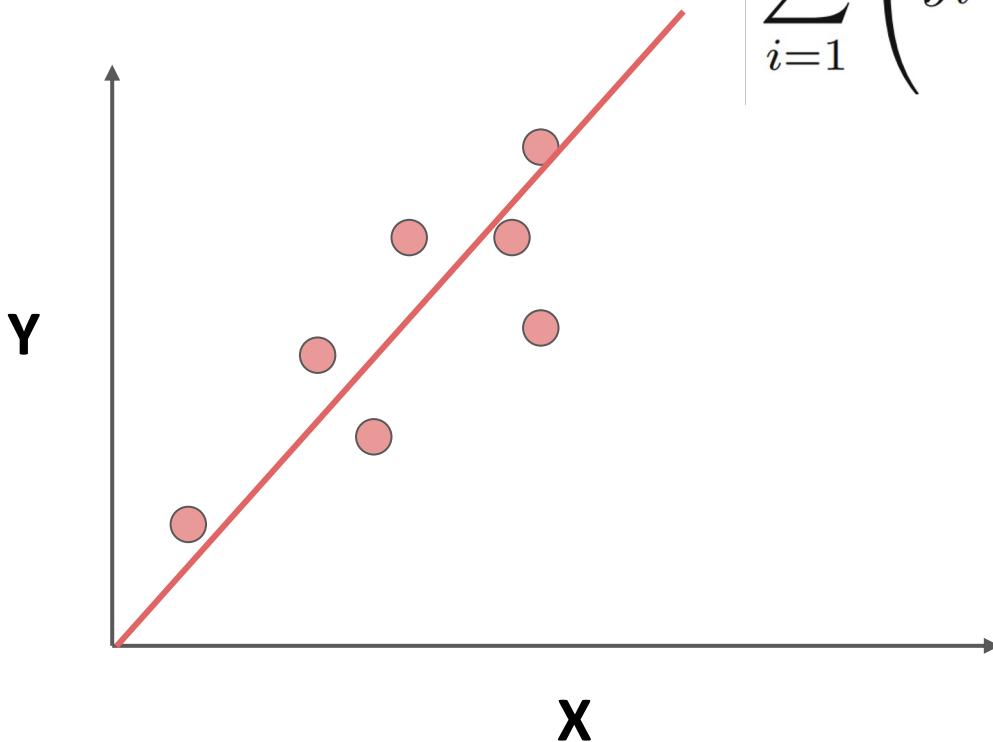
# Ridge Regression

- First let's fit using only RSS...



# Ridge Regression

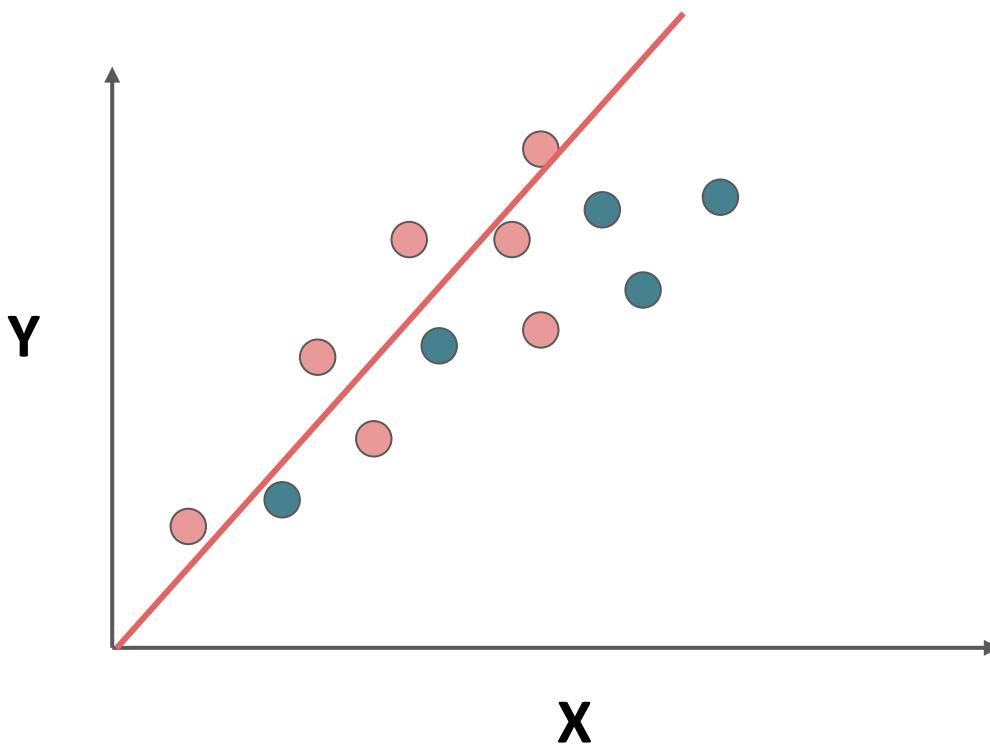
- Our fitted  $\hat{y} = \beta_1 x + \beta_0$



$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$

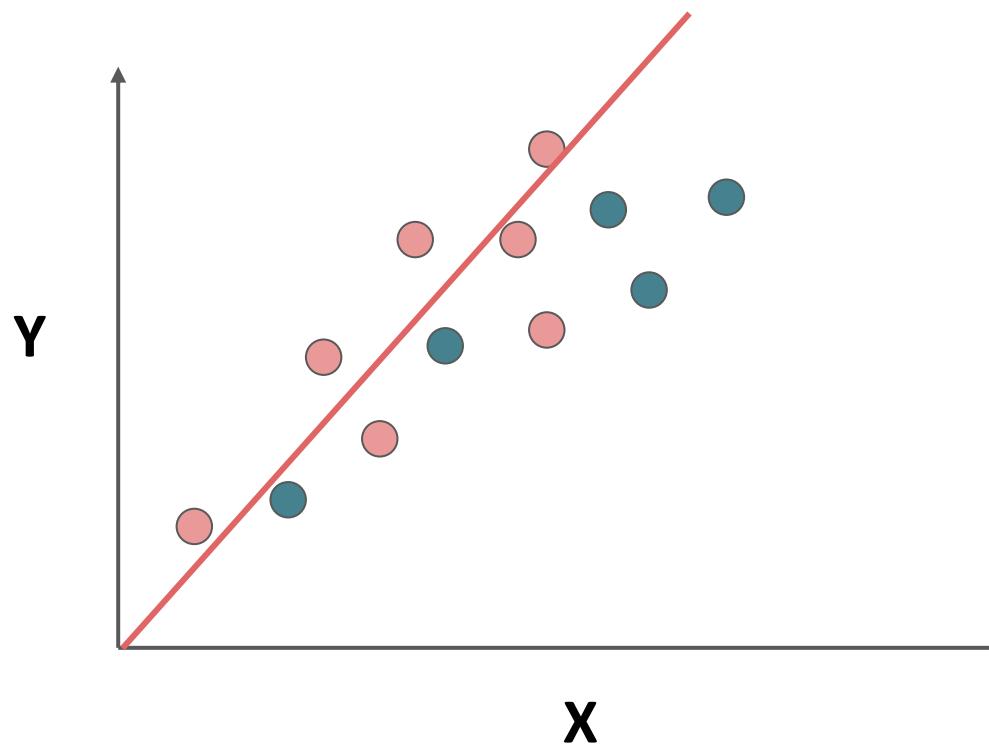
# Ridge Regression

- Appears to have over fit to training data.



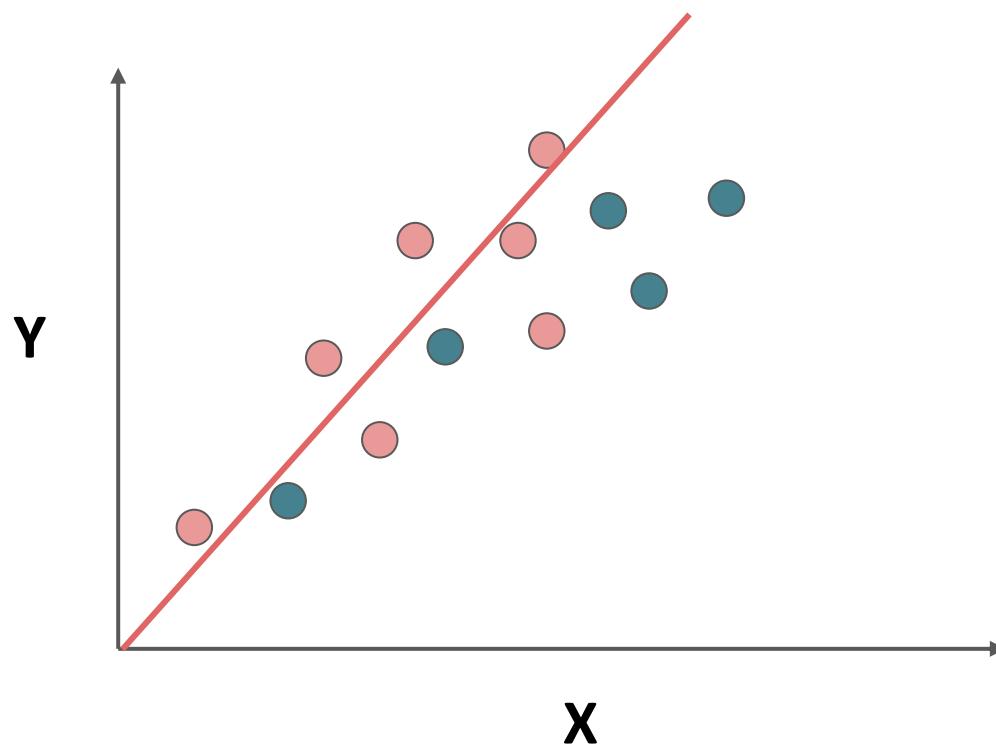
# Ridge Regression

- This means we have high **variance.**



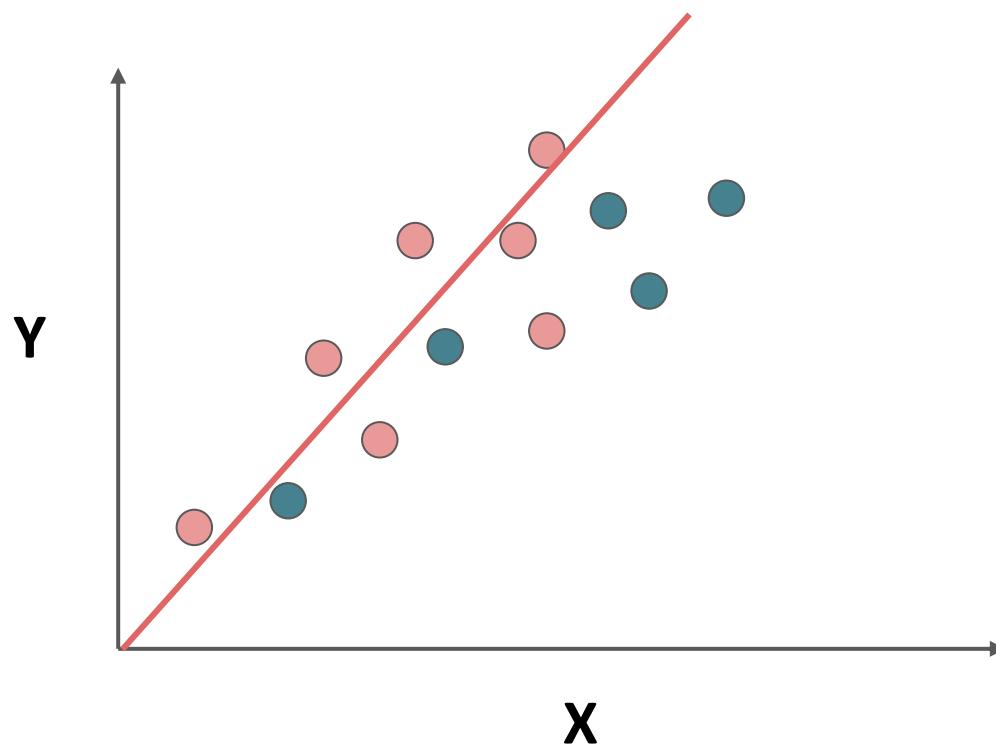
# Ridge Regression

- We know there is a **bias-variance** trade-off.



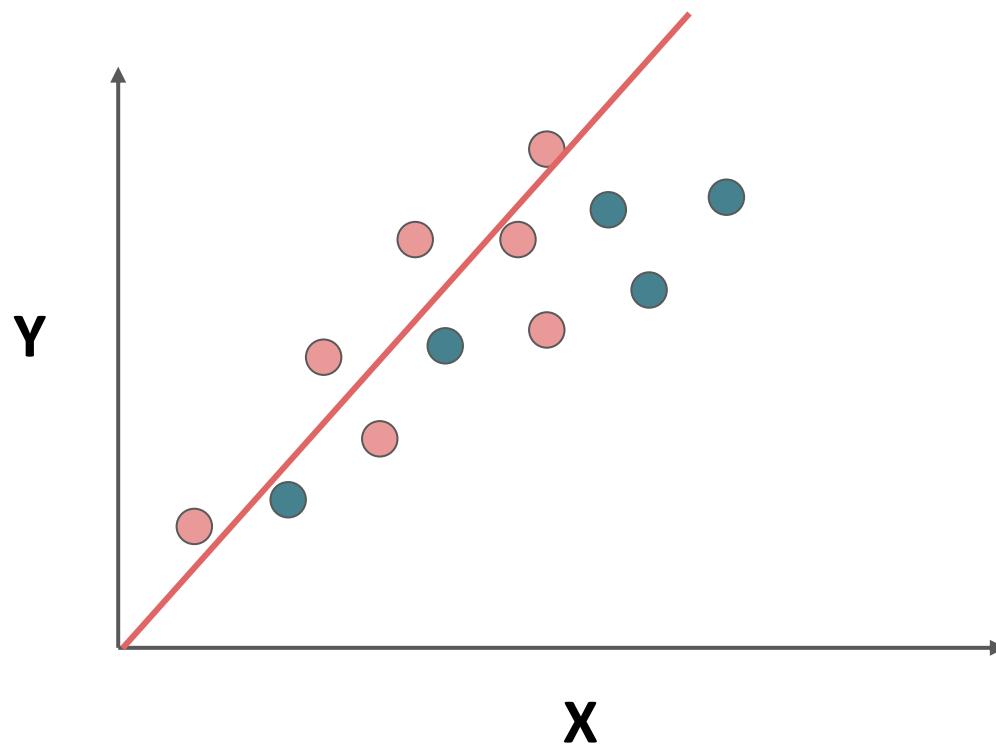
# Ridge Regression

- But could we introduce a little more **bias** to significantly **reduce** variance?



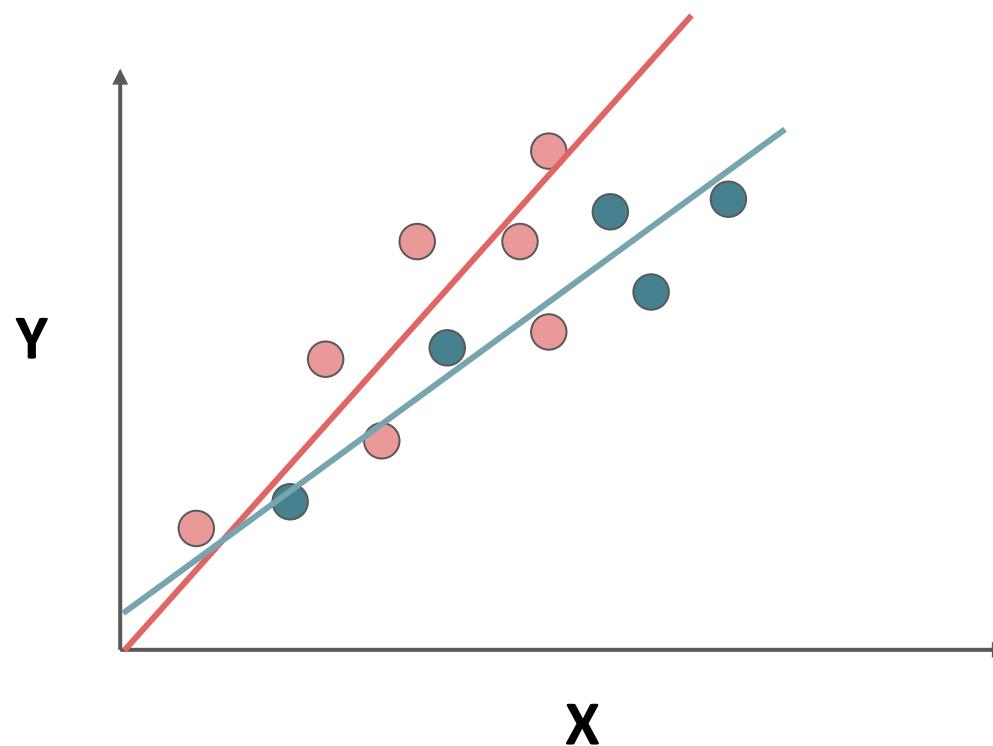
# Ridge Regression

- Would adding the penalty term help generalize with more **bias**?



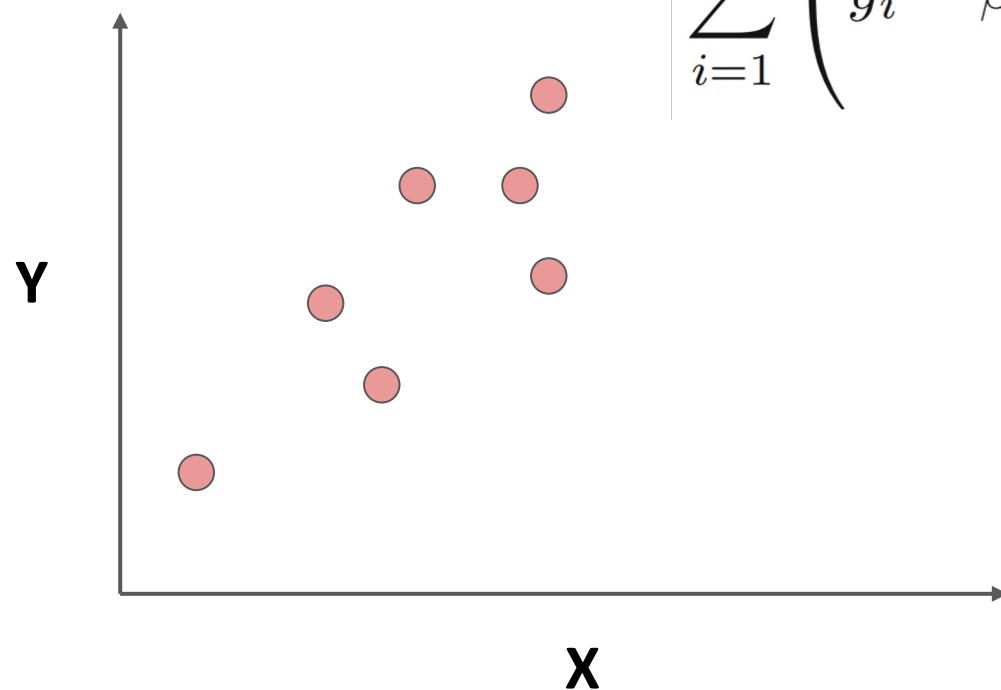
# Ridge Regression

- Adding bias can help generalize  $\hat{y} = \beta_1 x + \beta_0$



# Ridge Regression

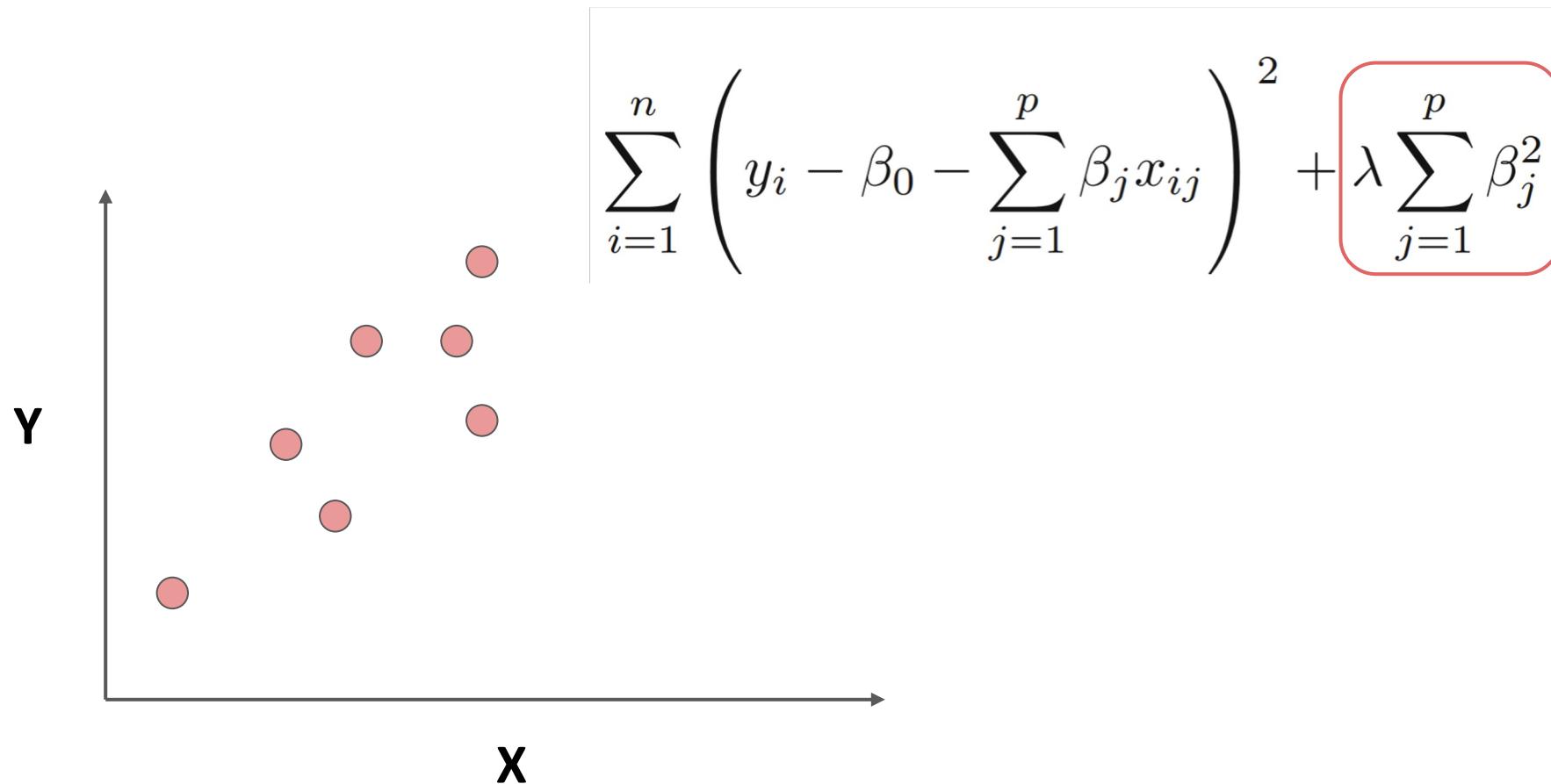
- Let's imagine trying to reduce the Ridge Regression error term:



$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

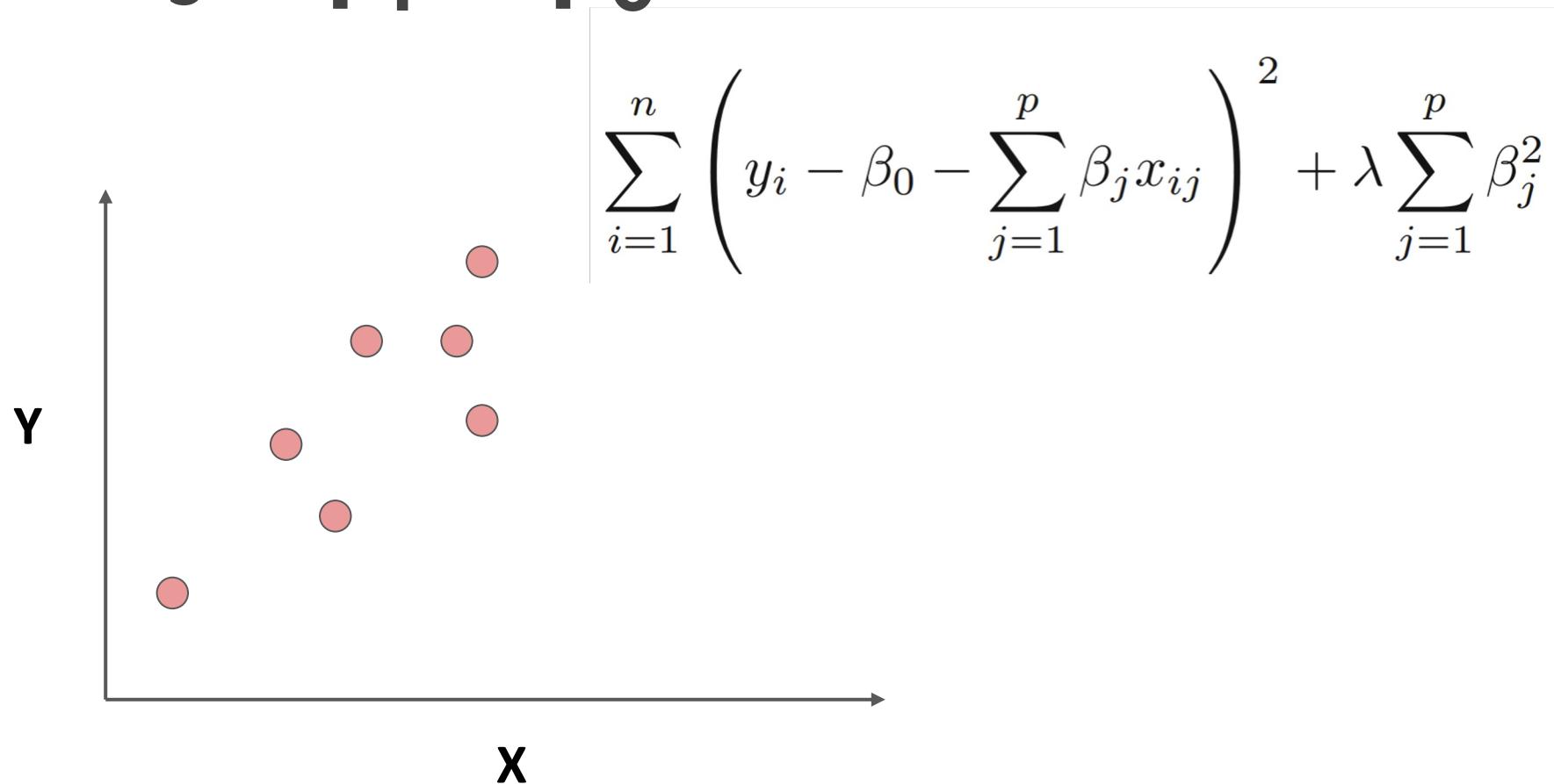
# Ridge Regression

- There is  $\lambda$  and the squared slope coefficient.



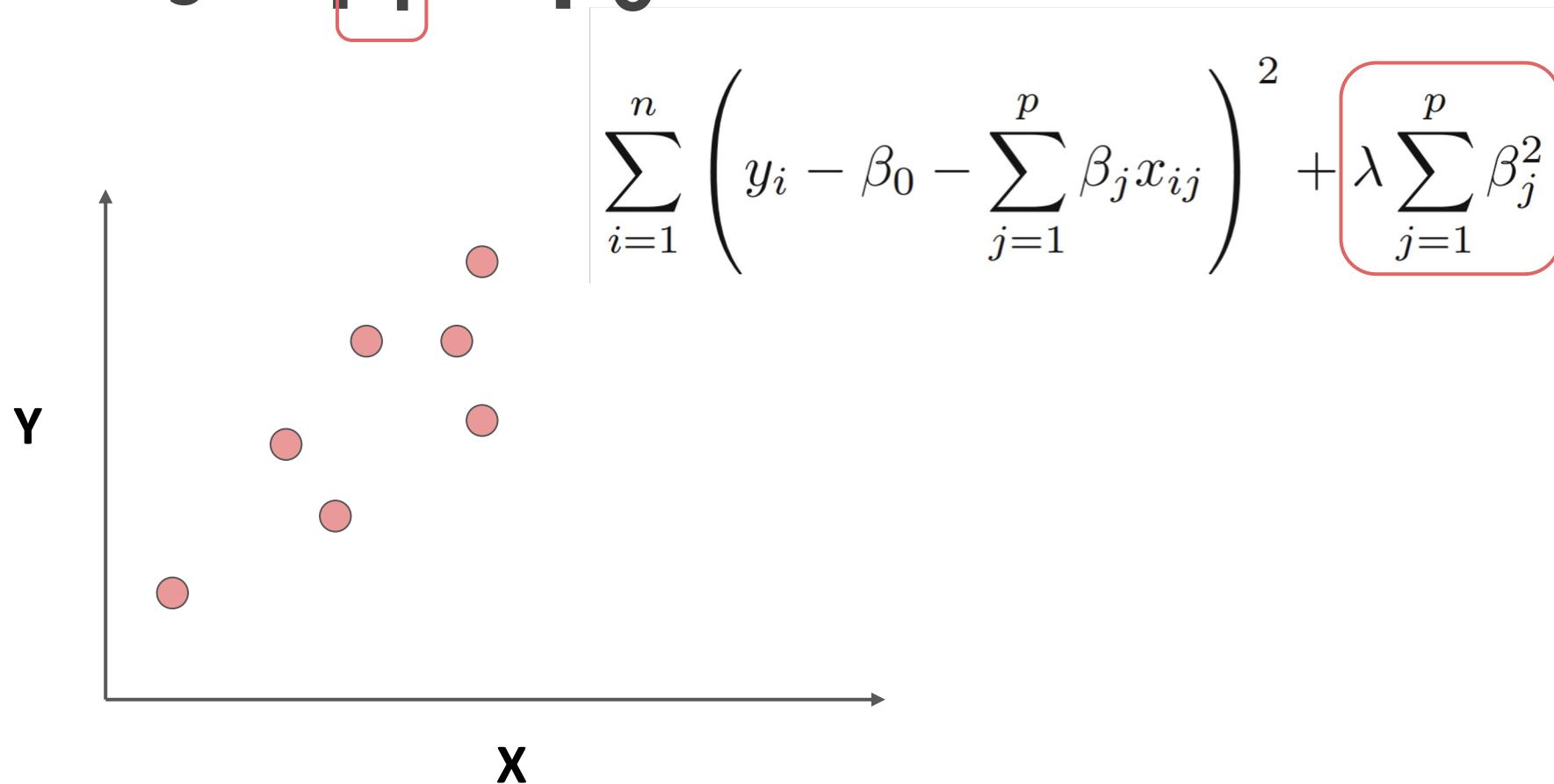
# Ridge Regression

- In the case of  $\hat{y} = \beta_1 x + \beta_0$



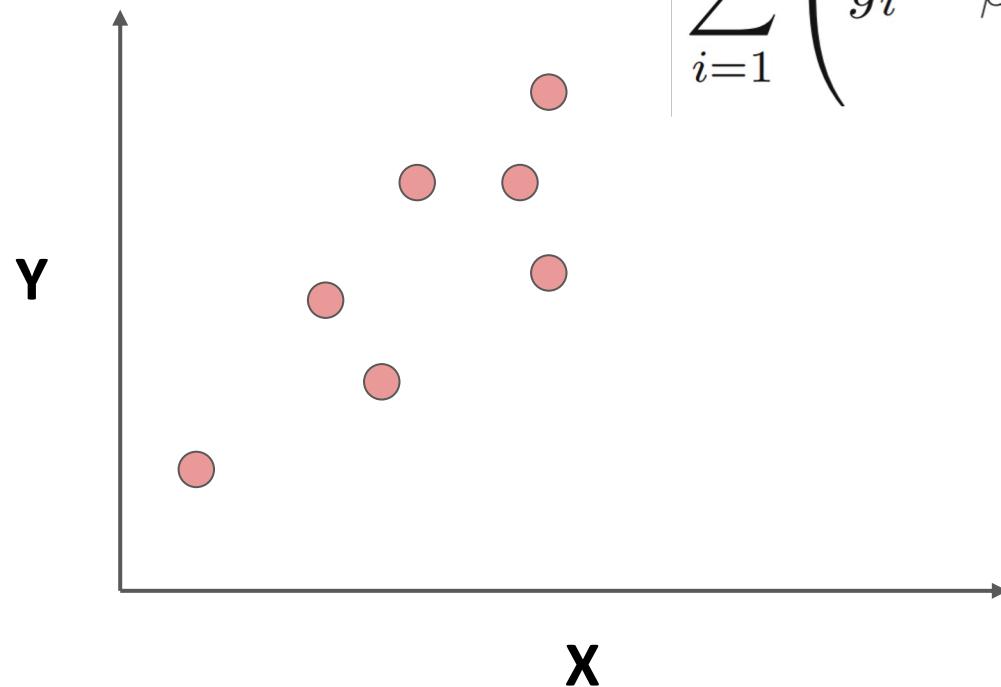
# Ridge Regression

- In the case of  $\hat{y} = \beta_1 x + \beta_0$



# Ridge Regression

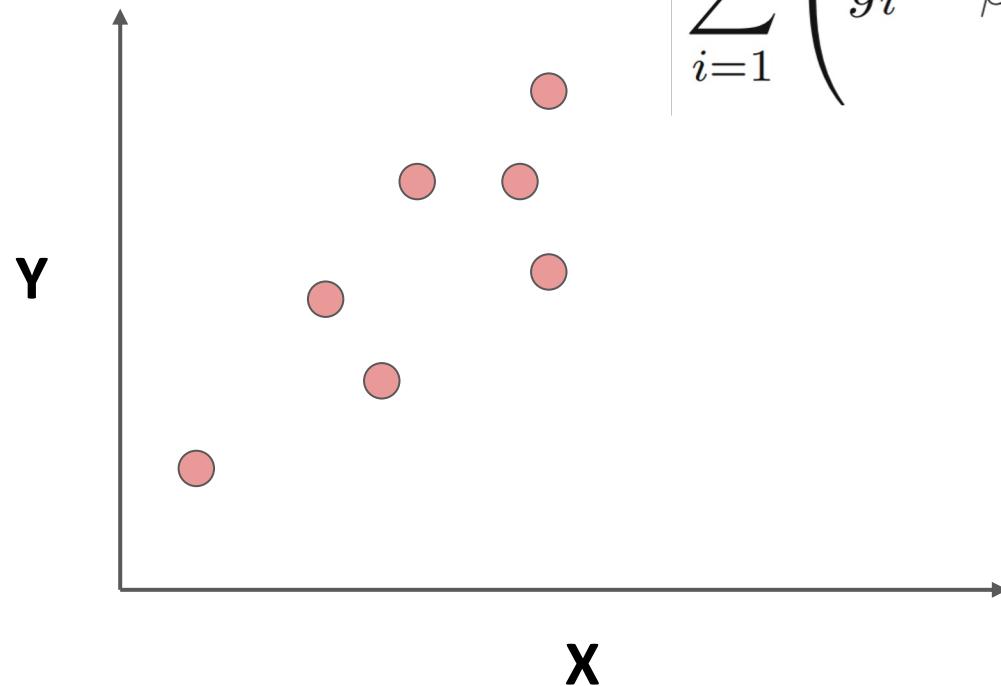
- Let's assume  $\lambda = 1$



$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

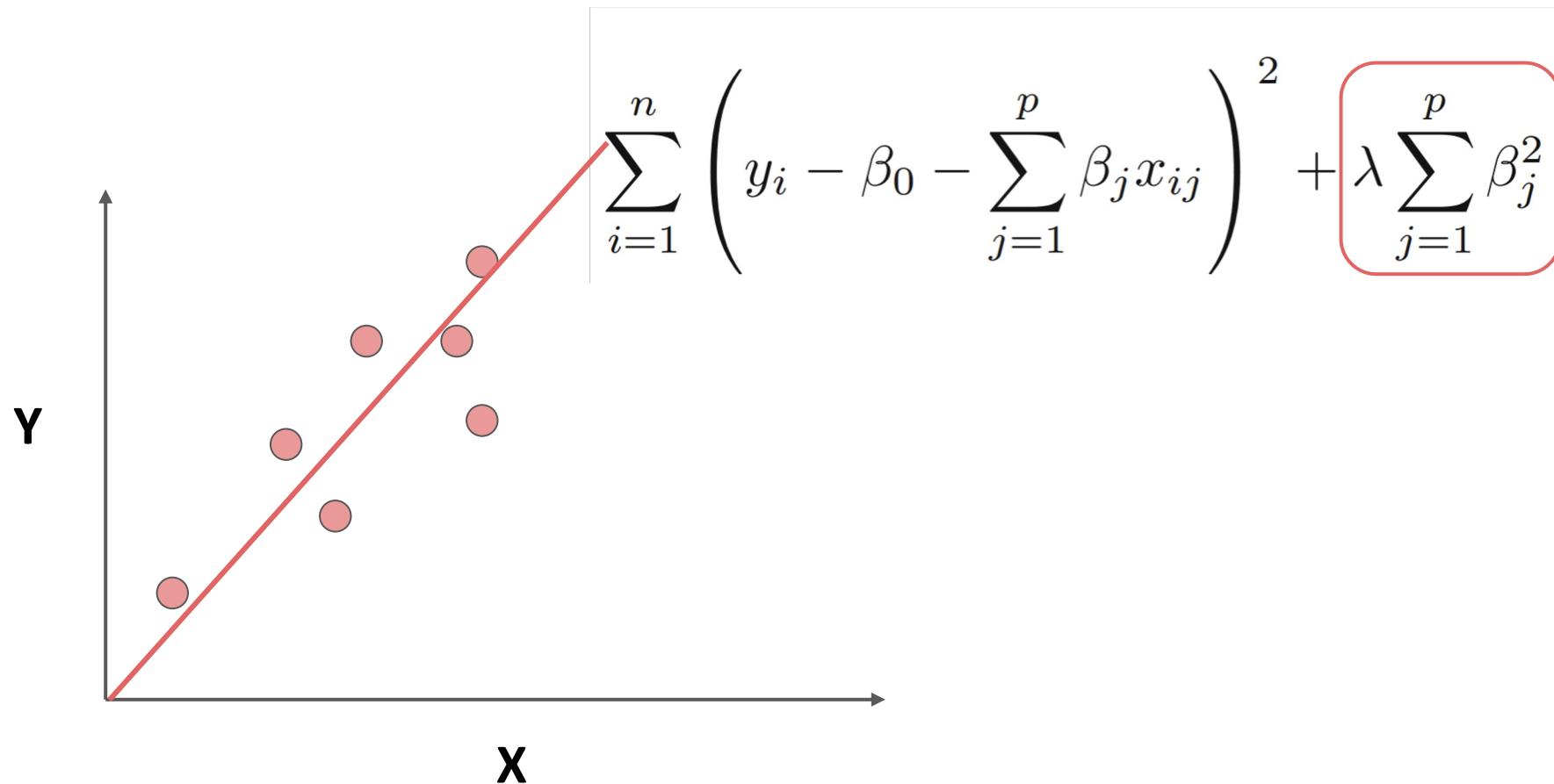
- This punishes a large slope for  $\hat{y} = \beta_1 x + \beta_0$



$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

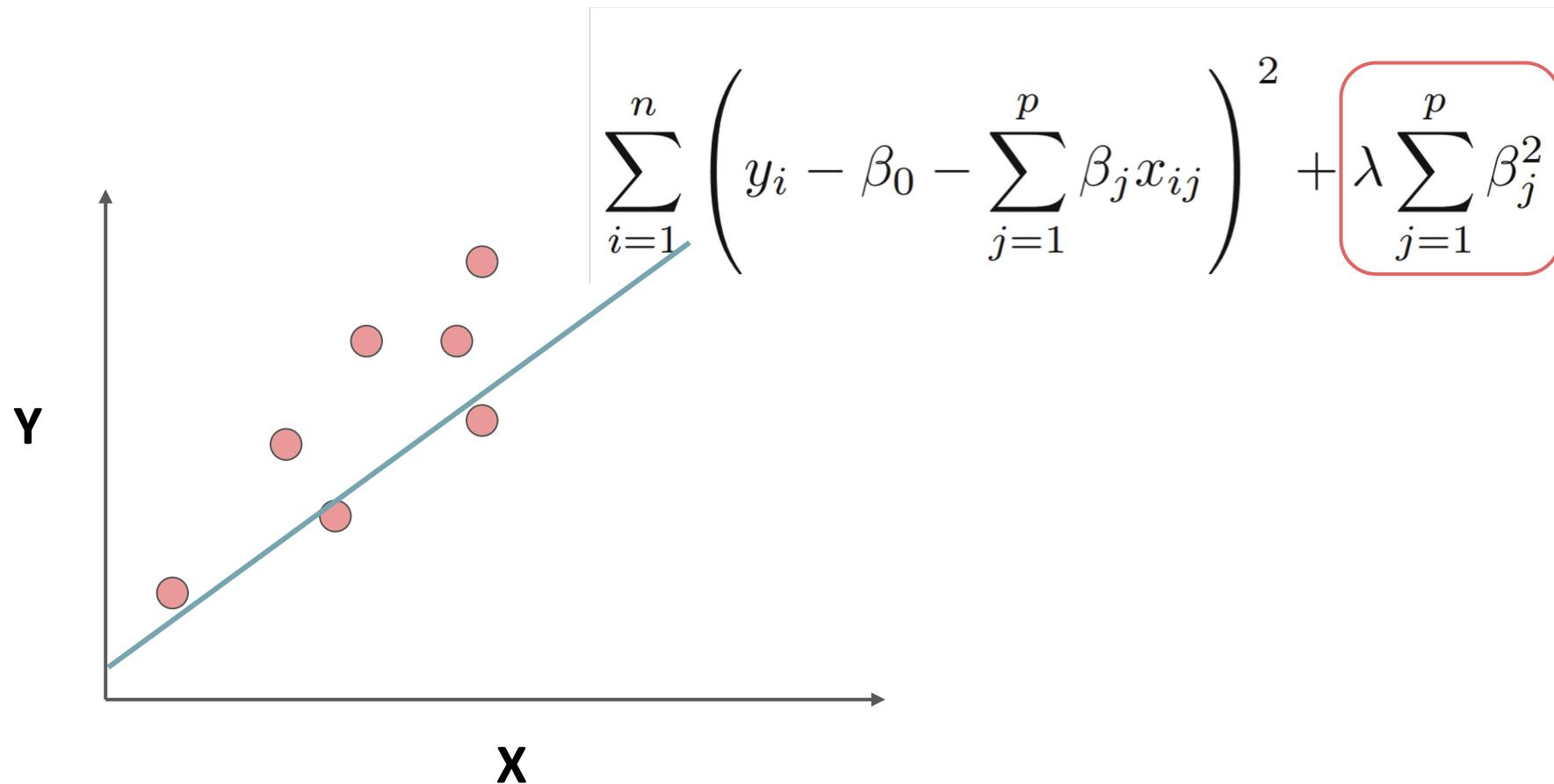
# Ridge Regression

- For single feature this lowers slope



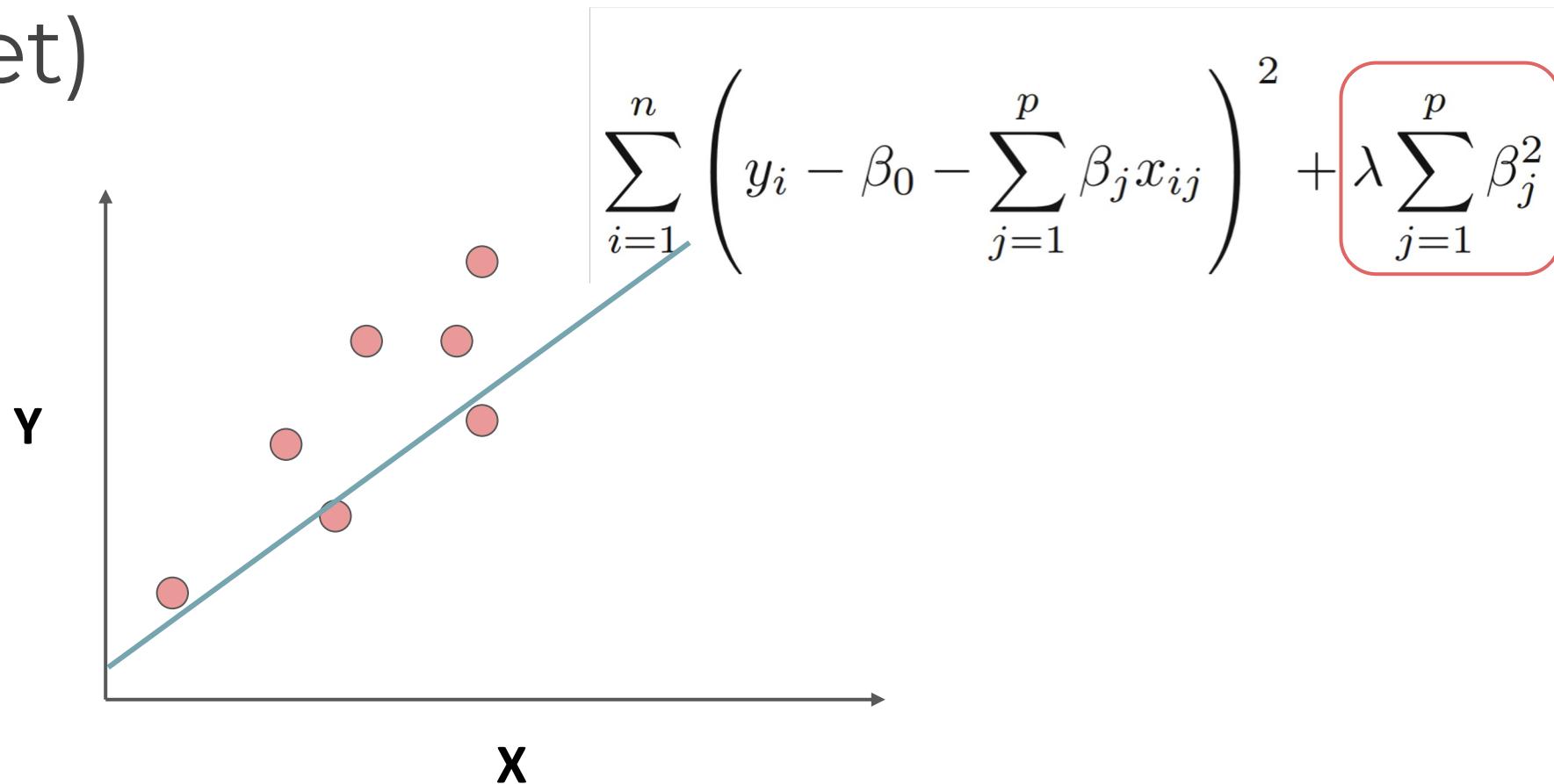
# Ridge Regression

- For single feature this lowers slope



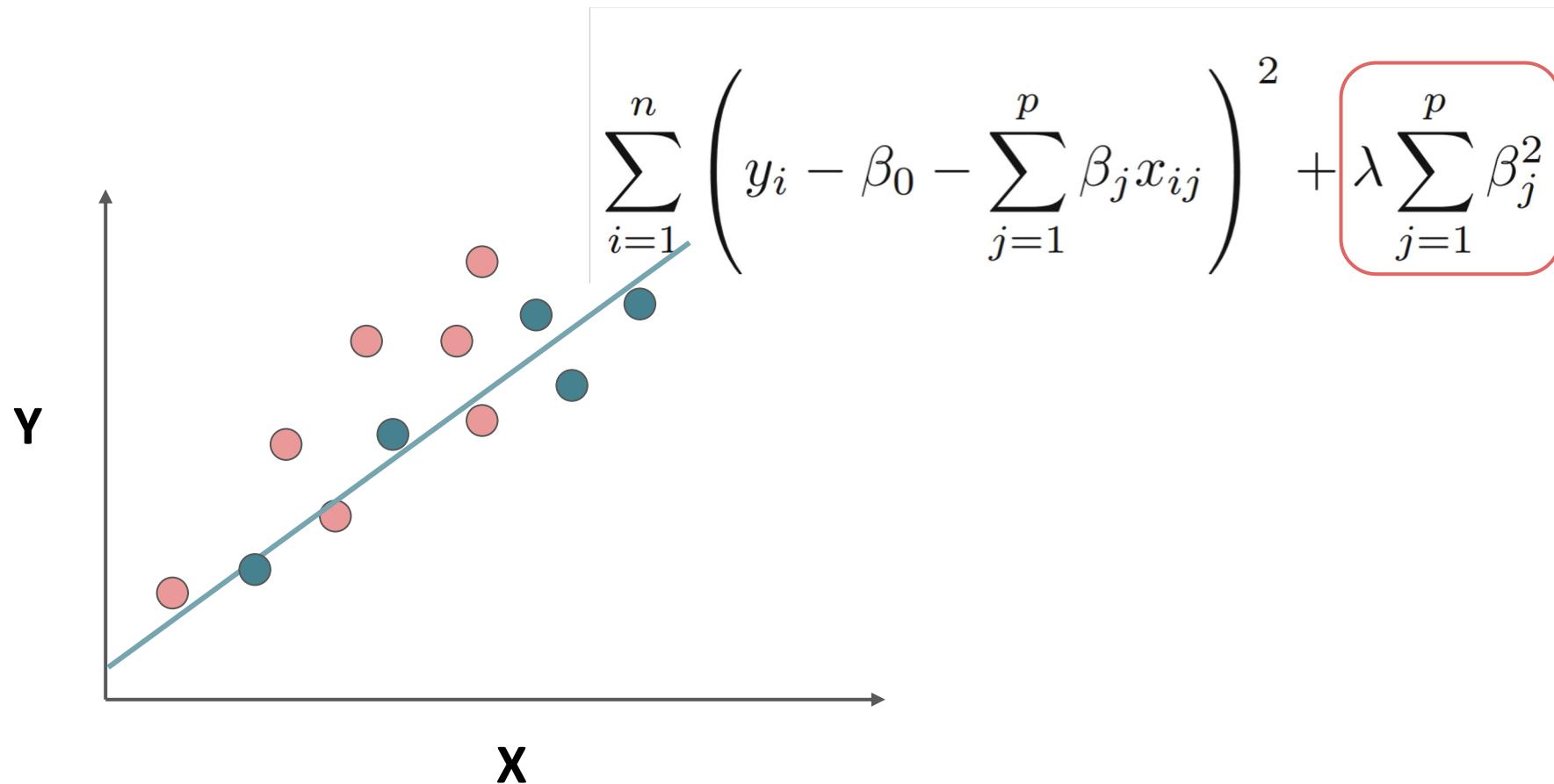
# Ridge Regression

- At the cost of some additional bias (error in training set)



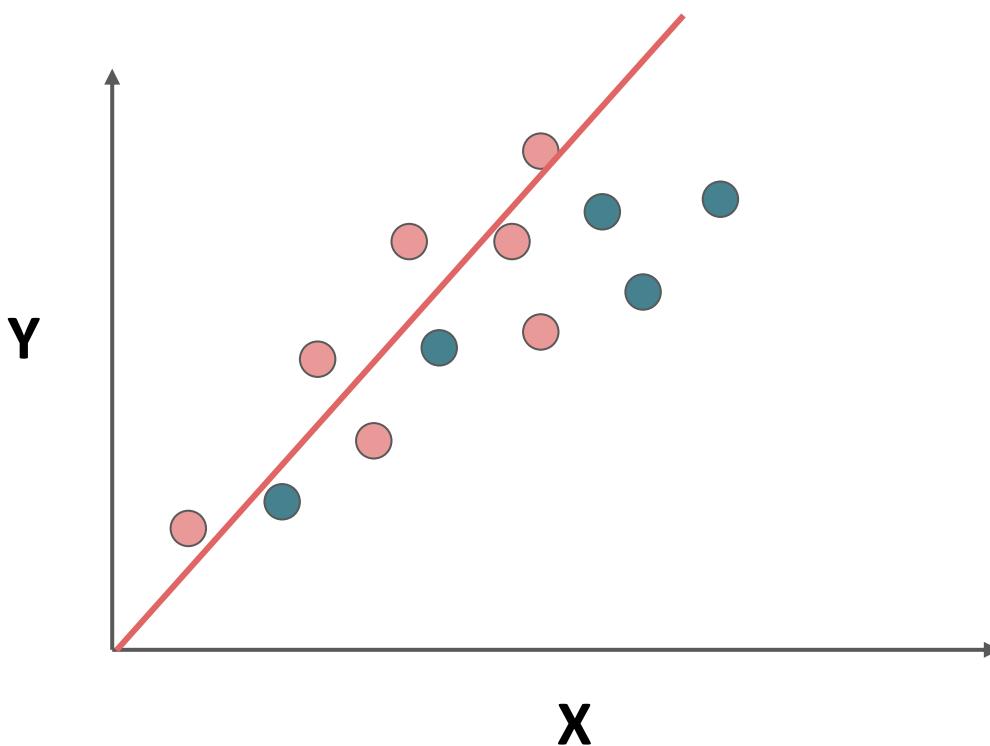
# Ridge Regression

- We generalize better to unseen data



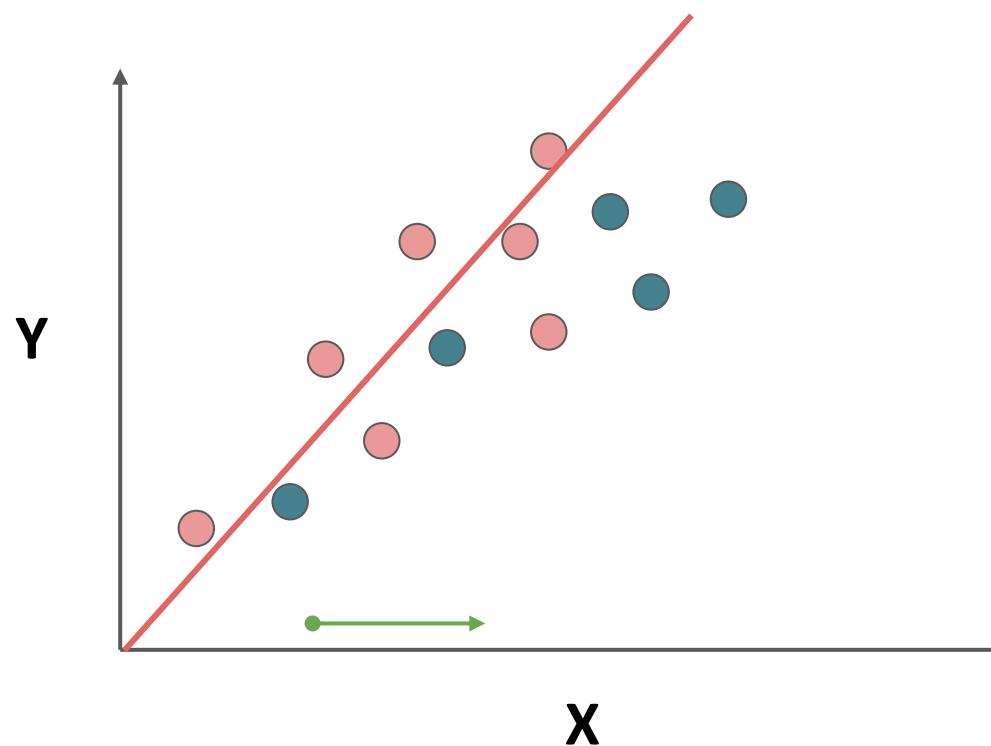
# Ridge Regression

- Consider overfitting to training set:



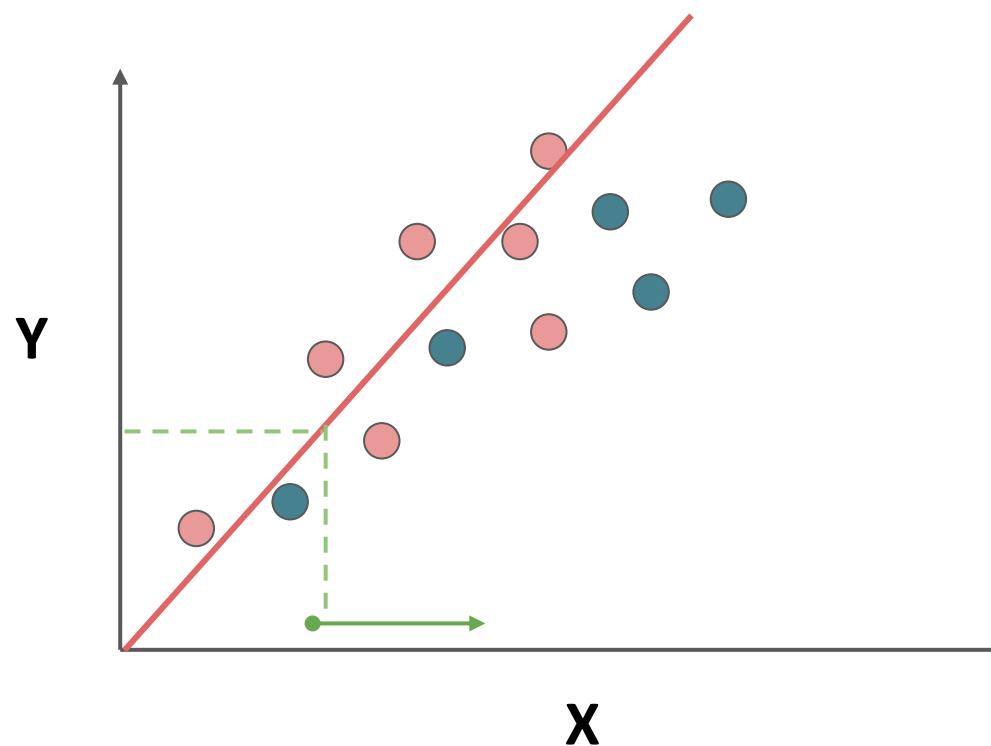
# Ridge Regression

- An increase in X results in a greater y response:



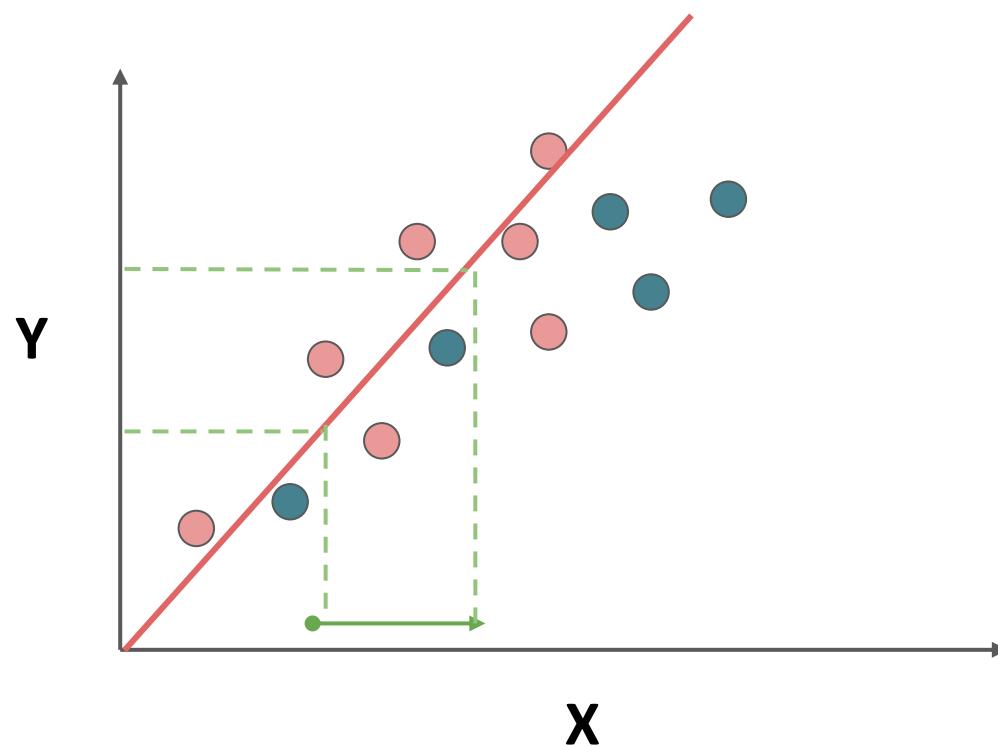
# Ridge Regression

- An increase in X results in a greater y response:



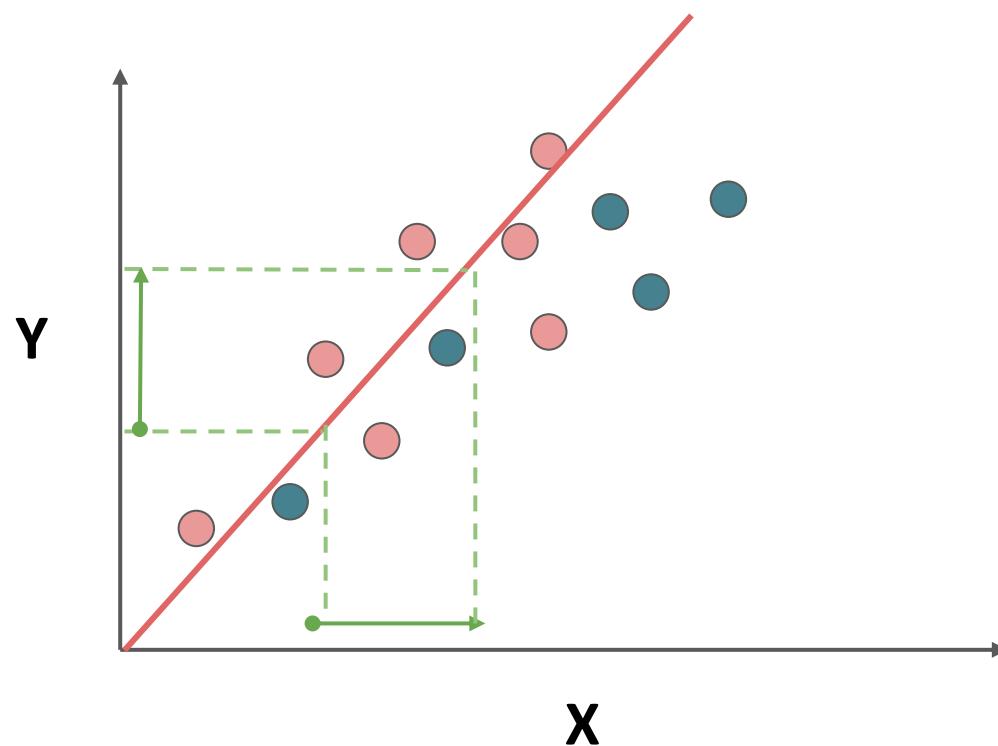
# Ridge Regression

- An increase in X results in a greater y response:



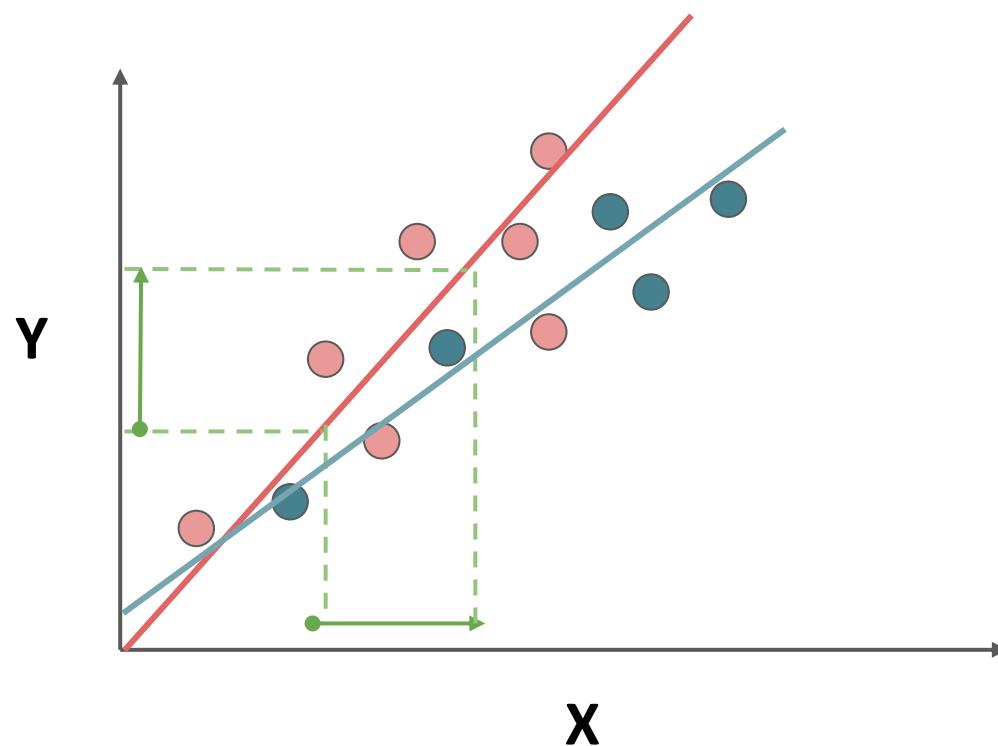
# Ridge Regression

- An increase in X results in a greater y response:



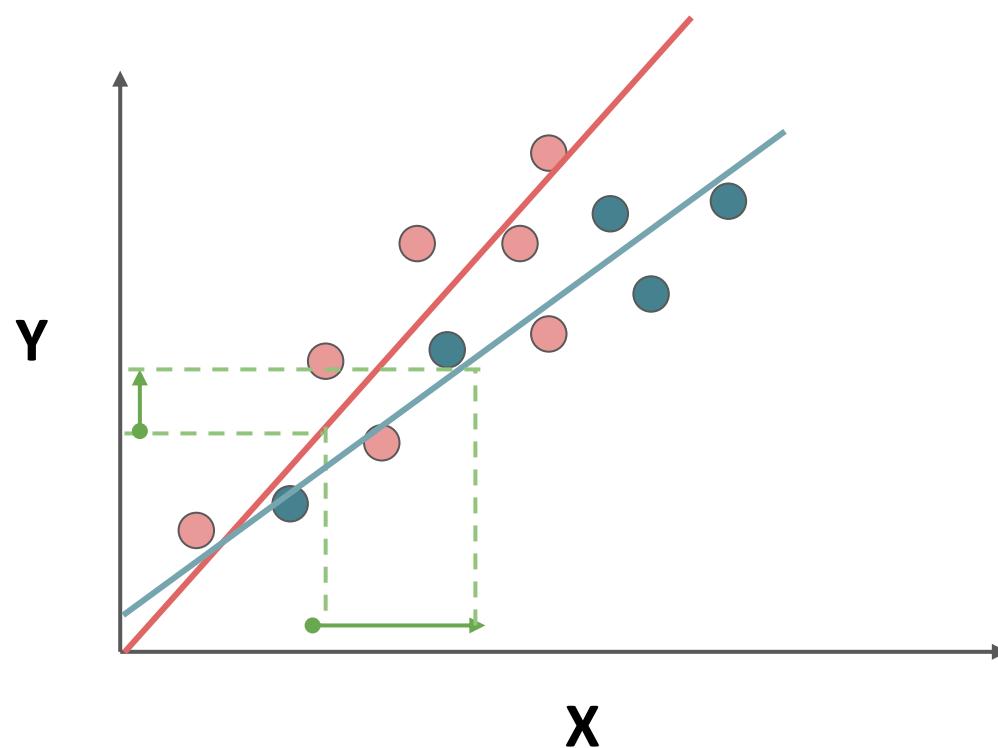
# Ridge Regression

- Compare to a more generalized model that used Ridge Regression:



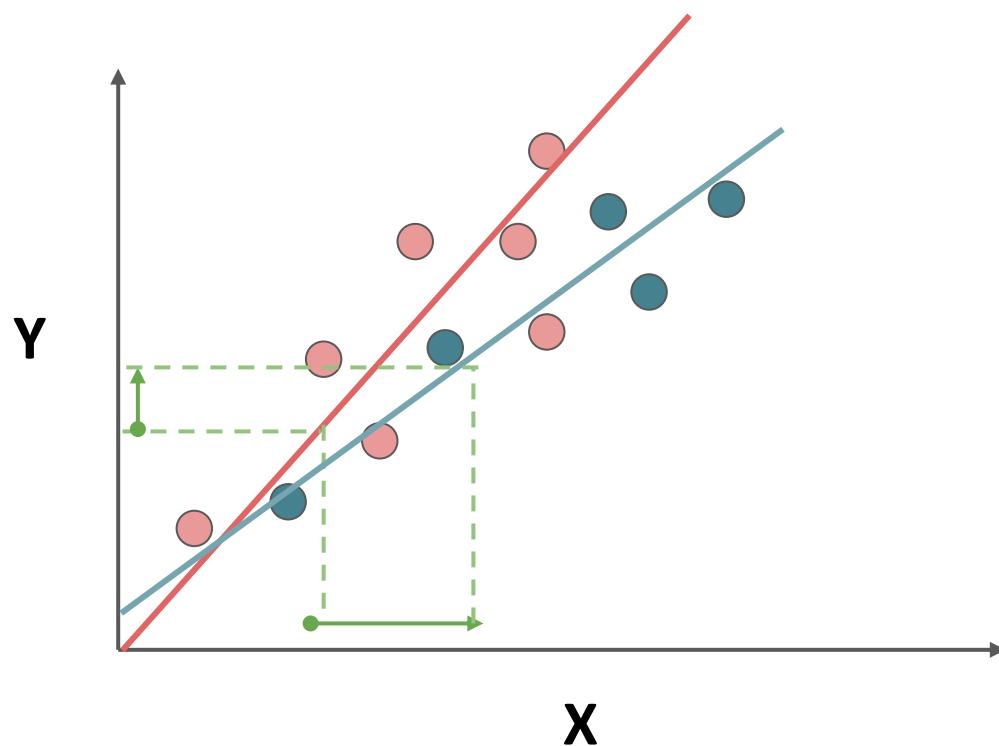
# Ridge Regression

- Same feature change does not produce as much y response:



# Ridge Regression

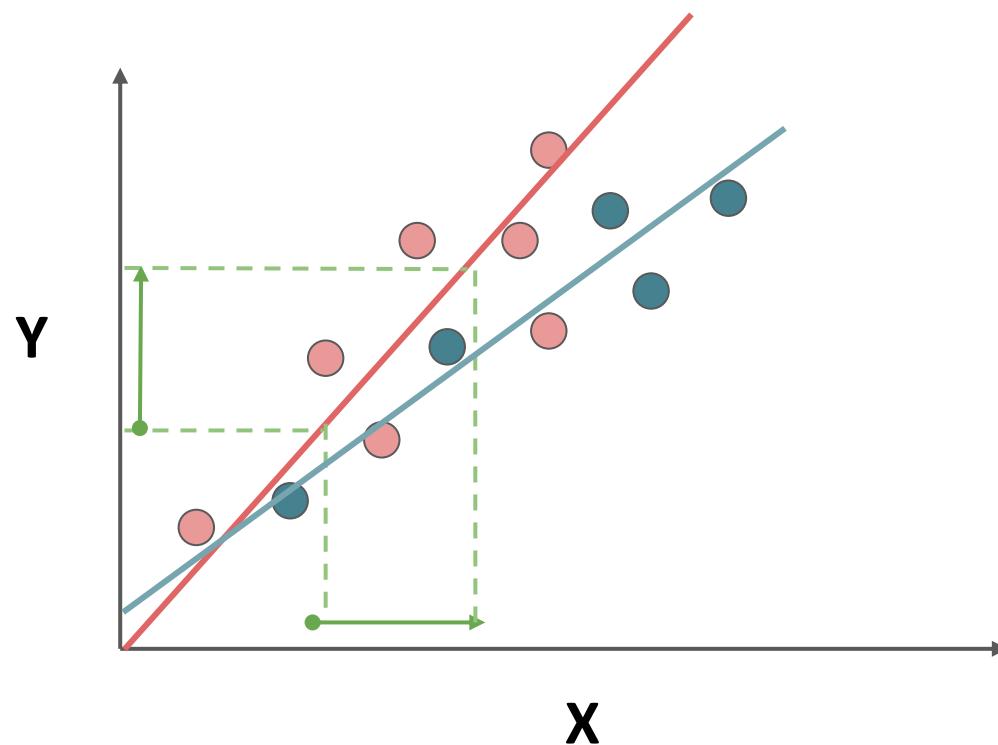
- Trying to minimize a squared Beta term leads us to punish larger coefficients.



$$\lambda \sum_{j=1}^p \beta_j^2$$

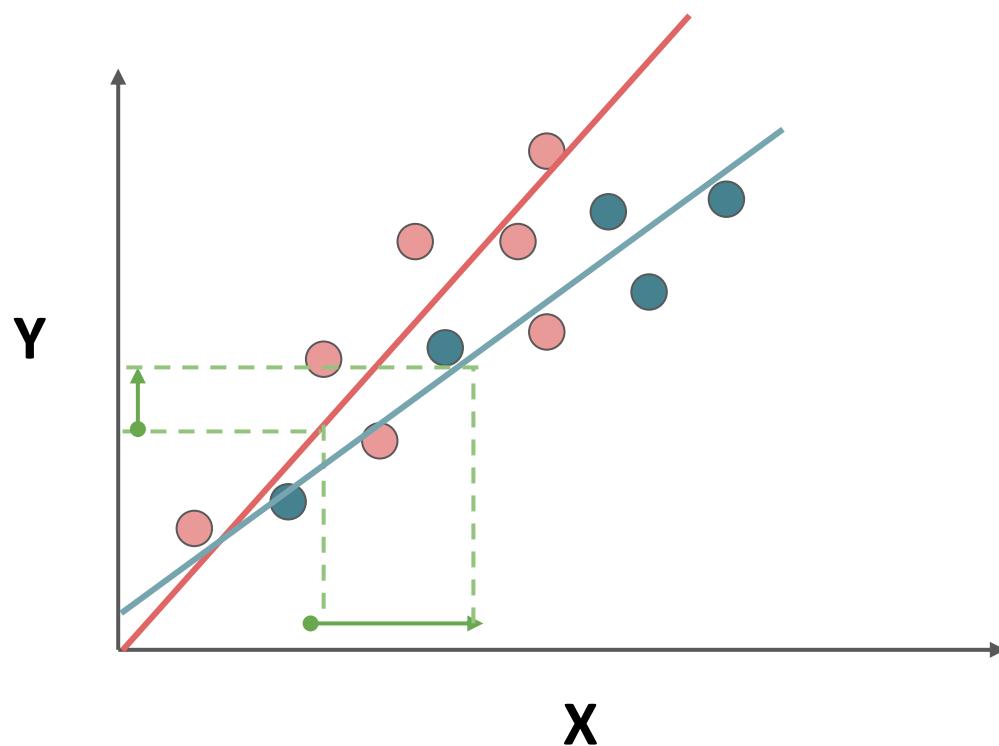
# Ridge Regression

- In the case of a single feature, a larger Beta means a steeper sloped line.



# Ridge Regression

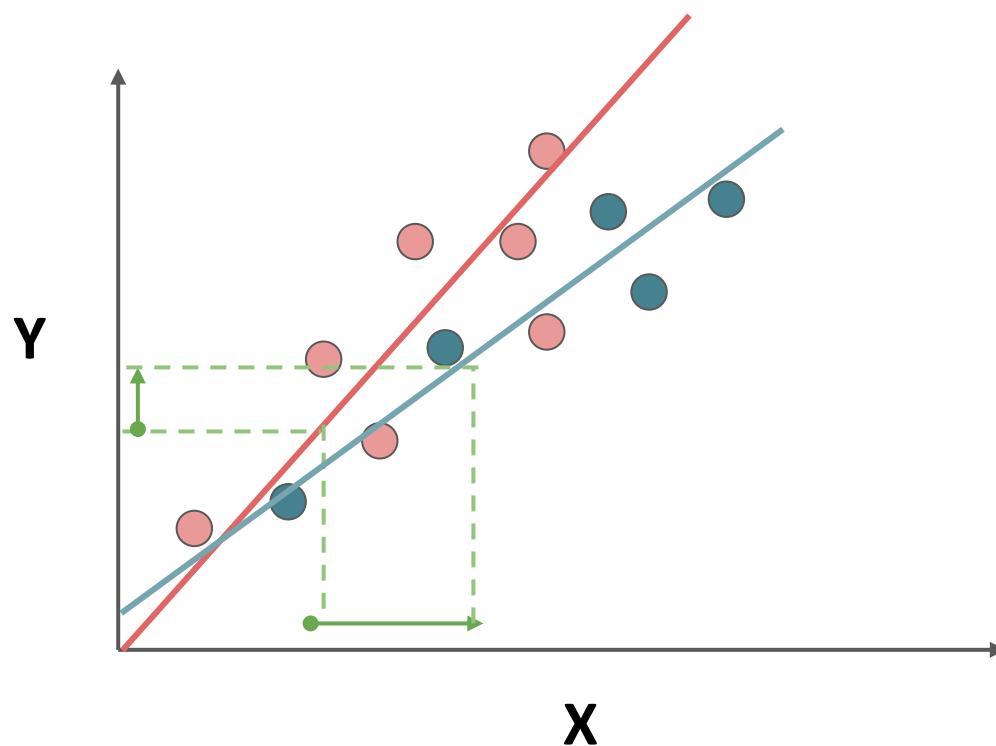
- A steeper sloped line would mean more response per increase in X value.



$$\lambda \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

- What about the lambda term? How much should we punish these larger coefficients?



$$\lambda \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

- We simply use cross-validation to explore multiple lambda options and then choose the best one!

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

L2 Regularization

# Ridge Regression

- Important Note!
  - Sklearn refers to lambda as alpha within the class call!

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \boxed{\lambda} \sum_{j=1}^p \beta_j^2$$

# Ridge Regression

- Important Note!
  - For cross validation metrics, sklearn uses a “scorer object”.
  - All scorer objects follow the convention that **higher** return values are **better** than lower return values.

# Ridge Regression

- Important Note!
  - For example, obviously higher accuracy is better.
  - But higher RMSE is actually worse!
  - So Scikit-Learn fixes this by using a **negative** RMSE as its scorer metric.

# Ridge Regression

- Important Note!
  - This allows for uniformity across **all** scorer metrics, even across different tasks types.
  - The same idea of uniformity across model classes applies to referring to the penalty strength parameter as **alpha**.

# Lasso Regression

L1 Regularization

# Regularization

- L1 regularization adds a penalty equal to the **absolute value** of the magnitude of coefficients.

$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| = \text{RSS} + \boxed{\lambda \sum_{j=1}^p |\beta_j|}$$

# Regularization

- L1 regularization adds a penalty equal to the **absolute value** of the magnitude of coefficients.
  - Limits the size of the coefficients.
  - Can yield sparse models where some coefficients can become zero.

# Regularization

- LASSO can force some of the coefficient estimates to be exactly equal to zero when the tuning parameter  $\lambda$  is sufficiently large.
- Similar to subset selection, the LASSO performs variable selection.
- Models generated from the LASSO are generally much easier to interpret.

# Regularization

- LassoCV with sklearn operates on checking a number of alphas within a range, instead of providing the alphas directly.
- Let's explore the results of LASSO in Python and Scikit-Learn!

# Elastic Net

L1 and L2 Regularization

# Regularization

- We've been able to perform Ridge and Lasso regression.
- We know Lasso is able to shrink coefficients to zero, but we haven't taken a deeper dive into how or why that is.
- This ability becomes more clear when learning about **elastic net** which combines Lasso and Ridge together!

# Regularization

- Let's dive a little deeper into Lasso.
- Lasso was originally introduced in geophysics literature in 1986 by Symes and Santosa.
- It was later independently rediscovered and popularized in 1996 by Robert Tibshirani who coined the term “Lasso”.

# Regularization

- Does the name Robert Tibshirani sound familiar?

# Regularization

- We can rewrite Lasso and Ridge:

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^p |\beta_j| \leq s$$

and

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^p \beta_j^2 \leq s,$$

# Regularization

- There is some sum  $s$  which allows to rewrite the penalty as a requirement:

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^p |\beta_j| \leq s$$

and

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^p \beta_j^2 \leq s,$$

# Regularization

- There is some sum  $s$  which allows to rewrite the penalty as a requirement:

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \right\}$$

subject to  $\sum_{j=1}^p |\beta_j| \leq s$

and

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \right\}$$

subject to  $\sum_{j=1}^p \beta_j^2 \leq s$ ,

# Elastic Net

- Start with a simple thought experiment:
  - A simple equation:
    - $\hat{y} = \beta_1 x_1 + \beta_2 x_2$
  - We know that regularization can be expressed as an additional requirement that RSS is subject to.

# Elastic Net

- Start with a simple thought experiment:
  - A simple equation:
    - $\hat{y} = \beta_1 x_1 + \beta_2 x_2$
  - L1 constrains the sum of absolute values.
    - $\sum |\beta|$
  - L2 constrains the sum of squared values.
    - $\sum \beta^2$

# Elastic Net

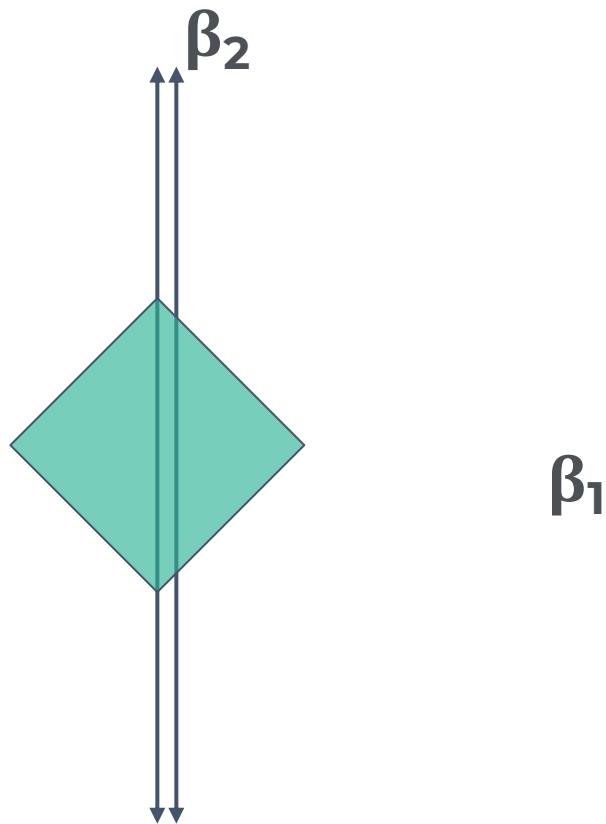
- Start with a simple thought experiment:
  - A simple equation:
    - $\hat{y} = \beta_1 x_1 + \beta_2 x_2$
  - There is some sum **s** that the penalty is less than.

# Elastic Net

- For the case of only two features:
  - $\hat{y} = \beta_1 x_1 + \beta_2 x_2$
- Lasso Regression Penalty:
  - $|\beta_1| + |\beta_2| \leq s$
- Ridge Regression Penalty:
  - $\beta_1^2 + \beta_2^2 \leq s$

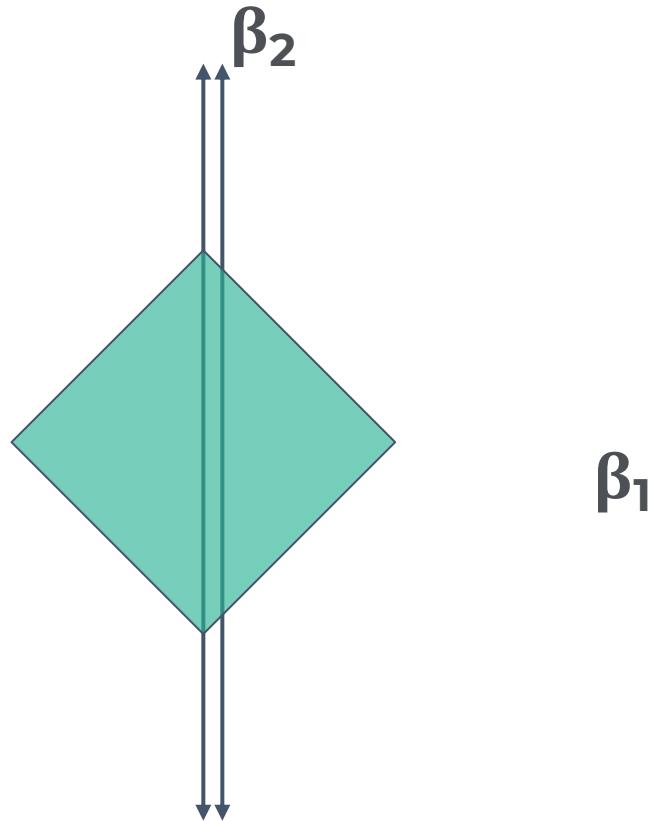
# Elastic Net

- Let's plot Lasso:  $|\beta_1| + |\beta_2| \leq s$



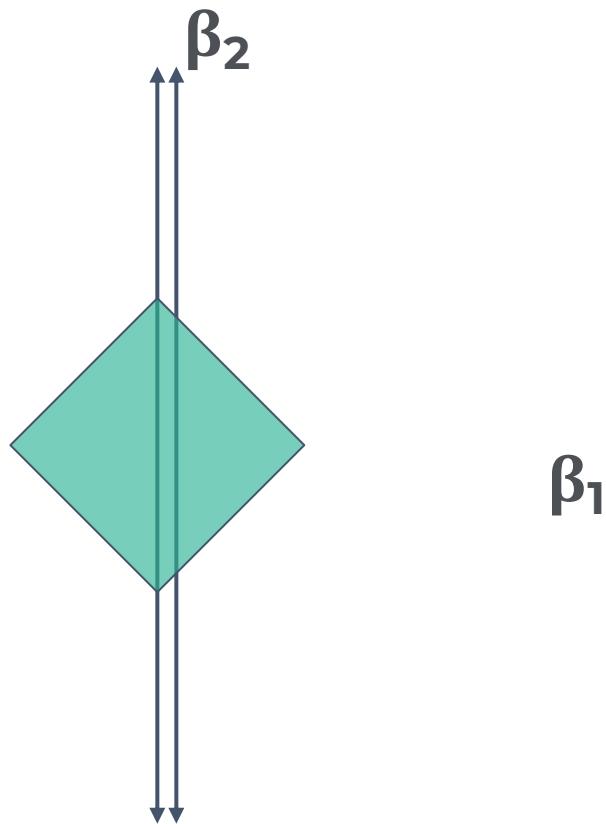
# Elastic Net

- Let's plot Lasso:  $|\beta_1| + |\beta_2| \leq s$



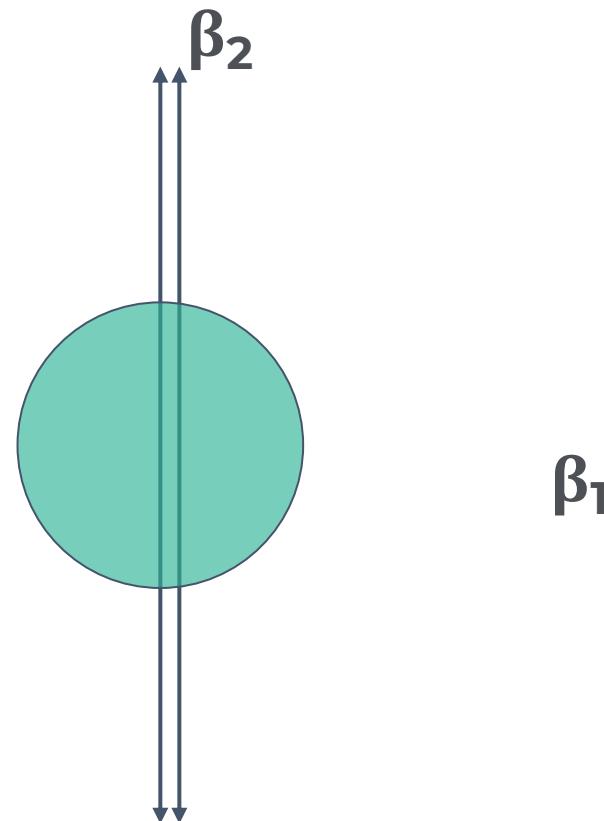
# Elastic Net

- Let's plot Lasso:  $|\beta_1| + |\beta_2| \leq s$



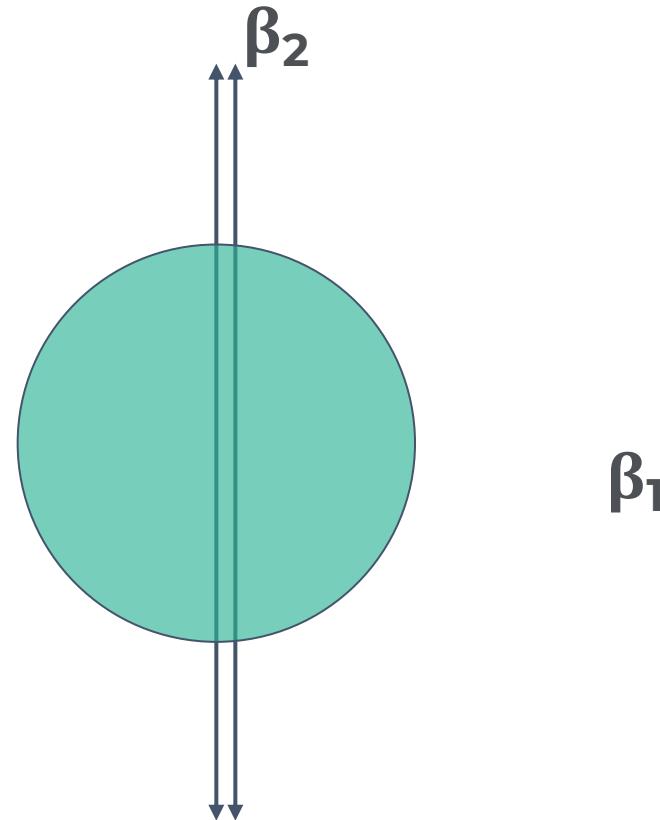
# Elastic Net

- Let's plot Ridge:  $\beta_1^2 + \beta_2^2 \leq s$



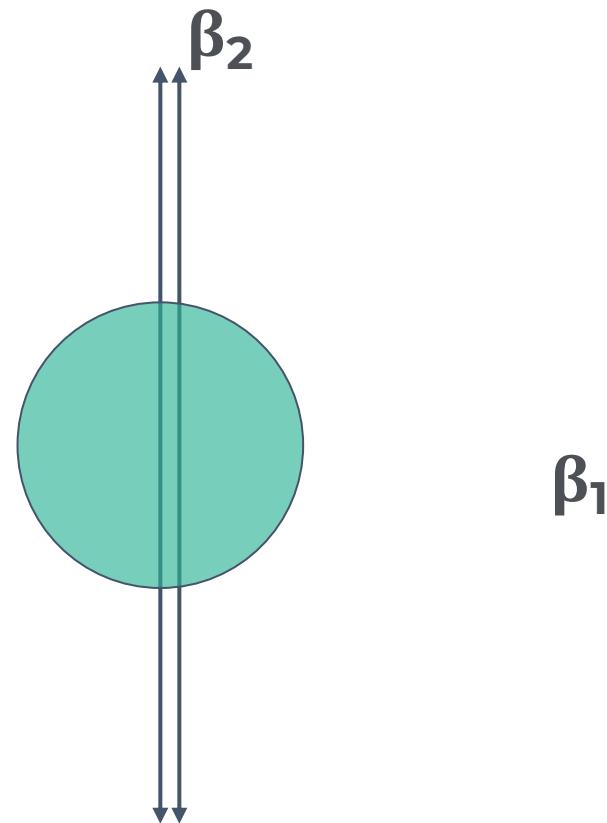
# Elastic Net

- Let's plot Ridge:  $\beta_1^2 + \beta_2^2 \leq s$



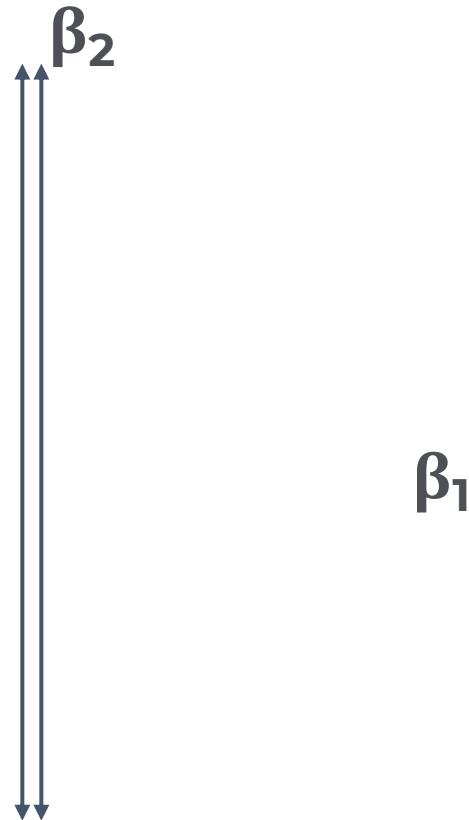
# Elastic Net

- Let's plot Ridge:  $\beta_1^2 + \beta_2^2 \leq s$



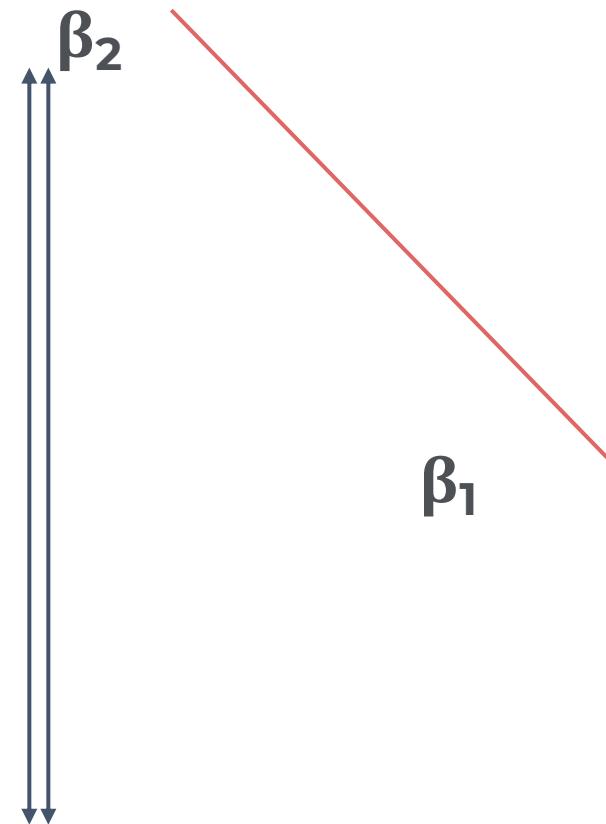
# Elastic Net

- What would RSS look like?



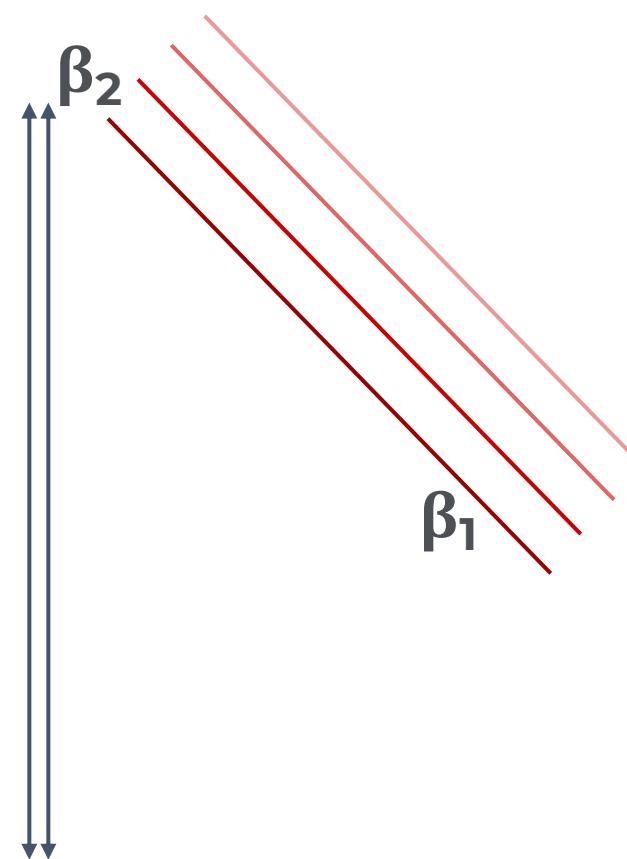
# Elastic Net

- What would RSS look like?



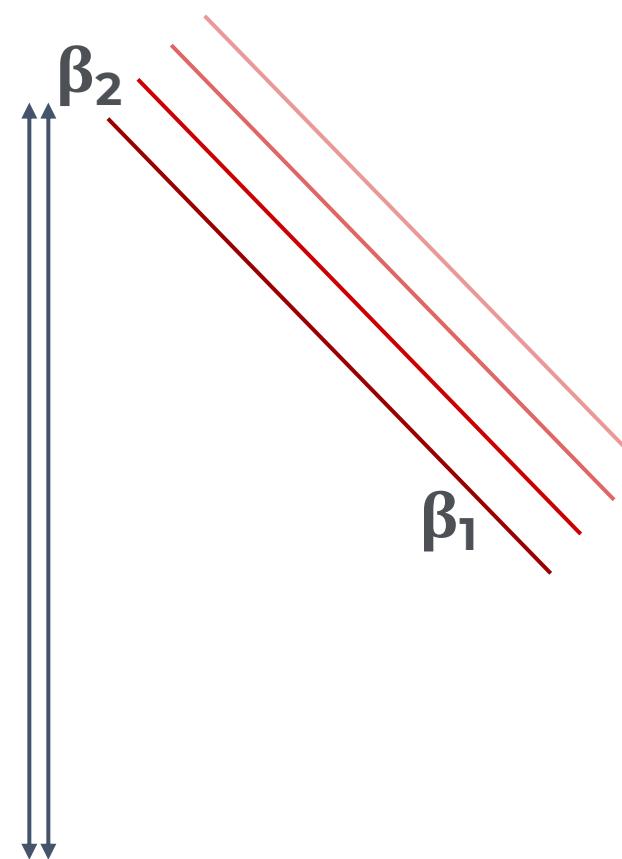
# Elastic Net

- What would RSS look like?



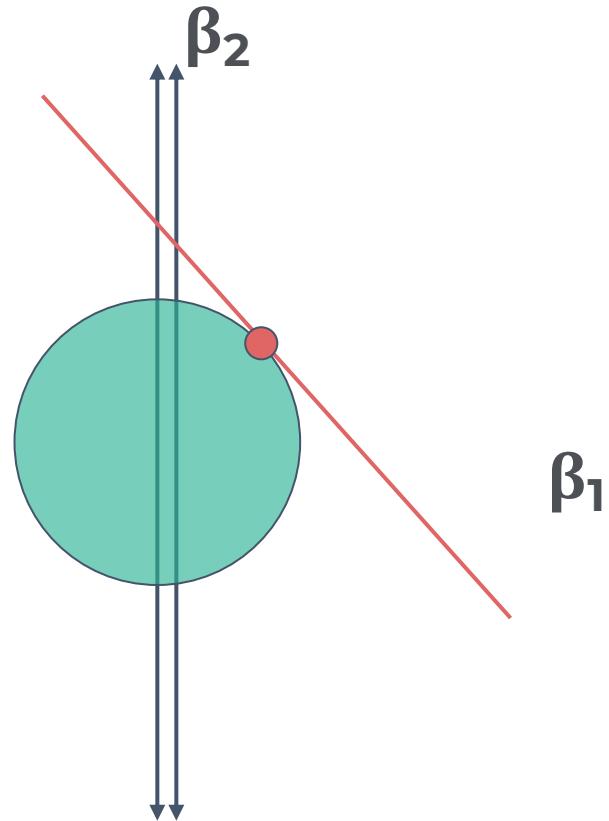
# Elastic Net

- But were subject to the penalty!



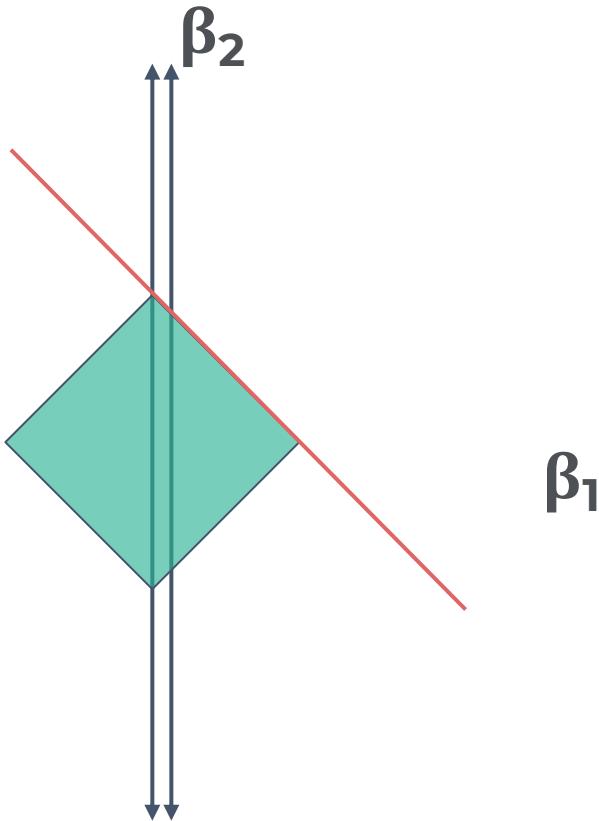
# Elastic Net

- Penalty for Ridge:  $\beta_1^2 + \beta_2^2 \leq s$



# Elastic Net

- Penalty for Lasso:  $|\beta_1| + |\beta_2| \leq s$



# Elastic Net

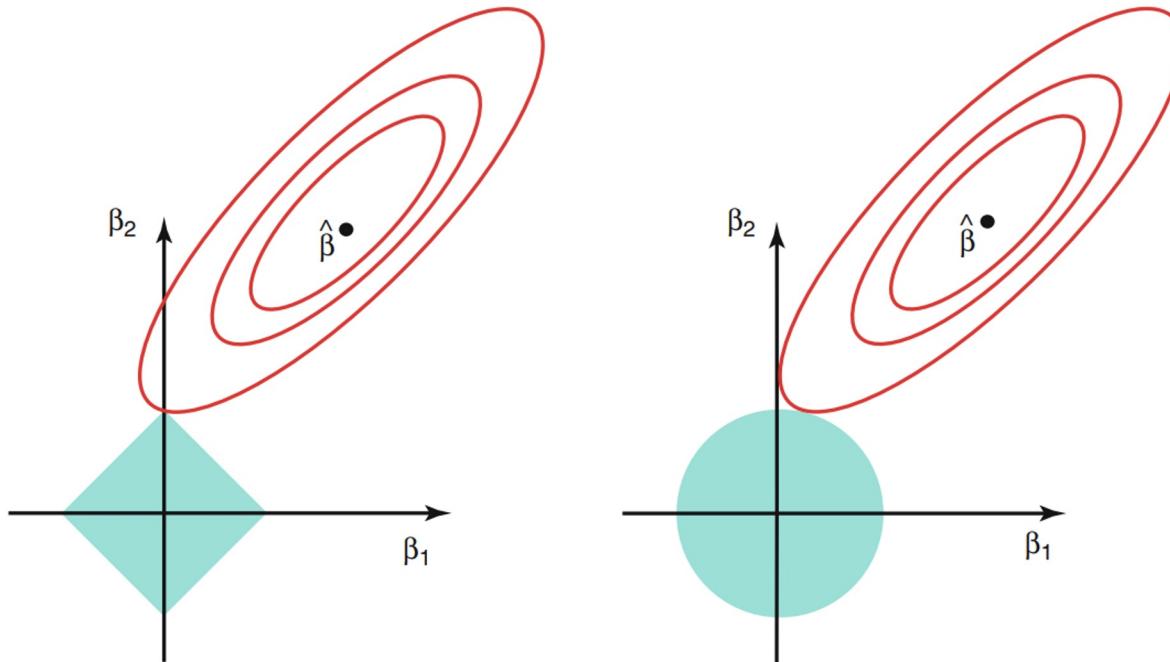
- Lasso:
  - A convex object that lies tangent to the boundary, is likely to encounter a corner of a hypercube, for which some components of  $\beta$  are identically zero.

# Elastic Net

- Ridge: In the case of an n-sphere, the points on the boundary for which some of the components of  $\beta$  are zero are not distinguished from the others and the convex object is no more likely to contact a point at which some components of  $\beta$  are zero than one for which none of them are.

# Elastic Net

- This is why Lasso is more likely to lead to coefficients as zero.
- This diagram is also commonly shown with contour RSS:



# Elastic Net

- Elastic Net seeks to improve on both L1 and L2 Regularization by combining them:
- 

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda_1 \sum_{j=1}^p \beta_j^2 + \lambda_2 \sum_{j=1}^p |\beta_j|$$

# Elastic Net

- Here we seek to minimize RSS and **both** the squared and absolute value terms:
- 

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda_1 \sum_{j=1}^p \beta_j^2 + \lambda_2 \sum_{j=1}^p |\beta_j|$$

# Elastic Net

- Notice there are **two** distinct lambda values for each penalty:
- 

$$\text{Error} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda_1 \sum_{j=1}^p \beta_j^2 + \lambda_2 \sum_{j=1}^p |\beta_j|$$

# Elastic Net

- We can alternatively express this as a ratio between L1 and L2:

$$\frac{\sum_{i=1}^n (y_i - x_i^J \hat{\beta})^2}{2n} + \lambda \left( \frac{1-\alpha}{2} \sum_{j=1}^m \hat{\beta}_j^2 + \alpha \sum_{j=1}^m |\hat{\beta}_j| \right)$$

# Elastic Net

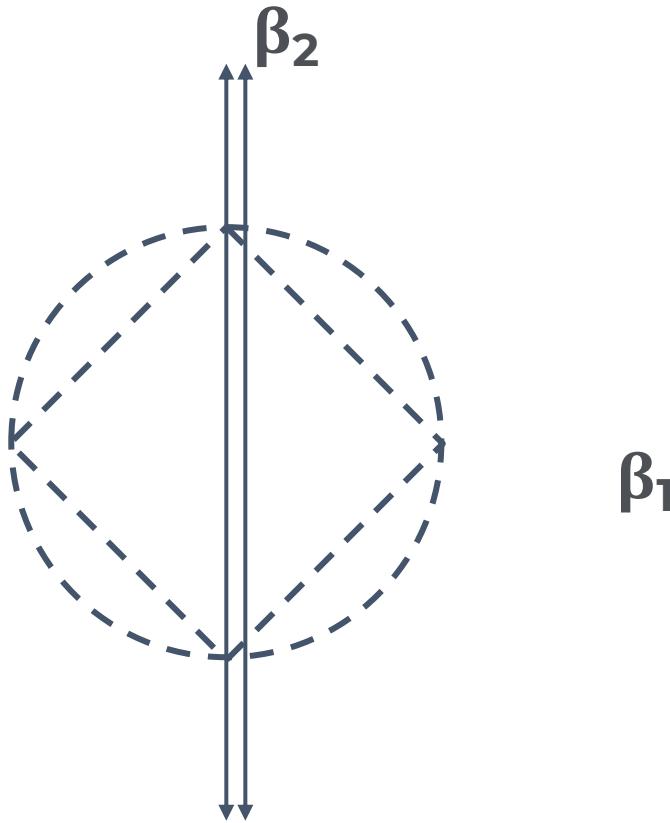
- We can also use simplified notation:

---

$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}} (\|y - X\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1)$$

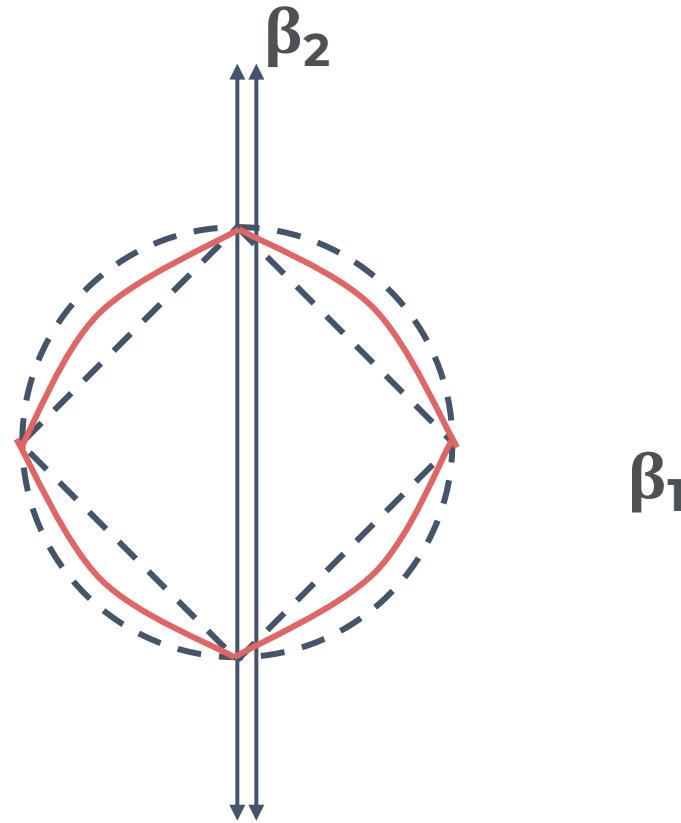
# Elastic Net

- Elastic Net Penalty Region:



# Elastic Net

- Elastic Net Penalty Region:



# Elastic Net

- Let's explore how to perform Elastic Net with Python and Scikit-learn!

$$\frac{\sum_{i=1}^n (y_i - x_i^J \hat{\beta})^2}{2n} + \lambda \left( \frac{1-\alpha}{2} \sum_{j=1}^m \hat{\beta}_j^2 + \alpha \sum_{j=1}^m |\hat{\beta}_j| \right)$$