

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!



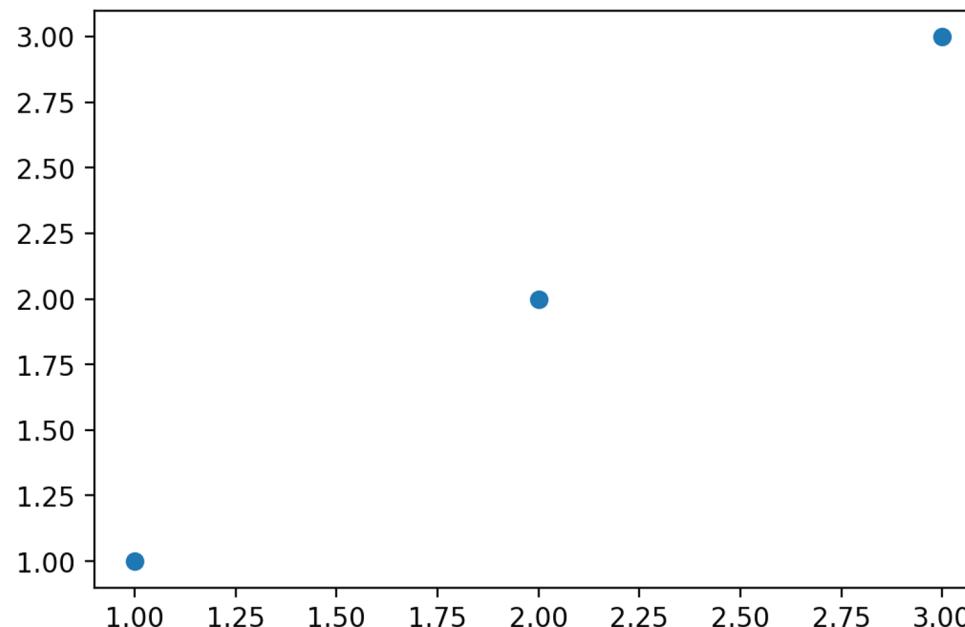
# Linear Regression

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



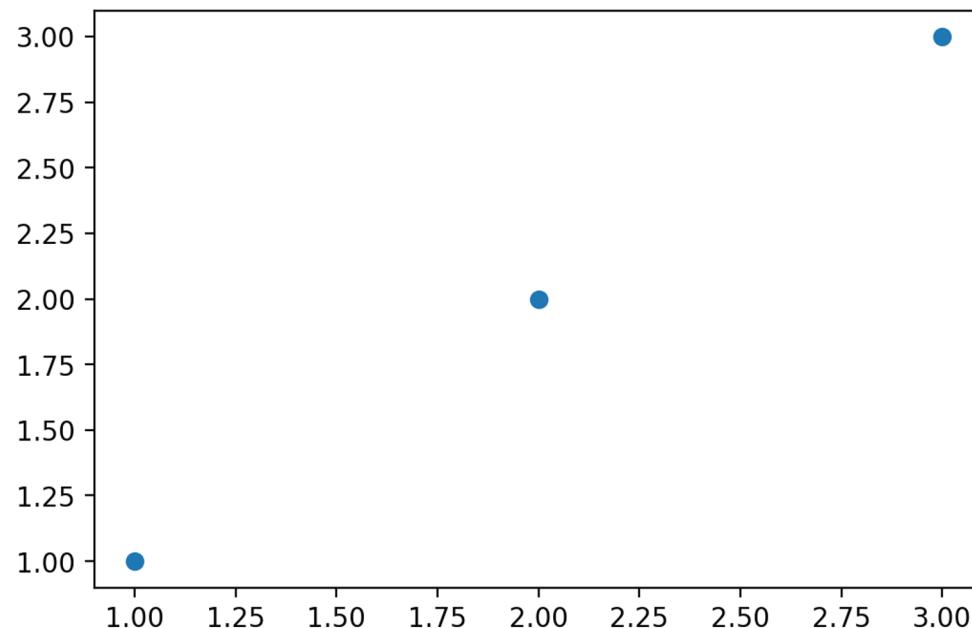
# Linear Regression

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



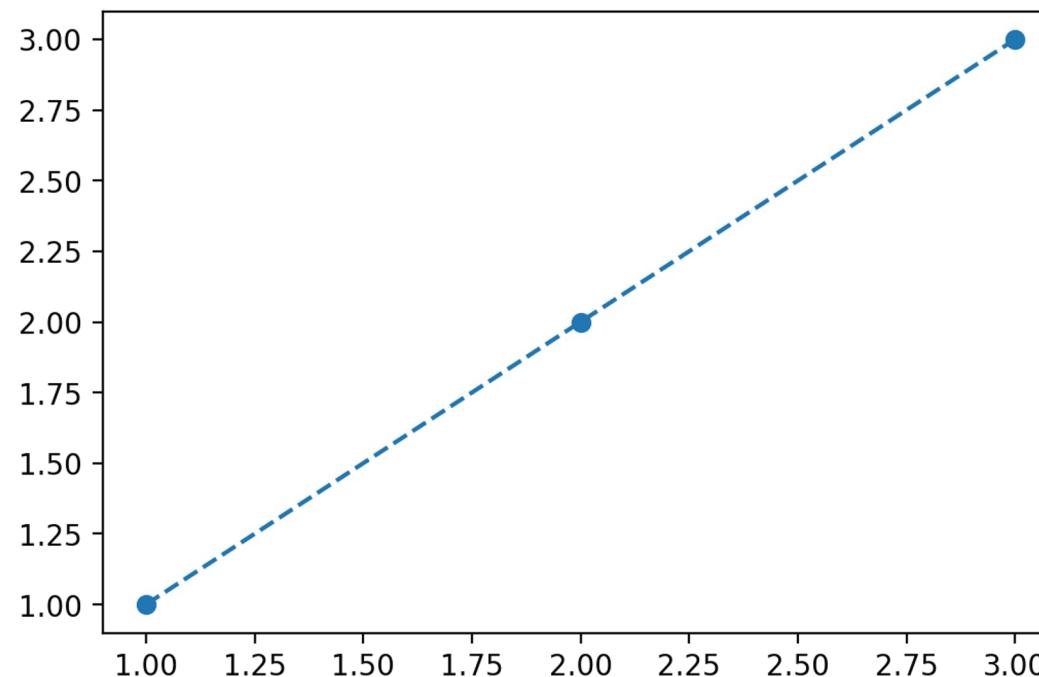
# Linear Regression

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



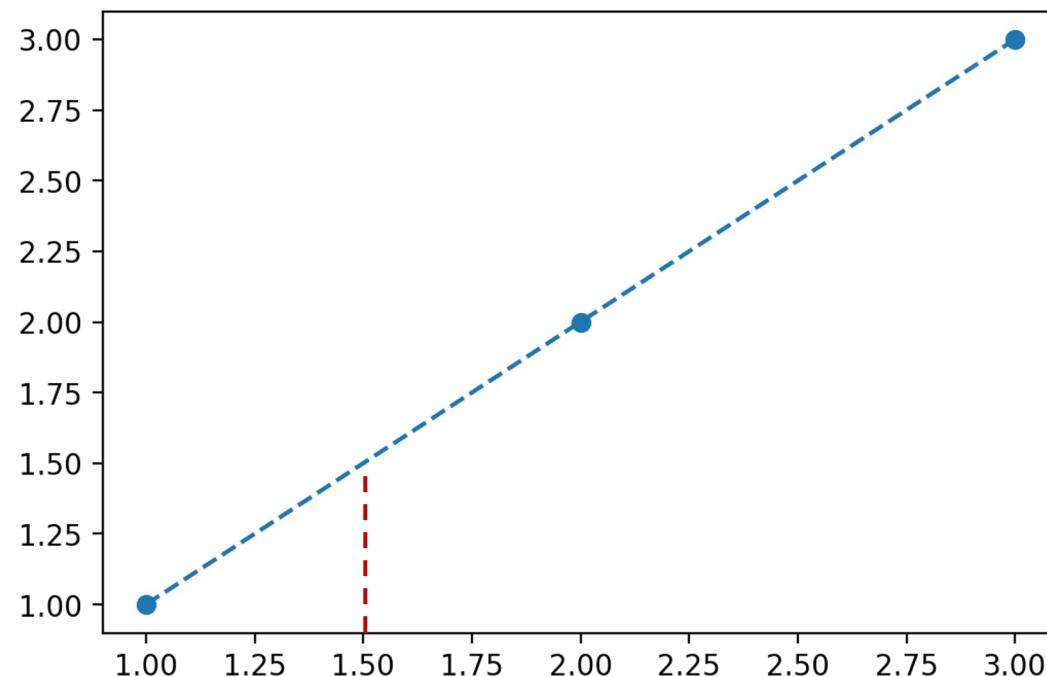
# Linear Regression

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



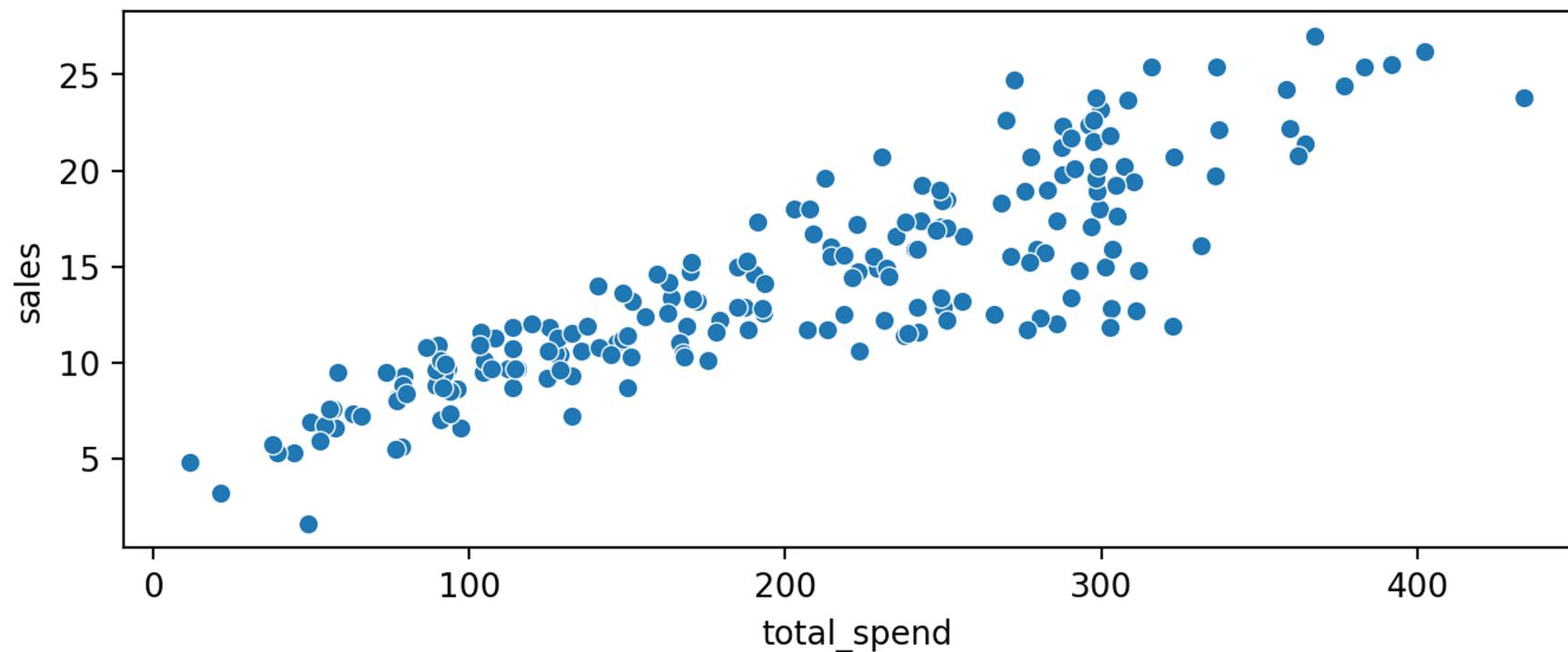
# Linear Regression

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



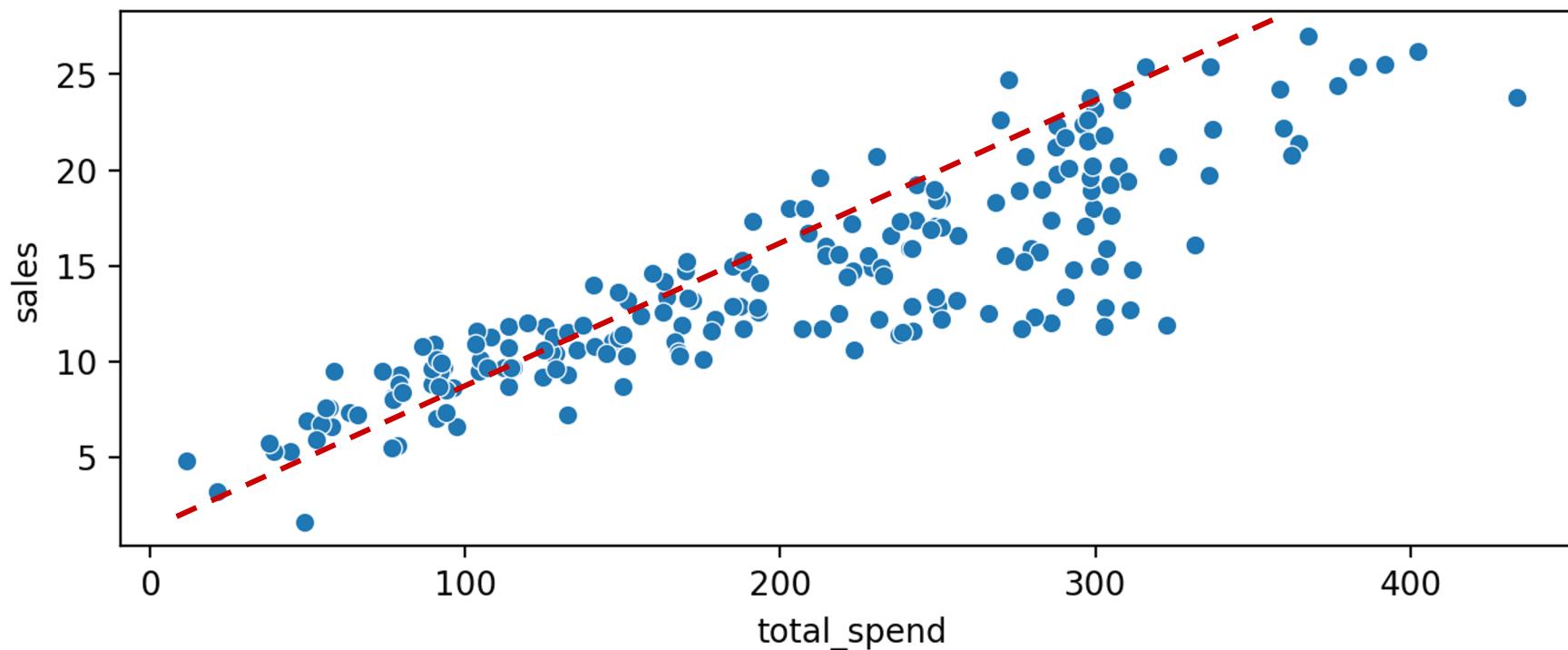
# Linear Regression

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



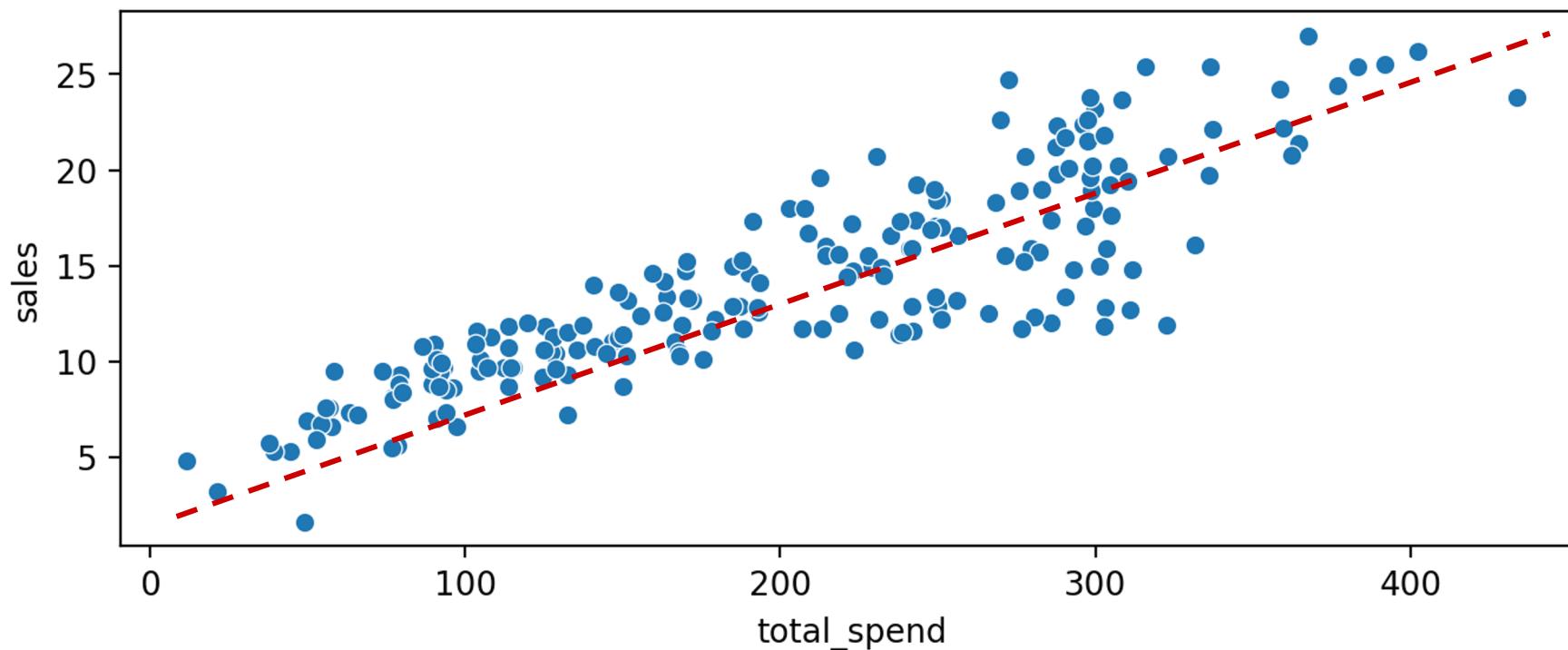
# Linear Regression

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



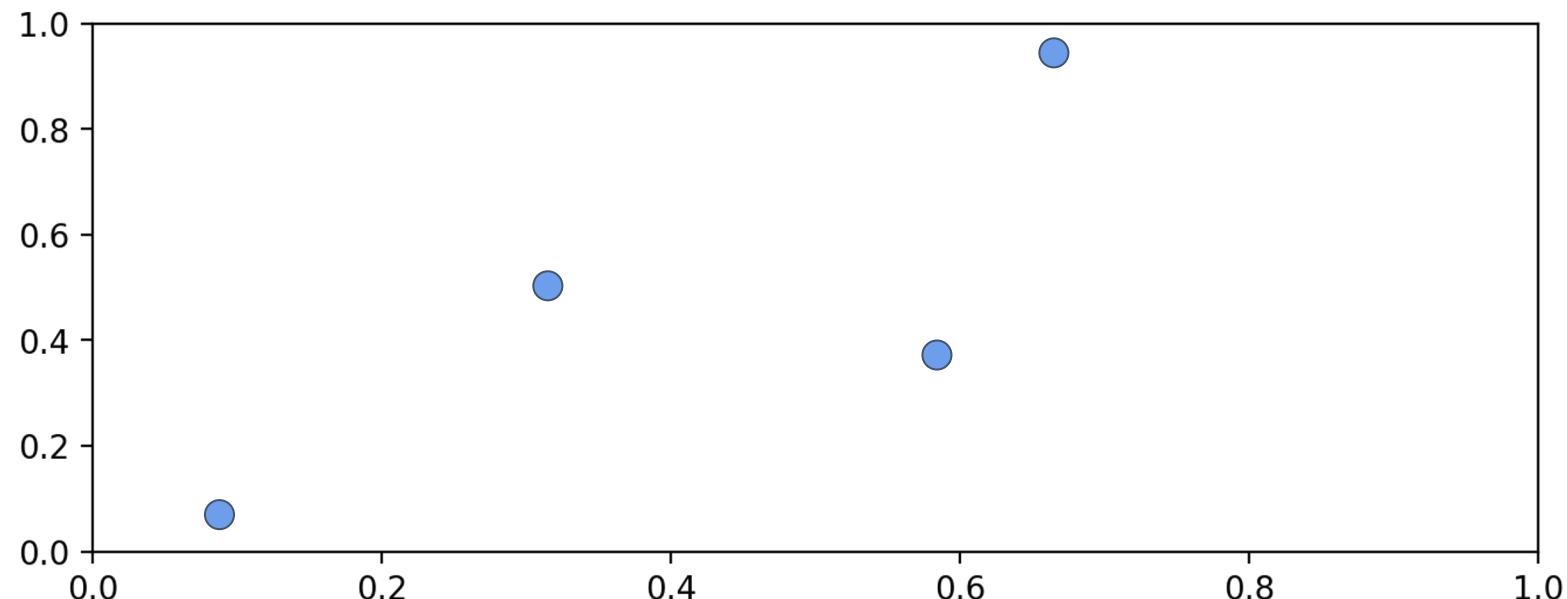
# Linear Regression

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



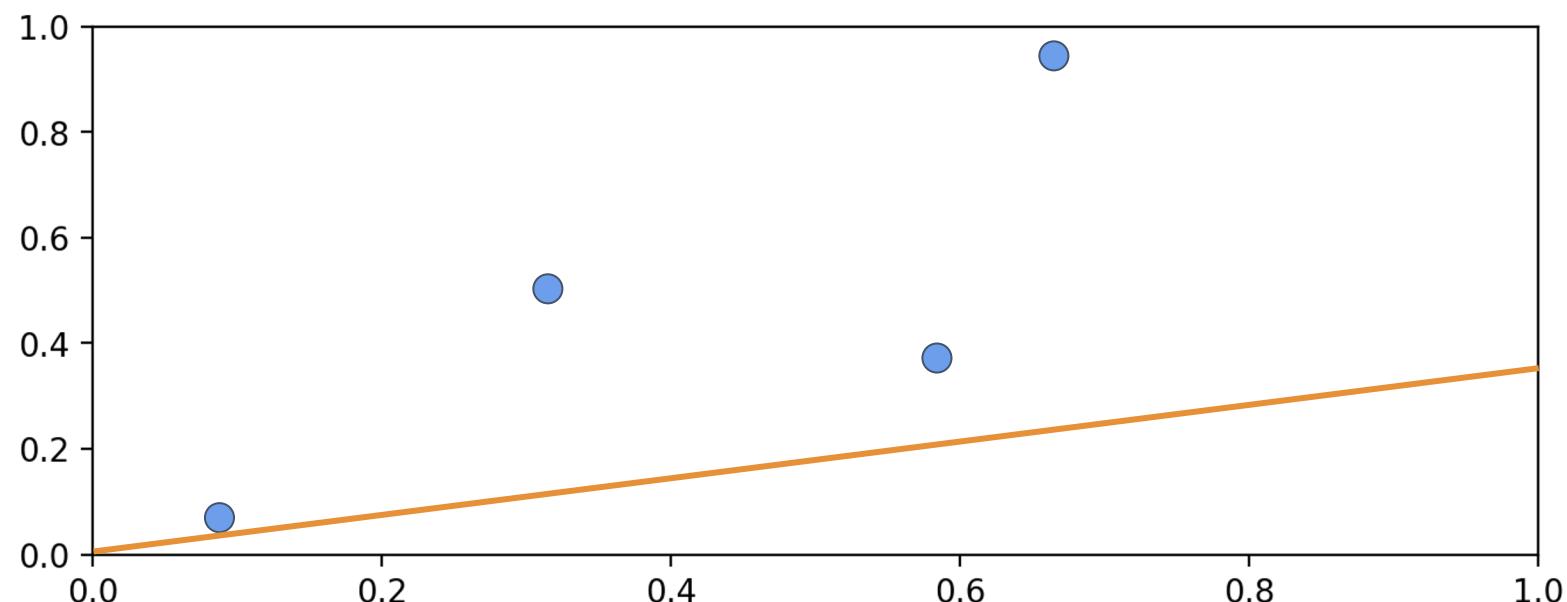
# Linear Regression

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



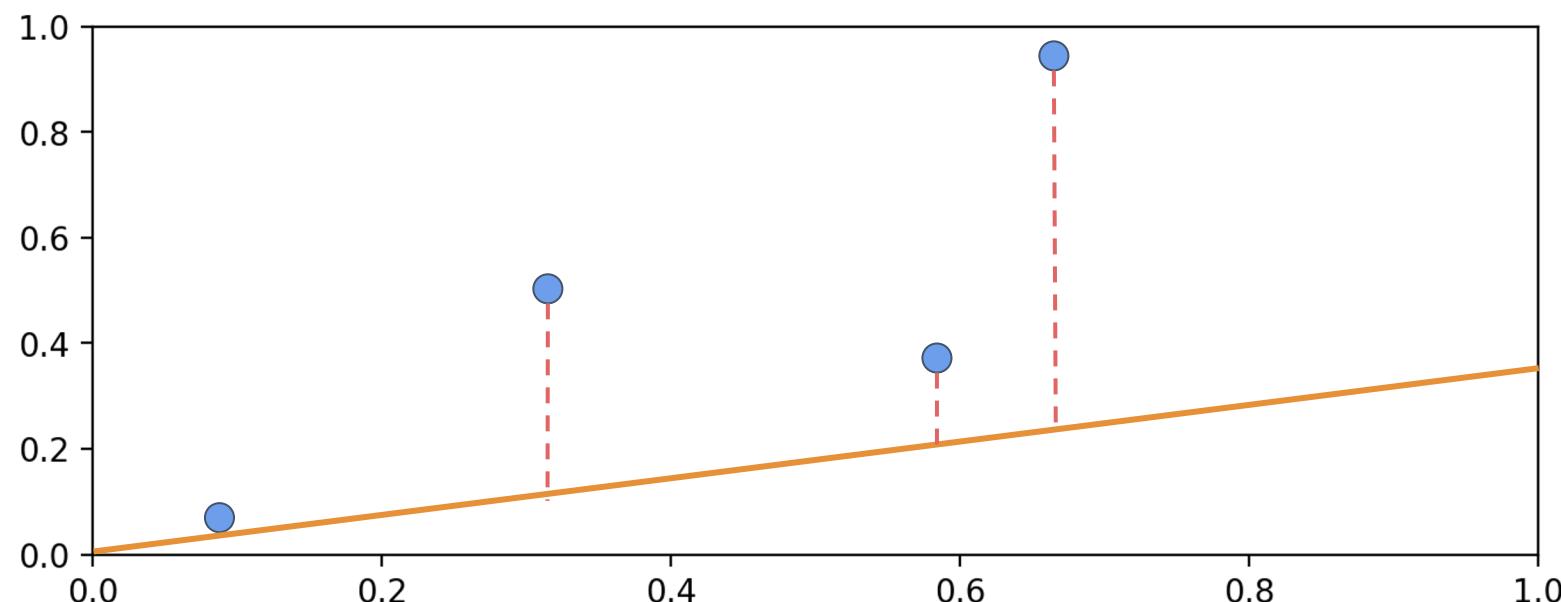
# Linear Regression

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



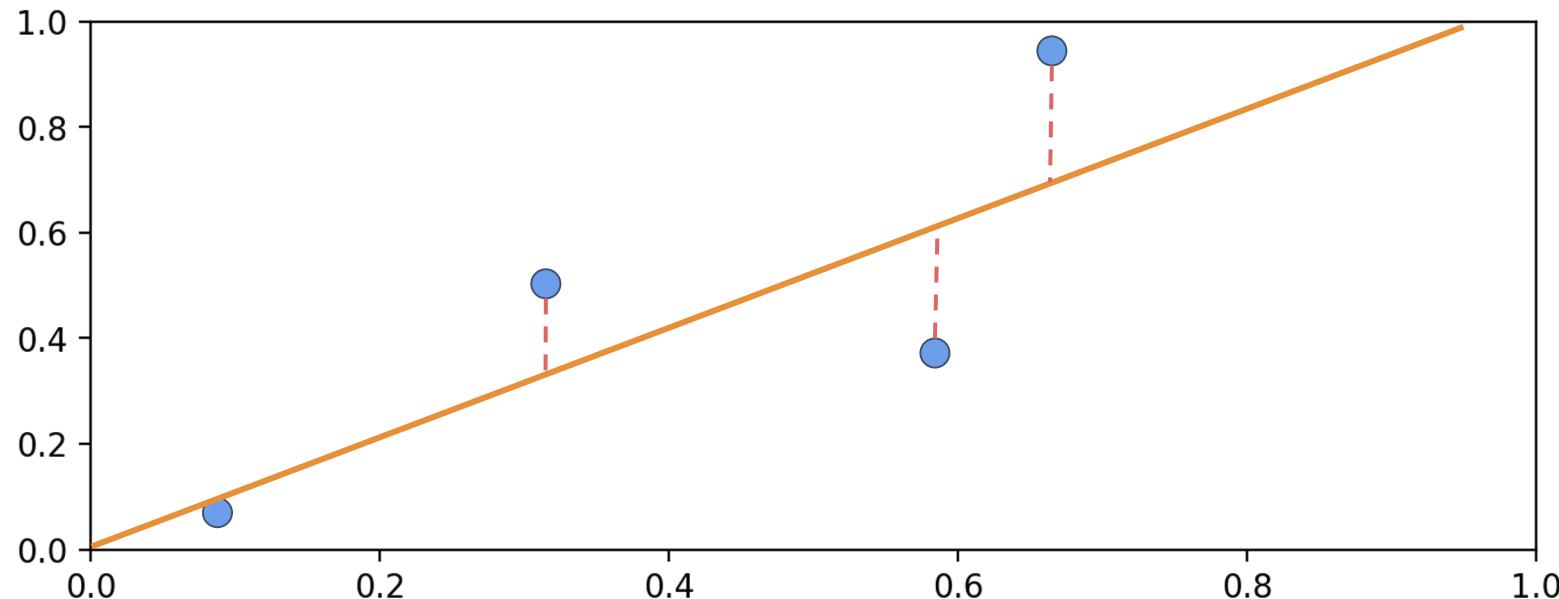
# Linear Regression

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



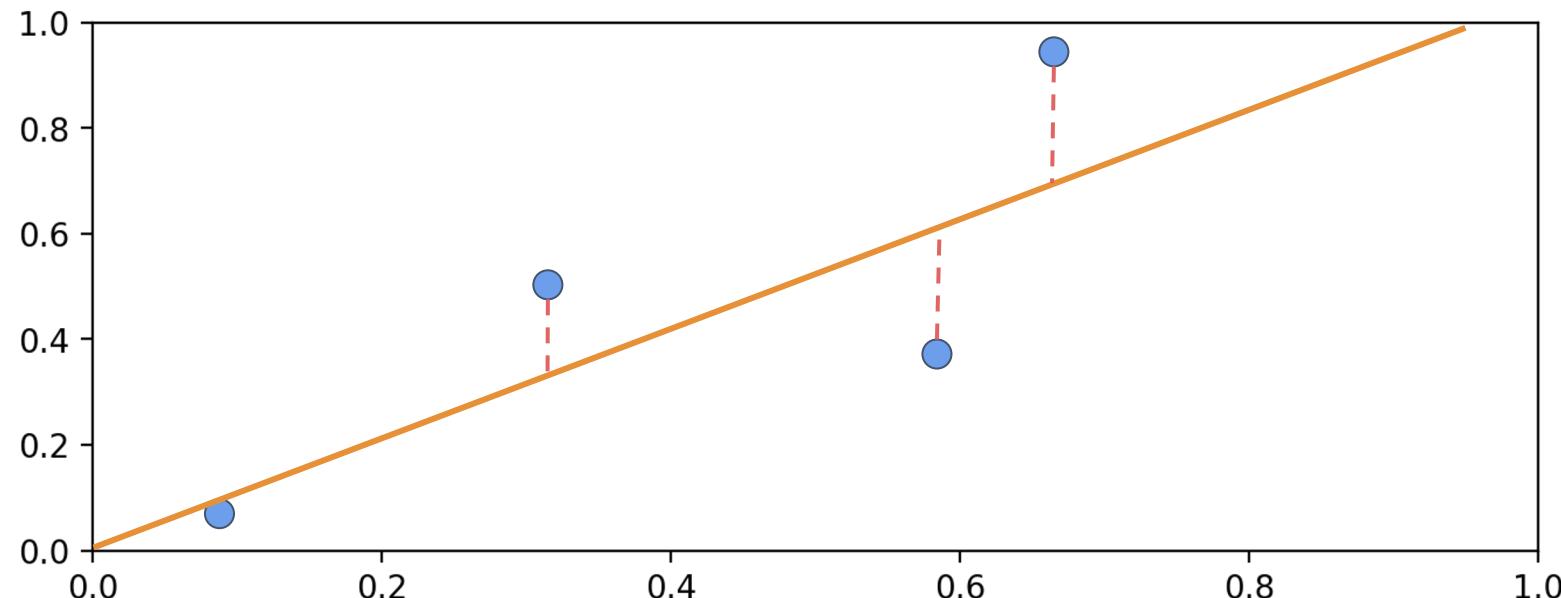
# Linear Regression

- Some lines will clearly be better fits than others.



# Linear Regression

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

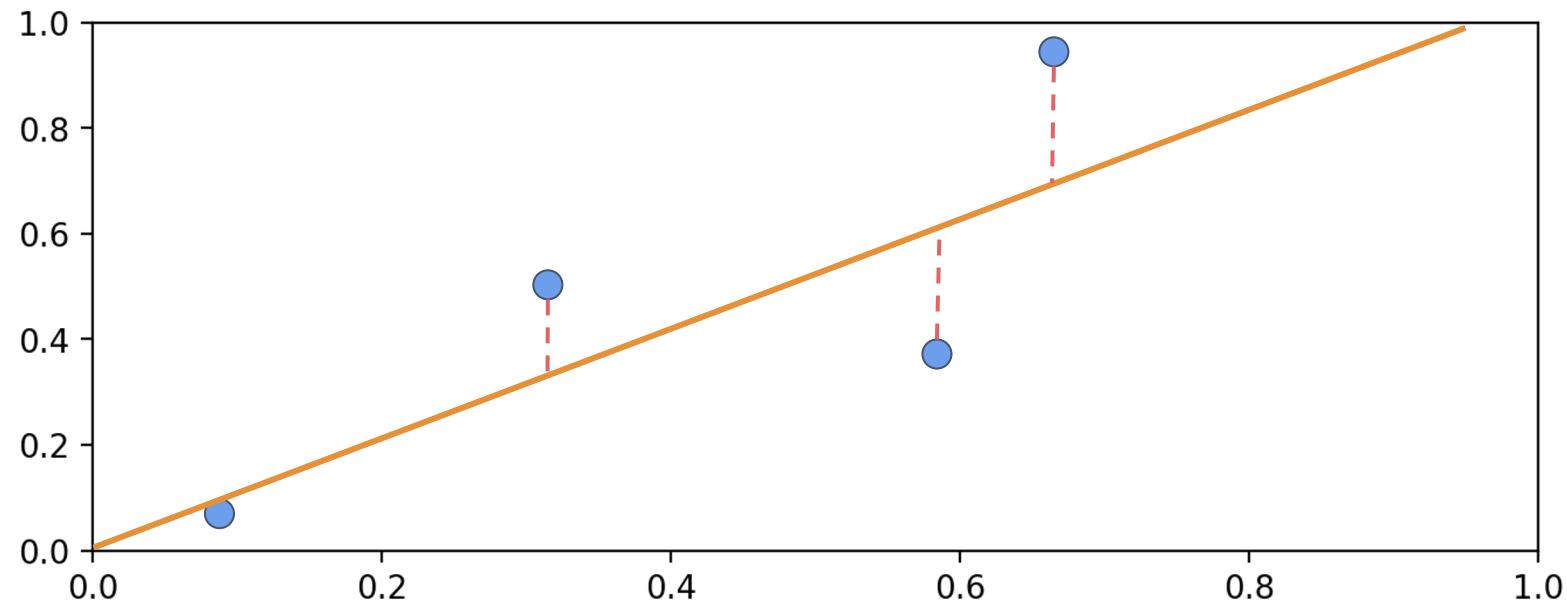


# Linear Regression

- **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.

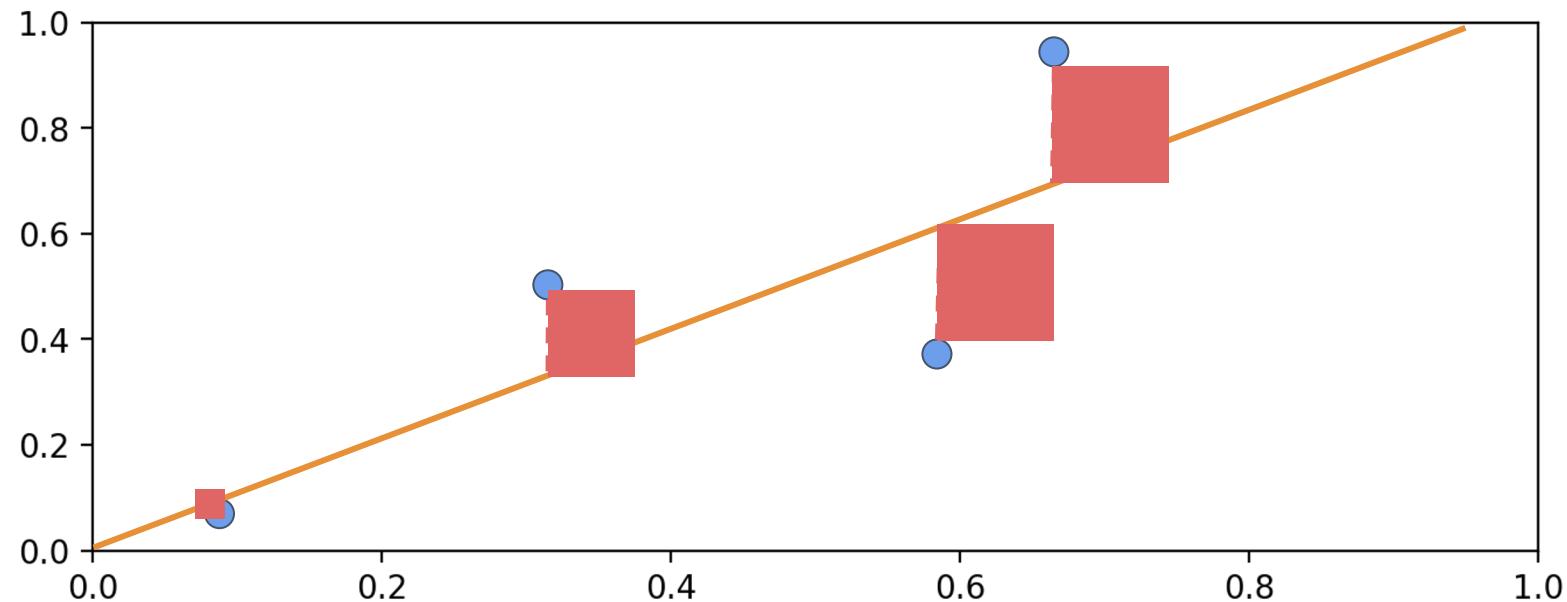
# Linear Regression

- 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

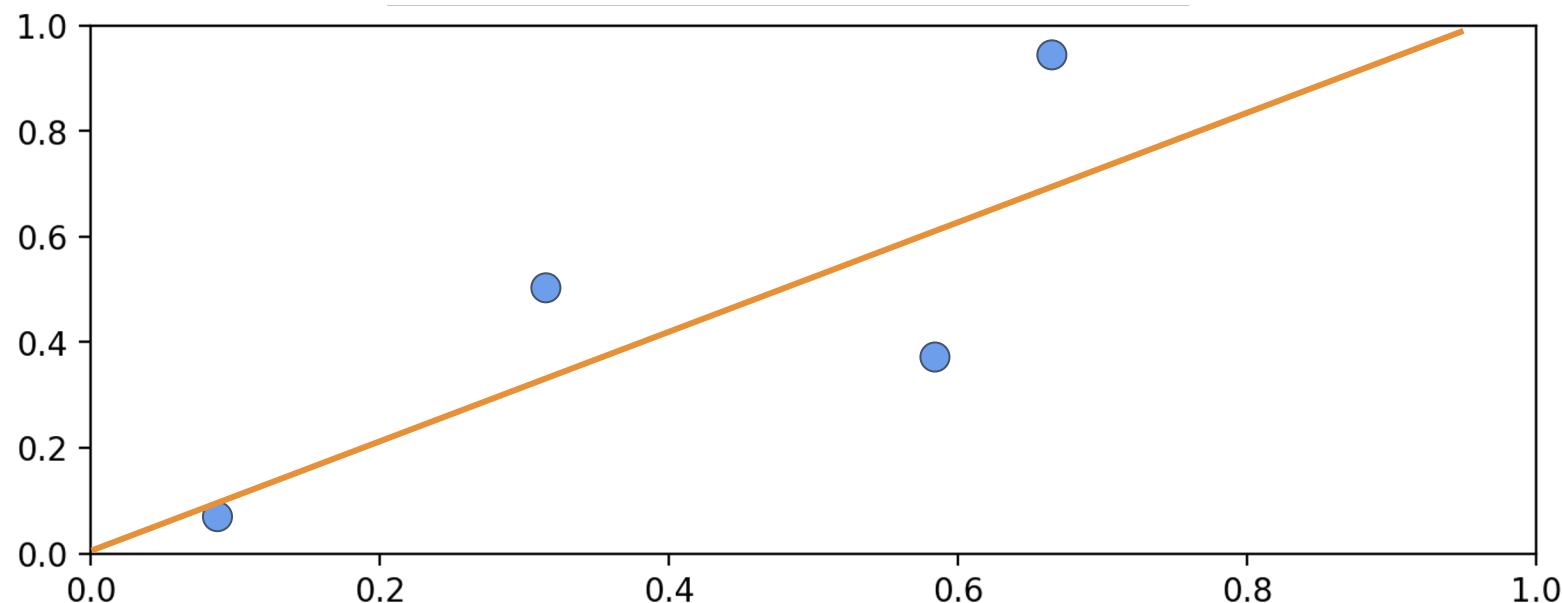
- Note the  $\hat{y}$  hat symbol displays a prediction. There is usually no set of Betas to create a perfect fit to  $y$ !

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

# Linear Regression

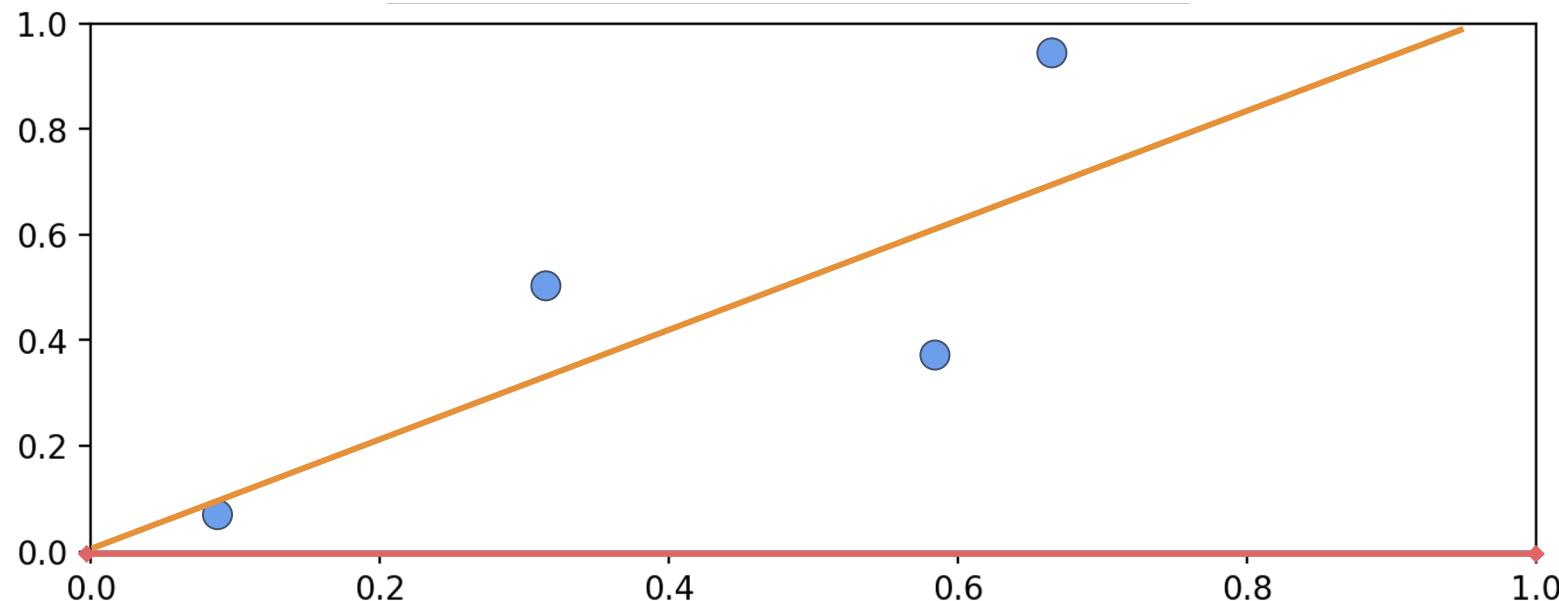
- Line equation:

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



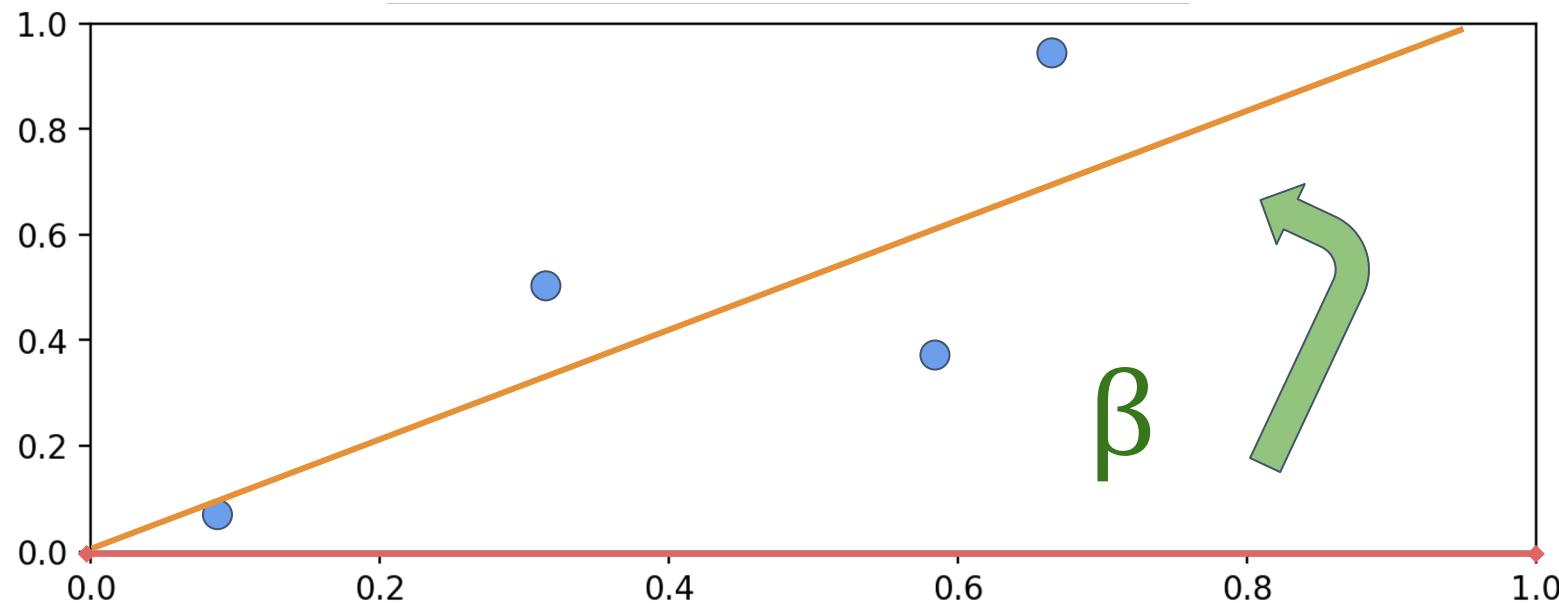
# Linear Regression

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



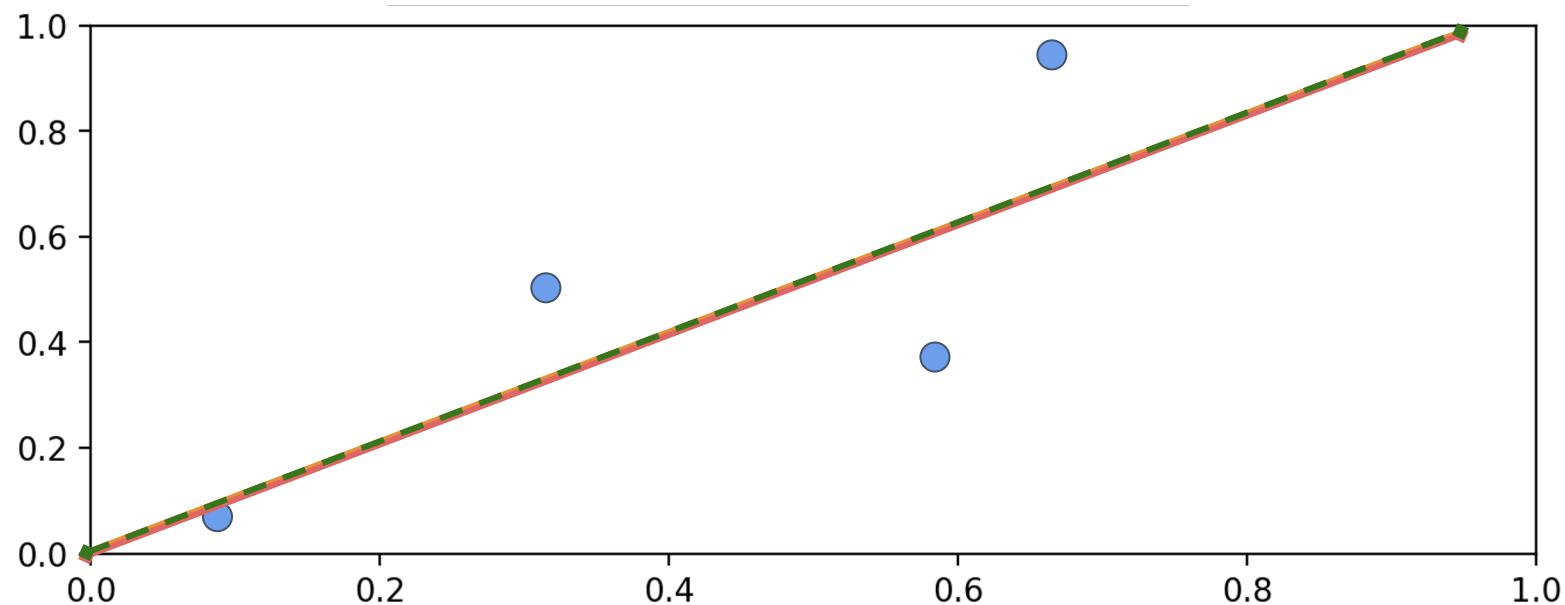
# Linear Regression

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



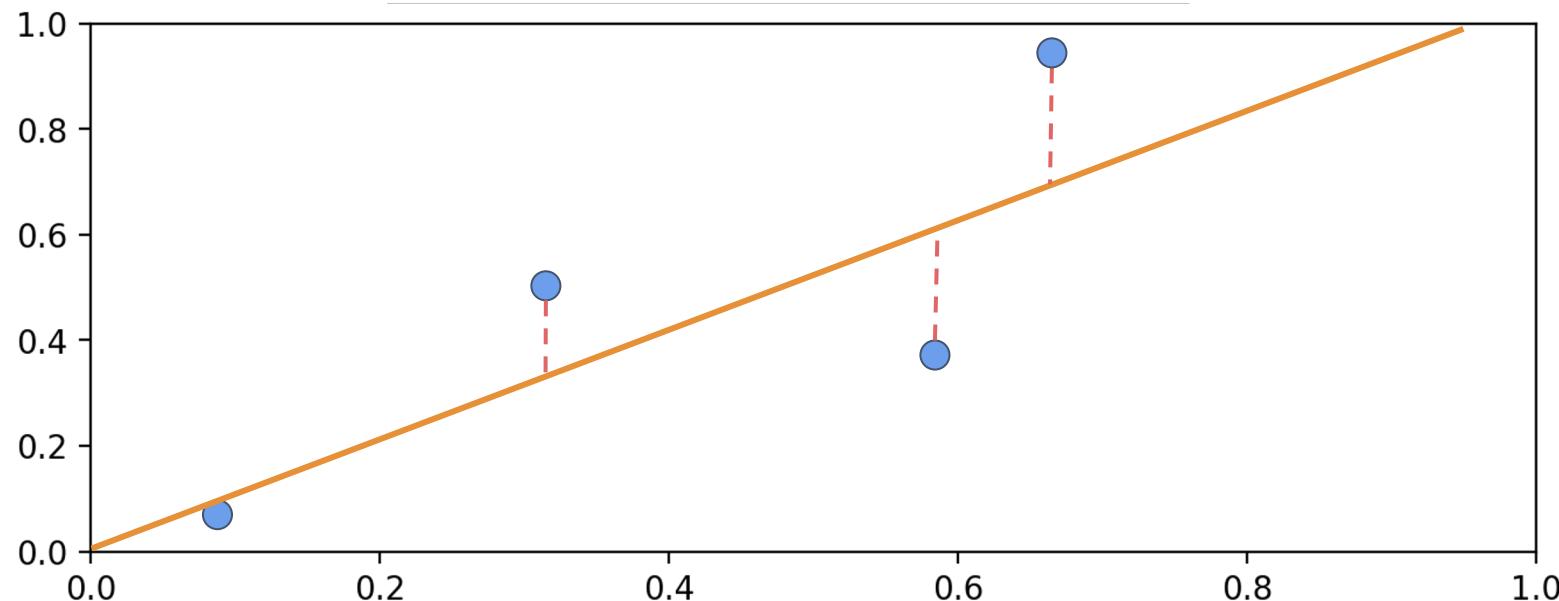
# Linear Regression

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



# 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**

**Algorithm Theory - Part Three  
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

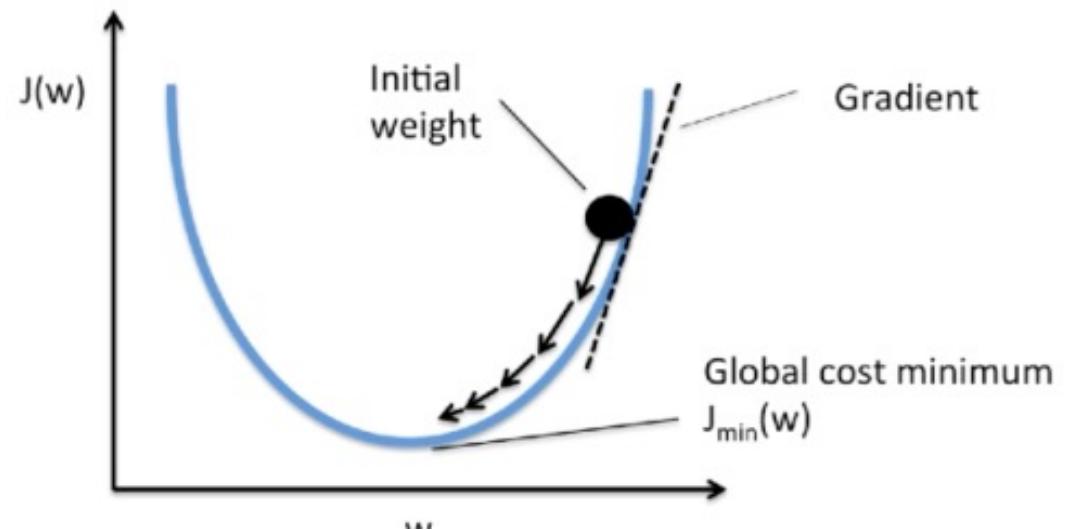
- Unfortunately, it is not scalable to try to get an analytical solution to minimize this cost function.
- In the next lecture we will learn to use **gradient descent** to minimize this **cost function**.

# **Introduction to Linear Regression**

**Algorithm Theory - Part Three  
Gradient Descent**

# Gradient Descent to find hidden parameters

```
# gradient descent  
# pseudocode  
  
 $\theta = (m, b) = \text{random}()$   
for i in range(n_epochs):  
     $\theta = \theta - \eta \nabla_{\theta} L$ 
```



$$L = MSE = \frac{1}{N} \sum_{i=1}^N (y_i - (mx_i + b))^2 = J(\beta)$$

# 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.

# 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.

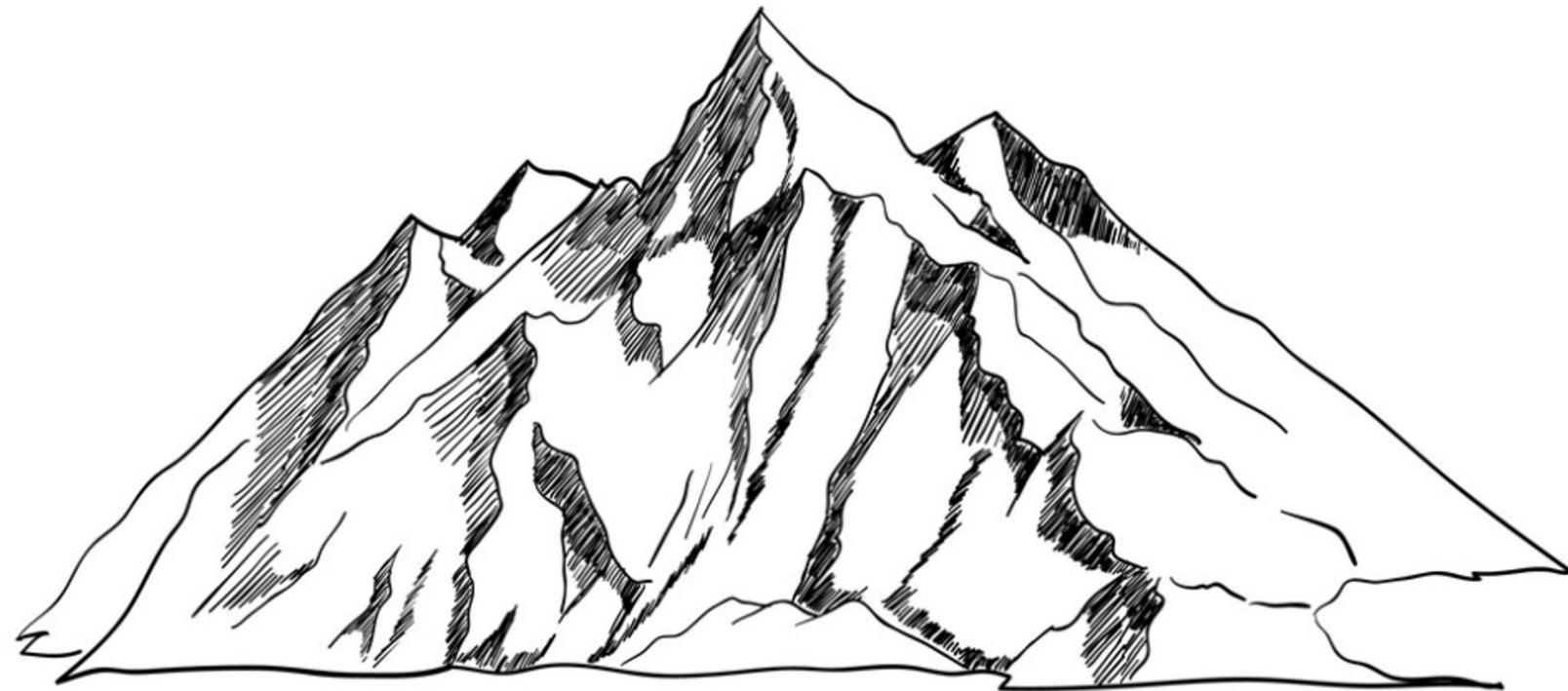
- **Title:** The Cost Function,  $J(\beta)$
- **Content:**
  - Defined as the Mean Squared Error (MSE):  
$$J(\beta) = \frac{1}{2m} \sum_{i=1}^m (h_\beta(x^{(i)}) - y^{(i)})^2$$
where:  
$$h_\beta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$
  - Objective: Minimize  $J(\beta)$ .

## 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  $\beta_j$ .

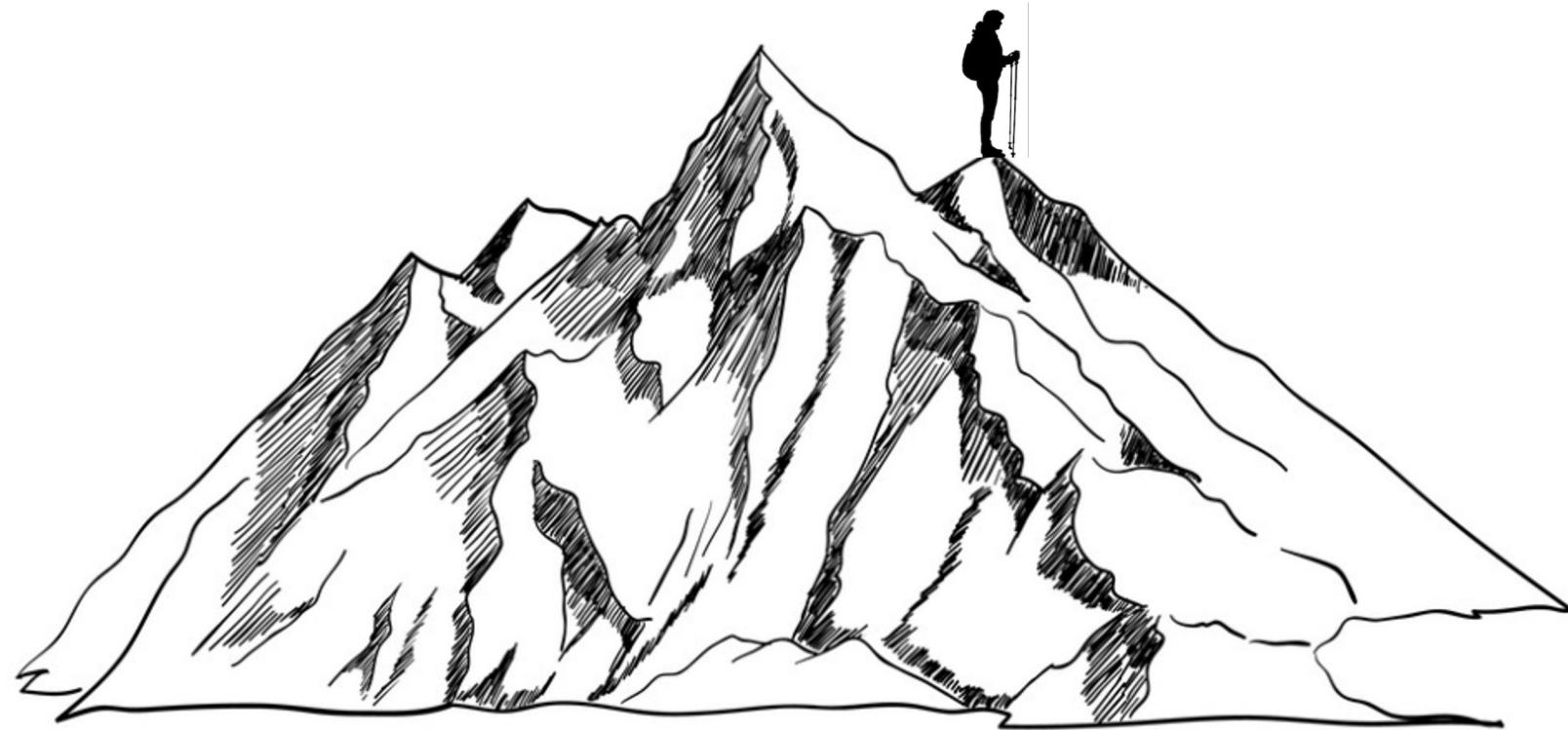
# Linear Regression

- Common mountain analogy



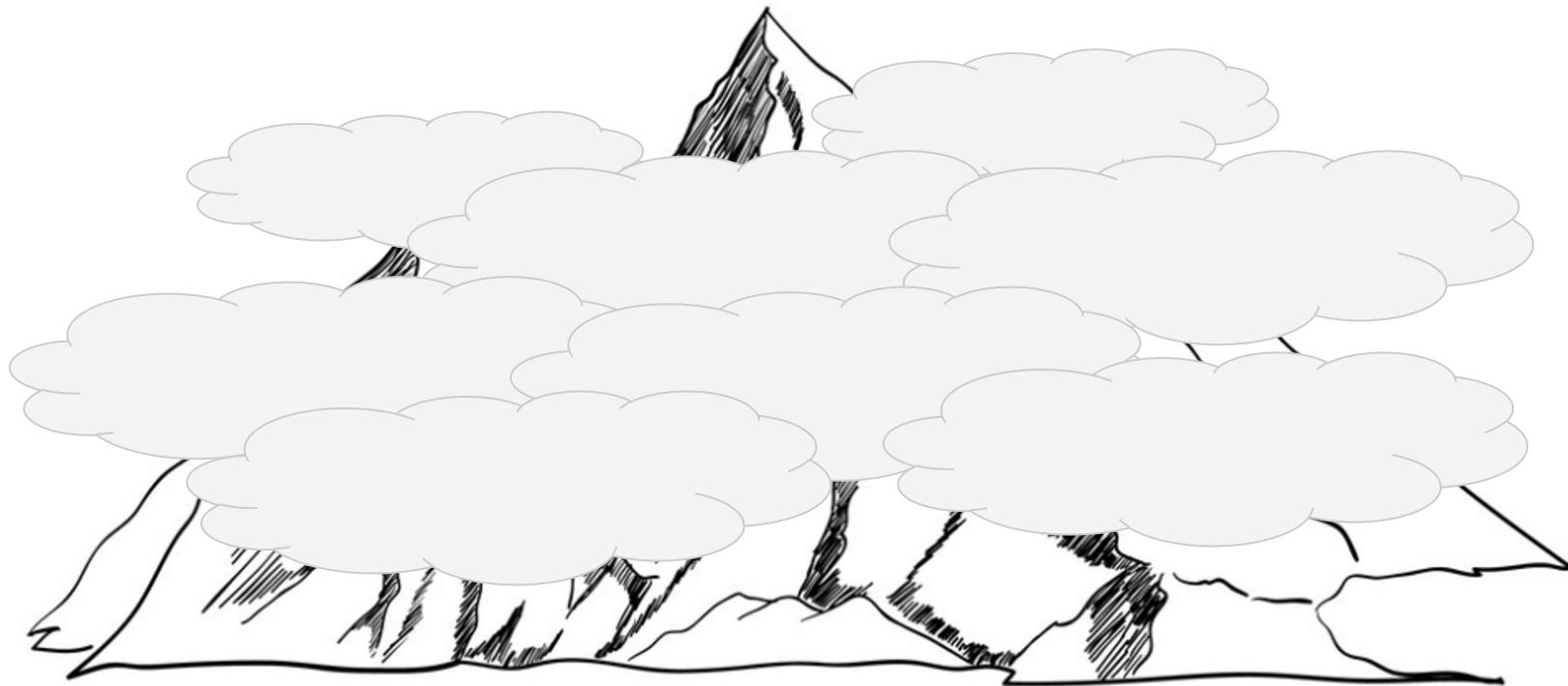
# Linear Regression

- Common mountain analogy



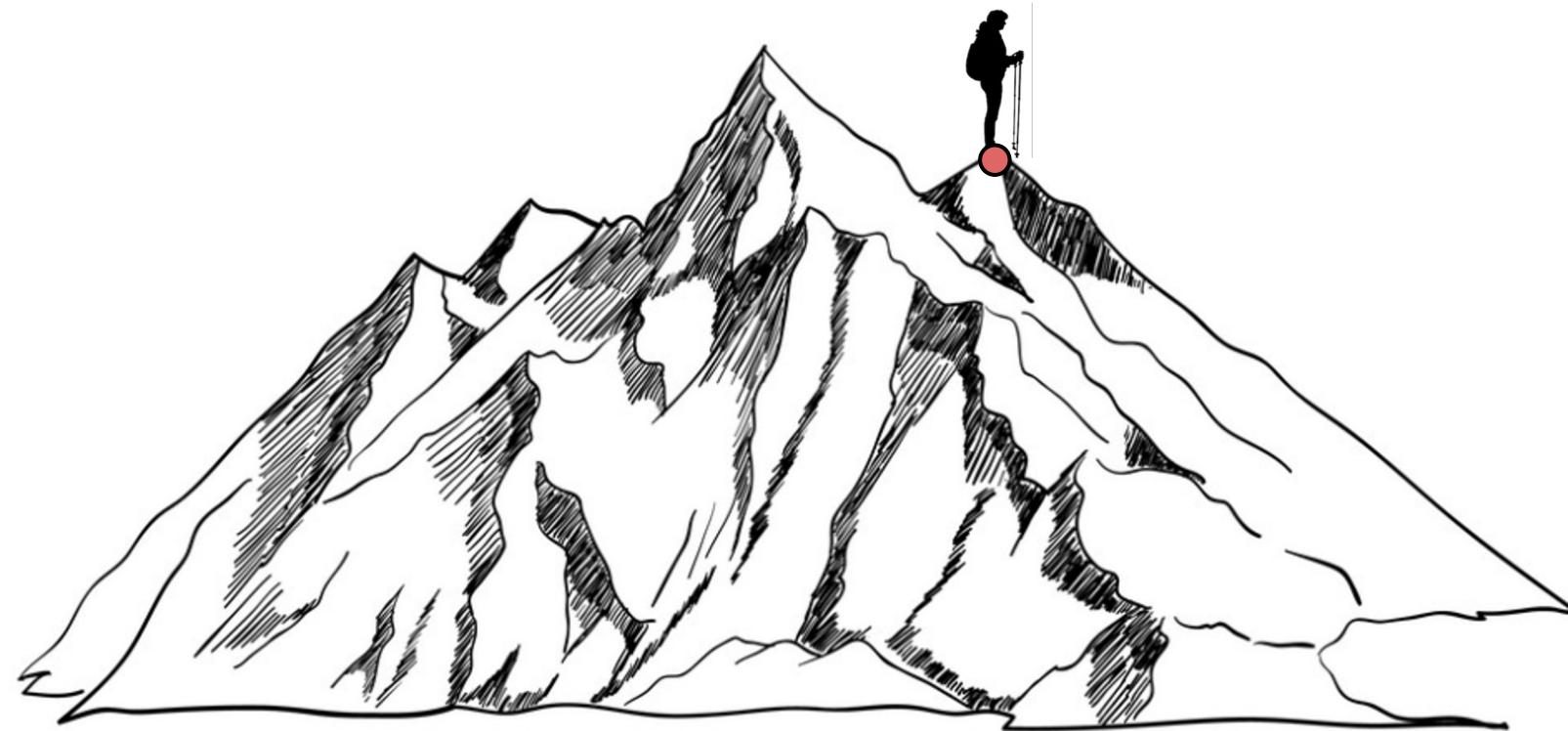
# Linear Regression

- Common mountain analogy



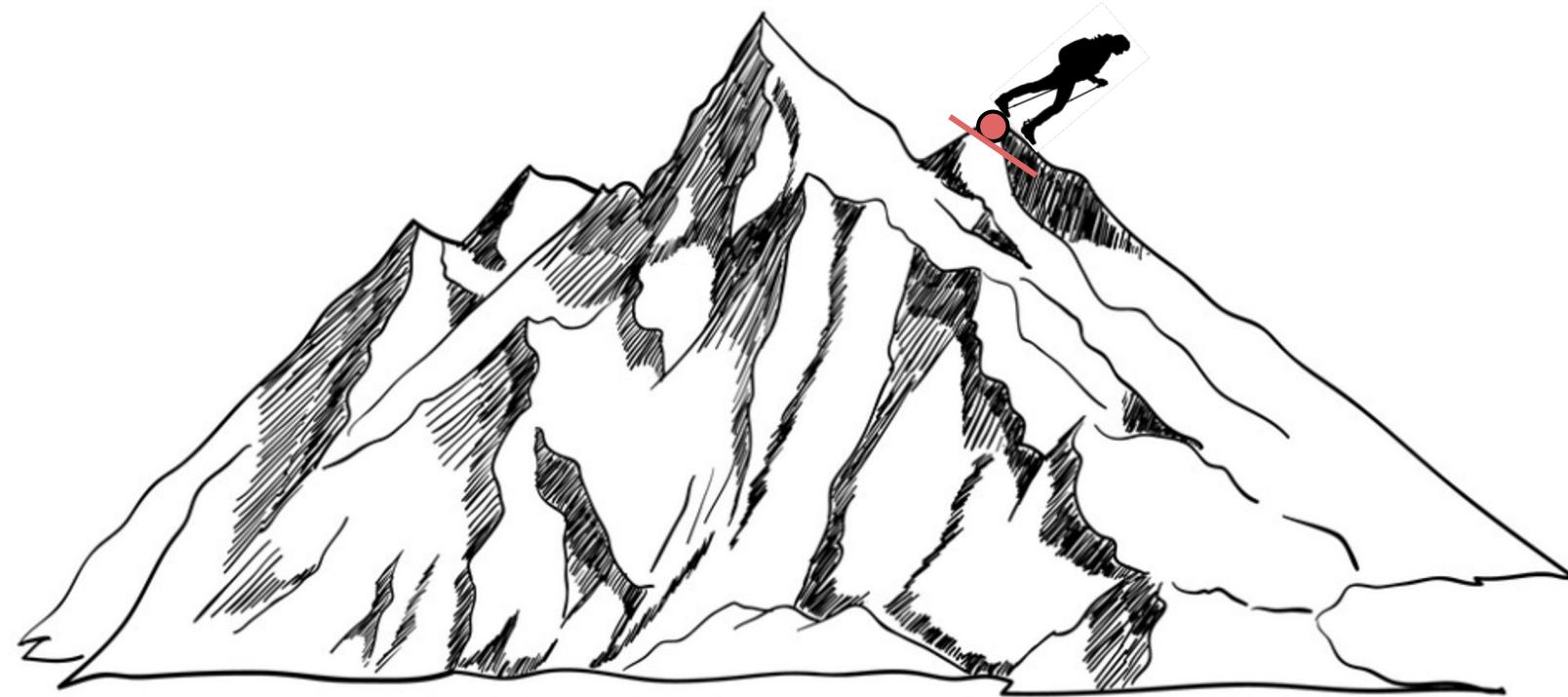
# Linear Regression

- Common mountain analogy



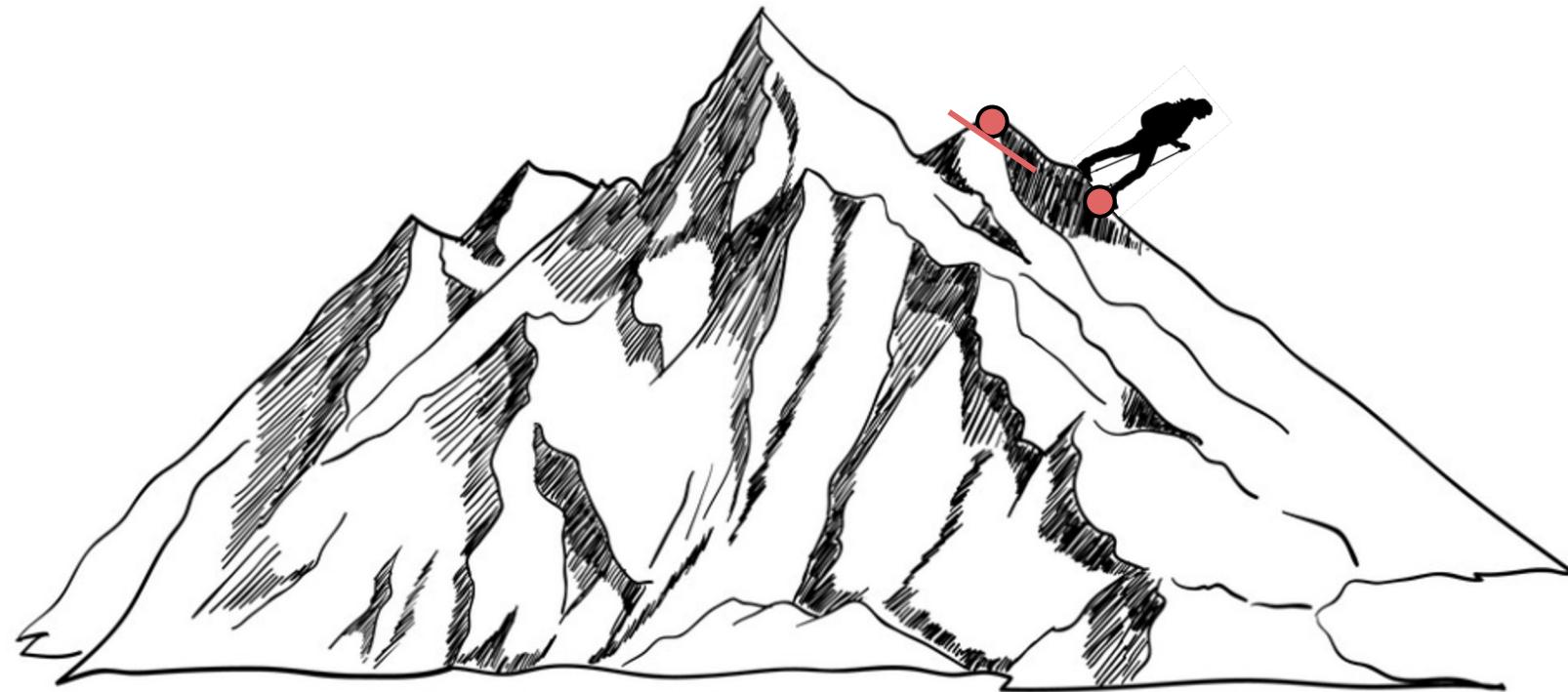
# Linear Regression

- Common mountain analogy



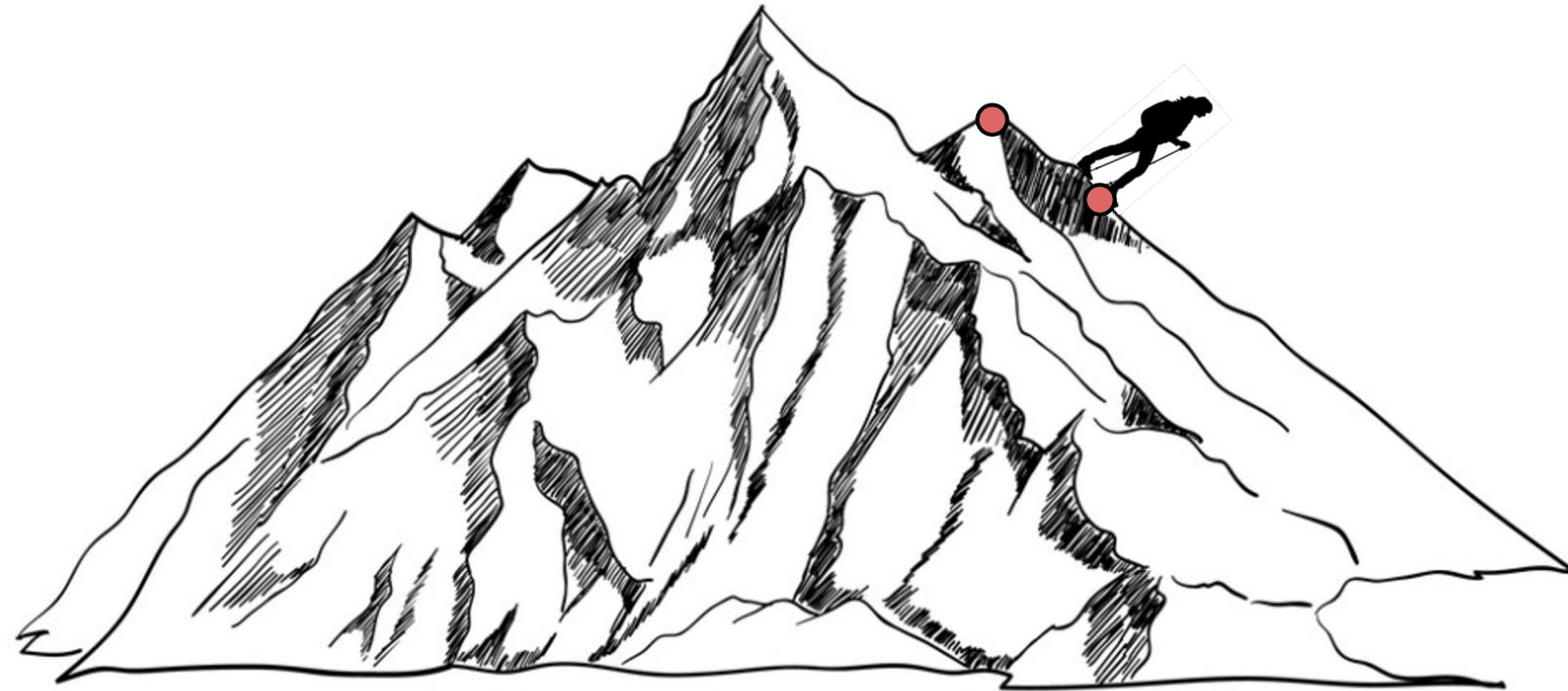
# Linear Regression

- Common mountain analogy



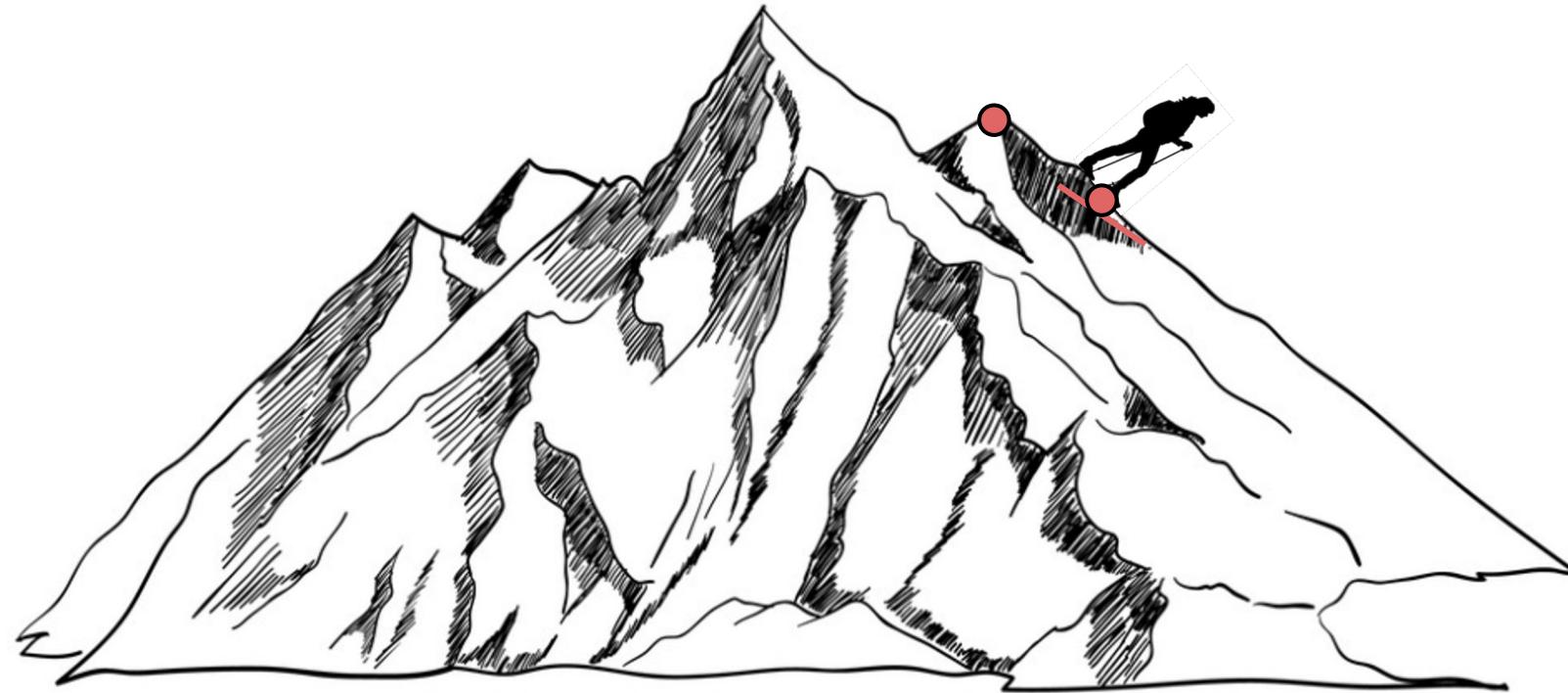
# Linear Regression

- Common mountain analogy



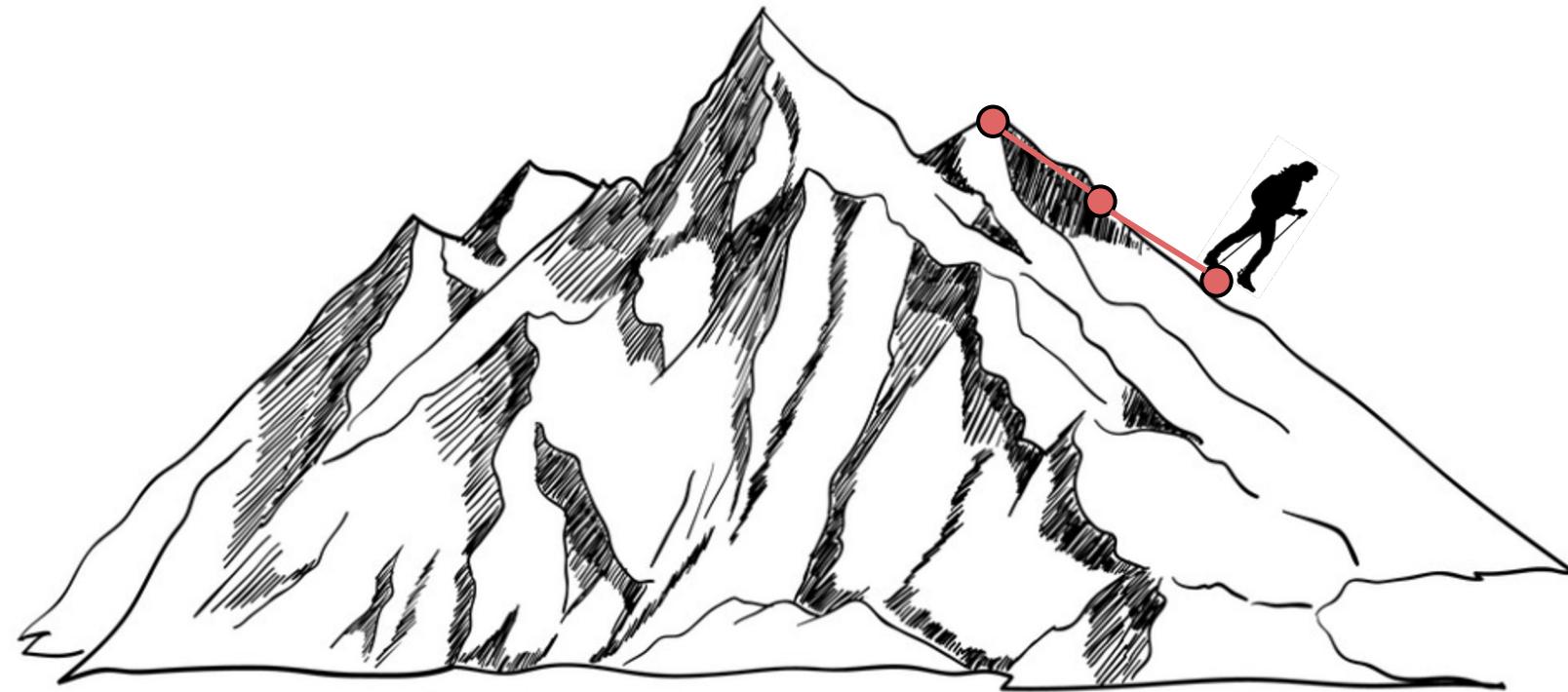
# Linear Regression

- Common mountain analogy



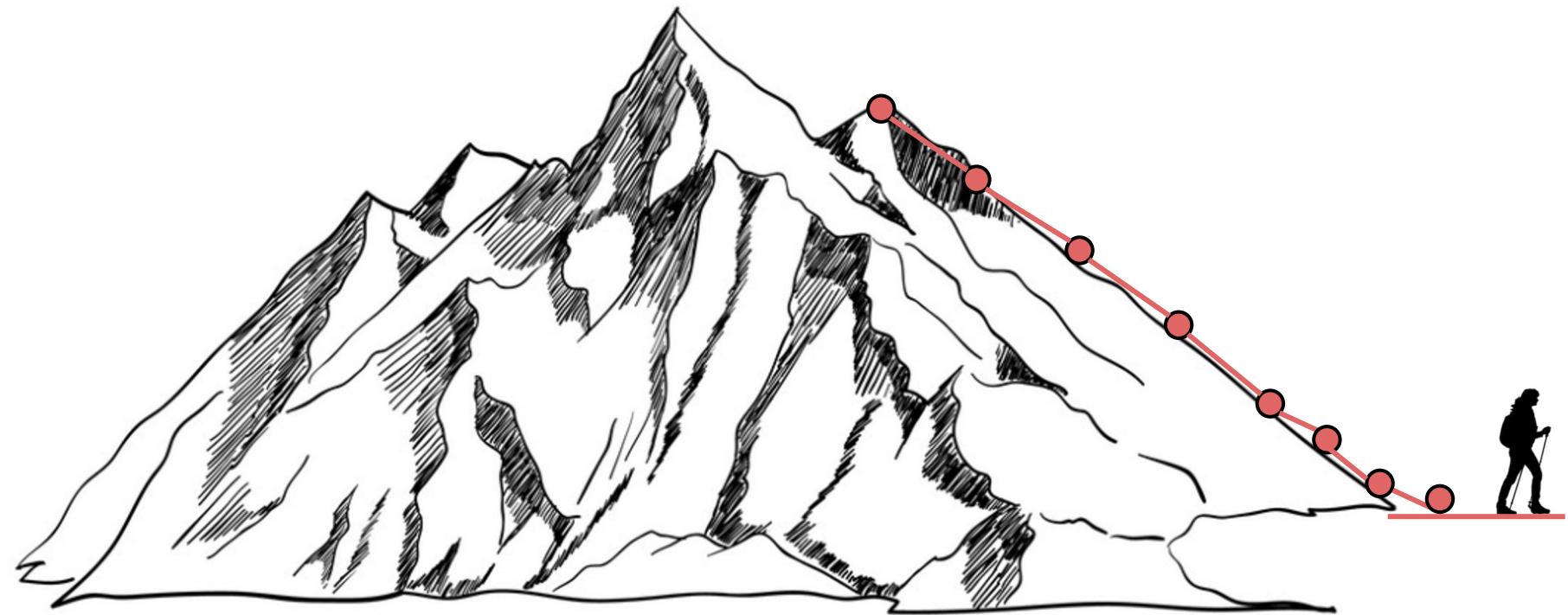
# Linear Regression

- Common mountain analogy



# Linear Regression

- Common mountain analogy

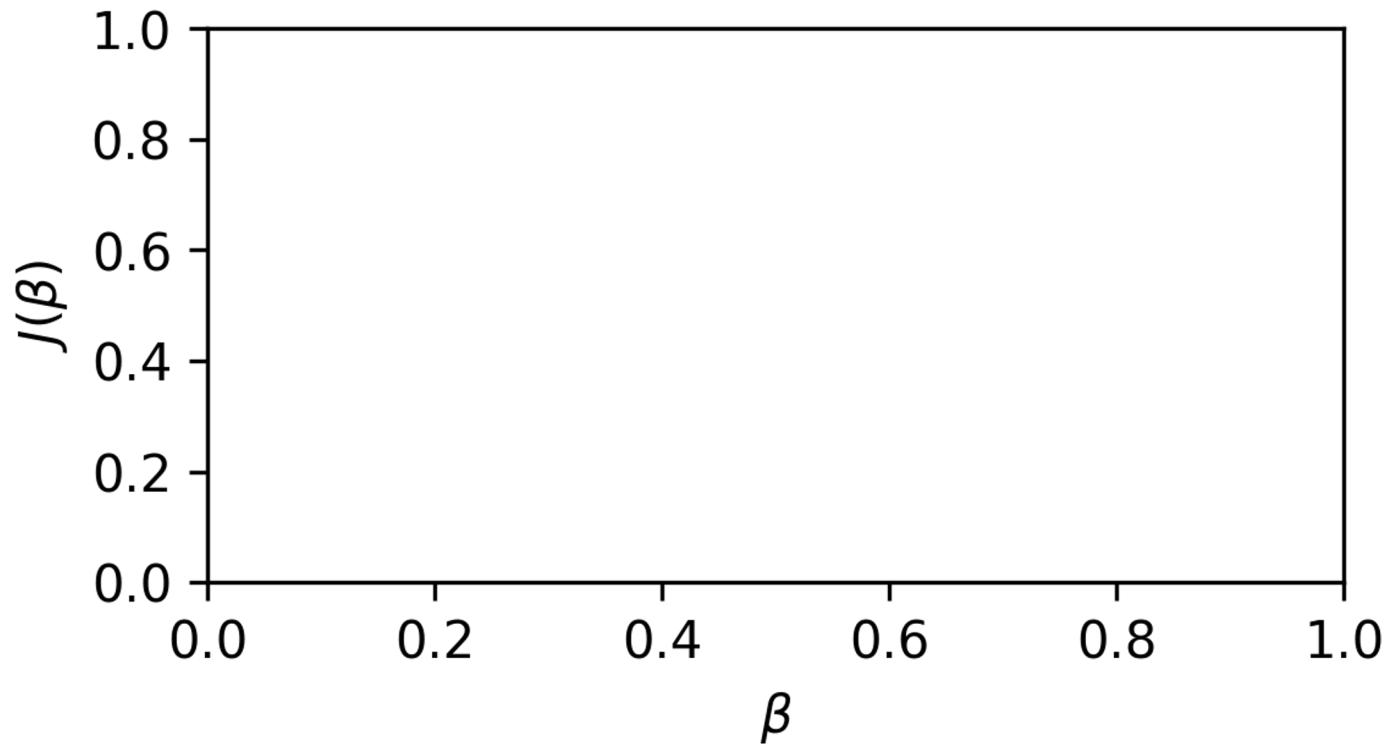


# Linear Regression

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

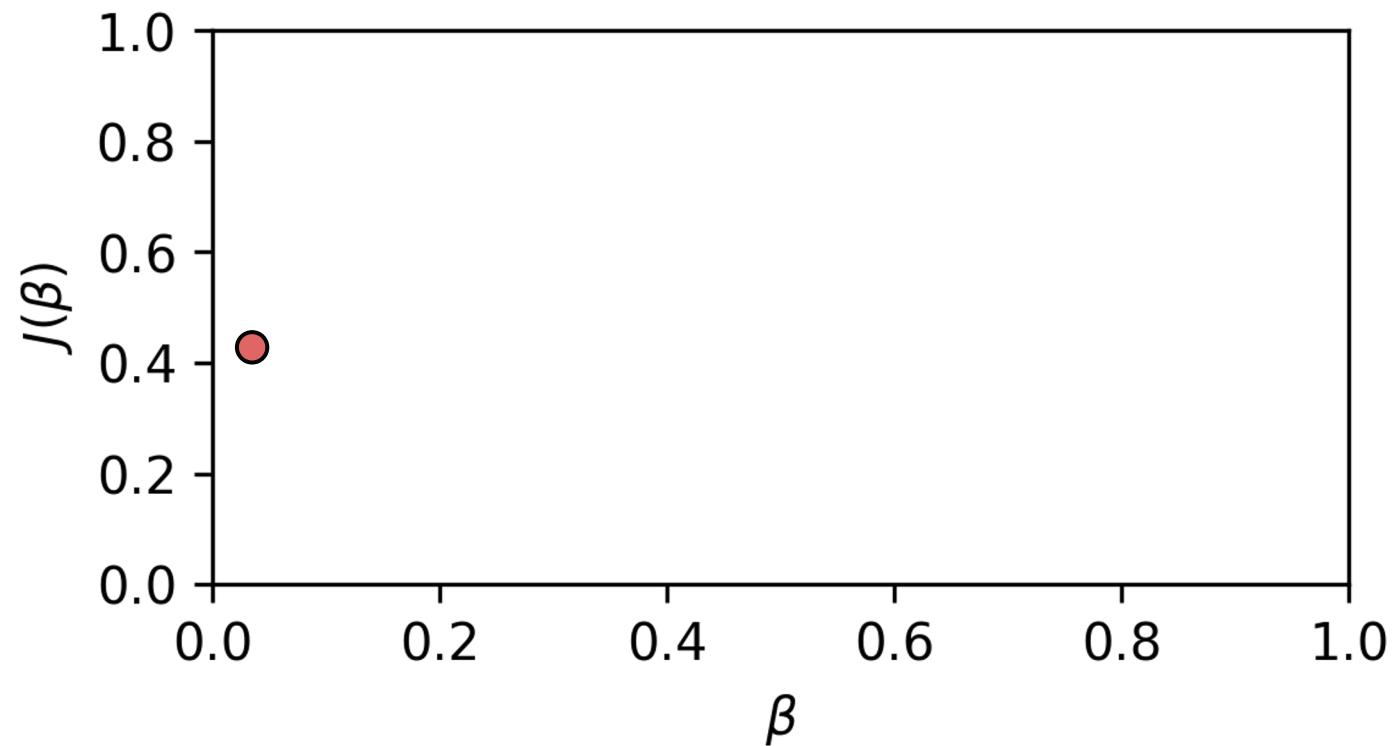
# Linear Regression

- 1 dimensional cost function (single Beta)



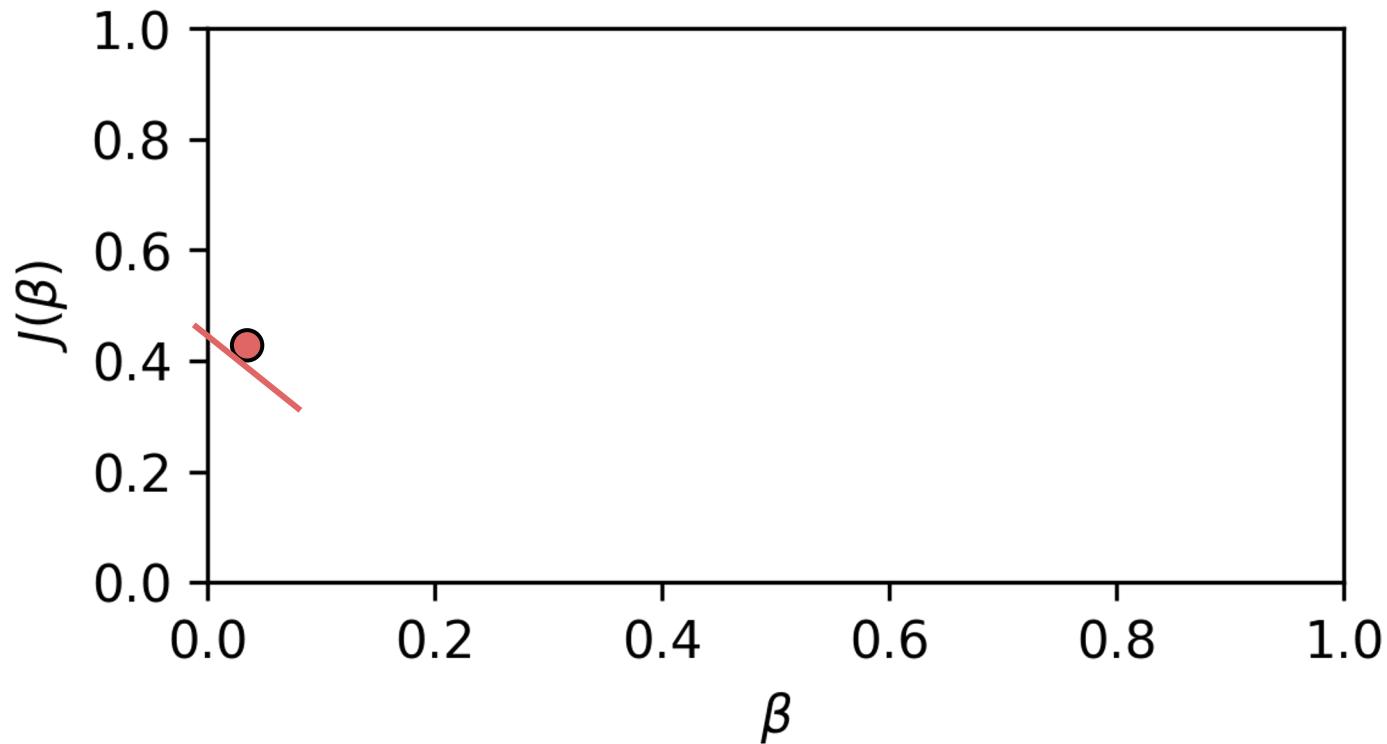
# Linear Regression

- Choose a starting point



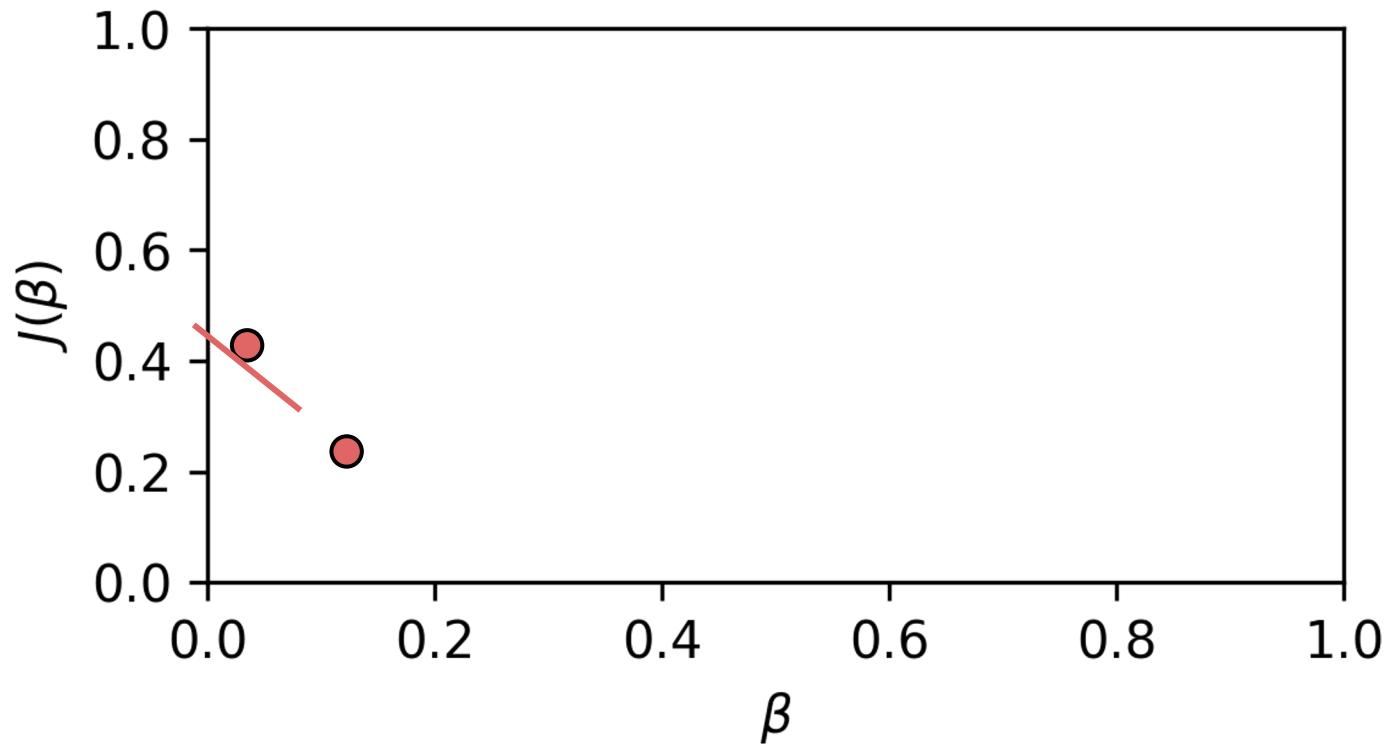
# Linear Regression

- Calculate gradient at that point



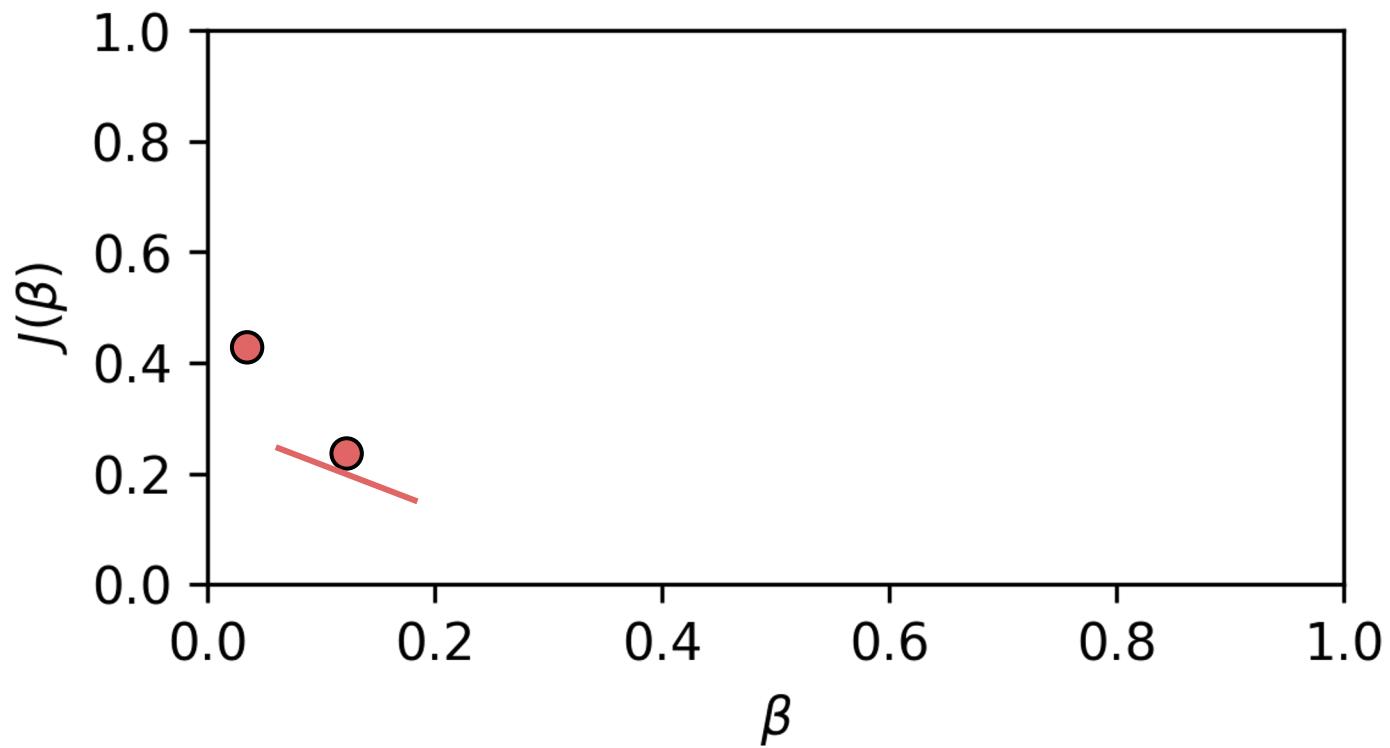
# Linear Regression

- Step forward proportional to **negative** gradient



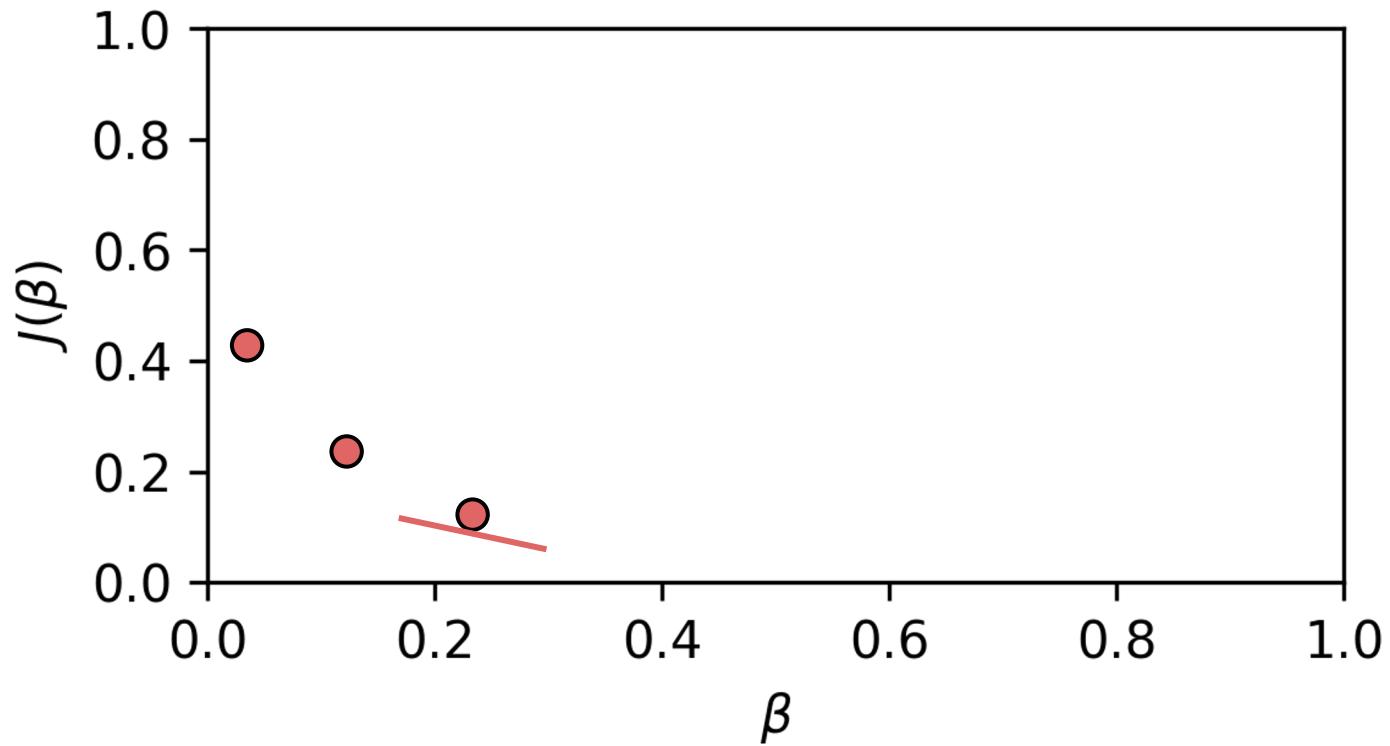
# Linear Regression

- Repeat the steps



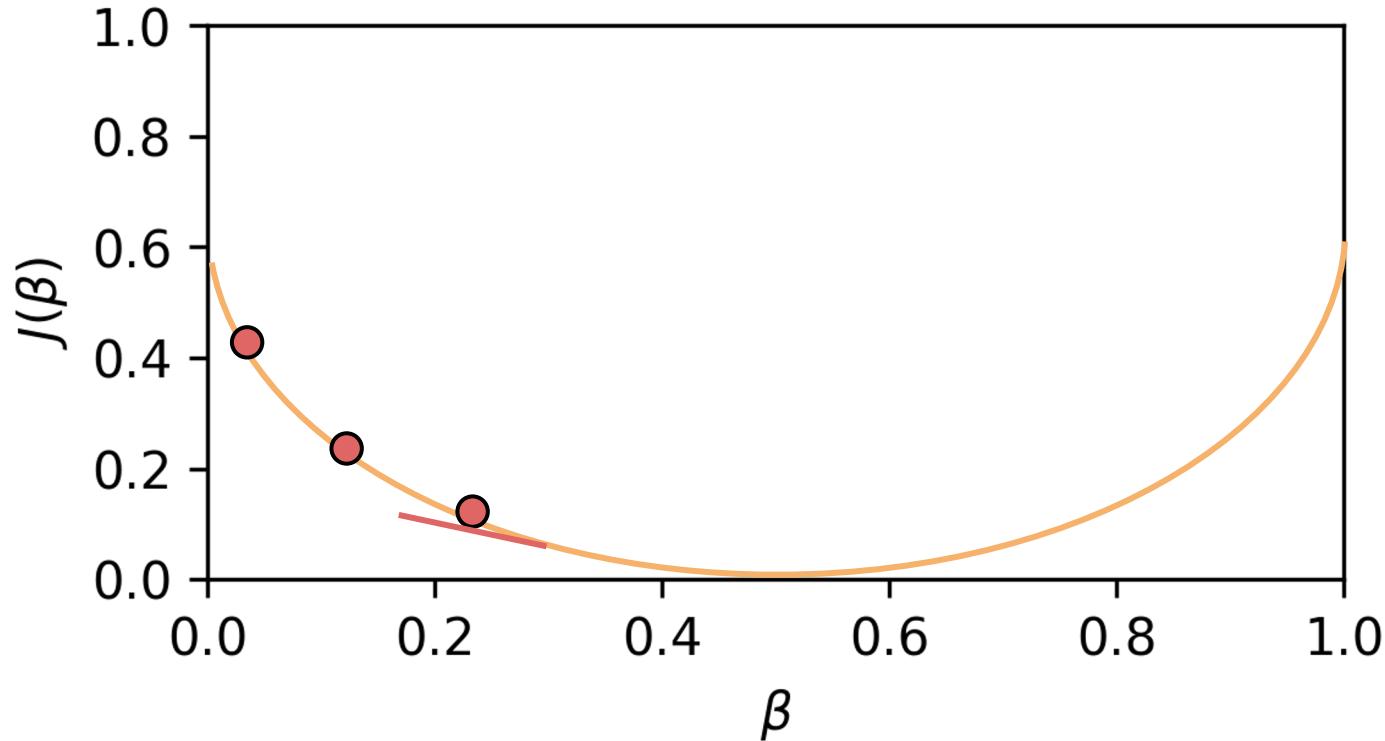
# Linear Regression

- Repeat the steps



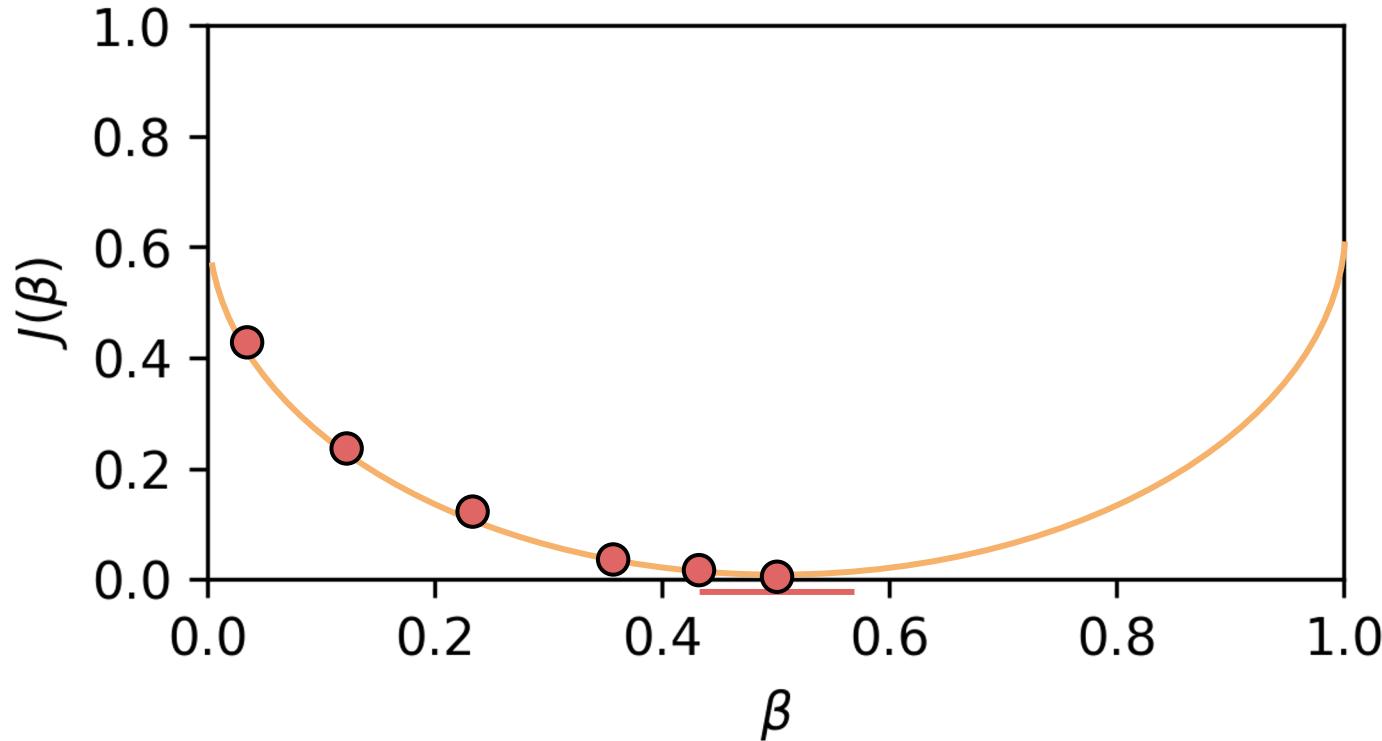
# Linear Regression

- Note how we are essentially mapping the gradient!



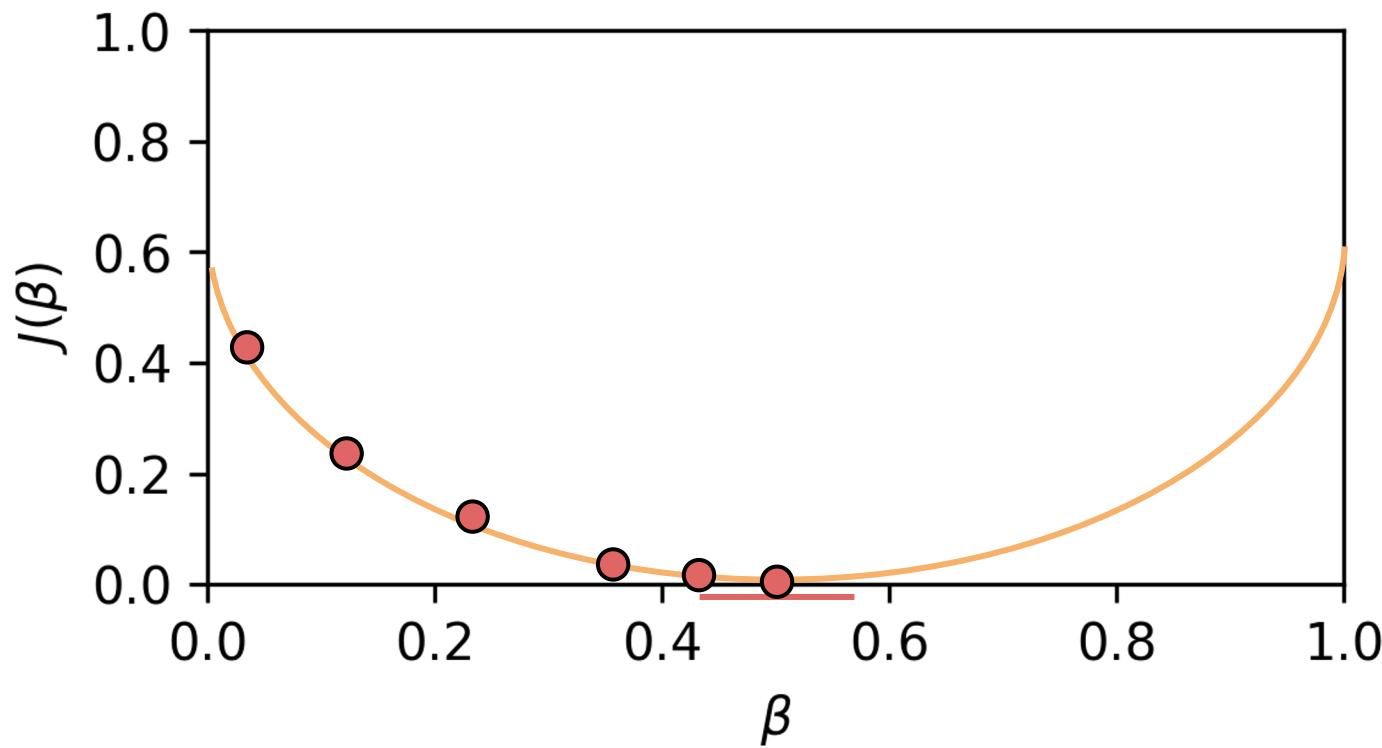
# Linear Regression

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



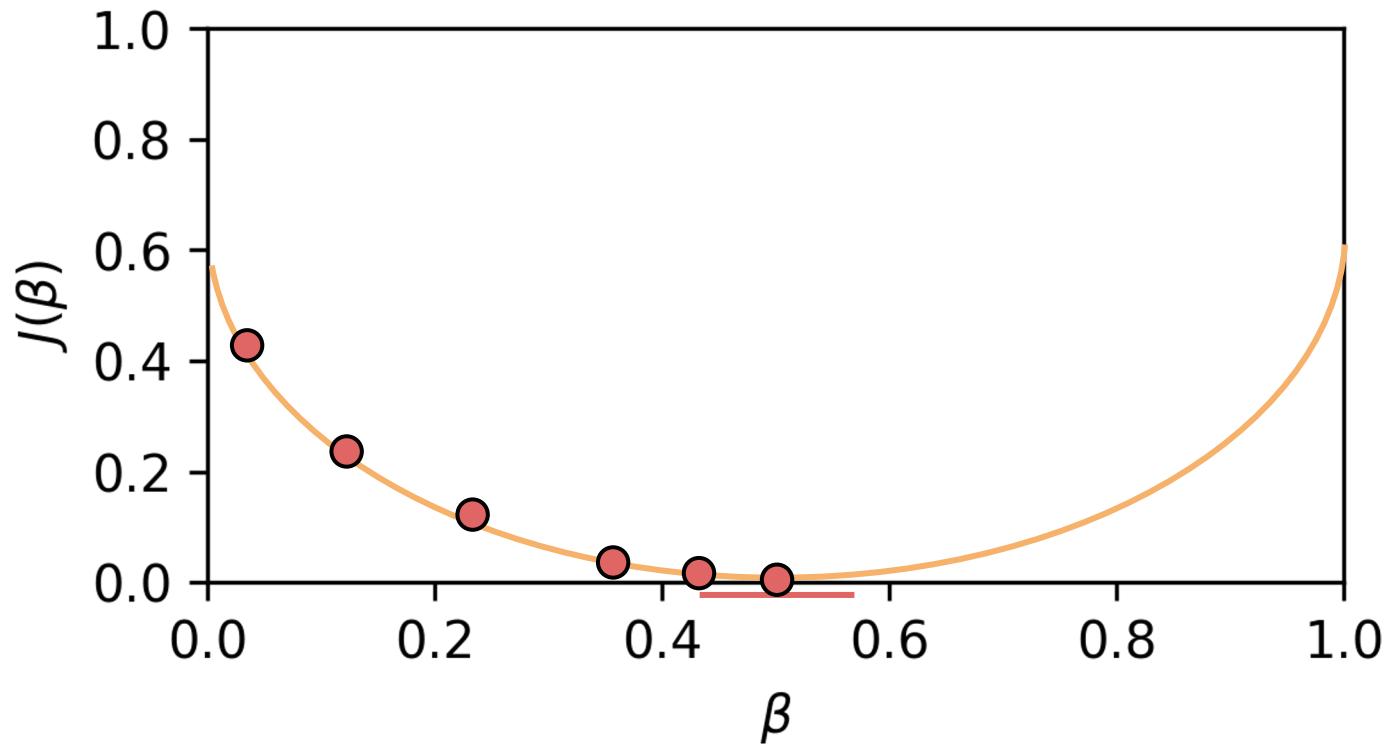
# Linear Regression

- Steps are proportional to negative gradient!



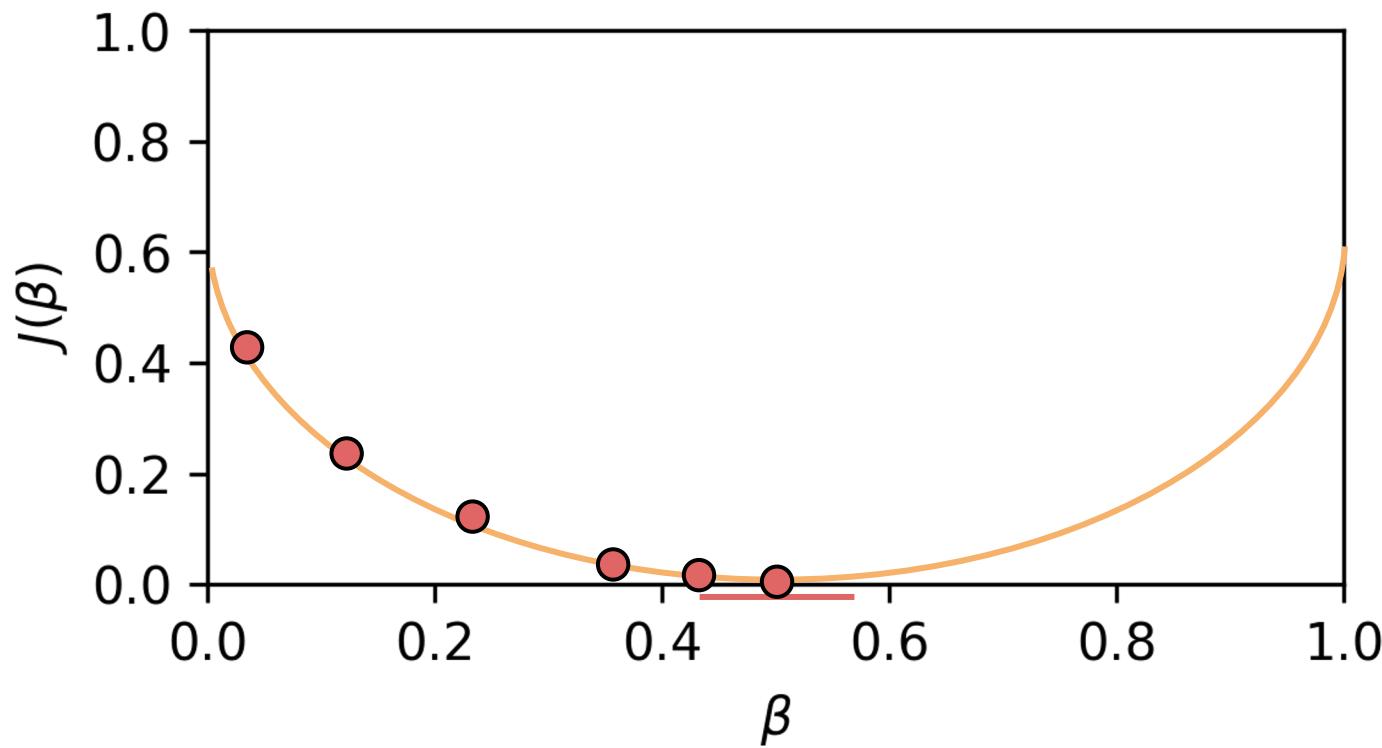
# Linear Regression

- Steeper gradient at start gives larger steps.



# Linear Regression

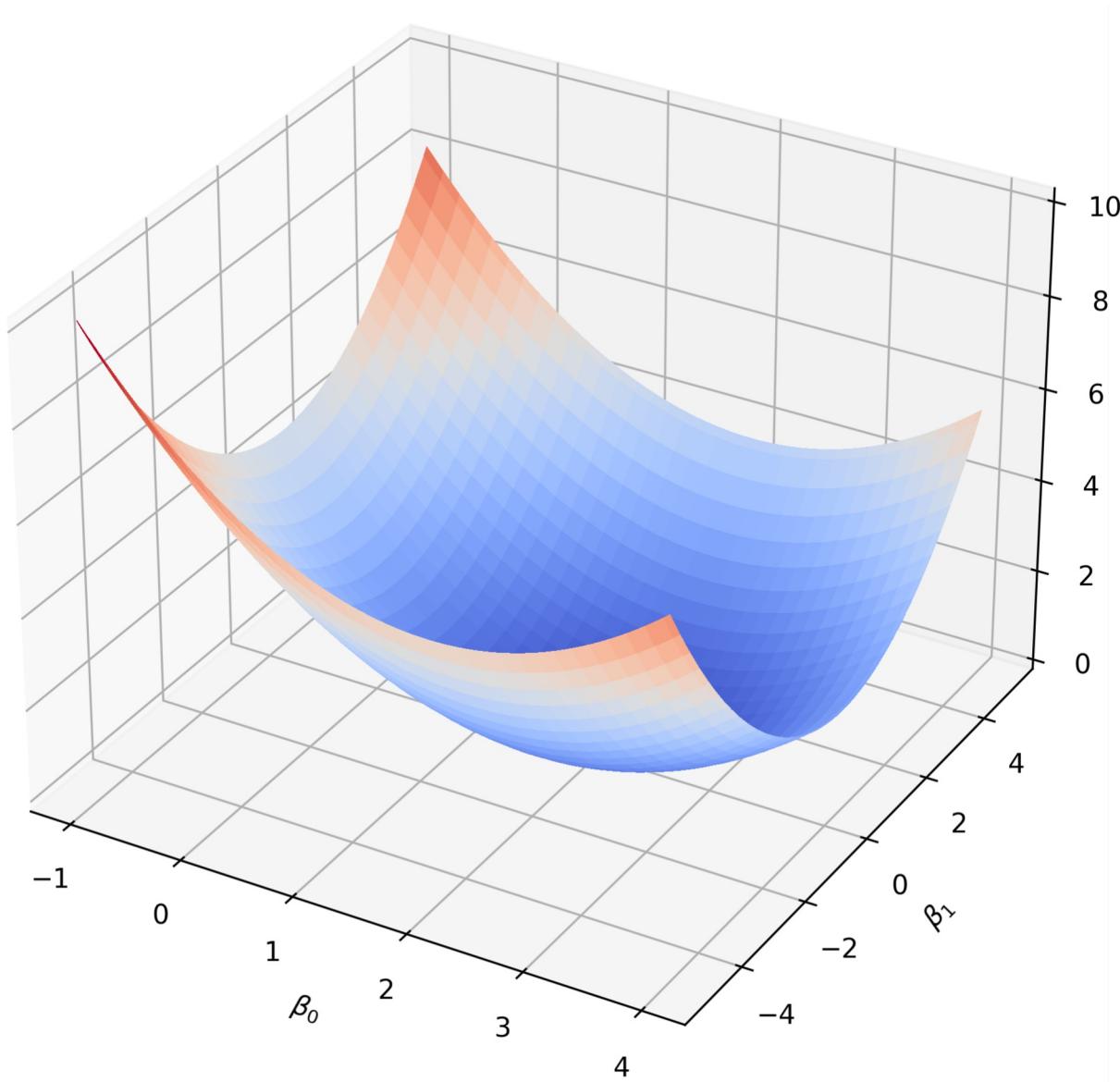
- Smaller gradient at end gives smaller steps.



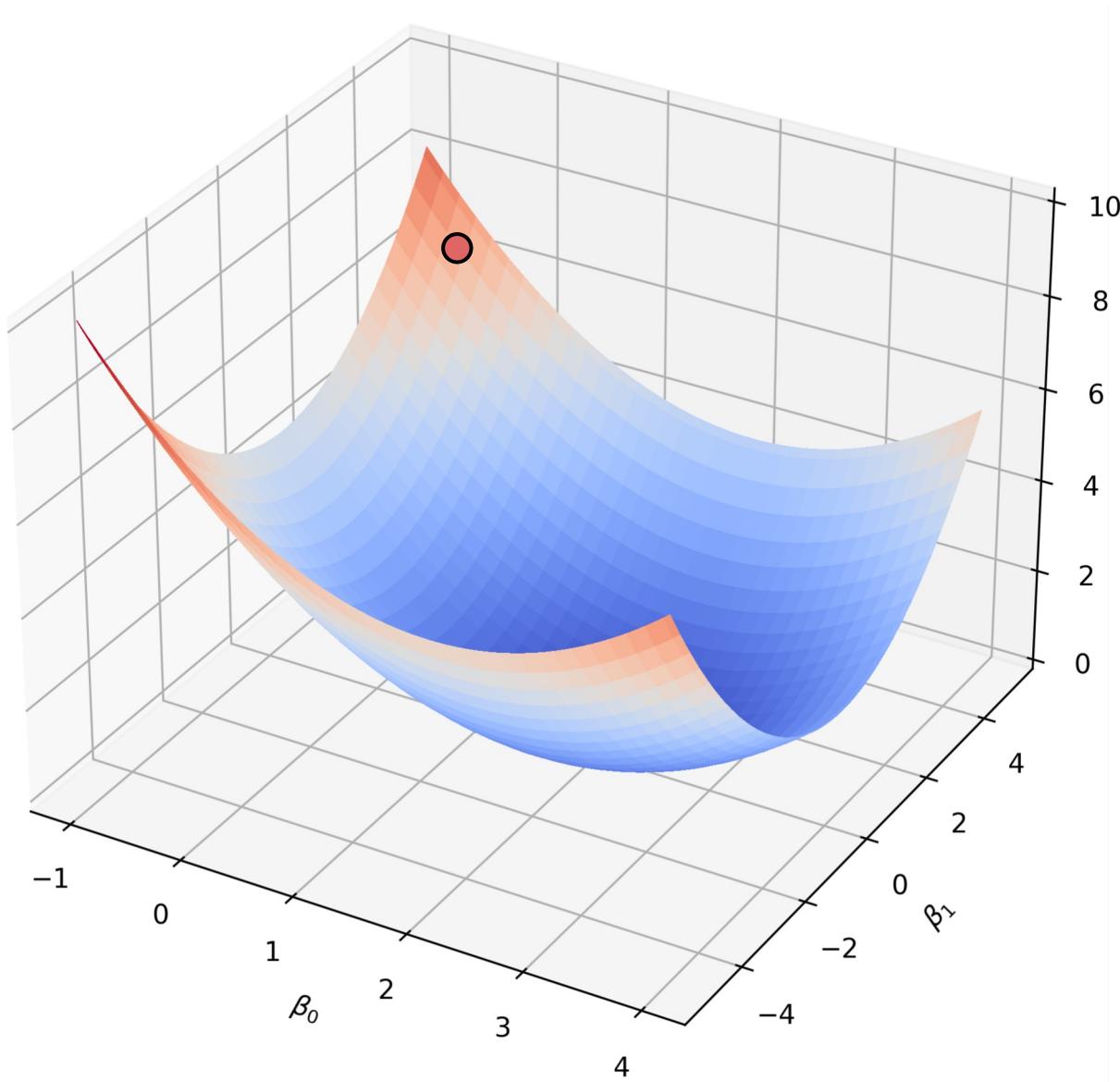
# Linear Regression

- 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.

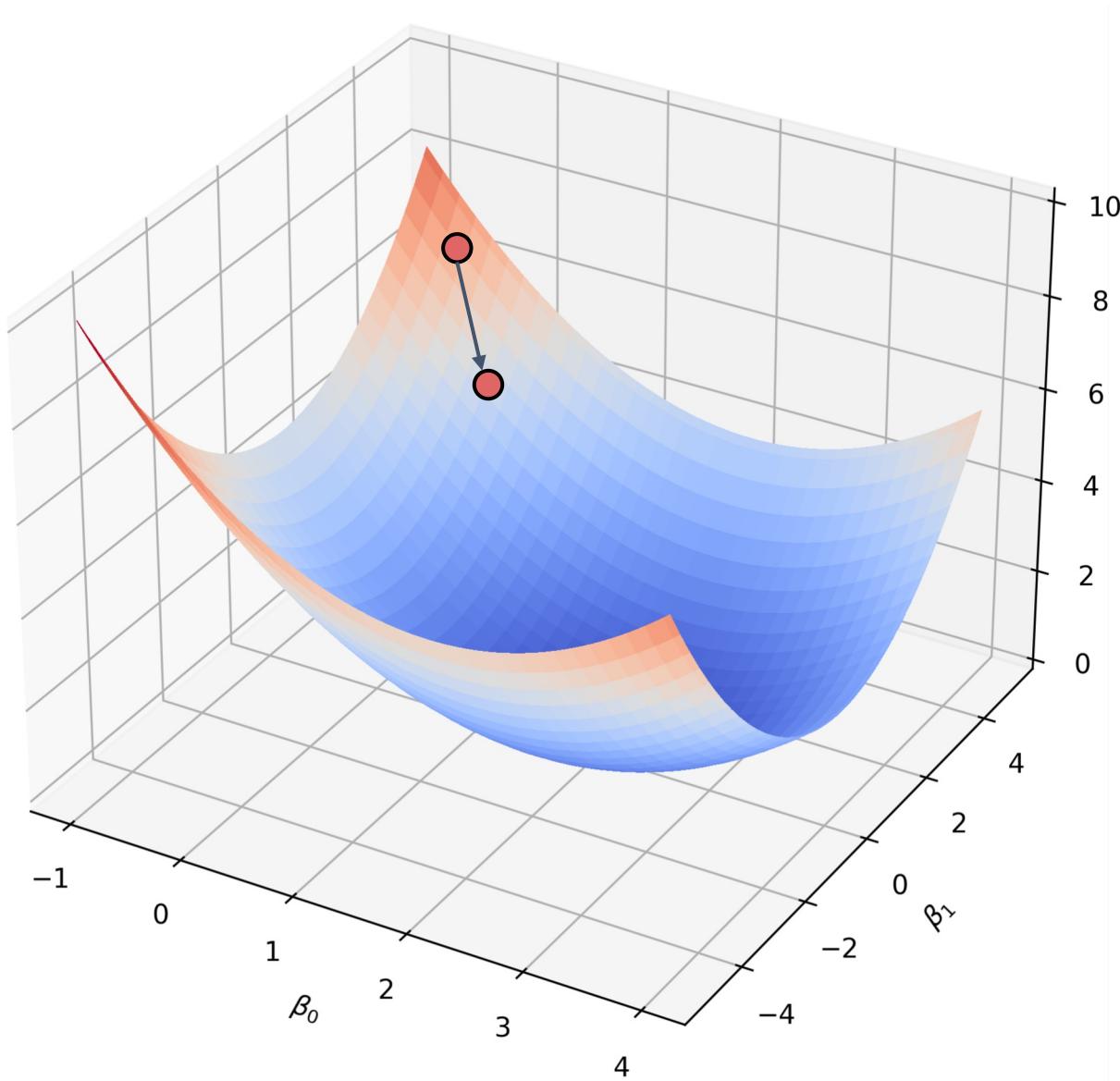
# Linear Regression



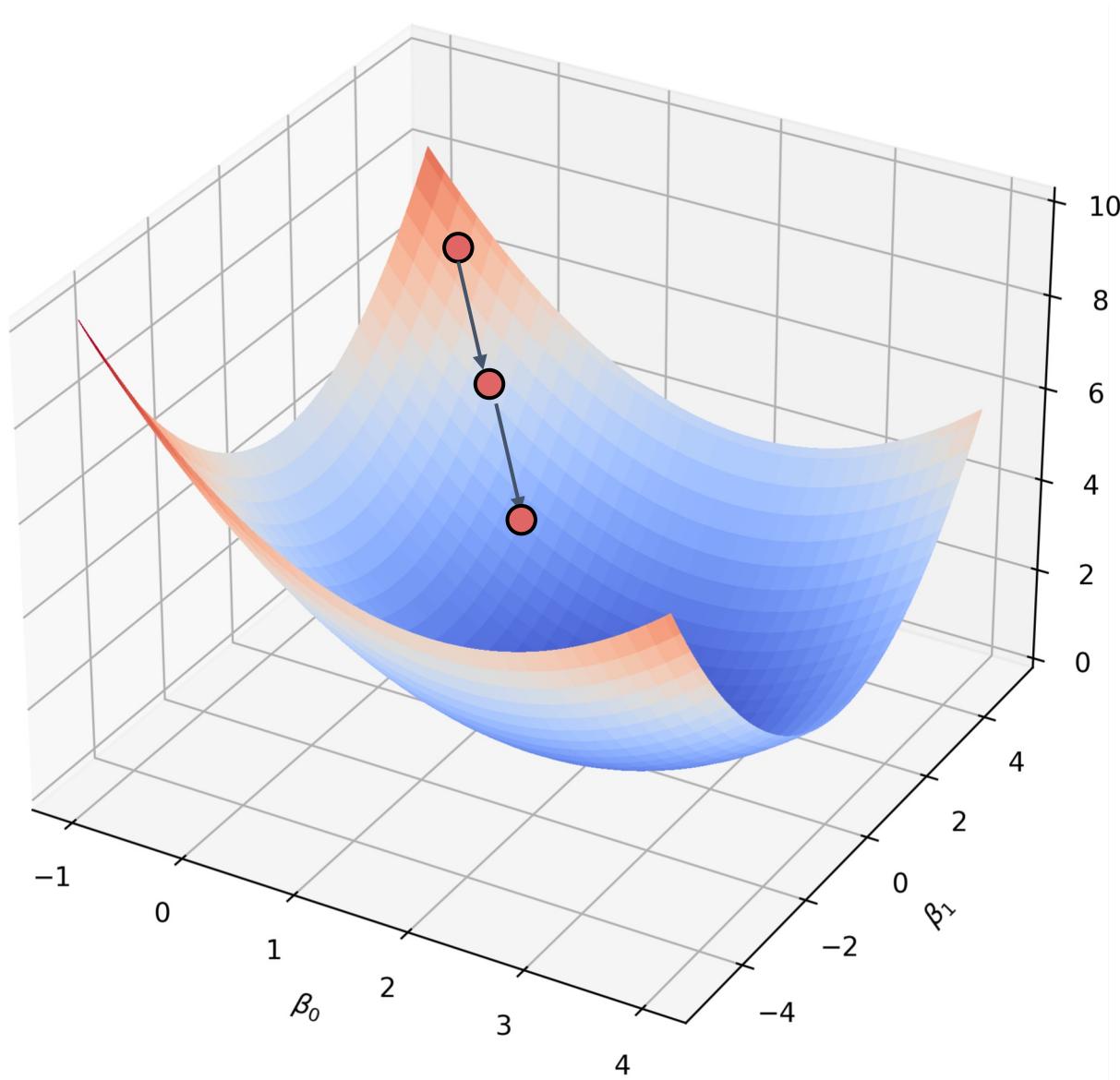
# Linear Regression



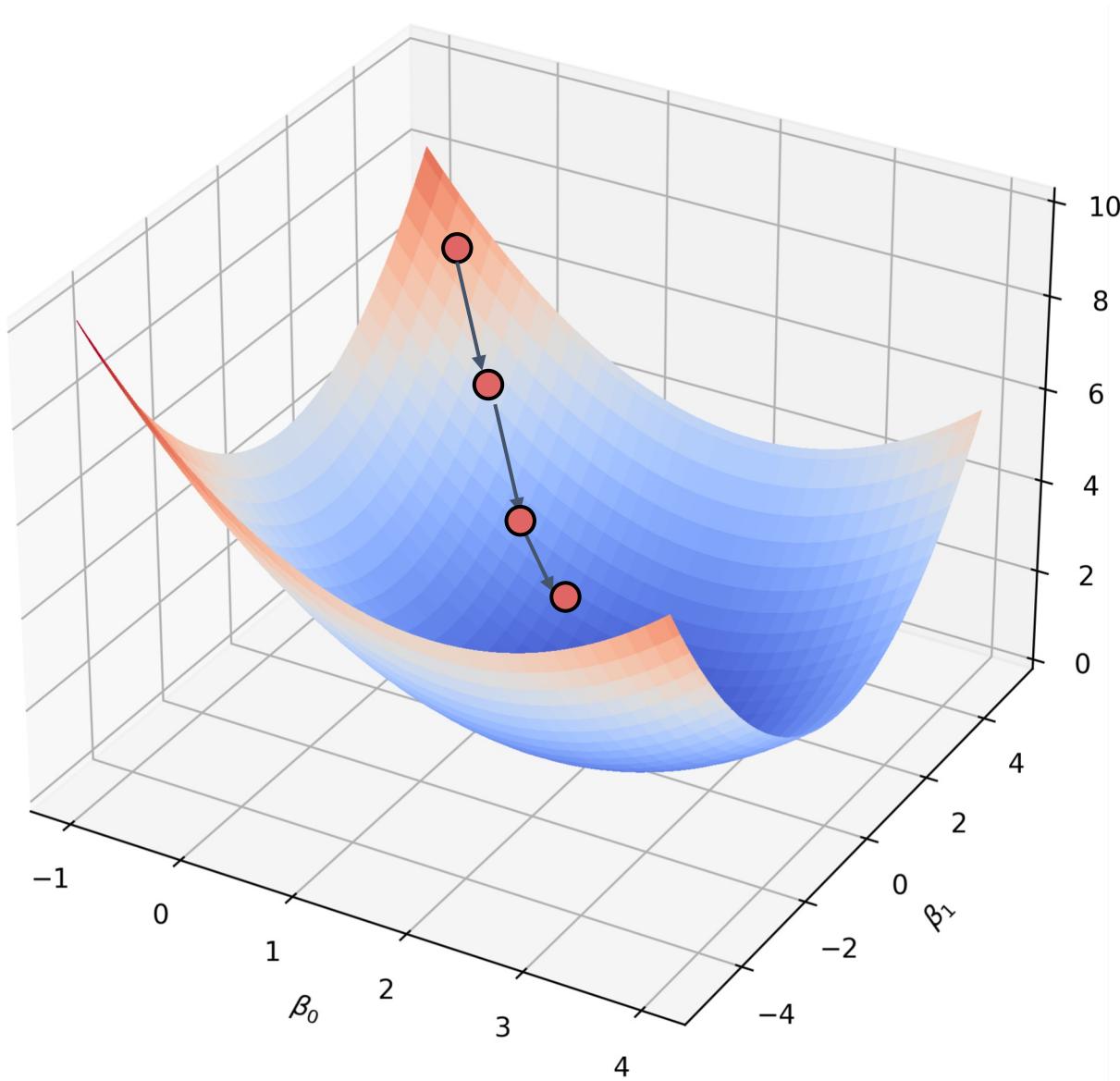
# Linear Regression



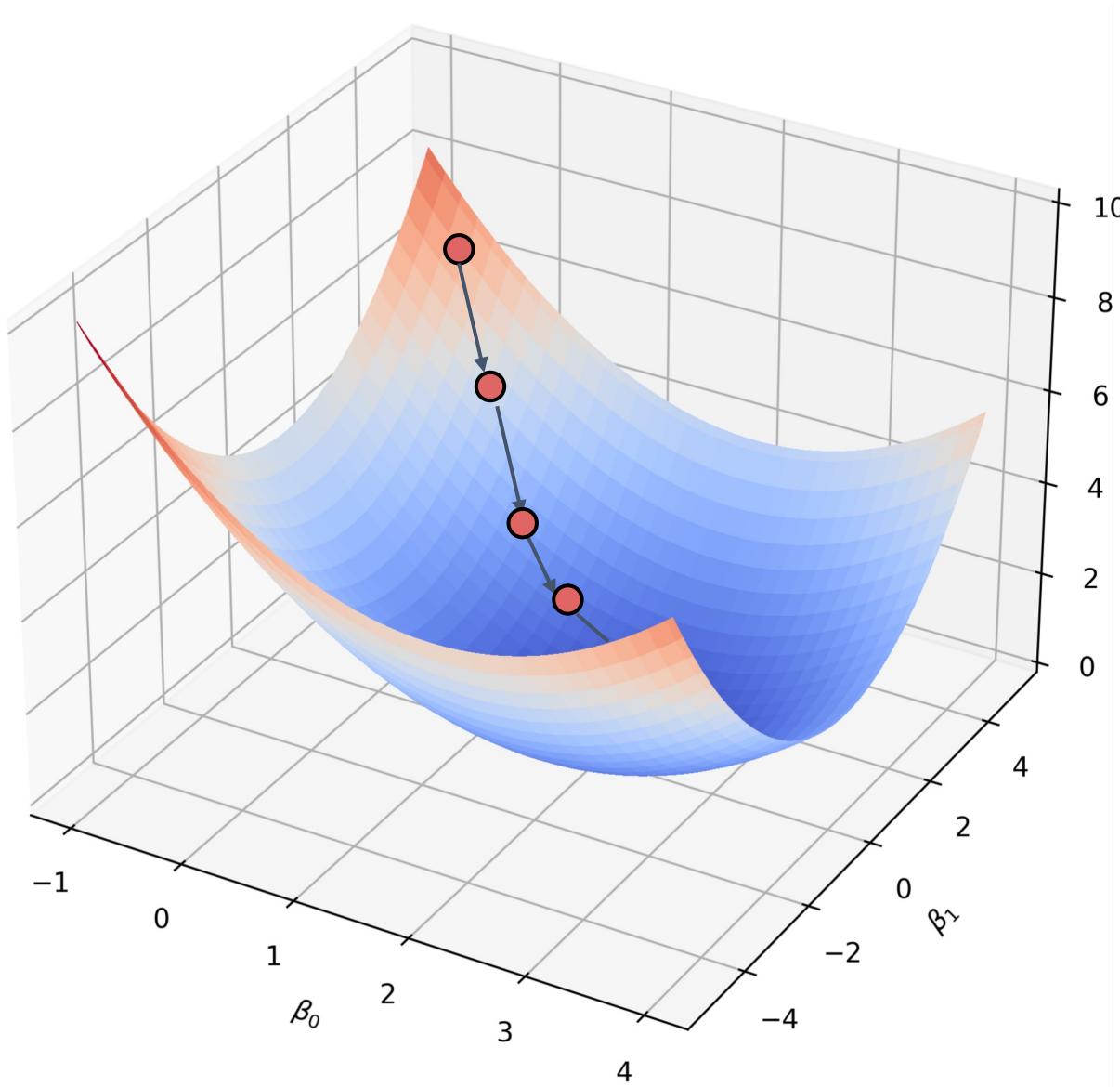
# Linear Regression



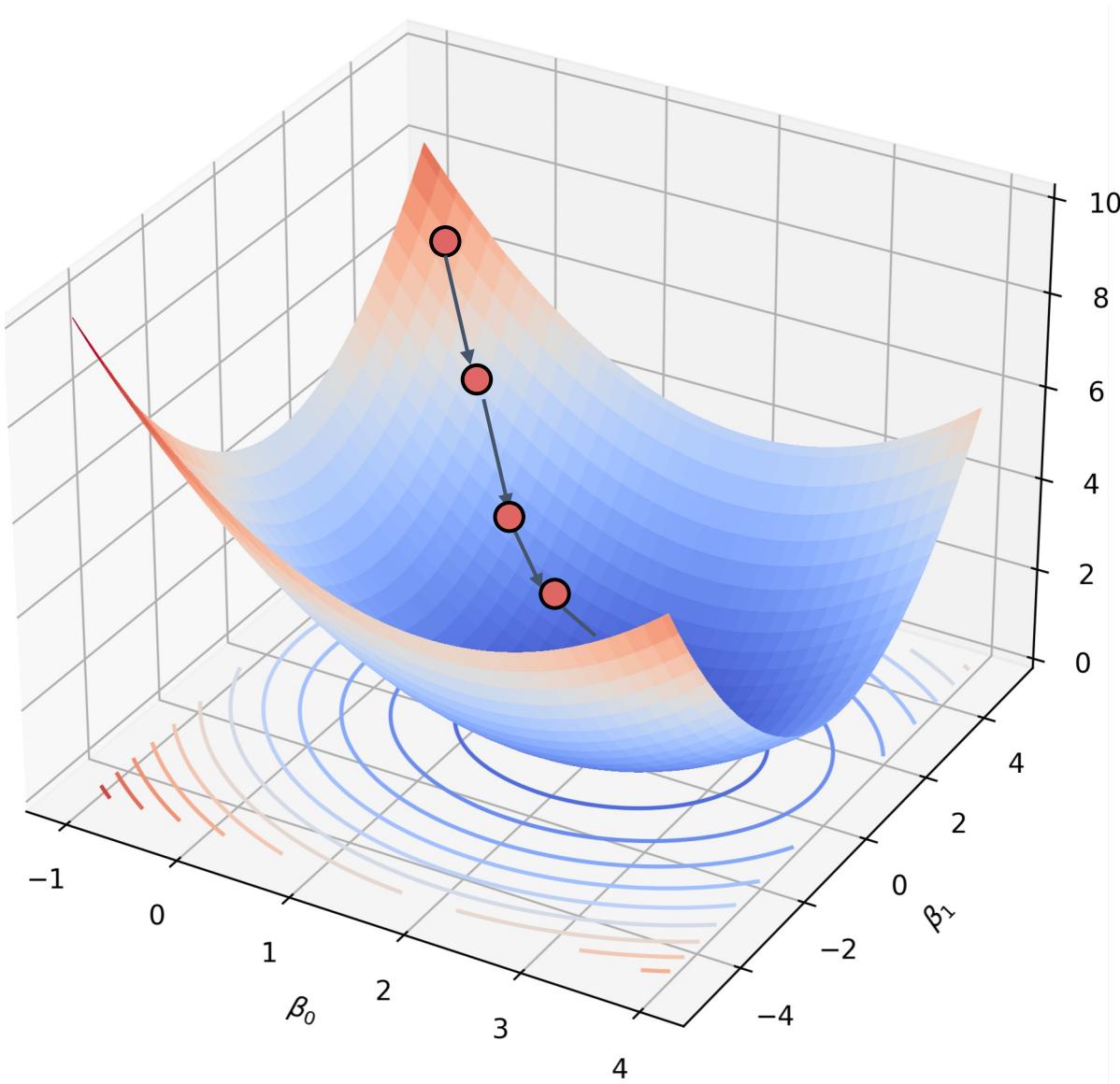
# Linear Regression



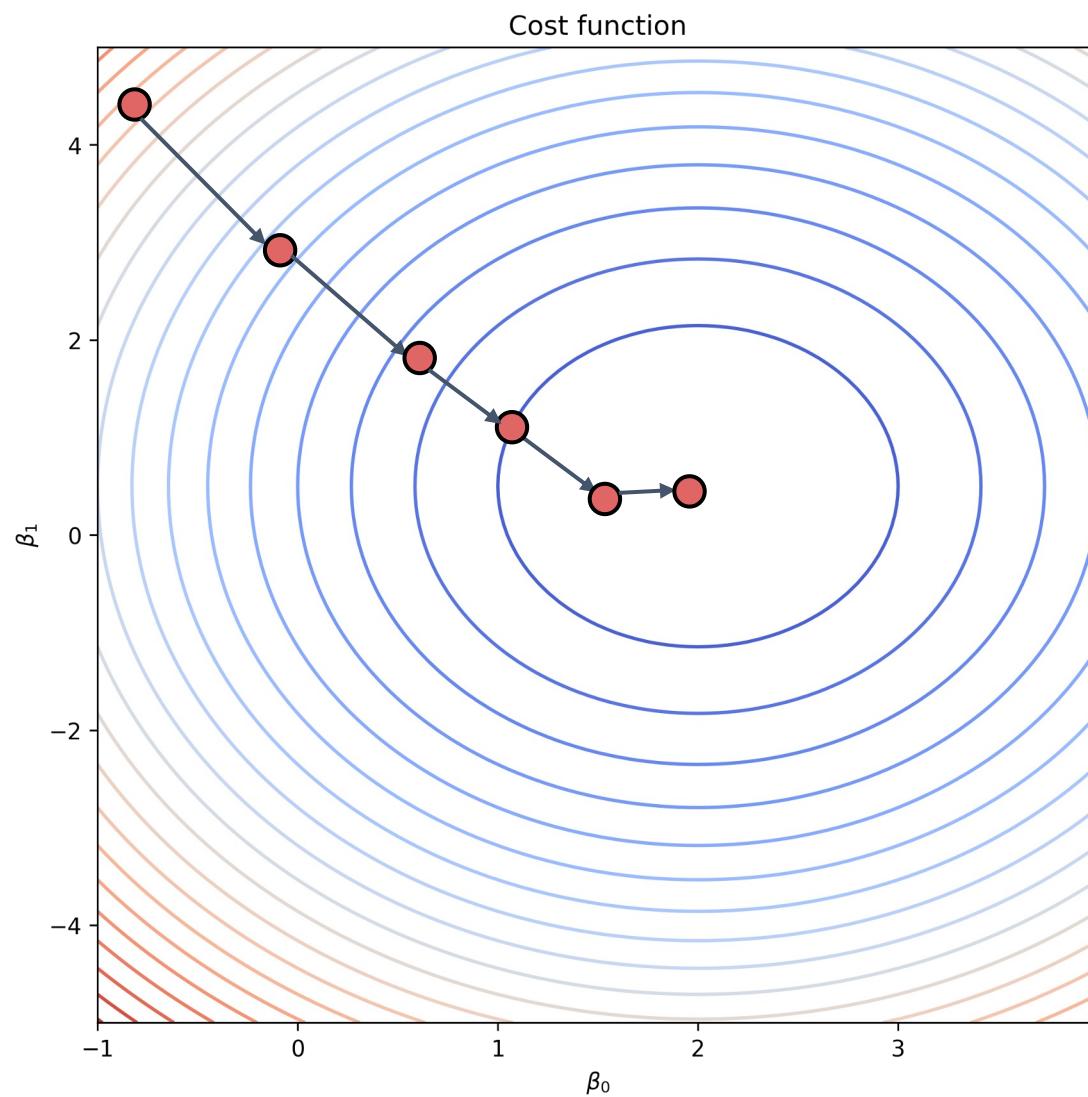
# Linear Regression



# 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!

# **Simple Linear Regression**

# Linear Regression

- Now that we understand what is happening “under the hood” for linear regression, let’s begin by coding through an example of **simple linear regression**.

# Linear Regression

- To help with our understanding, we will start by directly using the advertising data set mentioned in Chapter 3 of ISLR.
- This lecture also serves to start motivating us to think about performance evaluation and multivariate regression.

# Linear Regression

- Simple Linear Regression
  - Limited to one X feature ( $y=mx+b$ )
  - We will create a best-fit line to map out a linear relationship between total advertising spend and resulting sales.
- Let's head over to the notebook!

# **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

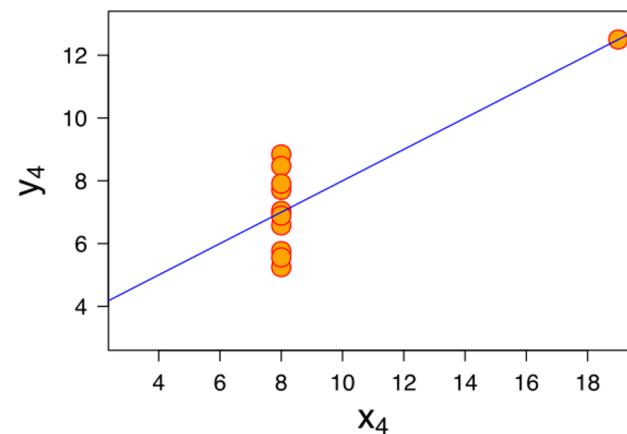
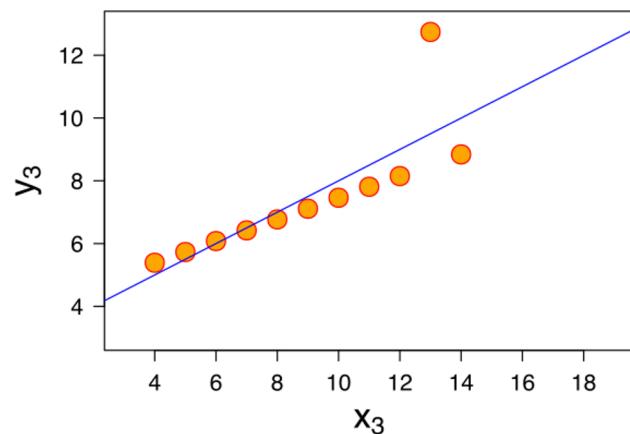
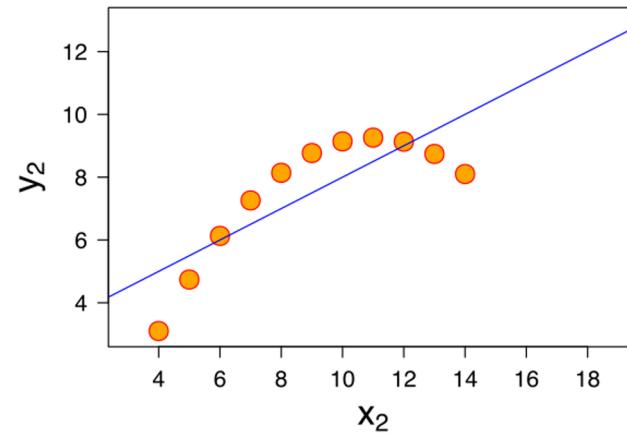
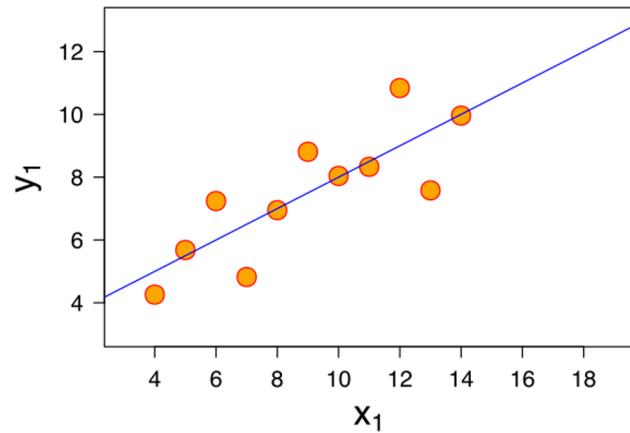
# Evaluating Regression

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

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

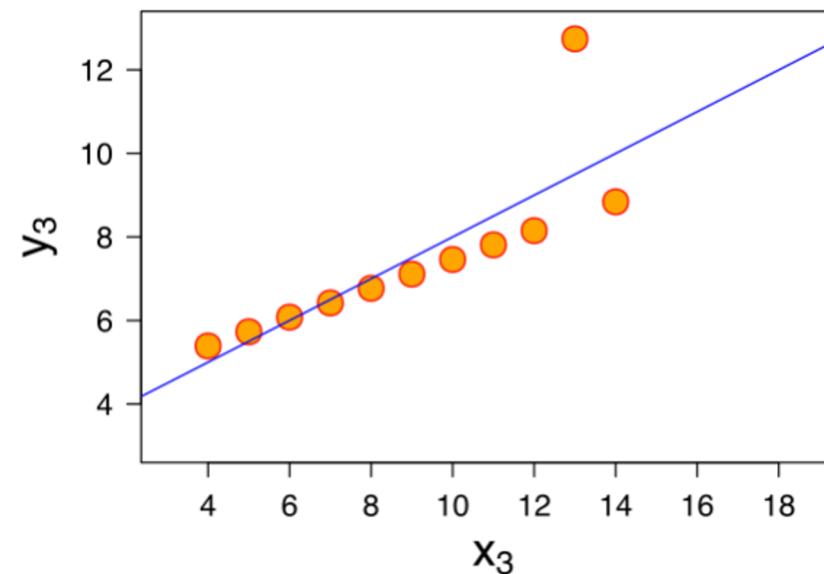
# Evaluating Regression

- MAE won't punish large errors however.



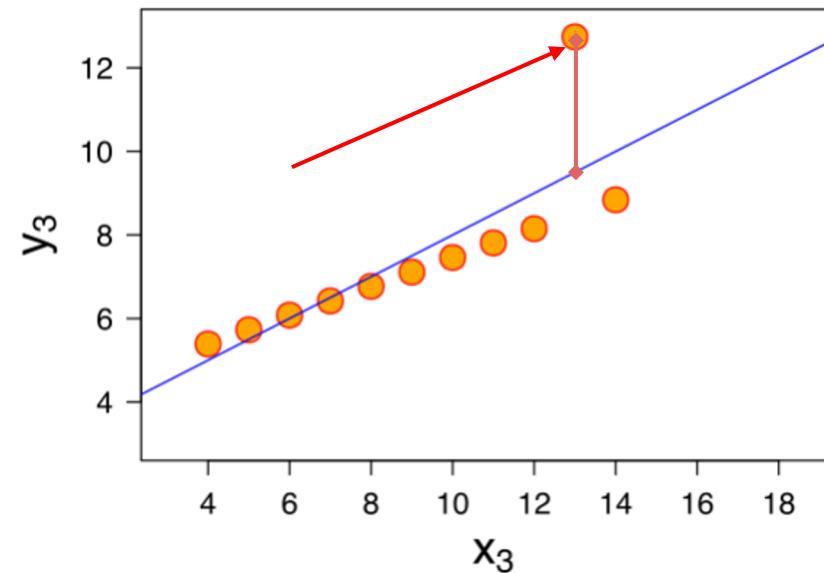
# Evaluating Regression

- 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!

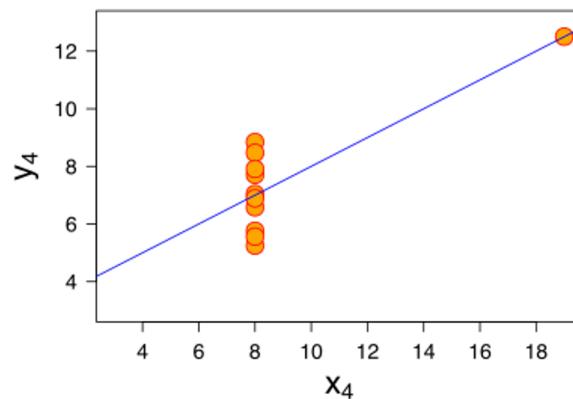
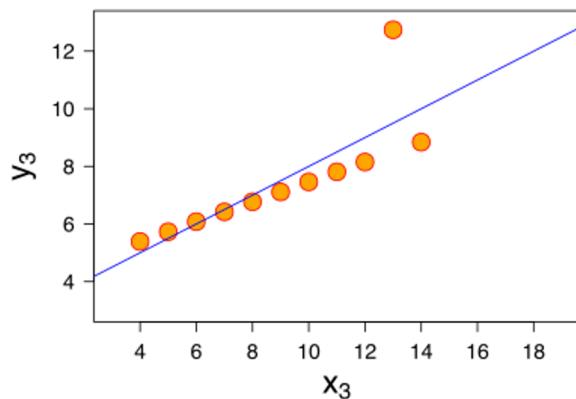
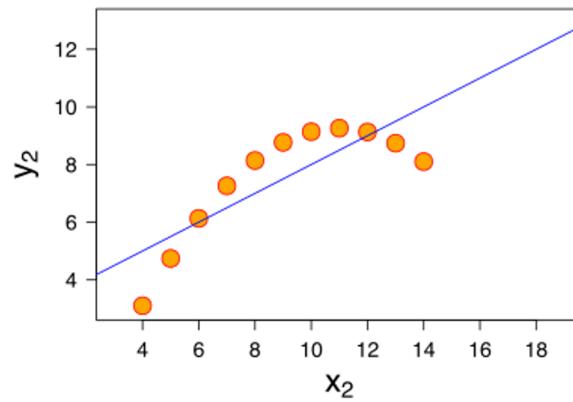
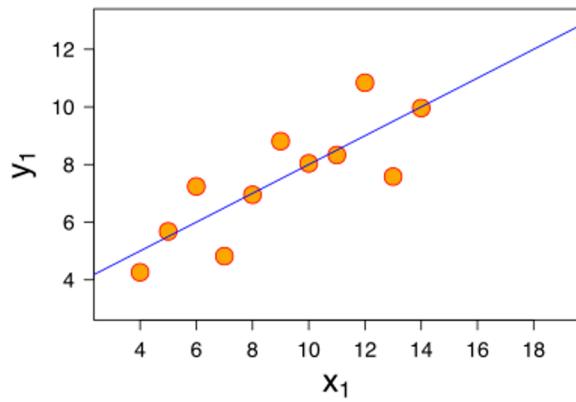
# Evaluating Residuals

# 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...

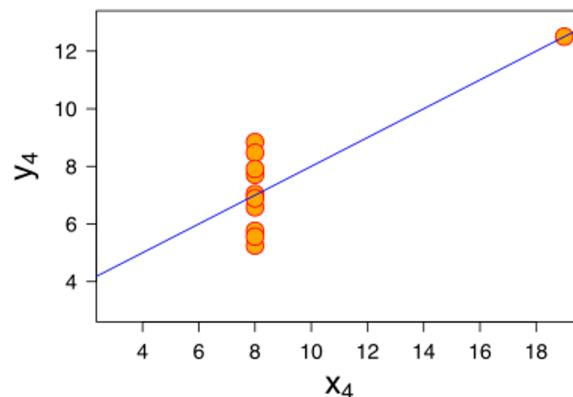
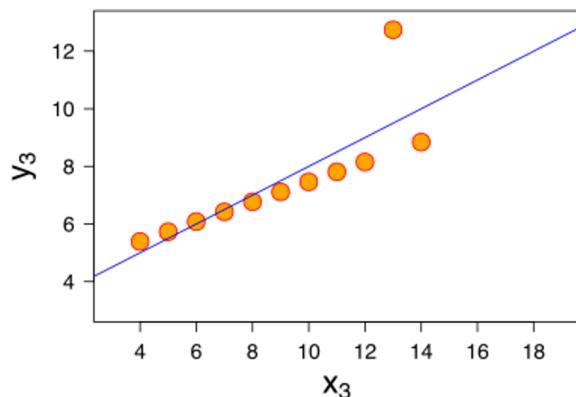
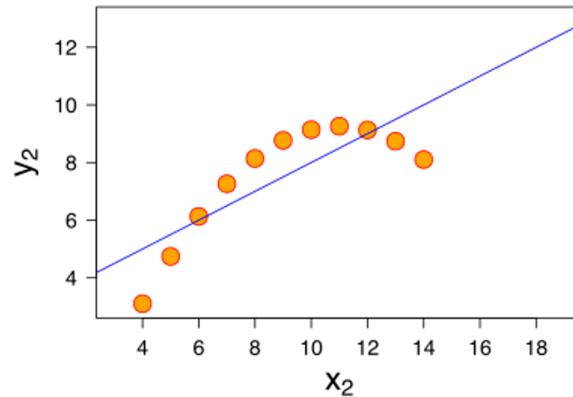
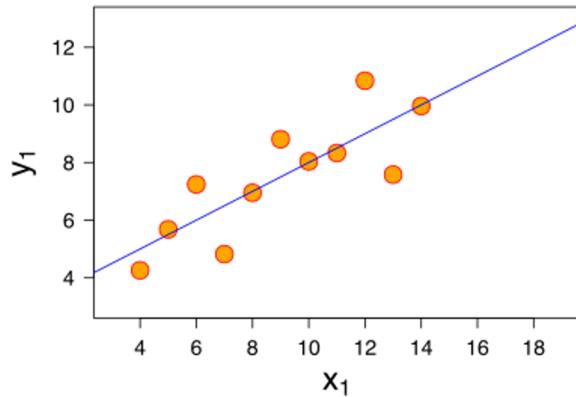
# Linear Regression

- Recall Anscombe's quartet:



# Linear Regression

- Clearly Linear Regression is not suitable!

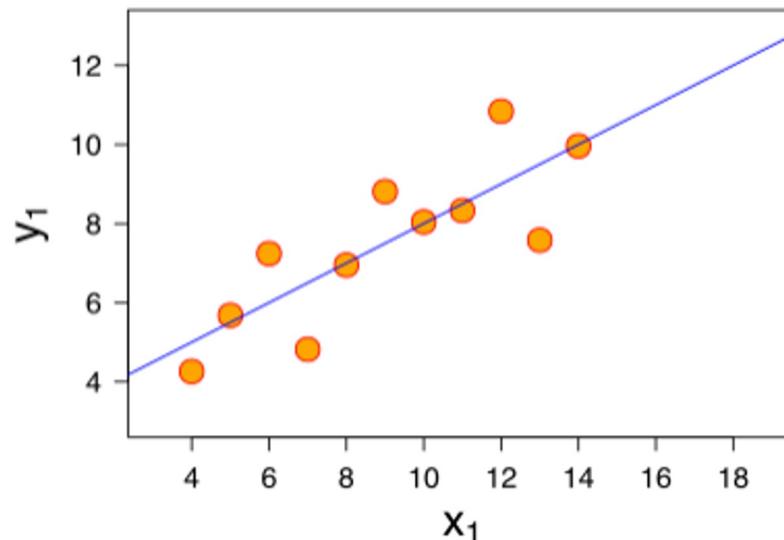


# Linear Regression

- 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!

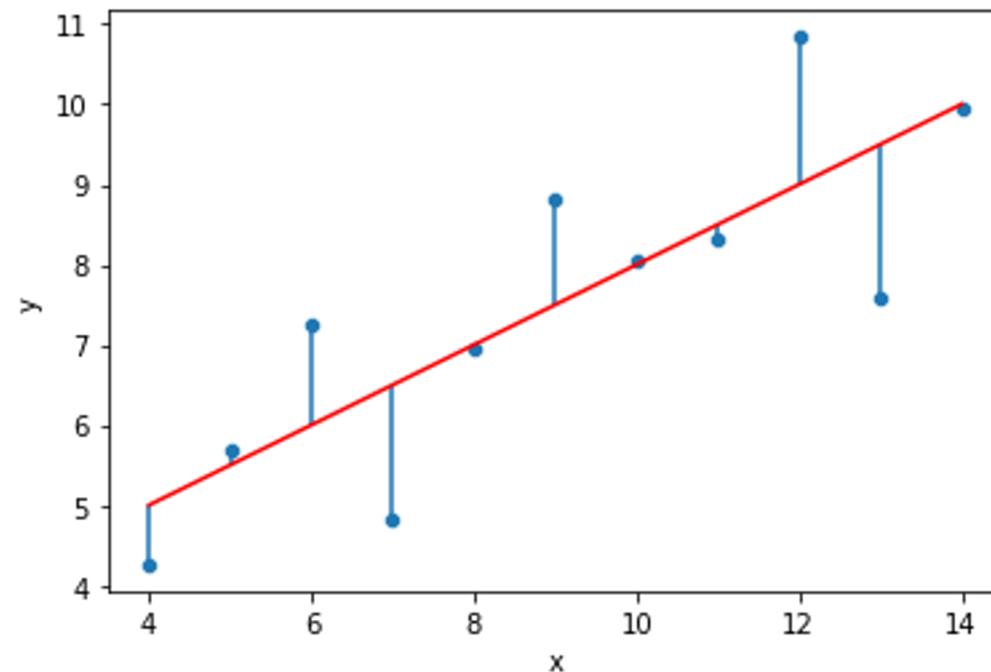
# Linear Regression

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



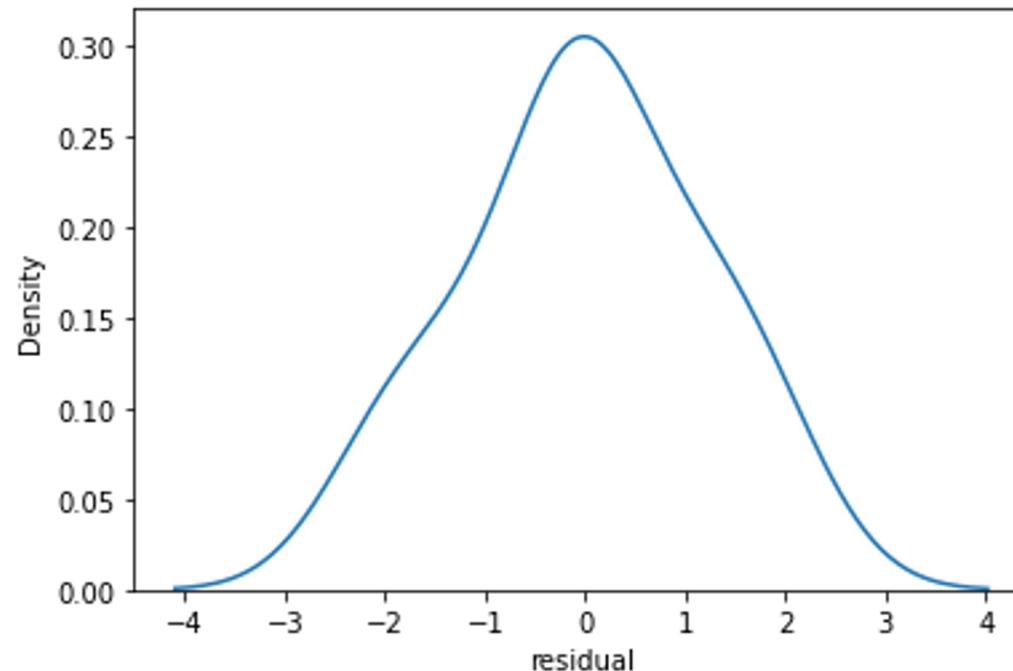
# Linear Regression

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



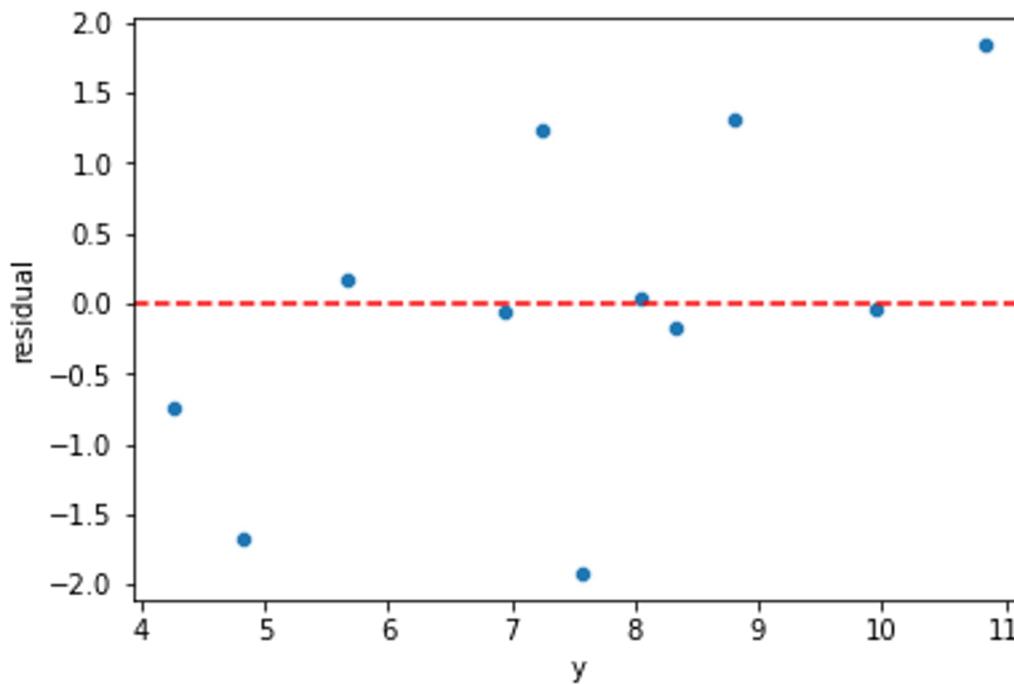
# Linear Regression

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



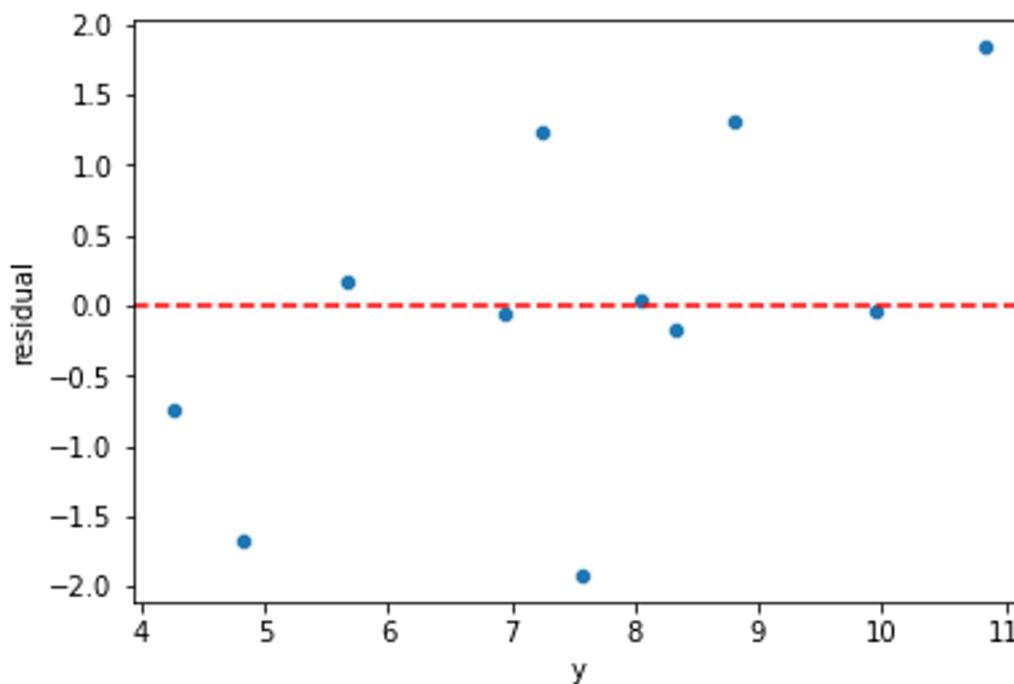
# Linear Regression

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



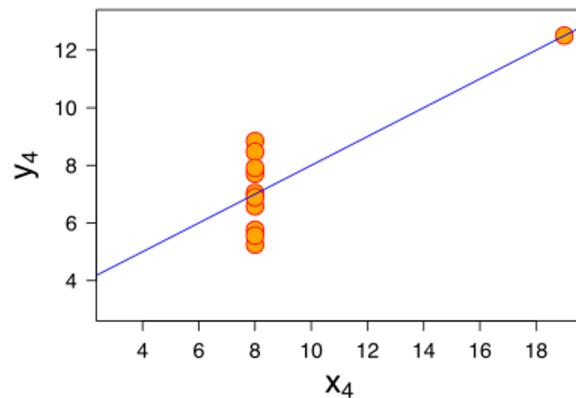
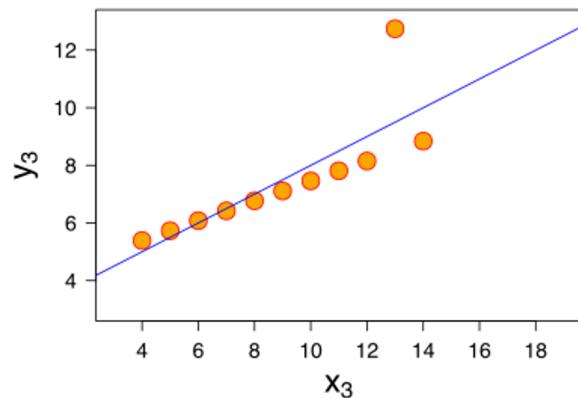
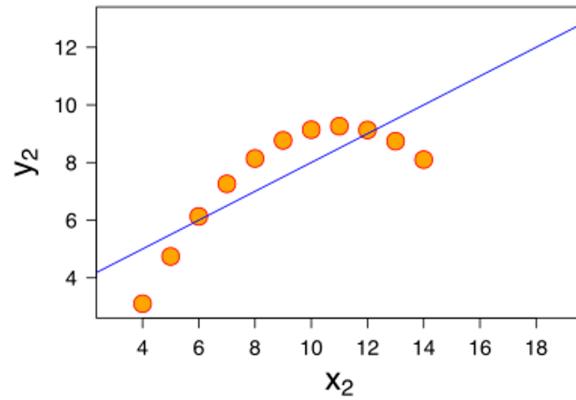
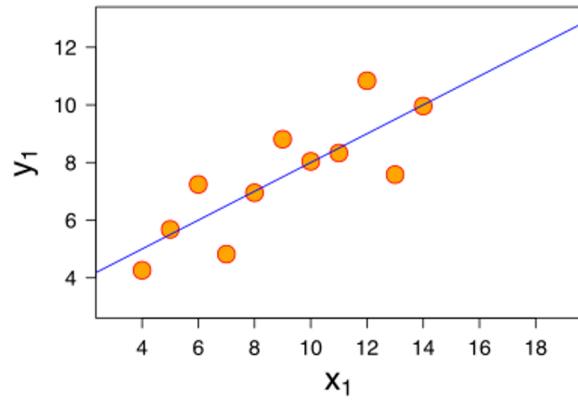
# Linear Regression

- There should be no clear line or curve.



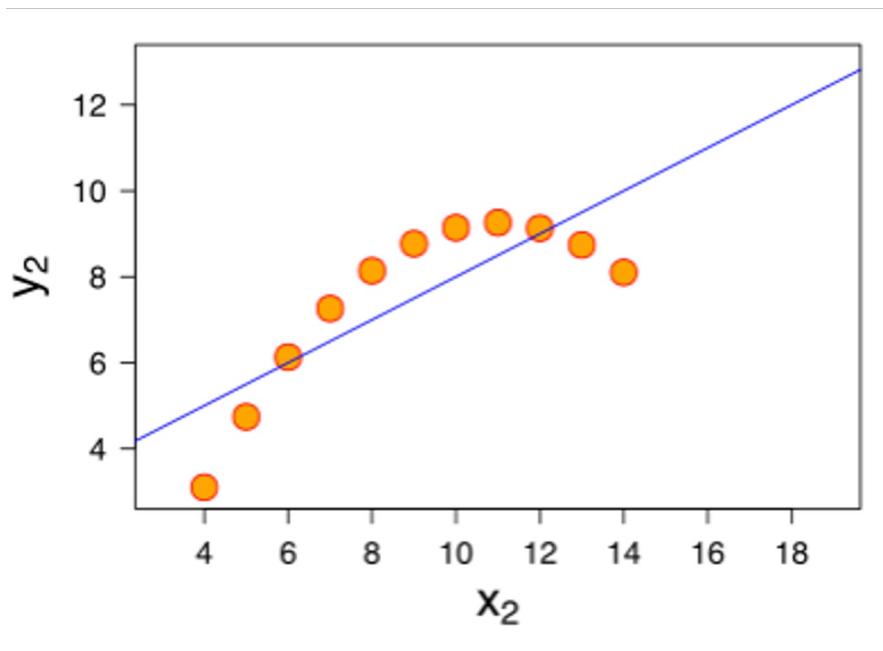
# Linear Regression

- What about non valid datasets?



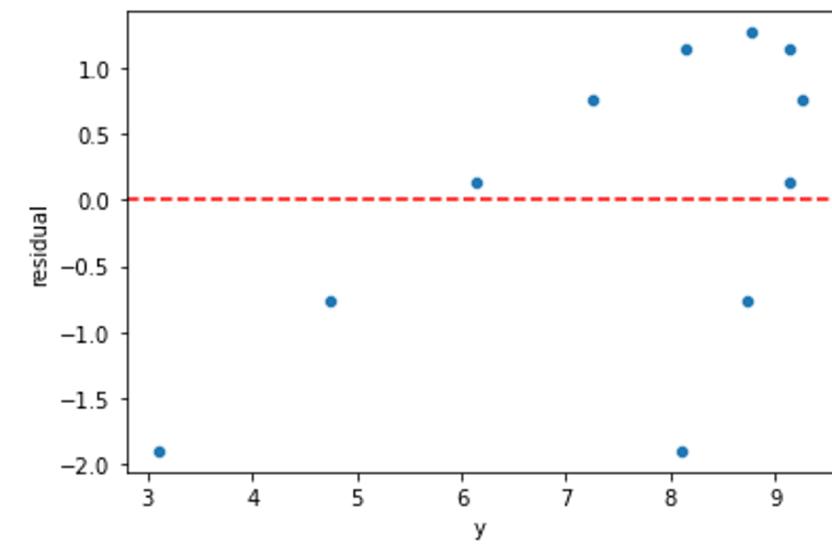
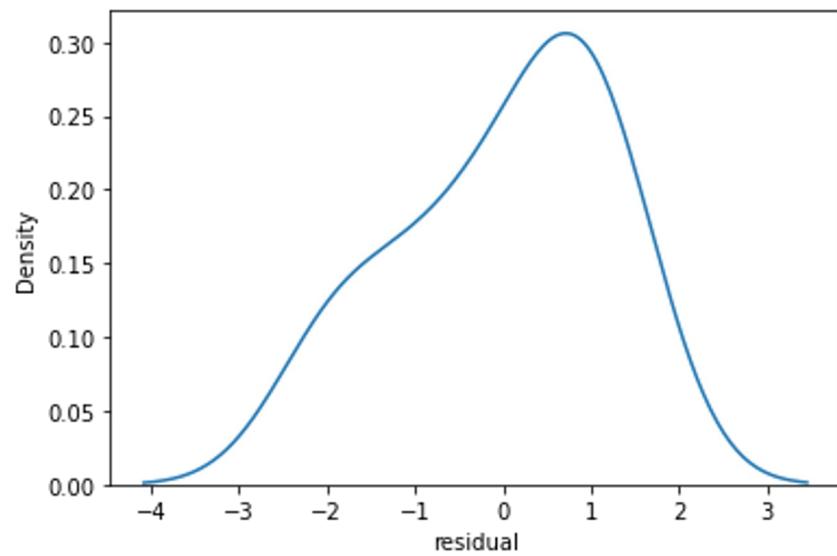
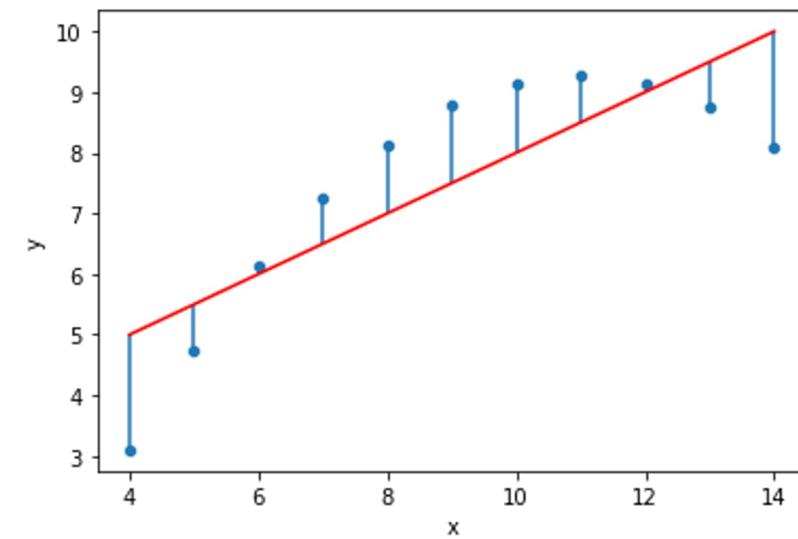
# Linear Regression

- What about non valid datasets?



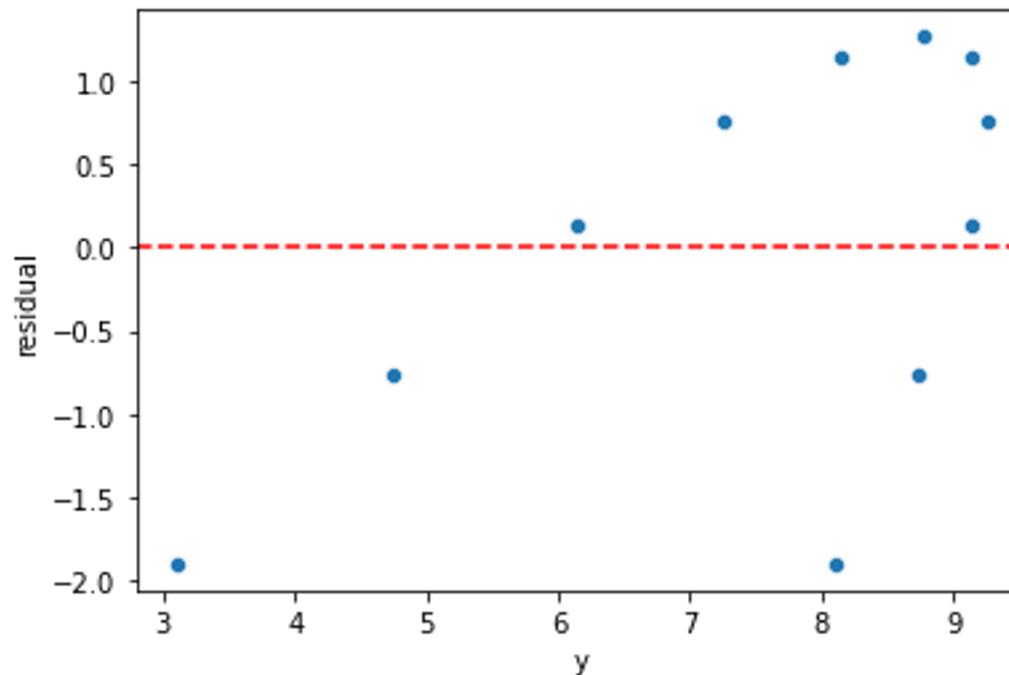
# Linear Regression

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



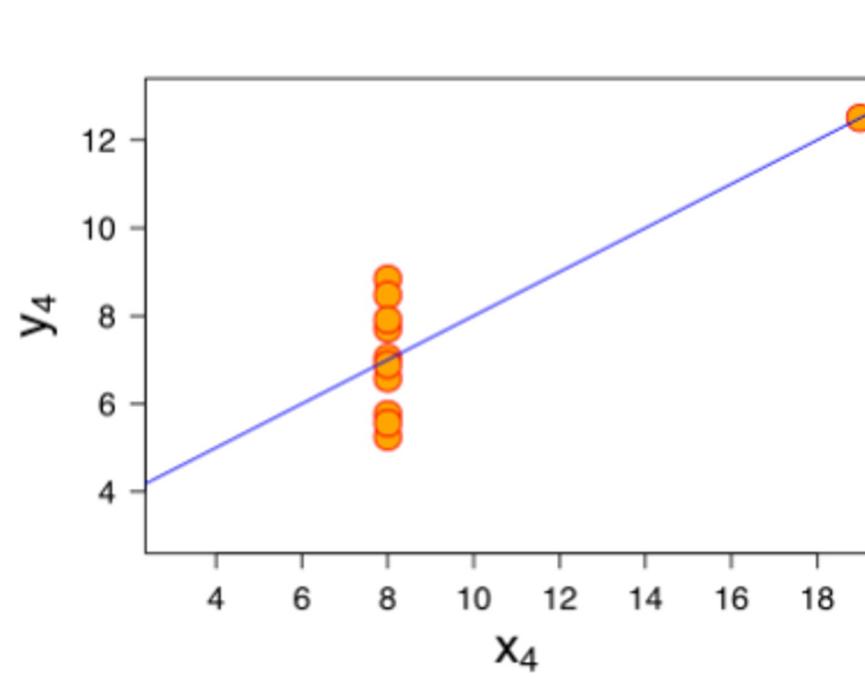
# Linear Regression

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



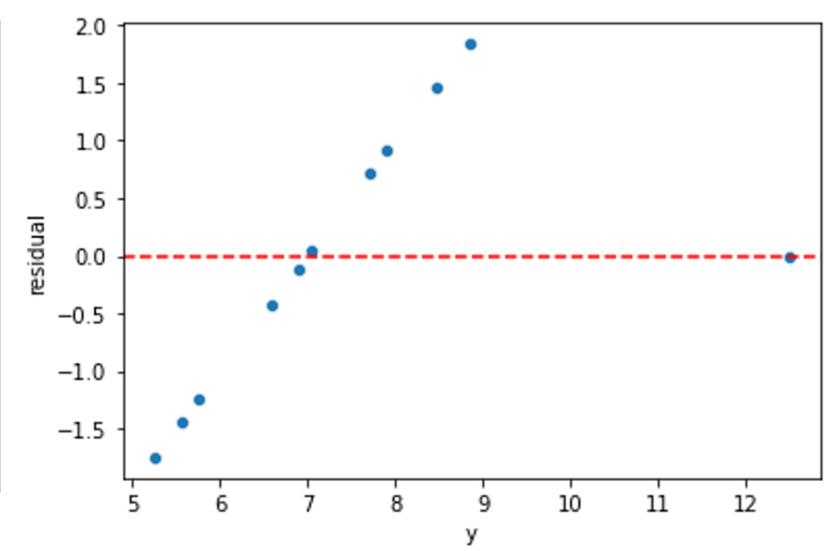
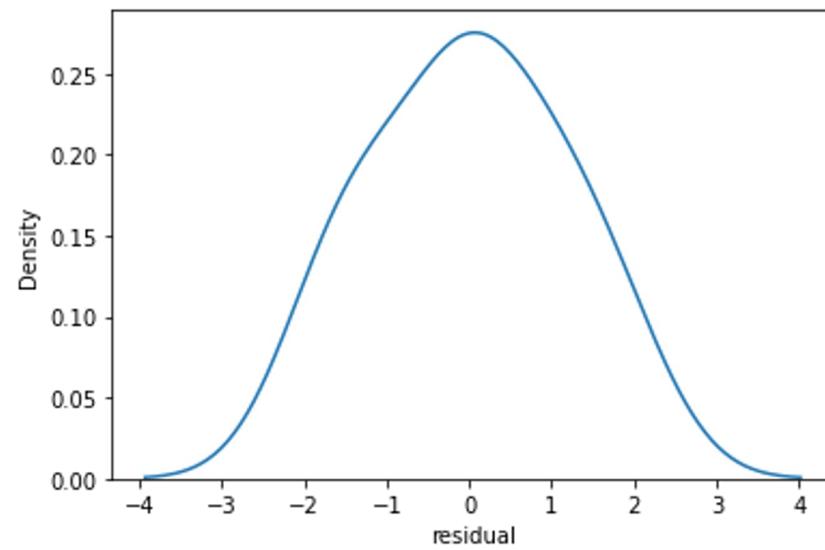
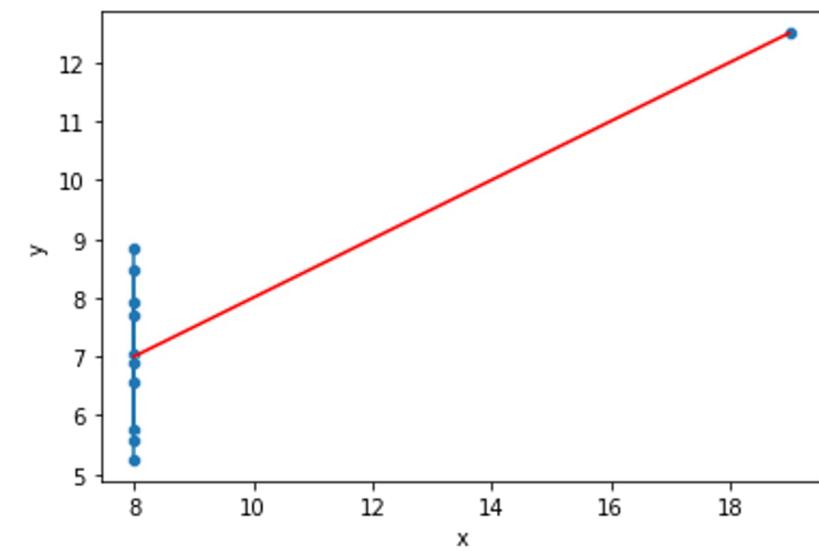
# Linear Regression

- What about non valid datasets?



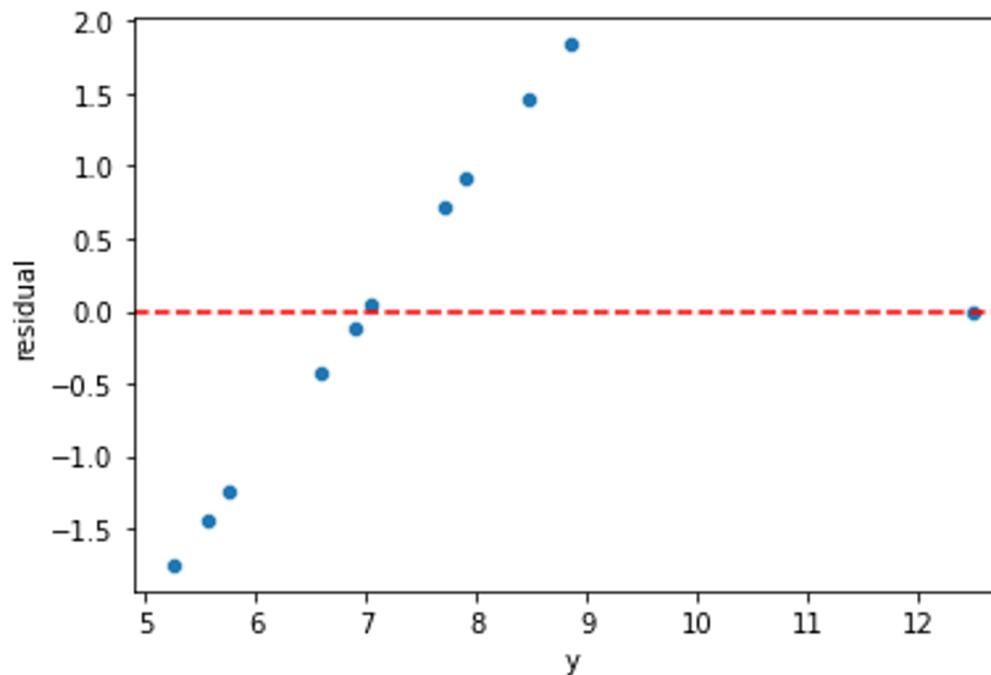
# Linear Regression

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



# Linear Regression

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



# Linear Regression

- Let's explore creating these plots with Python and our model results!

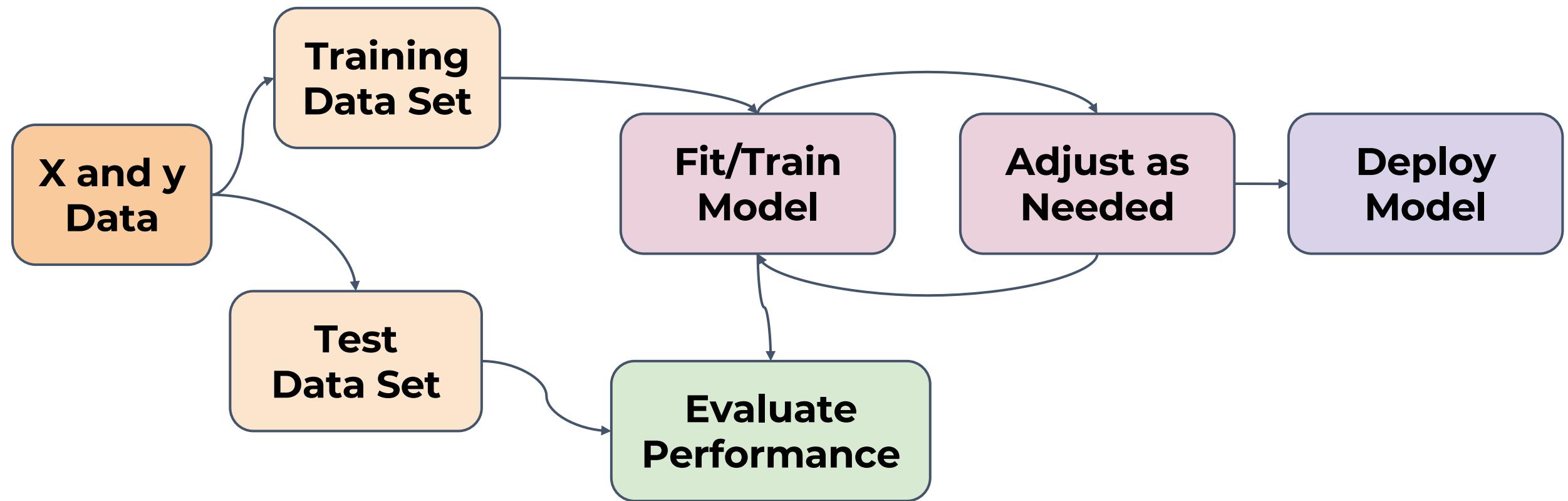
# **Model Deployment**

# Linear Regression

- We're almost done with our first machine learning run through!
- Let's quickly review what we've done so far in the ML process.

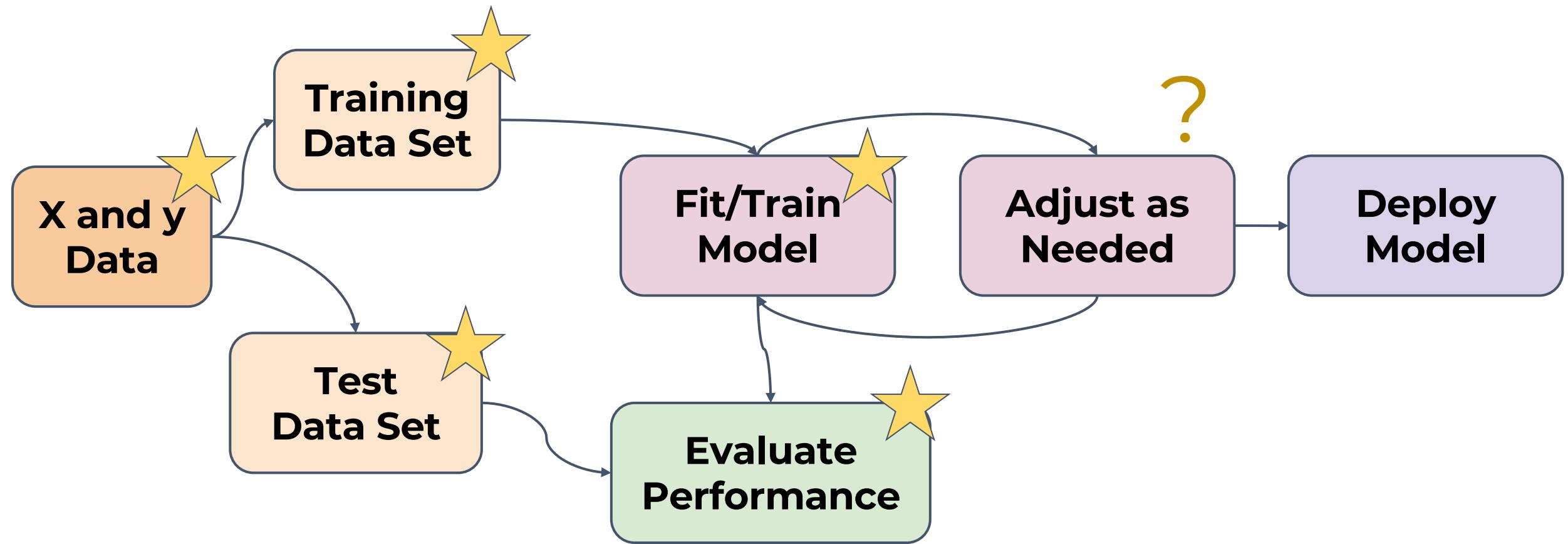
# Supervised Machine Learning Process

- Recall the Supervised ML Process



# Supervised Machine Learning Process

- Recall the Supervised ML Process



# Linear Regression

- Later on we will explore polynomial regression and regularization as model adjustments.
- For now, let's focus on a simple “deployment” of our model by saving and loading it, then applying to new data.

# **Polynomial Regression**

## Theory and Motivation

# Polynomial Regression

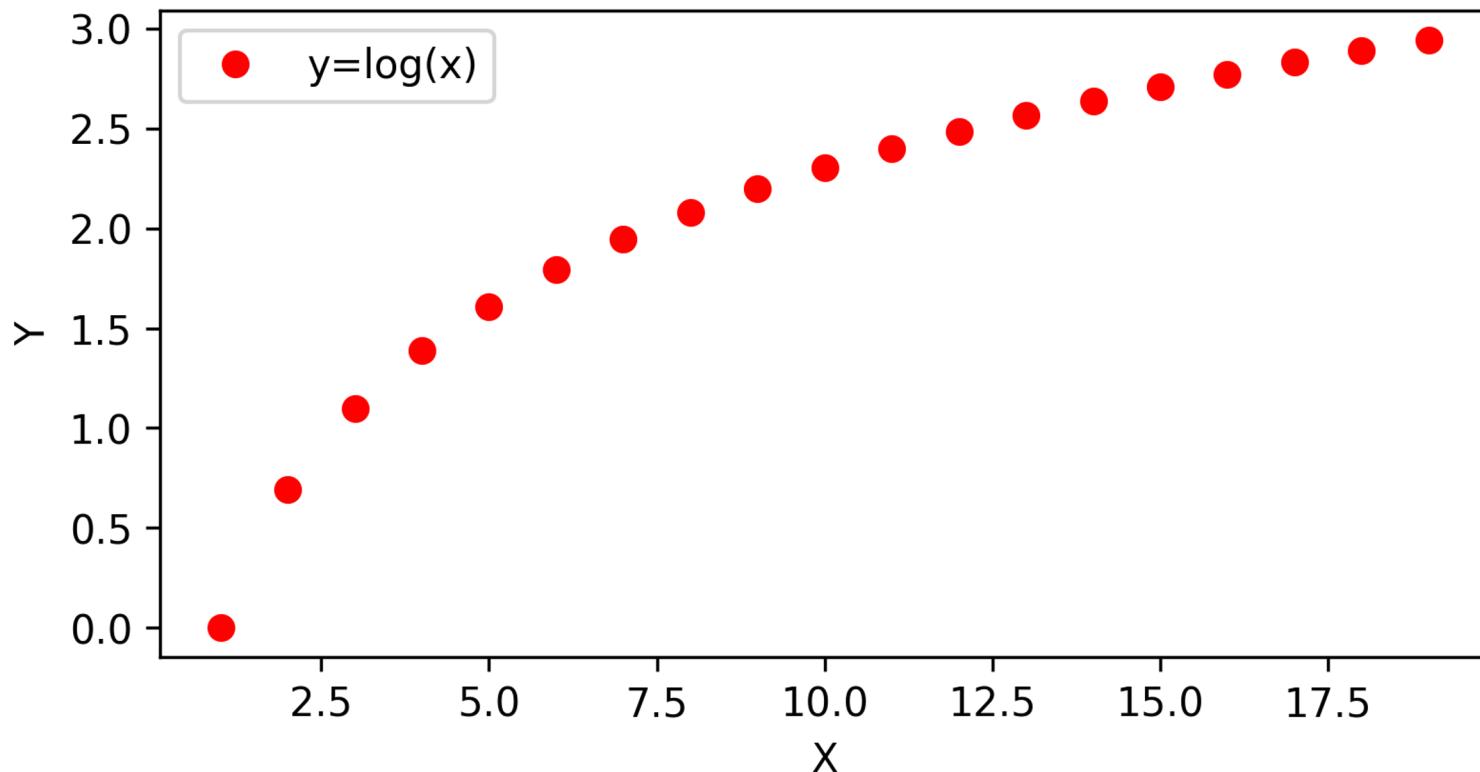
- We just completed a Linear Regression task, allowing us to predict future label values given a set of features!
- How can we now improve on a Linear Regression model?
- One approach is to consider **higher order relationships** on the features.

# Polynomial Regression

- There are two issues polynomial regression will address for us:
  - Non-linear feature relationships to label
  - Interaction terms between features
- Let's first explore non-linear relationships and how considering polynomial orders could help address this.

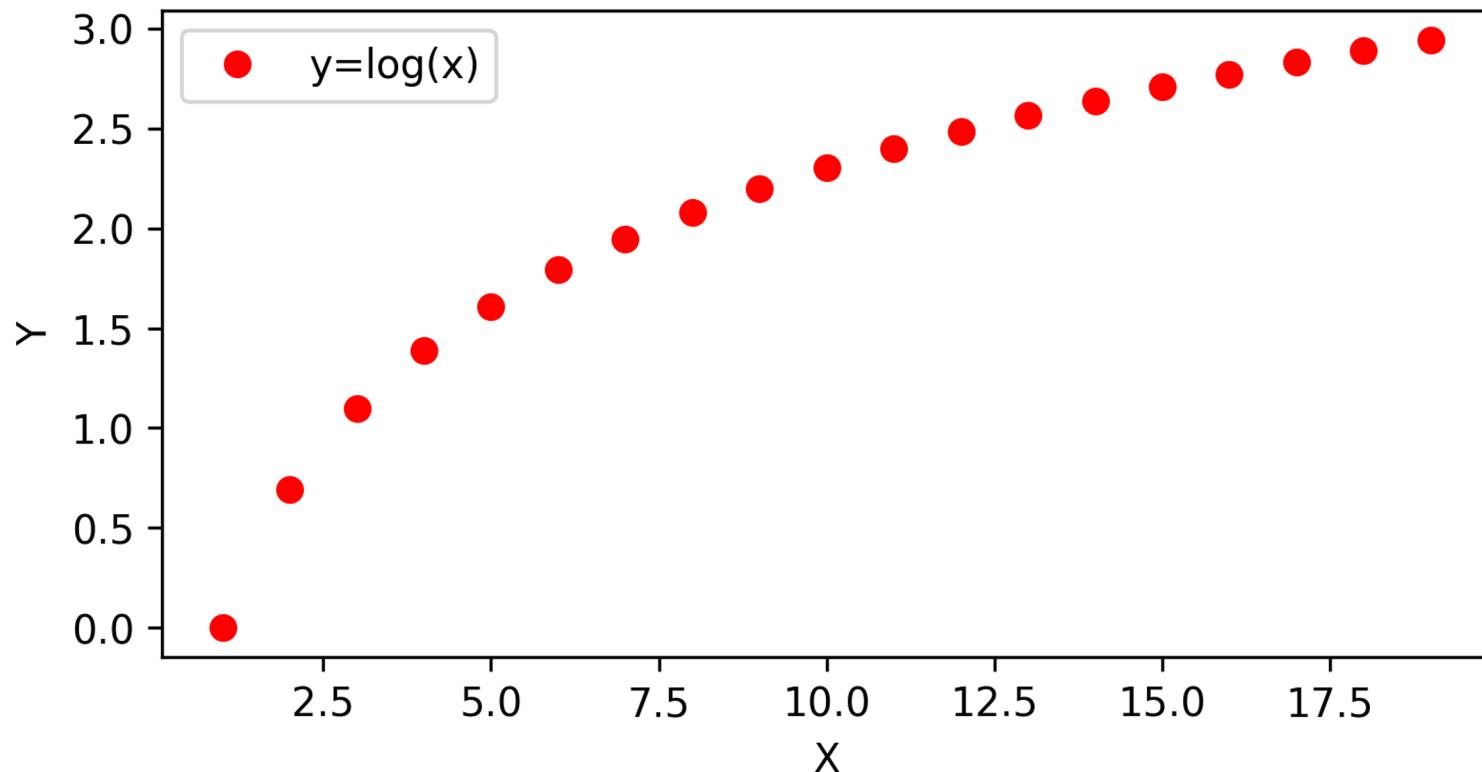
# Polynomial Regression

- Imagine a feature that is not linear:



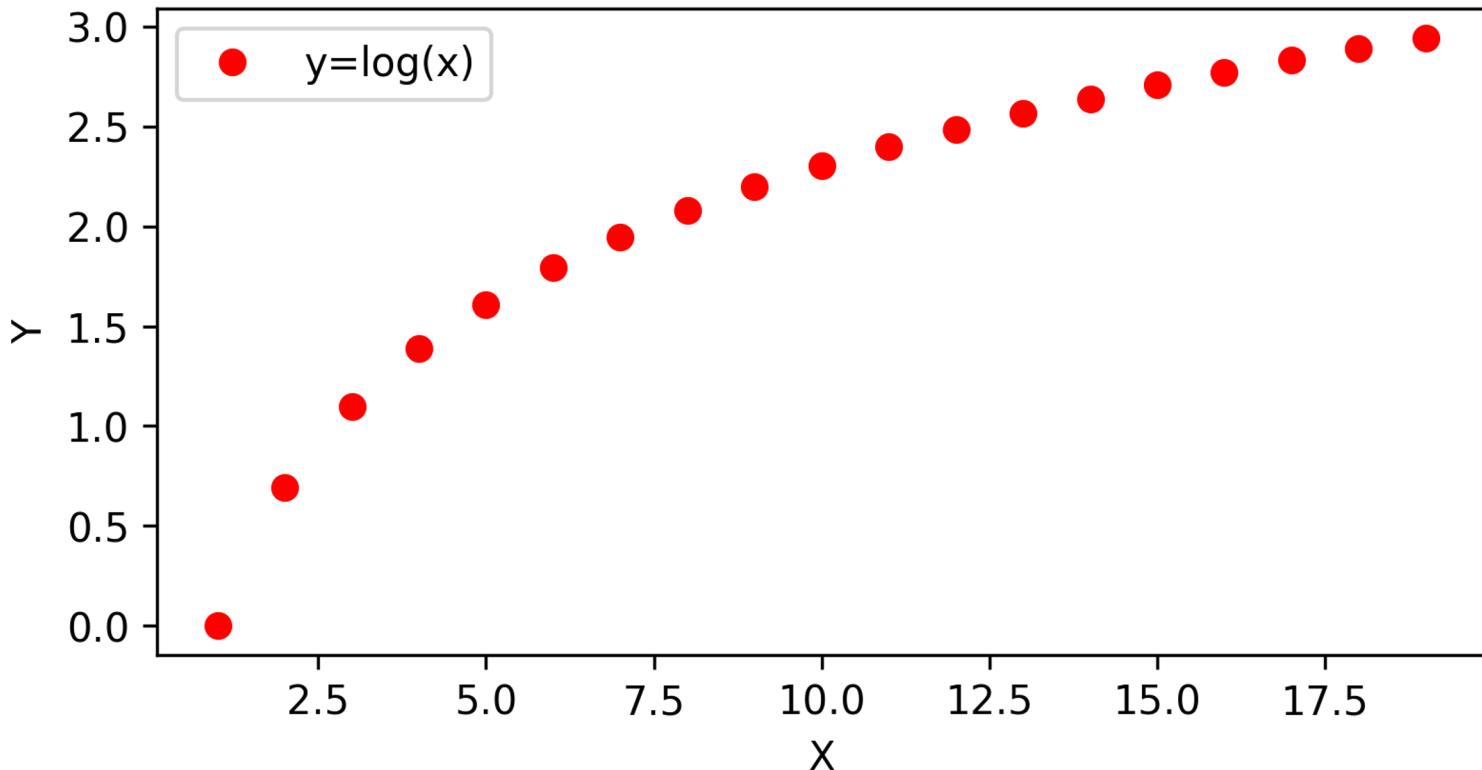
# Polynomial Regression

- We know  $\log(x)$  is not a linear relationship.



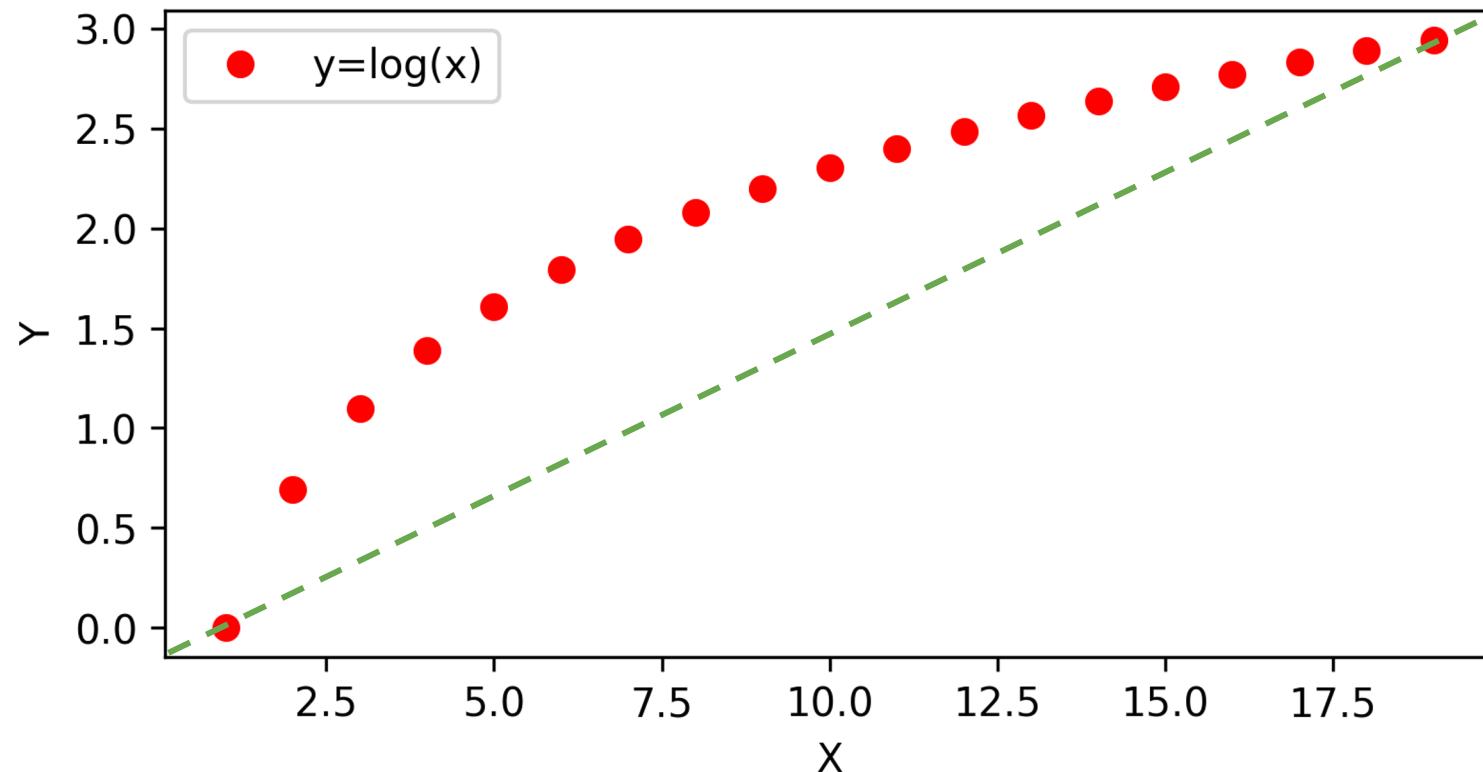
# Polynomial Regression

- What is a feature X behaved like  $\log(x)$ ?



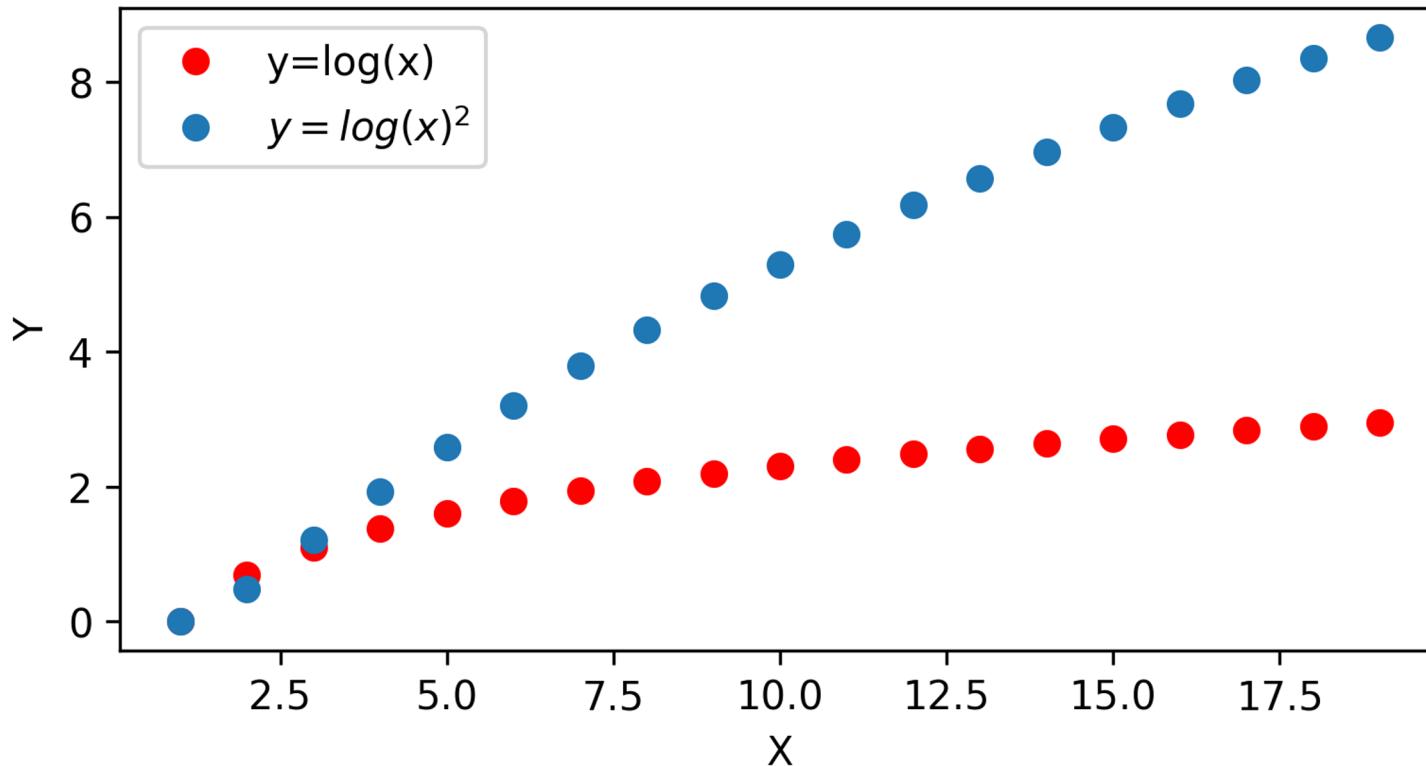
# Polynomial Regression

- Will be difficult to find a linear relationship



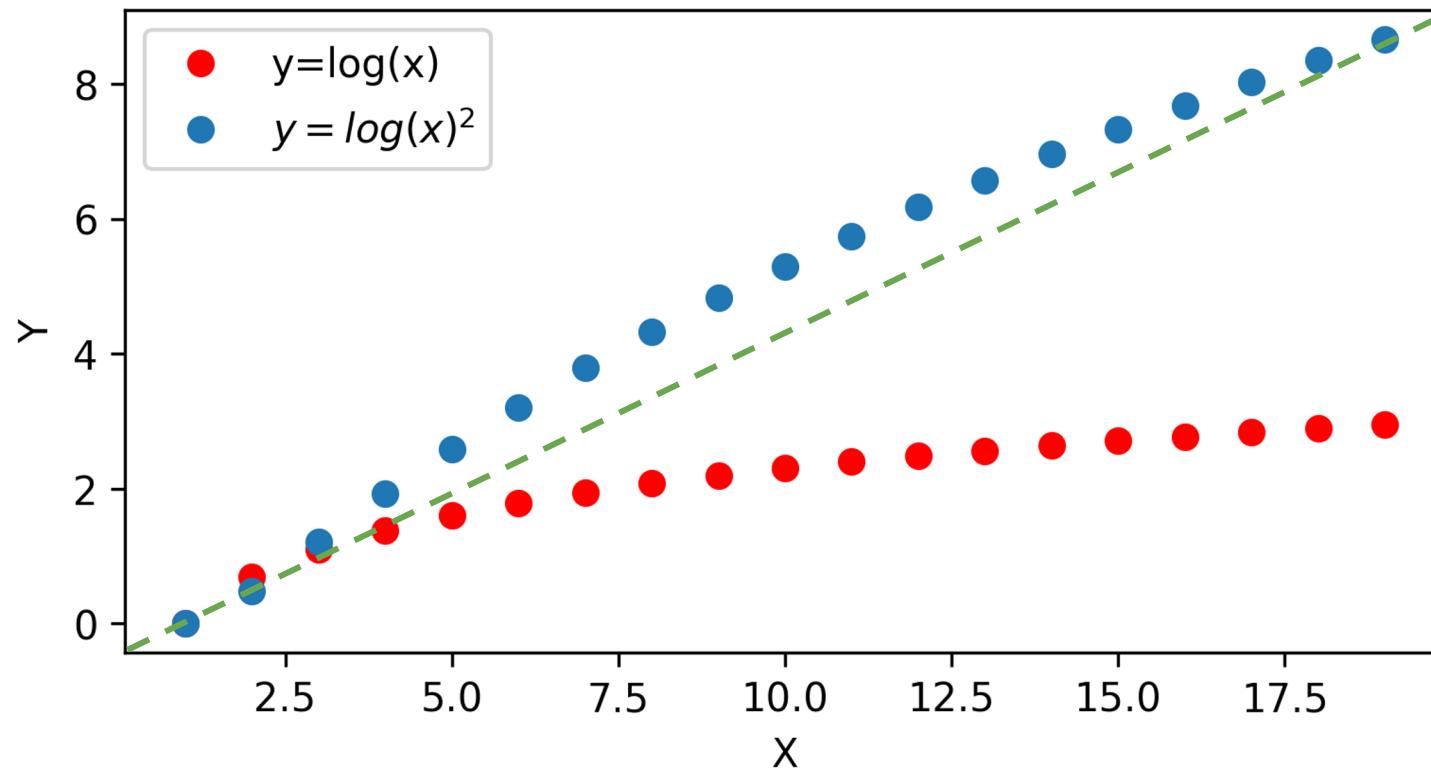
# Polynomial Regression

- What about the square of this feature?



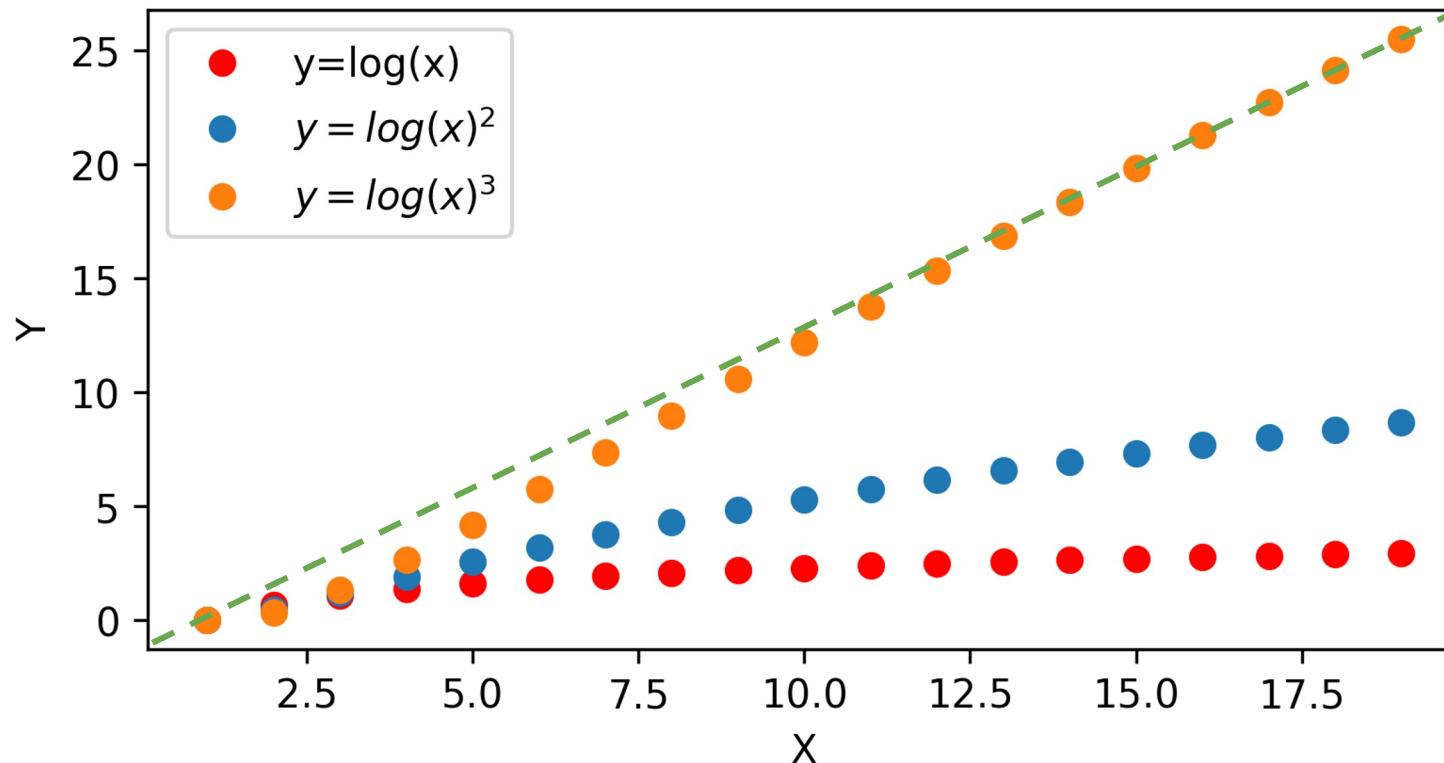
# Polynomial Regression

- Relationship could be more linear.



# Polynomial Regression

- Even more so for higher orders!



# Polynomial Regression

- Keep in mind this is an exaggerated example, and not every feature will have relationships at a higher order.
- The main point here is to show it could be reasonable to solve for a single linear Beta coefficient for polynomial of an original feature.

# Polynomial Regression

- Let's now also consider **interaction terms**.
- What if features are only significant when in sync with one another?
- For example:
  - Perhaps newspaper advertising spend by itself is not effective, but greatly increases effectiveness if added to a TV advertising campaign.

# Polynomial Regression

- Consumers only watching a TV ad will create some sales, but consumers who watch TV **and** are later “reminded” through a newspaper ad could contribute even more sales than TV or newspaper alone!
- How can we check for this?

# Polynomial Regression

- Simplest way is to create a new feature that multiplies two existing features together to create an **interaction term**.
- We can keep the original features, and add on this **interaction term**.
- Fortunately Scikit-Learn does this for us easily through a **preprocessing** call.

# Polynomial Regression

- Scikit-Learn's preprocessing library contains many useful tools to apply to the original data set **before** model training.
- One tool is the **PolynomialFeatures** which automatically creates both higher order feature polynomials and the interaction terms between all feature combinations.

# Polynomial Regression

- The features created include:
  - The bias (the value of 1.0)
  - Values raised to a power for each degree (e.g.  $x^1, x^2, x^3, \dots$ )
  - Interactions between all pairs of features (e.g.  $x_1 * x_2, x_1 * x_3, \dots$ )

# Polynomial Regression

- Converting Two Features **A** and **B**
  - **1, A, B, A<sup>2</sup>, AB, B<sup>2</sup>**

# Polynomial Regression

- Converting Two Features **A** and **B**
  - **1, A, B, A<sup>2</sup>, AB, B<sup>2</sup>**
- Generalized terms of features **X<sub>1</sub>** and **X<sub>2</sub>**
  - **1, X<sub>1</sub>, X<sub>2</sub>, X<sub>1</sub><sup>2</sup>, X<sub>1</sub>X<sub>2</sub>, X<sub>2</sub><sup>2</sup>**

# Polynomial Regression

- Converting Two Features **A** and **B**
  - **1, A, B, A<sup>2</sup>, AB, B<sup>2</sup>**
- Generalized terms of features **X<sub>1</sub>** and **X<sub>2</sub>**
  - **1, X<sub>1</sub>, X<sub>2</sub>, X<sub>1</sub><sup>2</sup>, X<sub>1</sub>X<sub>2</sub>, X<sub>2</sub><sup>2</sup>**
- Example if row was **X<sub>1</sub>=2** and **X<sub>2</sub>=3**
  - **1, 2, 3, 4, 6, 9**

# Polynomial Regression

- Let's explore how to perform polynomial regression with Scikit-Learn in the next lecture!