Linear Regression

Linear Regression

- The first machine learning algorithm we will explore is also one of the oldest!
- Let’s have a quick overview of what is covered in this section of the course.

Linear Regression

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

Linear Regression

* Unlike future ML Algorithm sections, the exercise project for linear regression will be spread over many sections, since we will first discuss feature engineering and cross-validation before tackling the full project exercise.

Let’s get started!

Introduction to Linear Regression

Algorithm Theory - Part One
History and Motivation

Linear Regression

- Before we do any coding, we will have a deep dive into building out an intuition of the theory and motivation behind Linear Regression.

Linear Regression

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

Linear Regression

- Relevant Reading in ISLR
    - Section 3 : Linear Regression
        - 3.1 Simple Linear Regression

Linear Regression

- The history of the “invention” of linear regression is a bit muddled.
- The linear regression methods based on least squares grew out of a need for mathematically improving navigation methods based on astronomy during the Age of Exploration in the 1700s.

Linear Regression

- 1722 - Roger Cotes discovers combining different observations yields better estimates of the true value.
- 1750 - Tobias Mayer explores averaging different results under similar conditions in studying librations of the moon.

Linear Regression

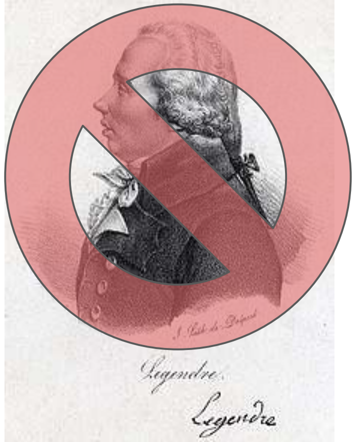
- 1757 - Roger Joseph Boscovich further develops combining observations studying the shape of the Earth.
- 1788 - Pierre-Simon LaPlace develops similar averaging theories in explaining the differences of motion between Jupiter and Saturn.

Linear Regression

- 1805 - First public exposition on Linear Regression with least squares method published by Adrien-Marie Legendre - *Nouvelles Méthodes pour la Détermination des Orbites des Comètes*

Linear Regression

- 2005 - Side portrait of Adrien-Marie Legendre is actually discovered to be Louis Legendre!
- Only one known sketch of Adrien-Marie Legendre...



Linear Regression

- Only one known watercolor caricature sketch of Adrien-Marie Legendre...

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

- Carl Friedrich Gauss was born in 1777, which would make him 18 years old at his proclaimed time of discovery!

Linear Regression

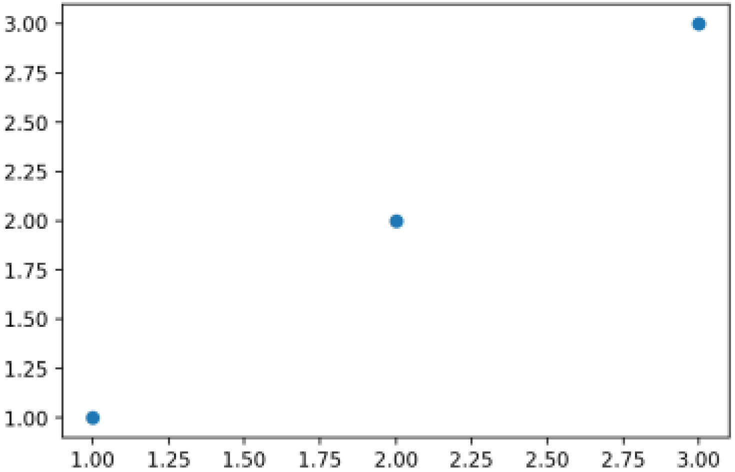
- 1808 - Robert Adrain published his formulation of least squares (a year before publication by Gauss).

Linear Regression

* Whatever the case of invention may be, let’s build an intuitive understanding of linear regression!

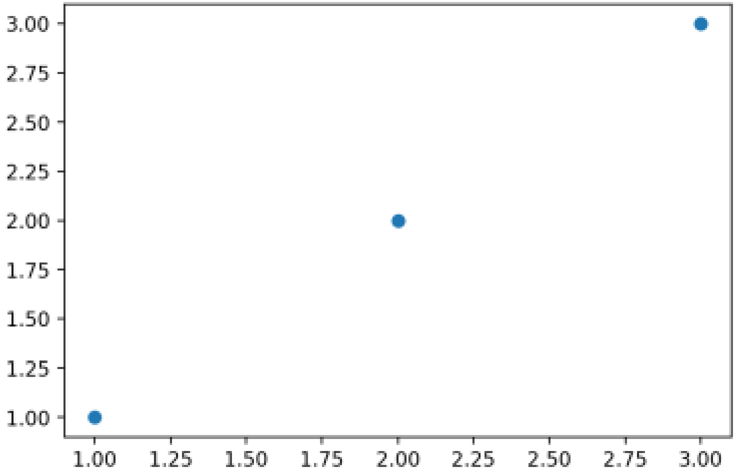
Linear Regression

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



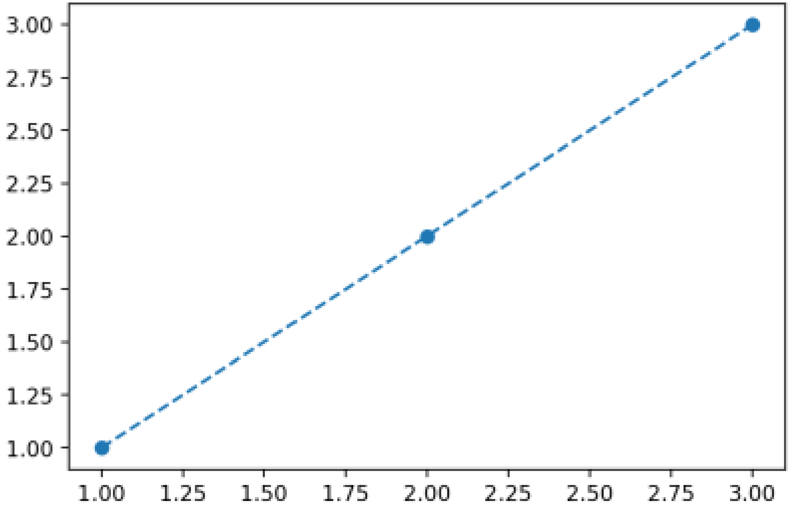
Linear Regression

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



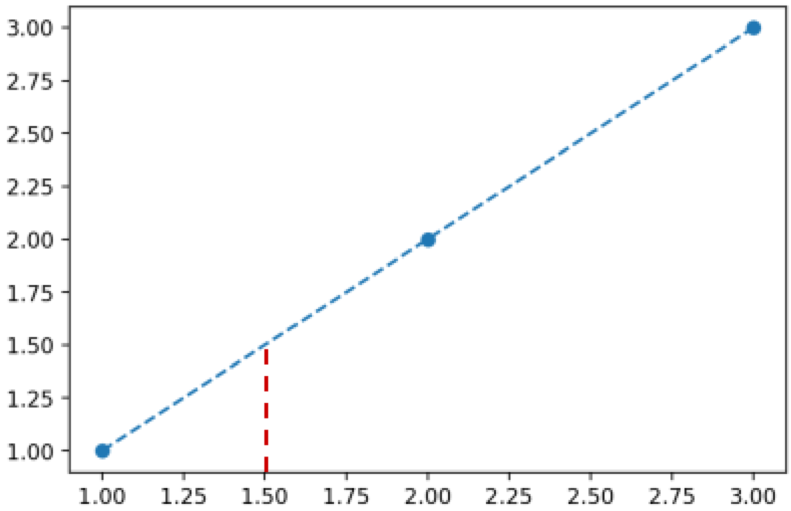
Linear Regression

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



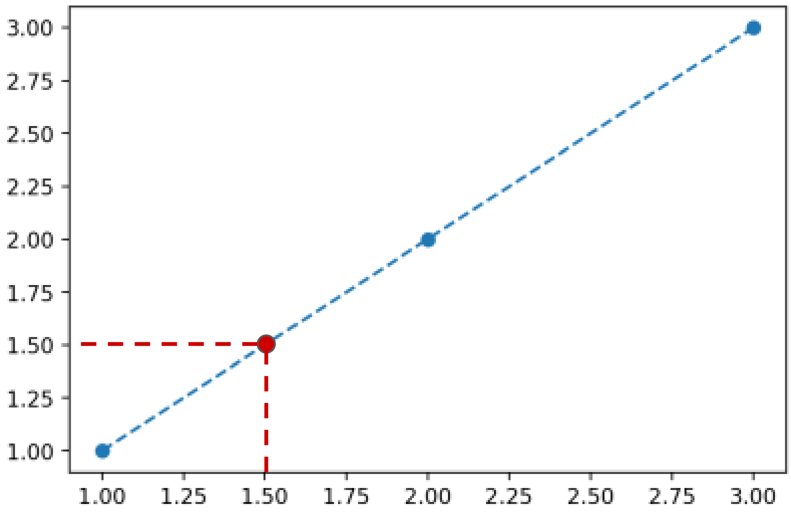
Linear Regression

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



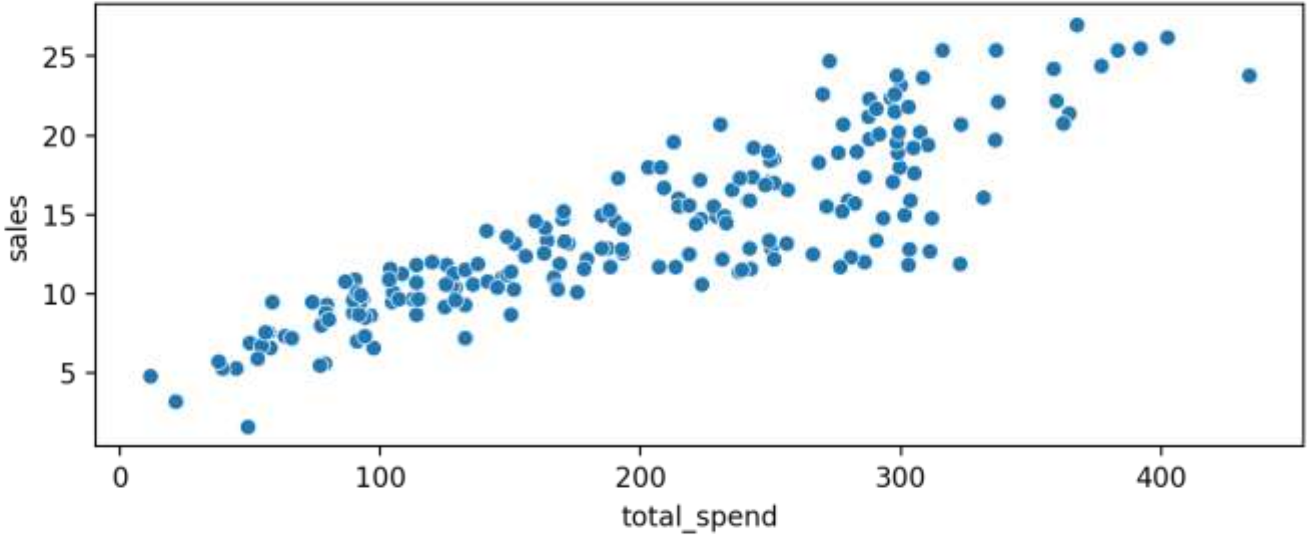
Linear Regression

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



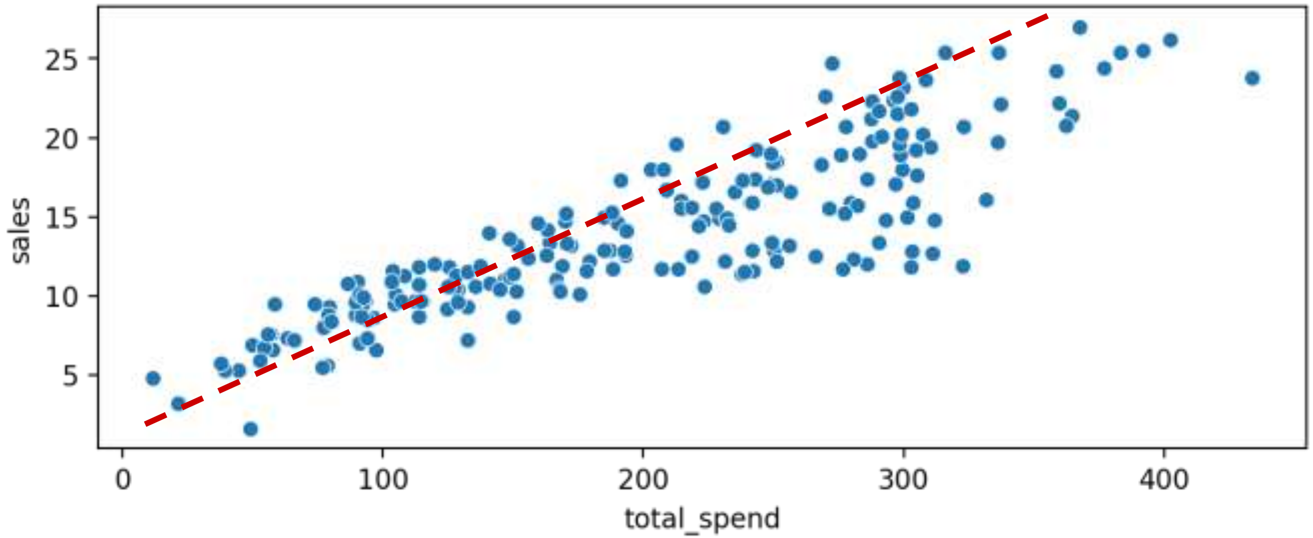
Linear Regression

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



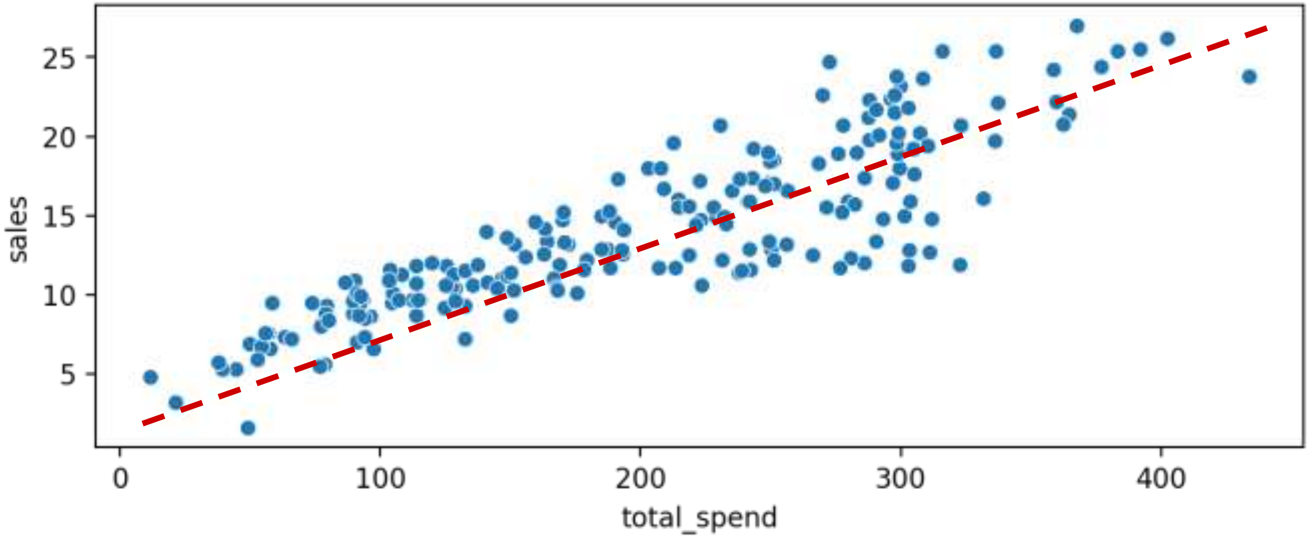
Linear Regression

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



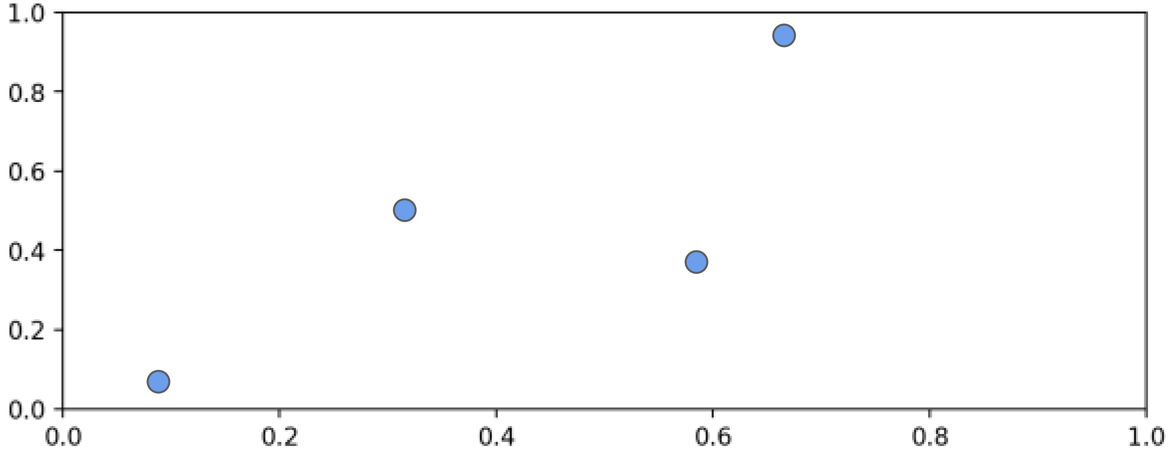
Linear Regression

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



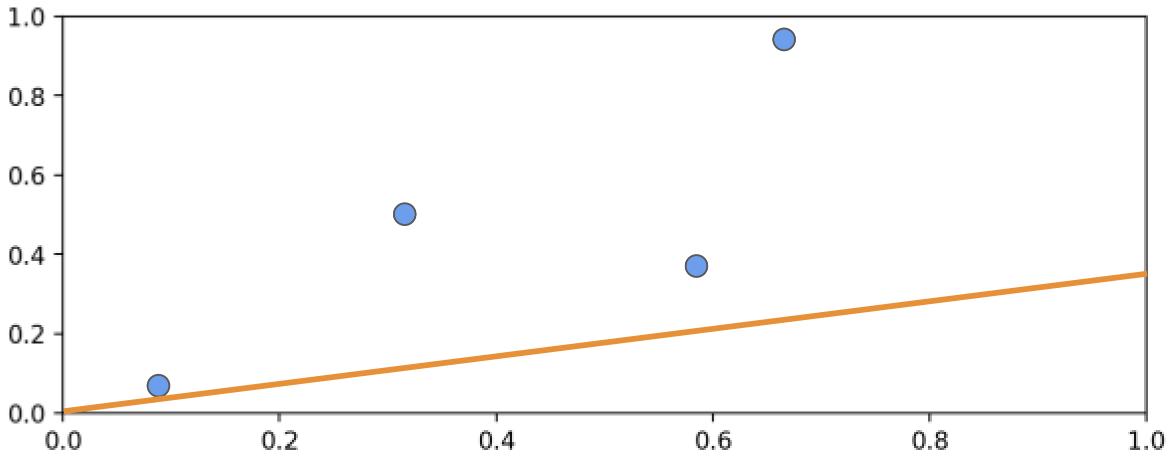
Linear Regression

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



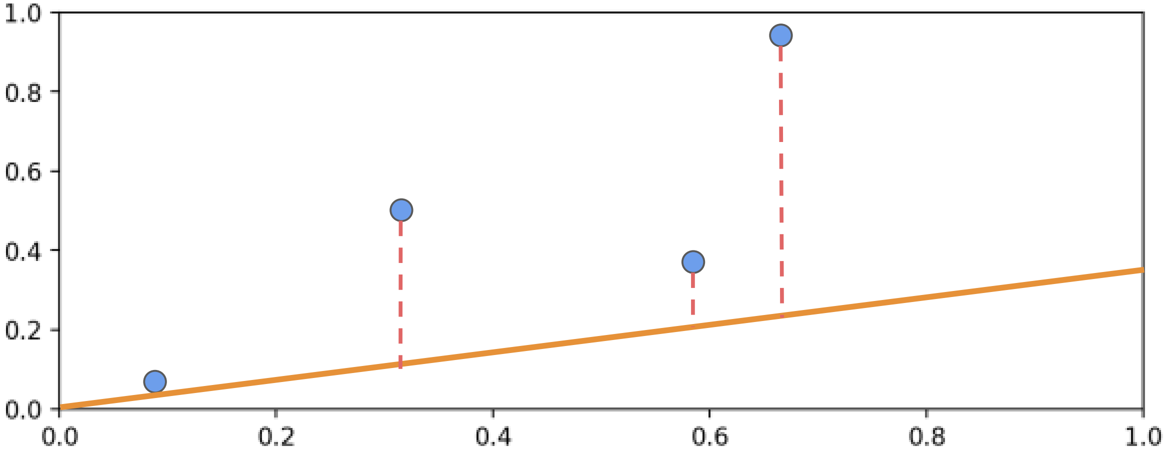
Linear Regression

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



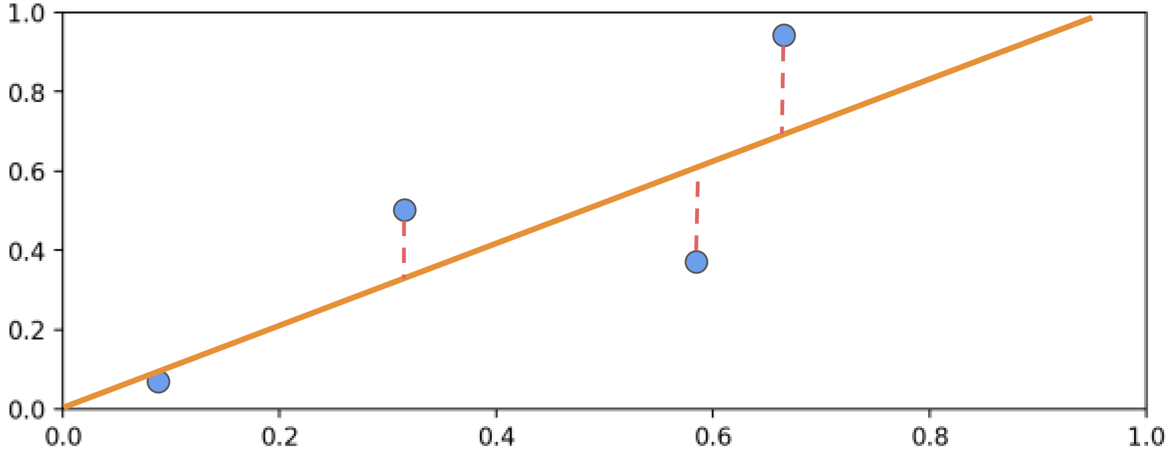
Linear Regression

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



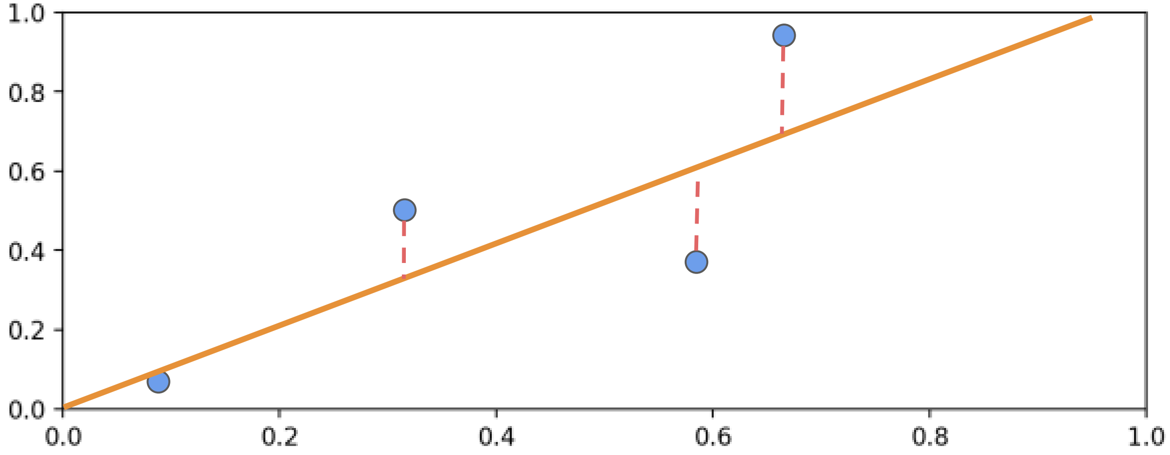
Linear Regression

- Some lines will clearly be better fits than others.



Linear Regression

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

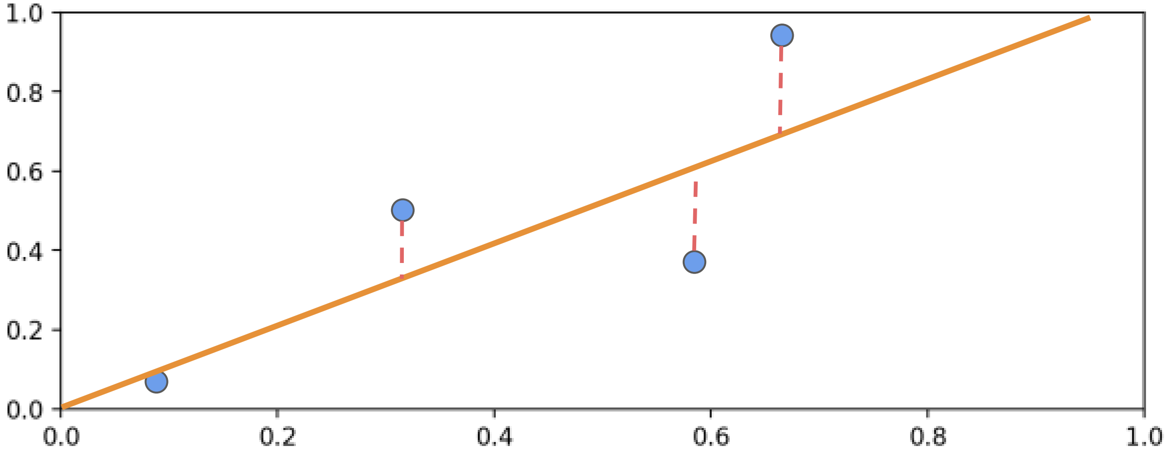


Linear Regression

- **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.

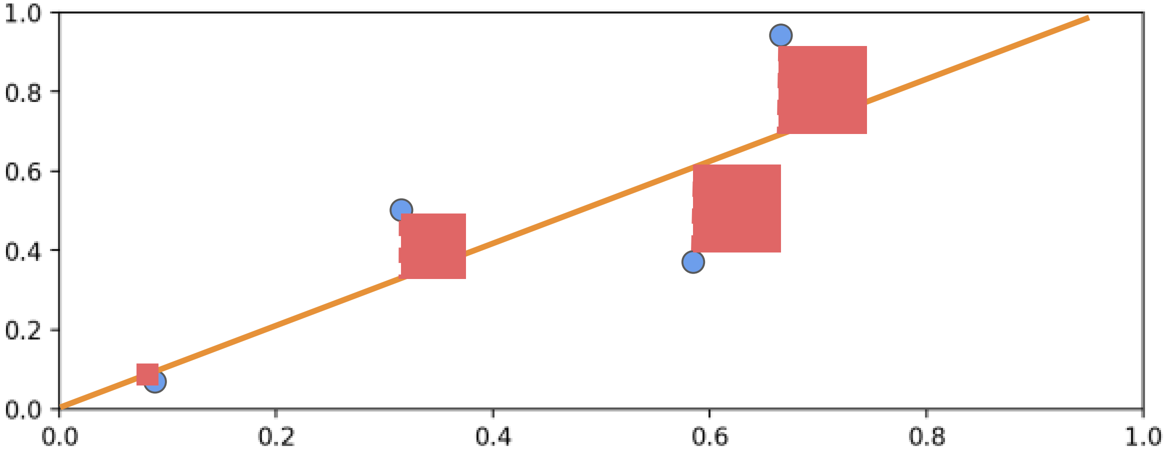
Linear Regression

- 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

PIERIAN DATA

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

PIERIAN DATA

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

- Let’s explore how we could translate a real data set into mathematical notation for linear regression.
- Then we’ll solve a simple case of one feature to explore OLS in action.
- Afterwards we’ll focus on gradient descent for real world data set situations.

Linear Regression

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

Area m^2	Bedrooms	Bathrooms	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…

X y

Area m^2	Bedrooms	Bathrooms	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…

X | **y**

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	y
x^1_1	3	2	$500,000
x^2_1	2	1	$450,000
x^3_1	3	3	$650,000
x^4_1	1	1	$400,000
x^5_1	2	2	$550,000

Linear Regression

- We can translate this data into generalized mathematical notation…

X			**y**
$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{y}$
$x^1{}_1$	$x^1{}_1$	$x^1{}_1$	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	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	x_3	y

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.

y	X		
y	x_1	x_2	x_3

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

Linear Regression

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

y	X		
y	x_1	x_2	x_3

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

Linear Regression

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

y	X		
y	x_1	x_2	x_3

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

Linear Regression

- We can also express this equation as a sum:

y	X		
y	x_1	x_2	x_3

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

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

Linear Regression

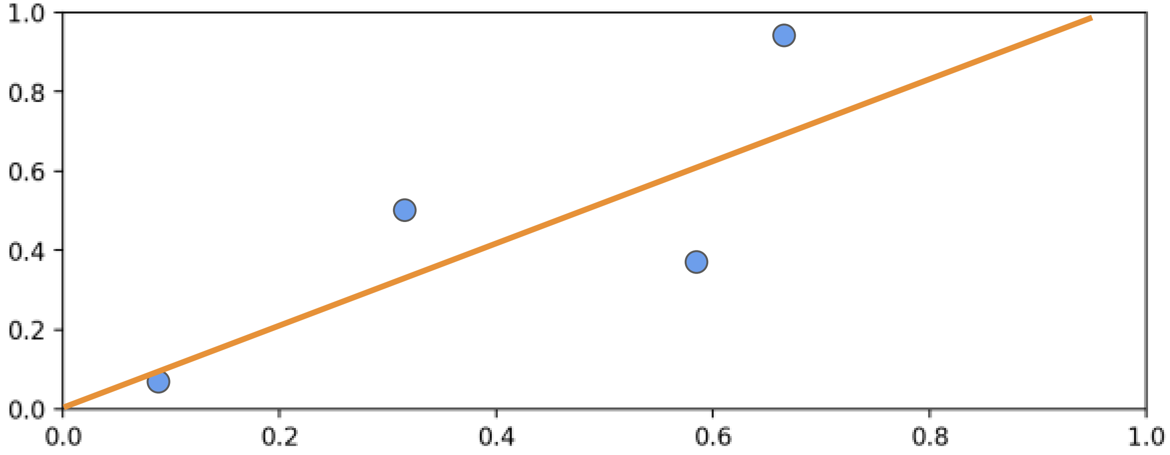
- Note the y hat symbol displays a prediction. There is usually no set of Betas to create a perfect fit to y!

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

Linear Regression

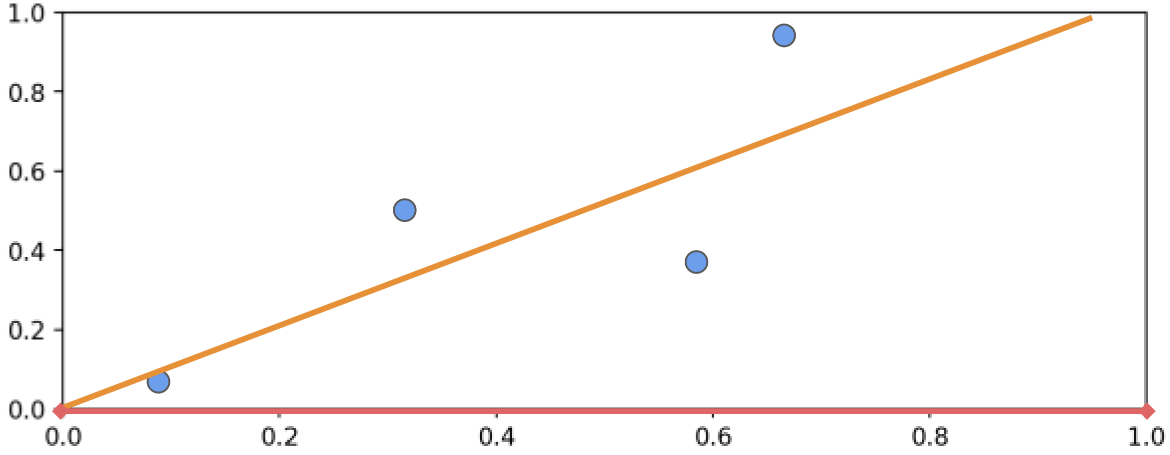
- Line equation:

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



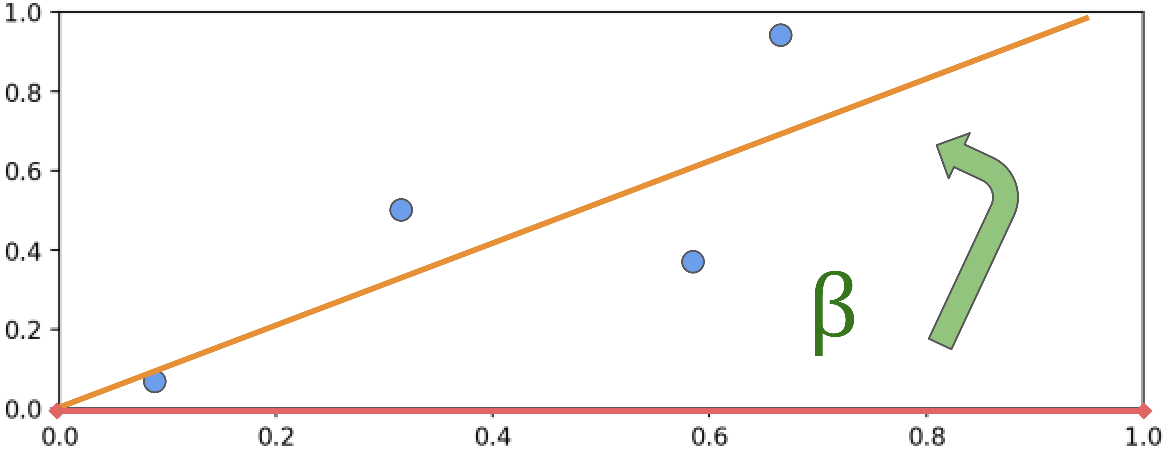
Linear Regression

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



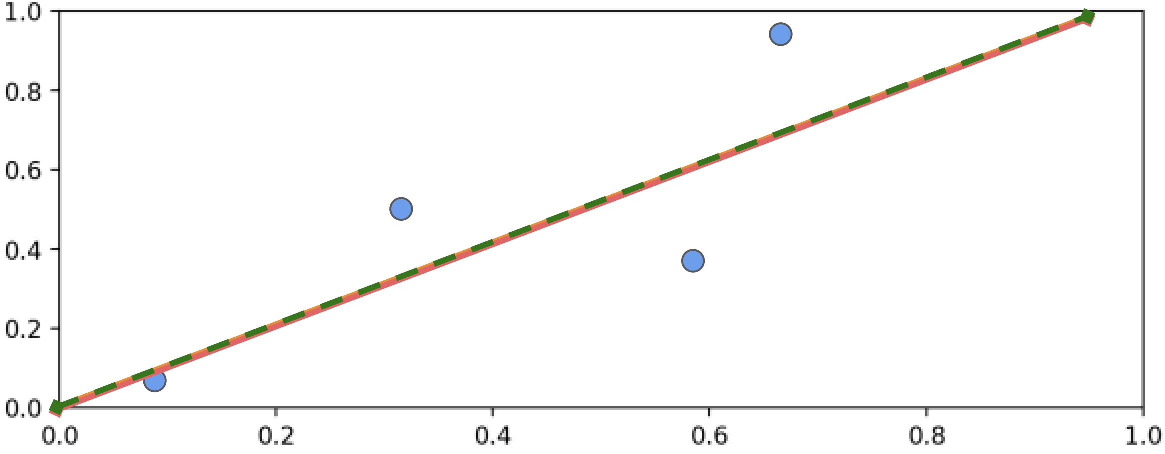
Linear Regression

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



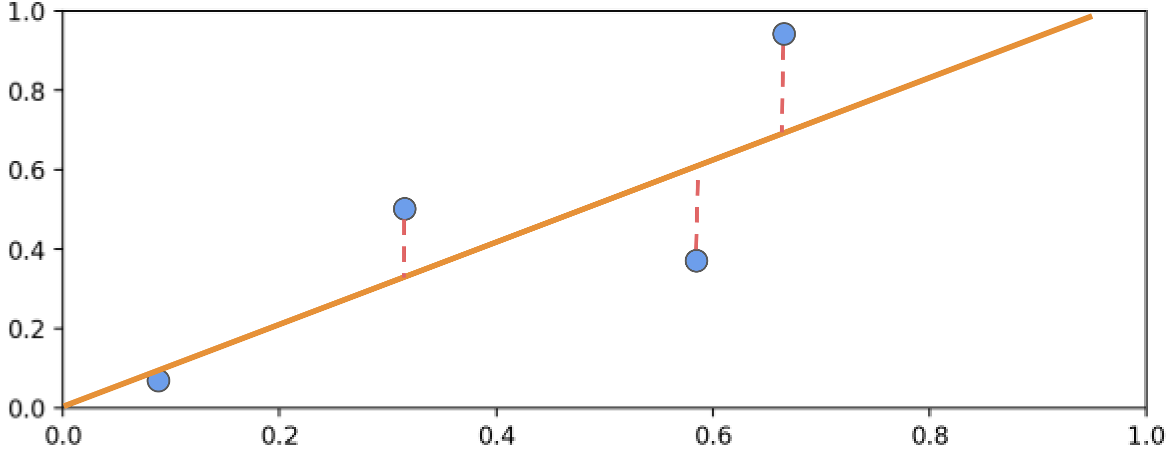
Linear Regression

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



Linear Regression

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



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!

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

Linear Regression

- We'll work on developing a cost function to minimize in the next lectures!

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

Introduction to Linear Regression

Algorithm Theory - Part Three
Cost Function

PIERIAN DATA

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:

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

Linear Regression

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

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

Linear Regression

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

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

Linear Regression

- Exactly what we need for a **cost function**!

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

Linear Regression

- Begin by defining a cost function **J.**

$$J(\boldsymbol{\beta})$$

Linear Regression

- A **cost function** is defined by some measure of error.

$$J(\boldsymbol{\beta})$$

Linear Regression

- This means we wish to **minimize** the cost function.

$$J(\boldsymbol{\beta})$$

Linear Regression

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

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

Linear Regression

- Note lowercase $\boldsymbol{j}$ is the specific data row.

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

Linear Regression

- Want to minimize cost for set of Betas.

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

Linear Regression

- Error between real **y** and predicted **ŷ**

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

Linear Regression

- Squaring corrects for negative and positive errors.

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

Linear Regression

- Summing error for m rows.

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

Linear Regression

- Summing error for m rows.

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

Linear Regression

- Divide by **m** to get **mean**

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

Linear Regression

- Additional ½ is for convenience for derivative.

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

Linear Regression

- What is $\hat{\mathbf{y}}$?

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

Linear Regression

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

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

$$= \frac{1}{2m}\sum_{j=1}^{m}\left(y^j - \sum_{i=0}^{n}\beta_i x_i^j\right)^2$$

Linear Regression

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

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

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

Linear Regression

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

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

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

Linear Regression

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

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

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

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

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

- Our goals:
  - Find a set of Beta coefficient values that minimizes the error (cost function)
  - Leverage computational power instead of having to manually attempt to analytically solve the derivative.

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

- Also recall our data will be in the form of a matrix **X** with a vector of labels **y.**

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

Linear Regression

- Which means we need a β for each feature, so we can express a vector of β valu

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

Linear Regression

- Use a **gradient** to express the derivative of the cost function with respect to each β

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

Linear Regression

- We can plug in our equation of the derivative of the loss function.

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

Linear Regression

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

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

Linear Regression

- We also know we can vectorize our data:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \dots & x_n^1 \\ 1 & x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^m & x_2^m & \dots & x_n^m \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}$$

Linear Regression

- We can split the gradient of the cost function into two parts:

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

Linear Regression

- We can split the gradient of the cost function into two parts:

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

Linear Regression

- We can split the gradient of the cost function into two parts:

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

Linear Regression

- This then results in the following:

$$\nabla_{\beta} J = -\frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} y^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} y^j x_n^j \end{bmatrix} + \frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} \sum_{i=0}^{n} \beta_i x_i^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} \sum_{i=0}^{n} \beta_i x_i^j x_n^j \end{bmatrix}$$

Linear Regression

- We can now calculate the gradient for any set of Beta values!

$$\nabla_\beta J = -\frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} y^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} y^j x_n^j \end{bmatrix} + \frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m}\sum_{i=0}^{n} \beta_i x_i^j x_0^j \\ \vdots \\ \sum_{j=1}^{m}\sum_{i=0}^{n} \beta_i x_i^j x_n^j \end{bmatrix}$$

Linear Regression

- In theory we could now guess and check Beta values that minimize this gradient.

$$\nabla_{\beta} J = -\frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} y^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} y^j x_n^j \end{bmatrix} + \frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} \sum_{i=0}^{n} \beta_i x_i^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} \sum_{i=0}^{n} \beta_i x_i^j x_n^j \end{bmatrix}$$

Linear Regression

- Note how the Beta coefficients are the only unknown variable here:

$$\nabla_\beta J = -\frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} y^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} y^j x_n^j \end{bmatrix} + \frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} \sum_{i=0}^{n} \boxed{\beta_i} x_i^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} \sum_{i=0}^{n} \boxed{\beta_i} x_i^j x_n^j \end{bmatrix}$$

Linear Regression

- The other variables are from our known data matrix values X and y.

$$\nabla_\beta J = -\frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} y^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} y^j x_n^j \end{bmatrix} + \frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} \sum_{i=0}^{n} \beta_i x_i^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} \sum_{i=0}^{n} \beta_i x_i^j x_n^j \end{bmatrix}$$

Linear Regression

- What is the best way to "guess" at the correct Beta values that minimize the gradient?

$$\nabla_\beta J = -\frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} y^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} y^j x_n^j \end{bmatrix} + \frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} \sum_{i=0}^{n} \beta_i x_i^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} \sum_{i=0}^{n} \beta_i x_i^j x_n^j \end{bmatrix}$$

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

- Given a cost function of $J(\boldsymbol{\beta})$ how can we computationally search for the correct value of $\boldsymbol{\beta}$ that minimizes the gradient of the cost function?
- What would the search process look like for single β value?

Linear Regression

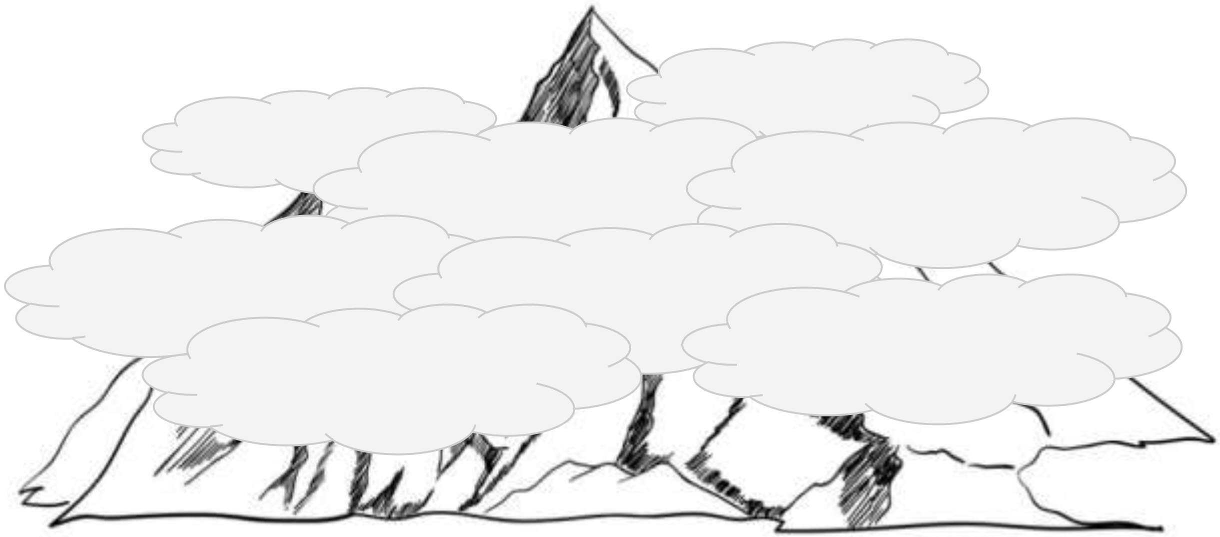
- Common mountain analogy

Linear Regression

- Common mountain analogy

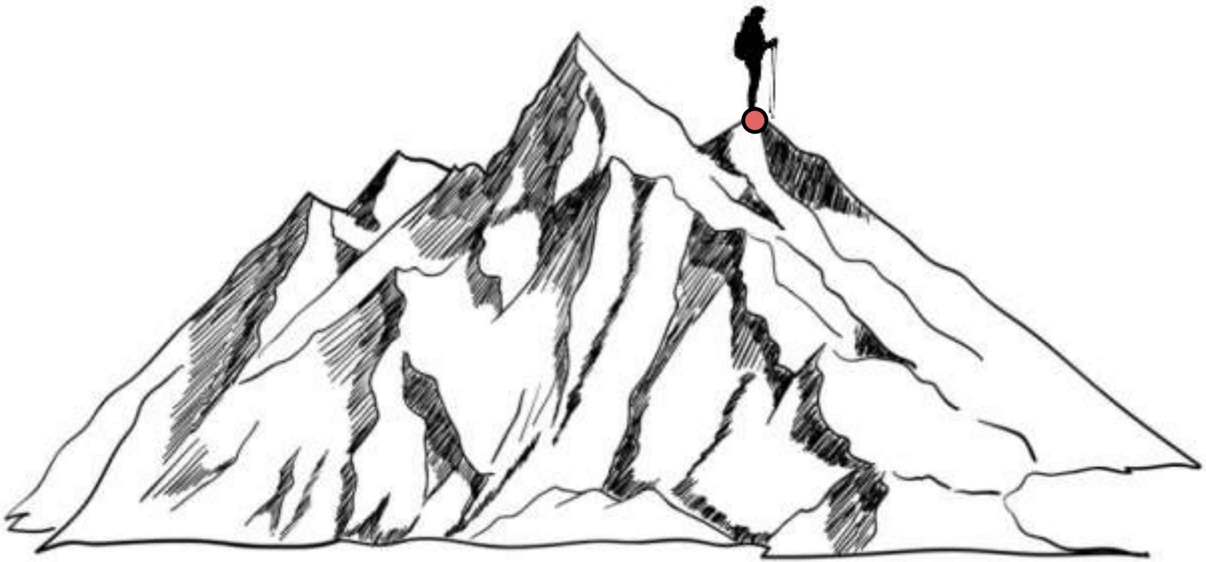
Linear Regression

- Common mountain analogy



Linear Regression

- Common mountain analogy

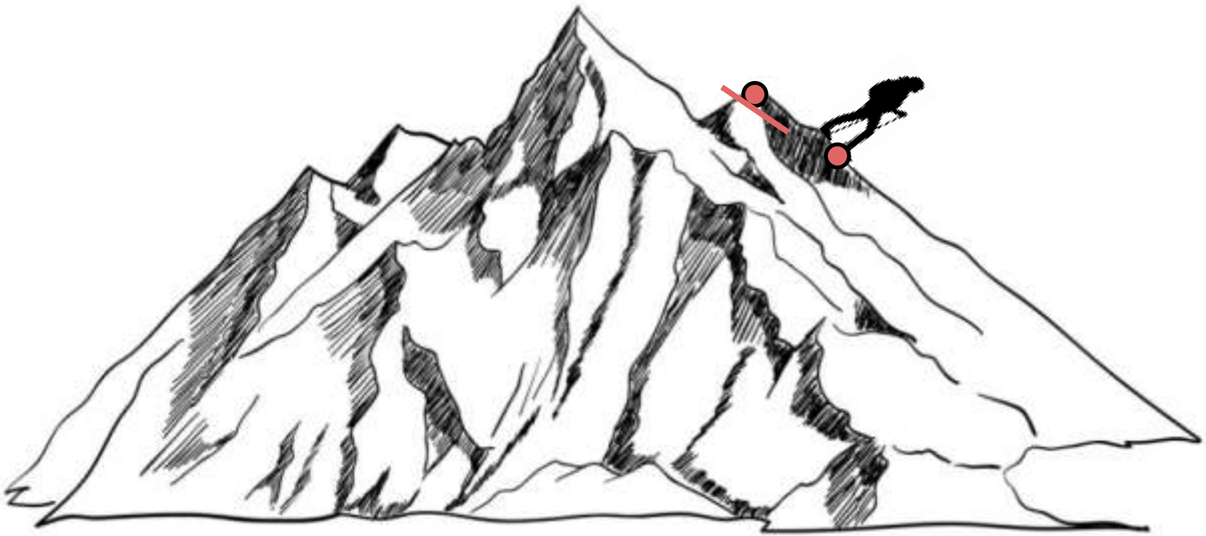


Linear Regression

- Common mountain analogy

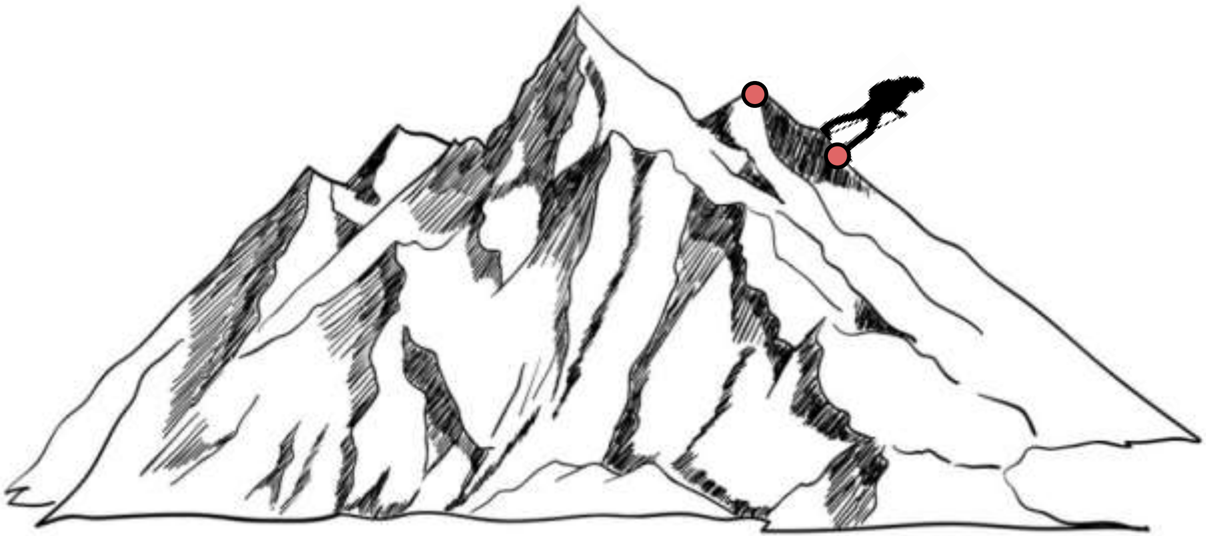
Linear Regression

- Common mountain analogy



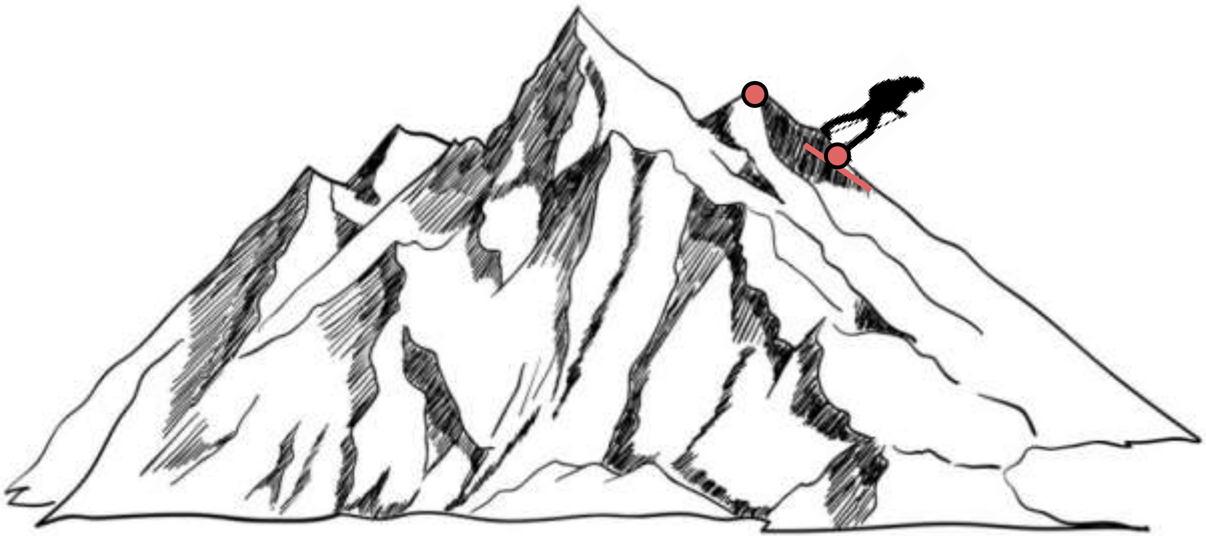
Linear Regression

- Common mountain analogy



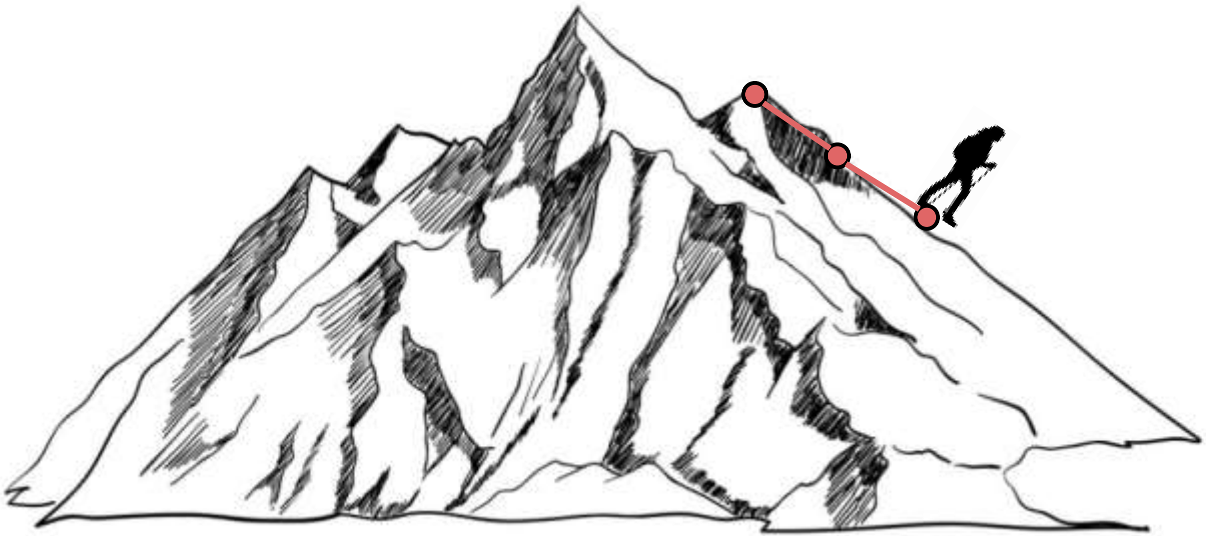
Linear Regression

- Common mountain analogy



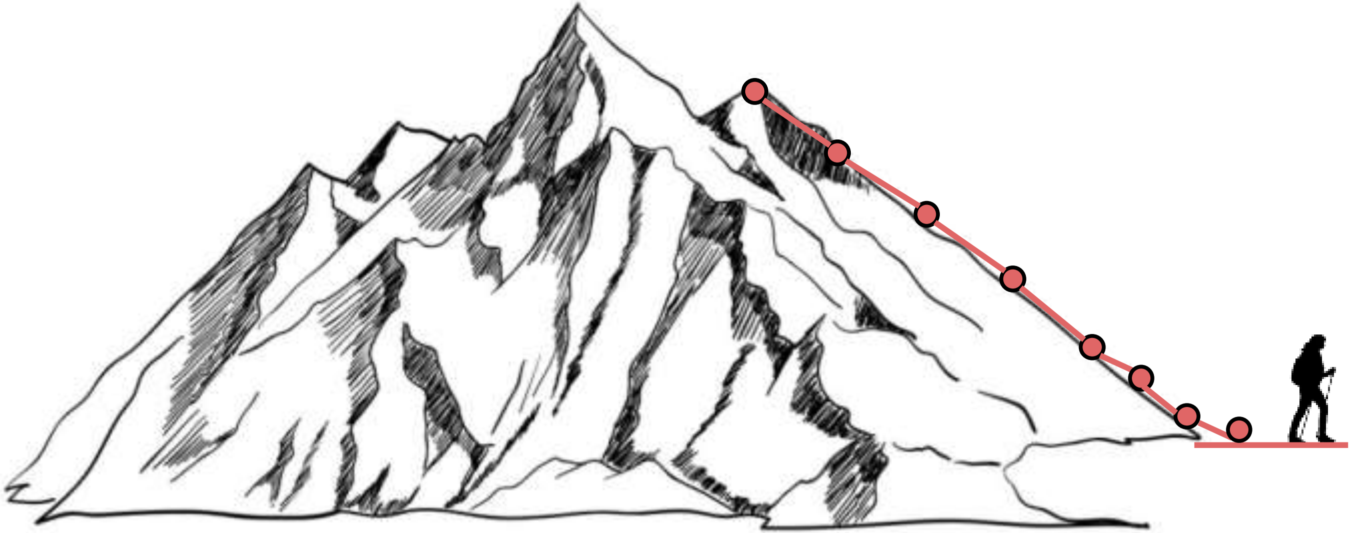
Linear Regression

- Common mountain analogy



Linear Regression

- Common mountain analogy

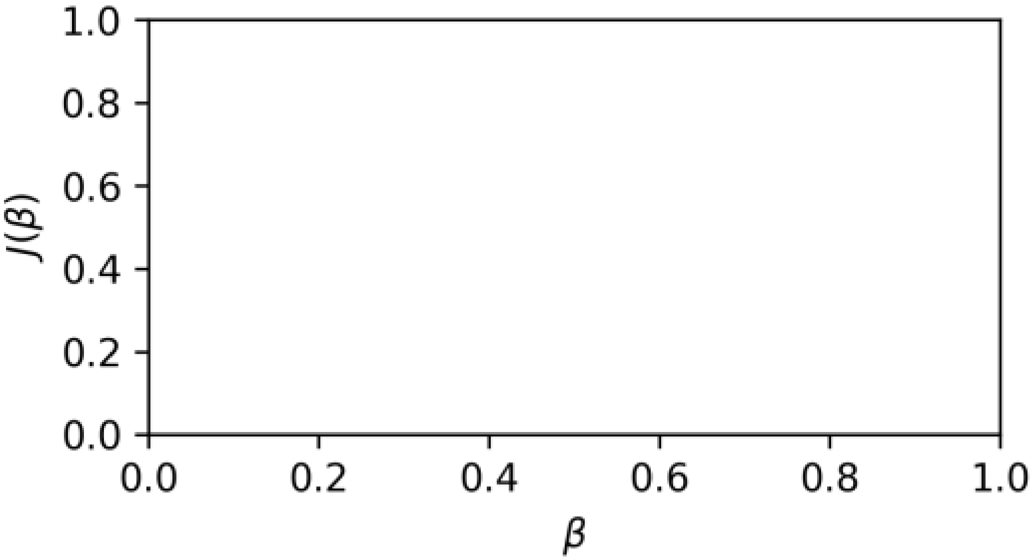


Linear Regression

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

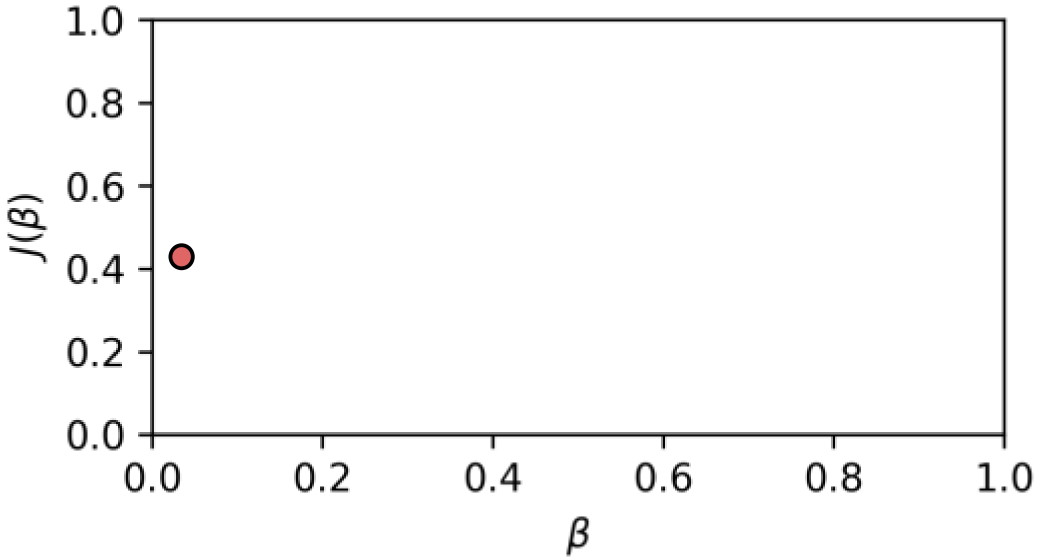
Linear Regression

- 1 dimensional cost function (single Beta)



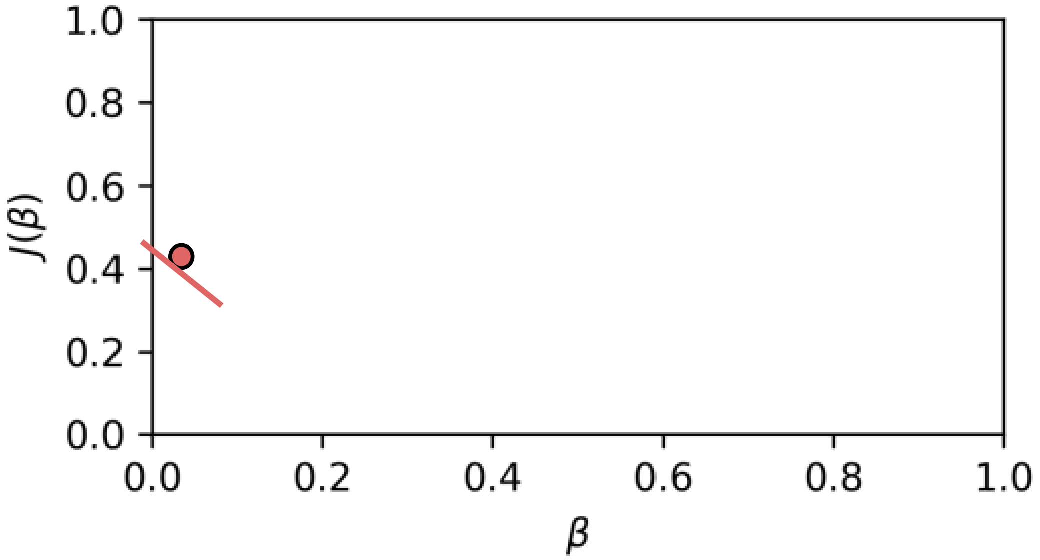
Linear Regression

- Choose a starting point



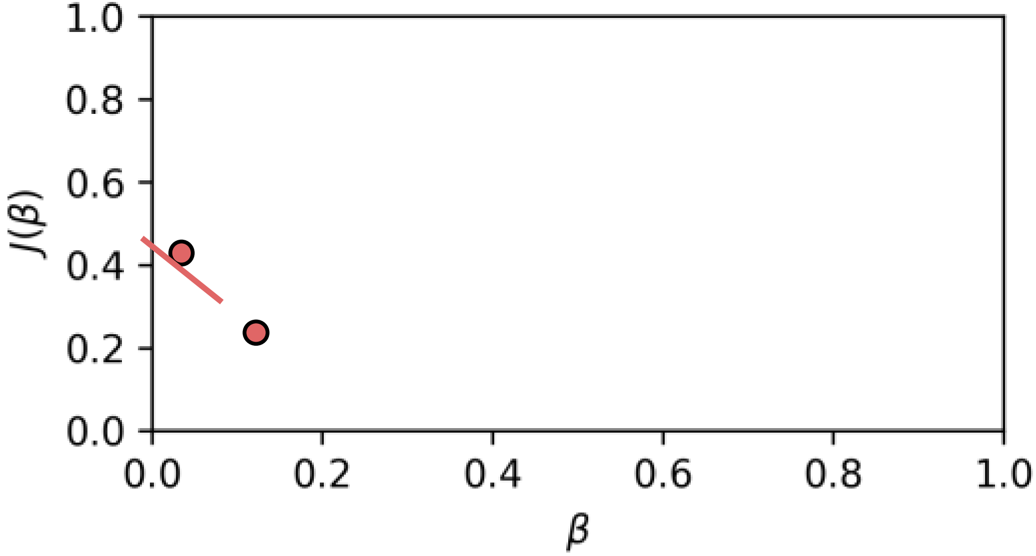
Linear Regression

- Calculate gradient at that point



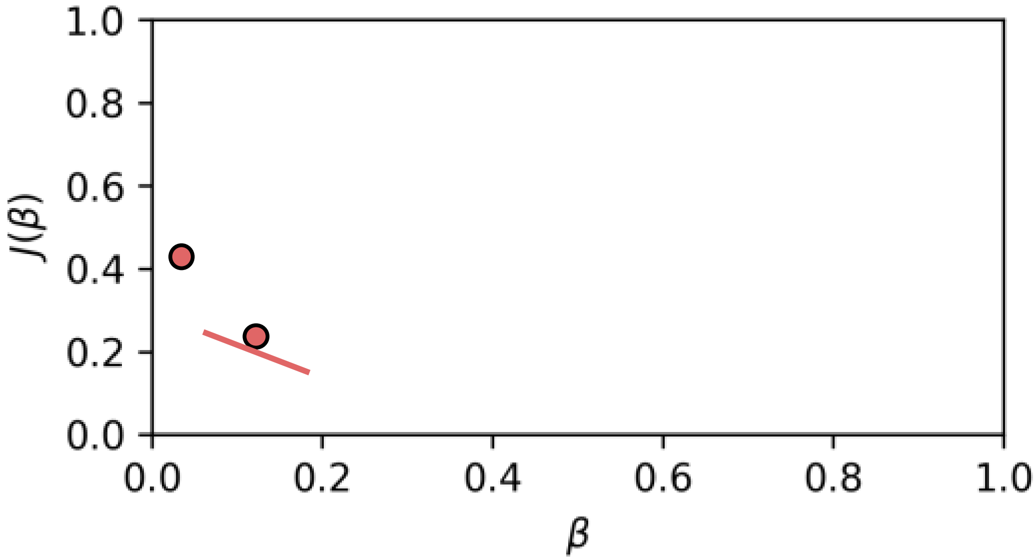
Linear Regression

- Step forward proportional to **negative** gradient



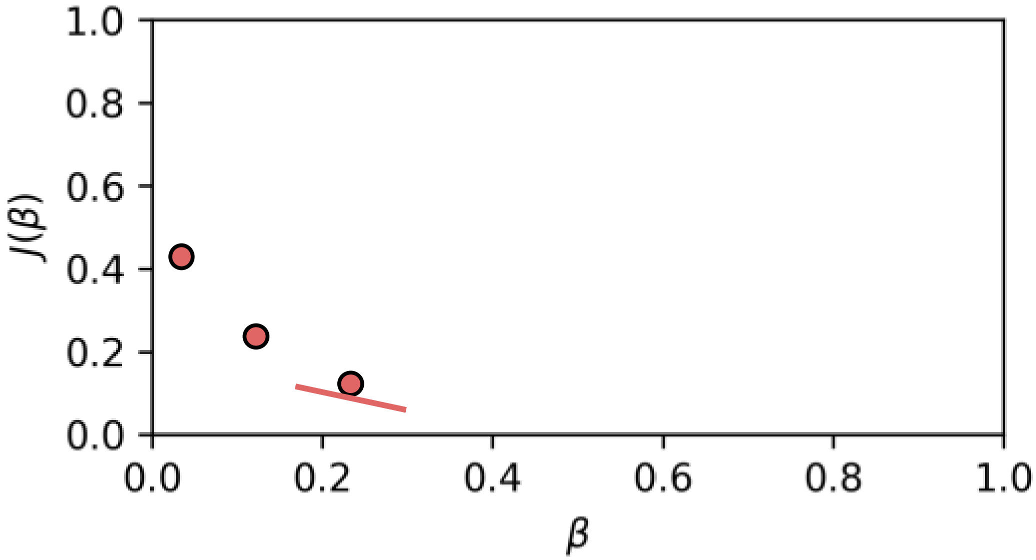
Linear Regression

- Repeat the steps



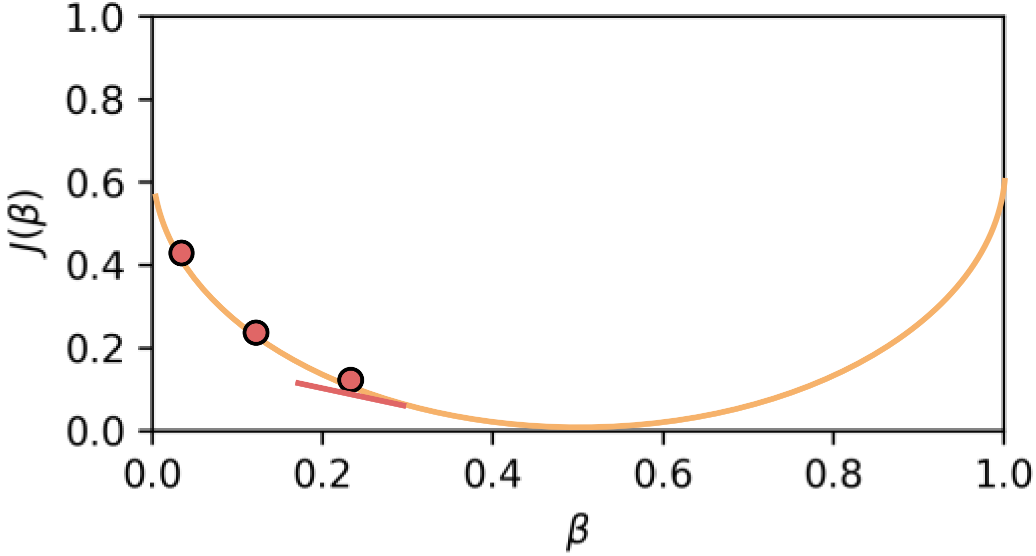
Linear Regression

- Repeat the steps



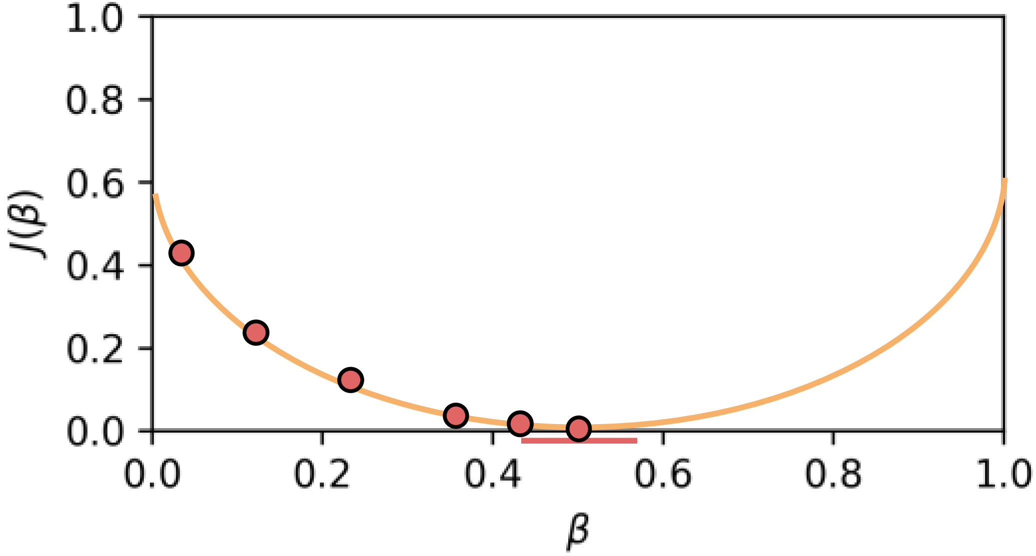
Linear Regression

- Note how we are essentially mapping the gradient!



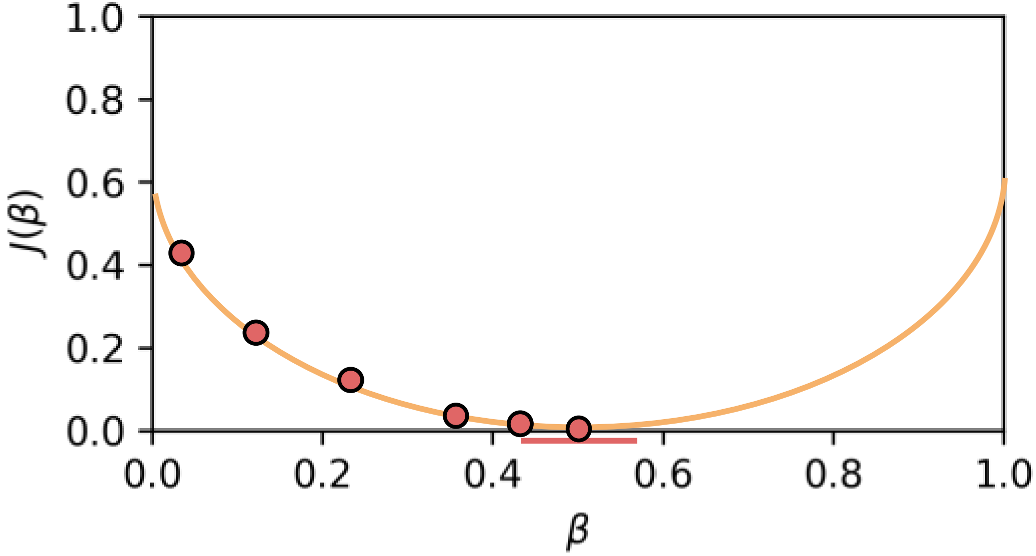
Linear Regression

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



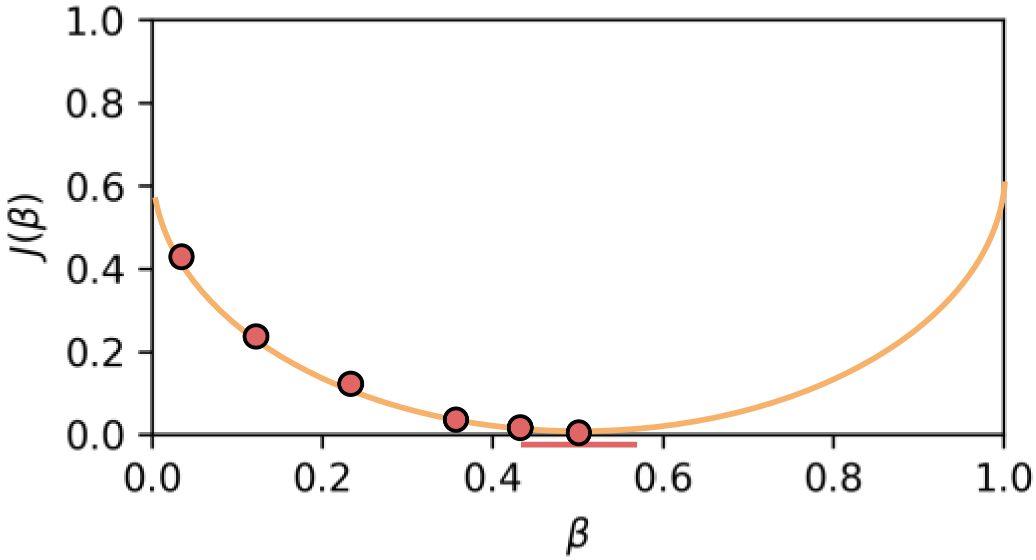
Linear Regression

- Steps are proportional to negative gradient!



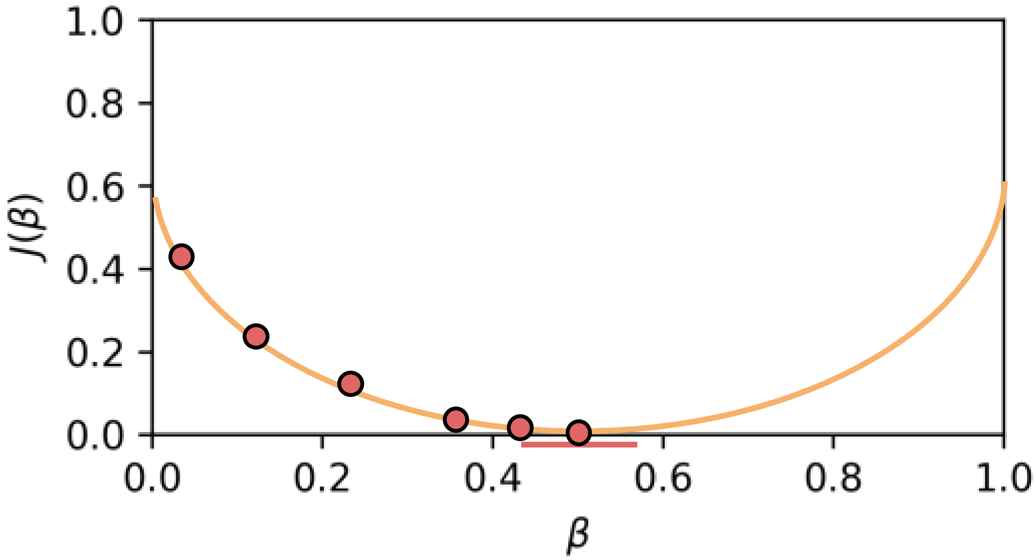
Linear Regression

- Steeper gradient at start gives larger steps.



Linear Regression

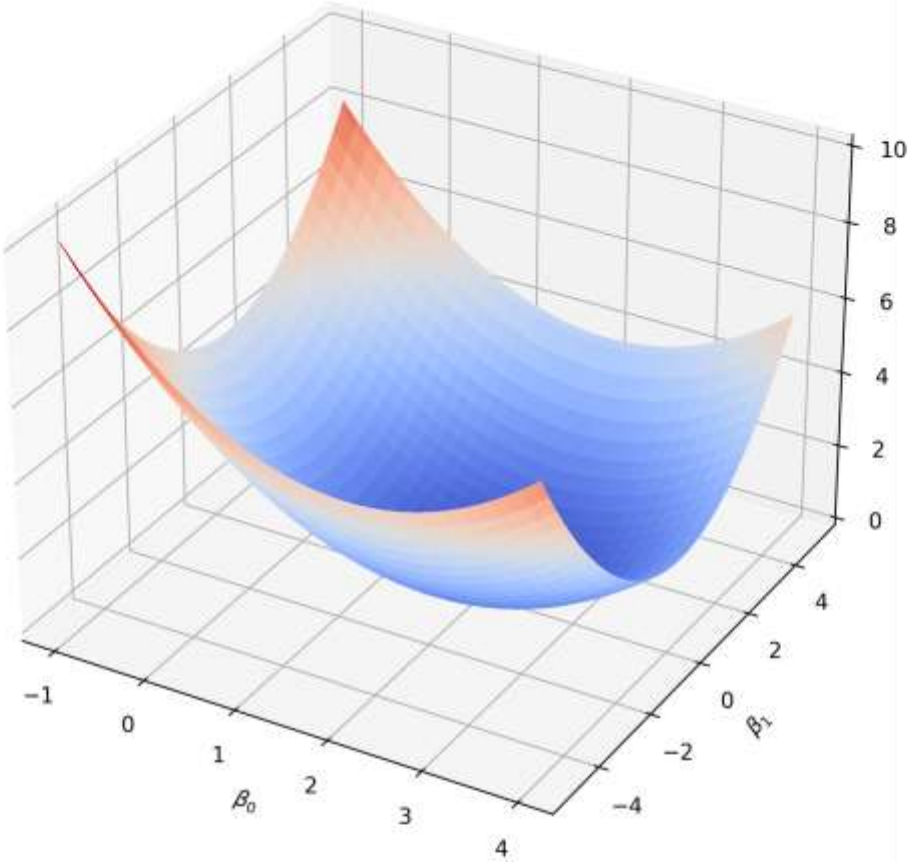
- Smaller gradient at end gives smaller steps.



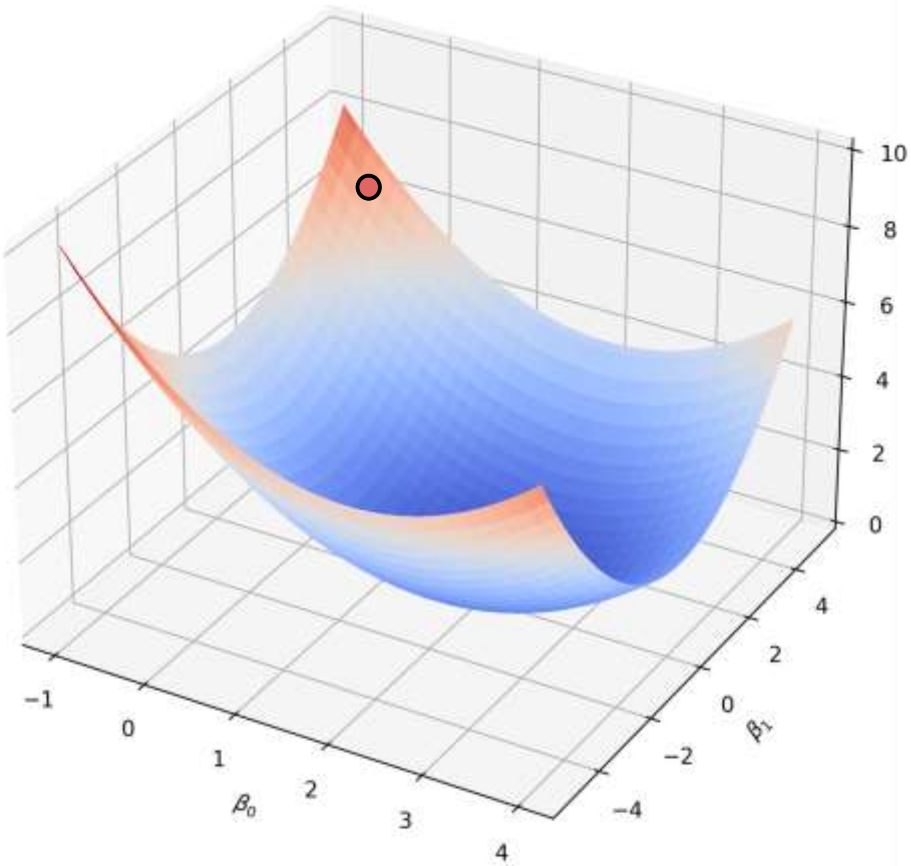
Linear Regression

- 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.

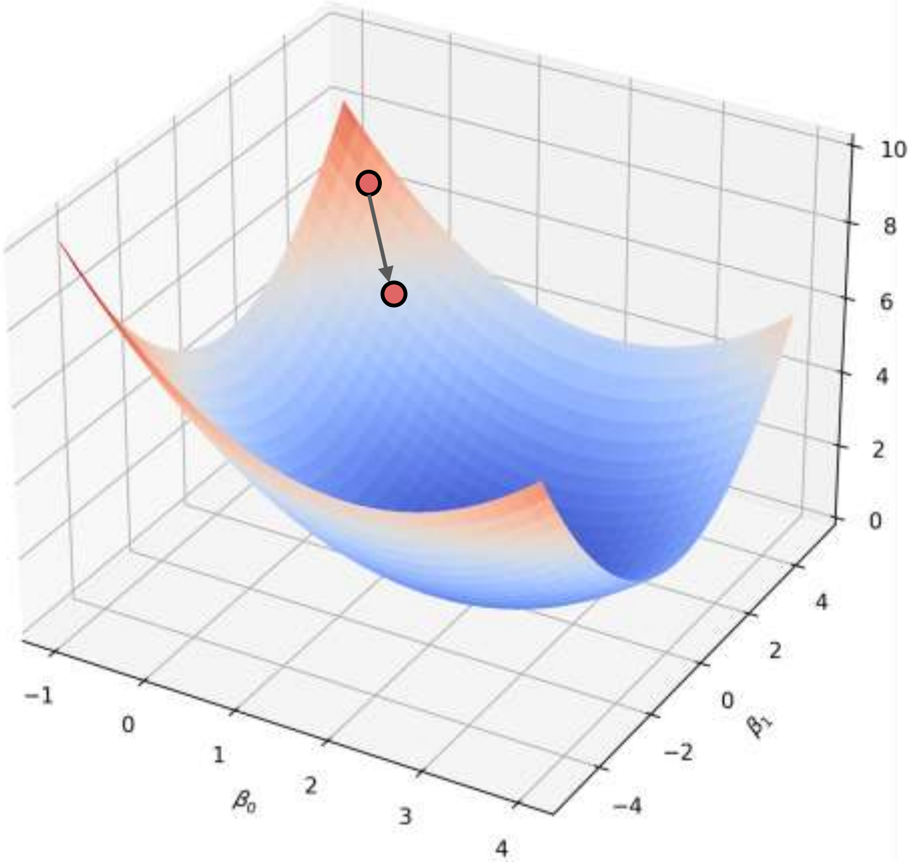
Linear Regression



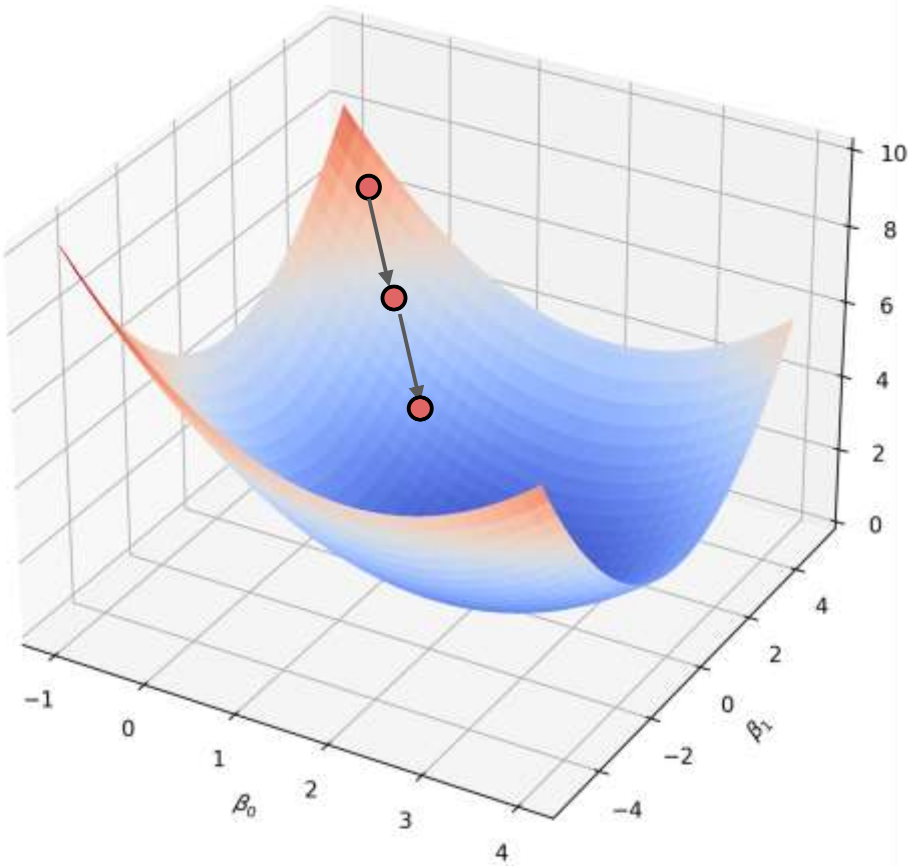
Linear Regression



Linear Regression

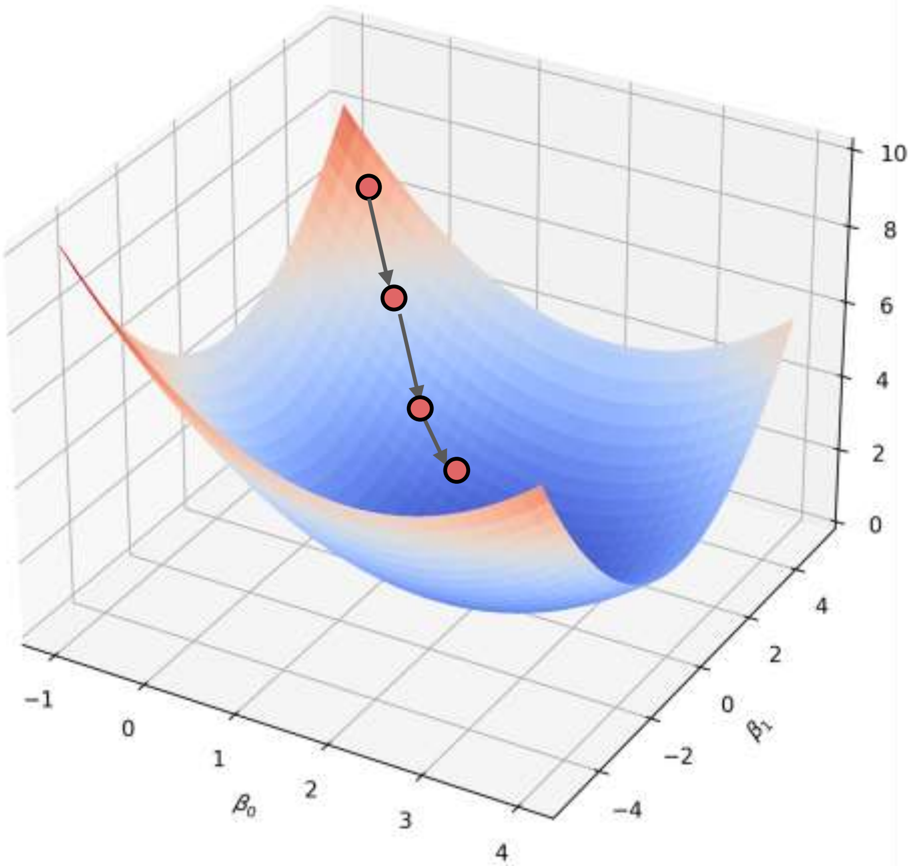


Linear Regression

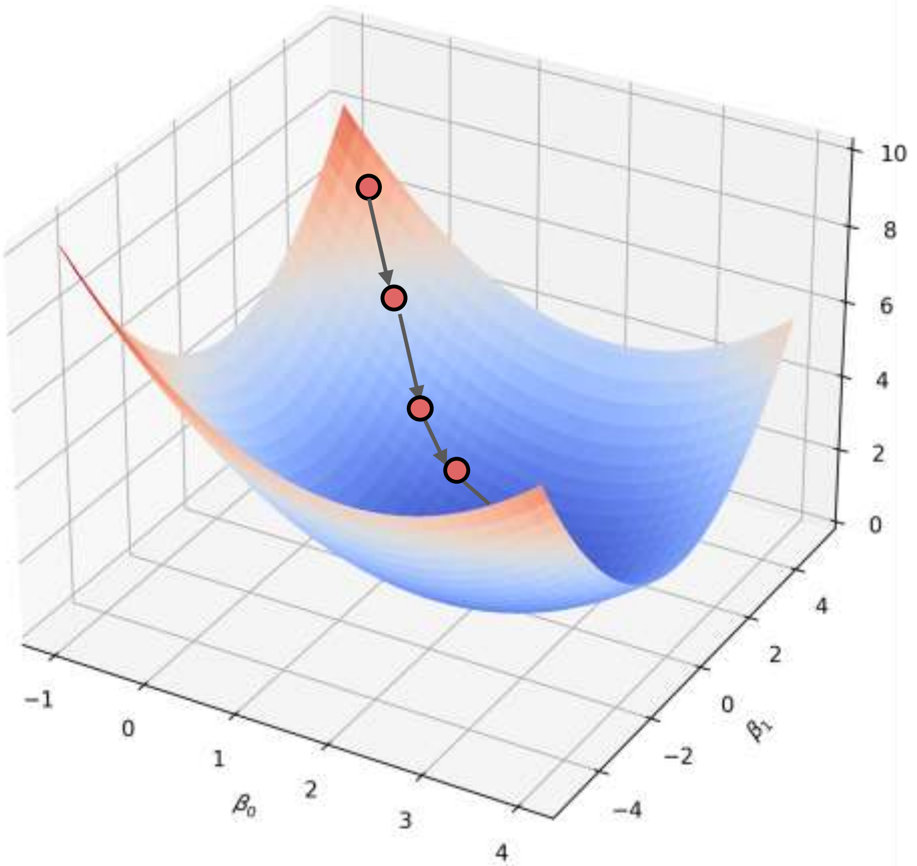


PIERIAN DATA

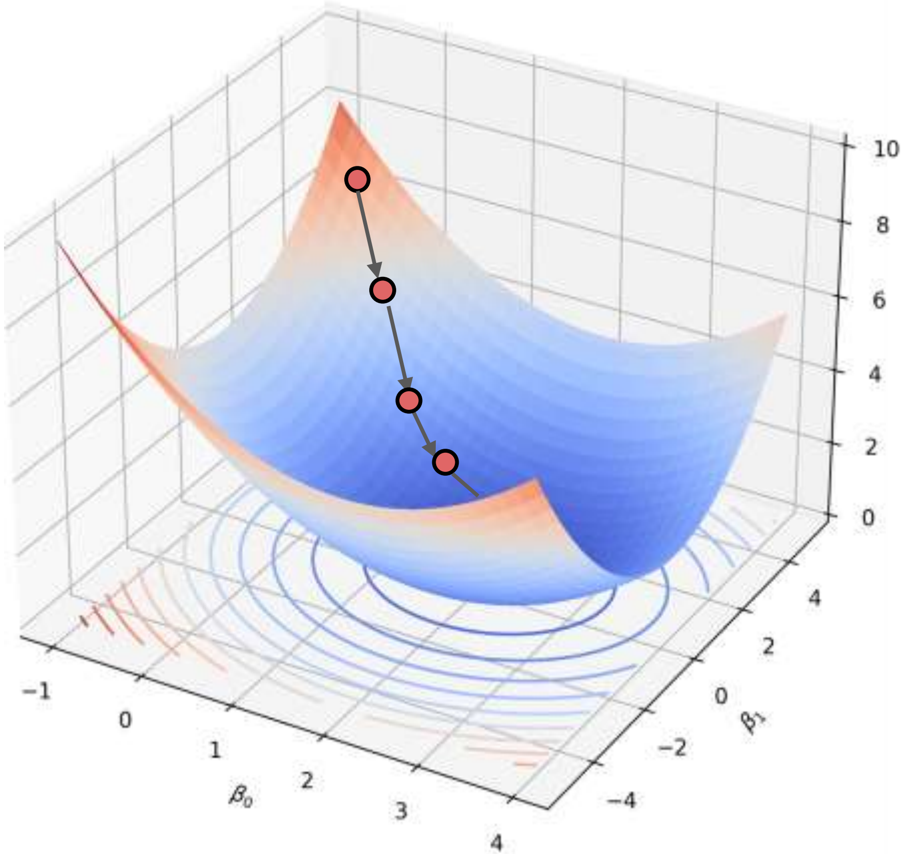
Linear Regression



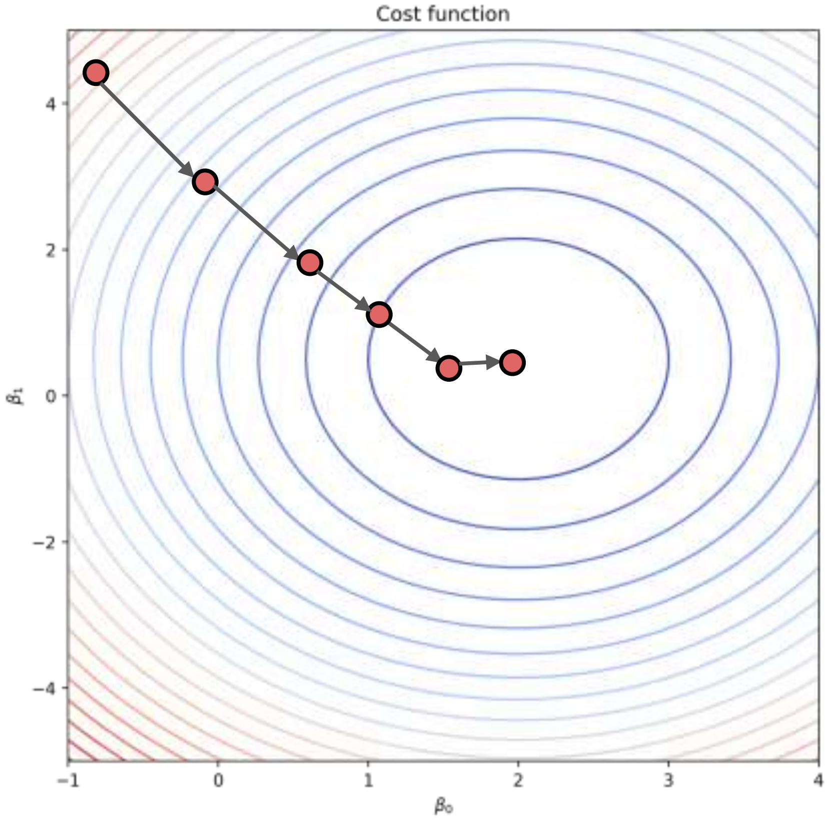
Linear Regression



Linear Regression



Linear Regression



PIERIAN DATA

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

PIERIAN DATA

Scikit-Learn

- We've seen NumPy had some built in capabilities for simple linear regression, but when it comes to more complex models, we'll need **Scikit-Learn**!
- Before we jump straight into machine learning with **Scikit-Learn** and Python, let's understand the philosophy behind **sklearn.**

Scikit-Learn

- Scikit-learn is a library containing many machine learning algorithms.
- It utilizes a generalized "estimator API" framework to calling the models.
- This means the way algorithms are imported, fitted, and used is uniform across all algorithms.

Scikit-Learn

- This allows users to easily swap algorithms in and out and test various approaches.
- Important Note:
    - *This uniform framework also means users can easily apply almost any algorithm effectively without truly understanding what the algorithm is doing!*

Scikit-Learn

- Scikit-learn also comes with many convenience tools, including train test split functions, cross validation tools, and a variety of reporting metric functions.
- This leaves Scikit-Learn as a “one-stop shop” for many of our machine learning needs.

Scikit-Learn

- Philosophy of Scikit-Learn
    - Scikit-Learn’s approach to model building focuses on **applying models** and **performance metrics**.
    - This is a more pragmatic industry style approach rather than an academic approach of describing the model and its parameters.

Scikit-Learn

- Philosophy of Scikit-Learn
  - Academic users used to **R** style reporting may also want to explore the **statsmodels** python library if interested in more statistical description of models such as significance levels.

Scikit-Learn

- Let's quickly review the framework of Scikit-Learn for the **supervised** machine learning process.
- We will quickly see how the code directly relates to the process theory!

Supervised Machine Learning Process

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

	Area m^2	Bedrooms	Bathrooms	Price
TRAIN	200	3	2	$500,000
	190	2	1	$450,000
	230	3	3	$650,000
TEST	180	1	1	$400,000
	210	2	2	$550,000

Supervised Machine Learning Process

- Also recall there are 4 main components after a Train | Test split:

	Area m^2	Bedrooms	Bathrooms	Price	
X TRAIN	200	3	2	$500,000	Y TRAIN
	190	2	1	$450,000	
	230	3	3	$650,000	
X TEST	180	1	1	$400,000	Y TEST
	210	2	2	$550,000	

Supervised Machine Learning Process

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

	Area m^2	Bedrooms	Bathrooms	Price
TRAIN	200	3	2	$500,000
	190	2	1	$450,000
	230	3	3	$650,000
TEST	180	1	1	$400,000
	210	2	2	$550,000

Scikit-Learn

```python
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.

Predictions	Area m^2	Bedrooms	Bathrooms	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)
```

Scikit-Learn

- This framework will be similar for any supervised machine learning algorithm.
- Let’s begin exploring it further with Linear Regression!

Linear Regression with Scikit-Learn

Part One:

Data Setup and Model Training

PIERIAN DATA

Linear Regression

- Previously, we explored "*Is there a relationship between* ***total advertising spend*** *and* ***sales****?* "
- Now we want to expand this to "*What is the relationship between* ***each advertising channel (TV,Radio,Newspaper)*** *and* ***sales?***"

Performance Evaluation

Regression Metrics

Evaluating Regression

- Now that we have a fitted model that can perform predictions based on features, how do we decide if those predictions are any good?
- Fortunately we have the known test labels to compare our results to.

Evaluating Regression

* Let’s take a moment now to discuss evaluating Regression Models
* Regression is a task when a model attempts to predict continuous values (unlike categorical values, which is classification)

Evaluating Regression

- For example, attempting to predict the price of a house given its features is a **regression task.**
- Attempting to predict the country a house is in given its features would be a classification task.

Evaluating Regression

- You may have heard of some evaluation metrics like accuracy or recall.
- These sort of metrics aren’t useful for regression problems, we need metrics designed for **continuous** values!

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

- The metrics shown here apply to any regression task, not just Linear Regression!

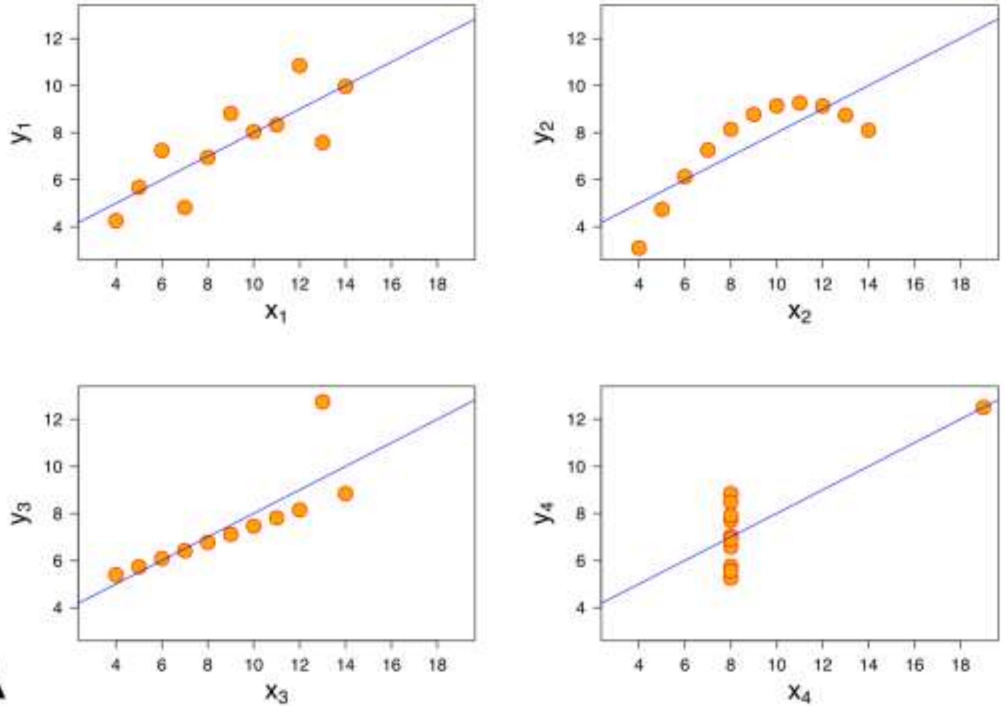
Evaluating Regression

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

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

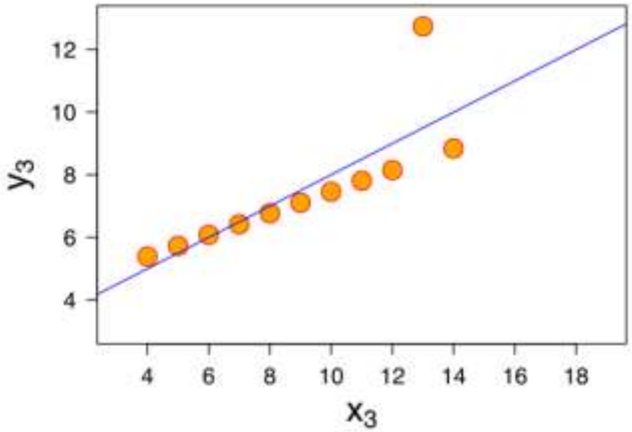
Evaluating Regression

- MAE won't punish large errors however.



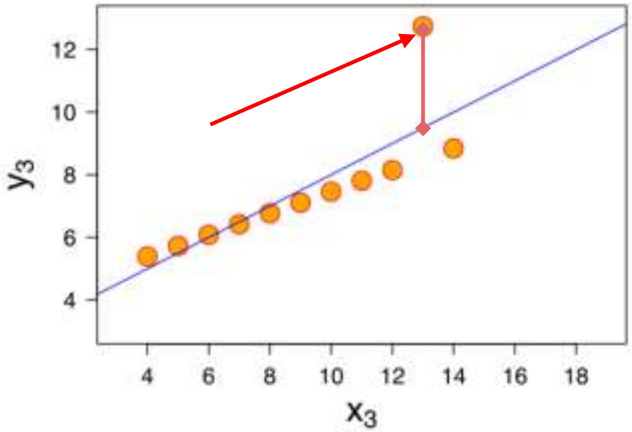
Evaluating Regression

- 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)

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

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!

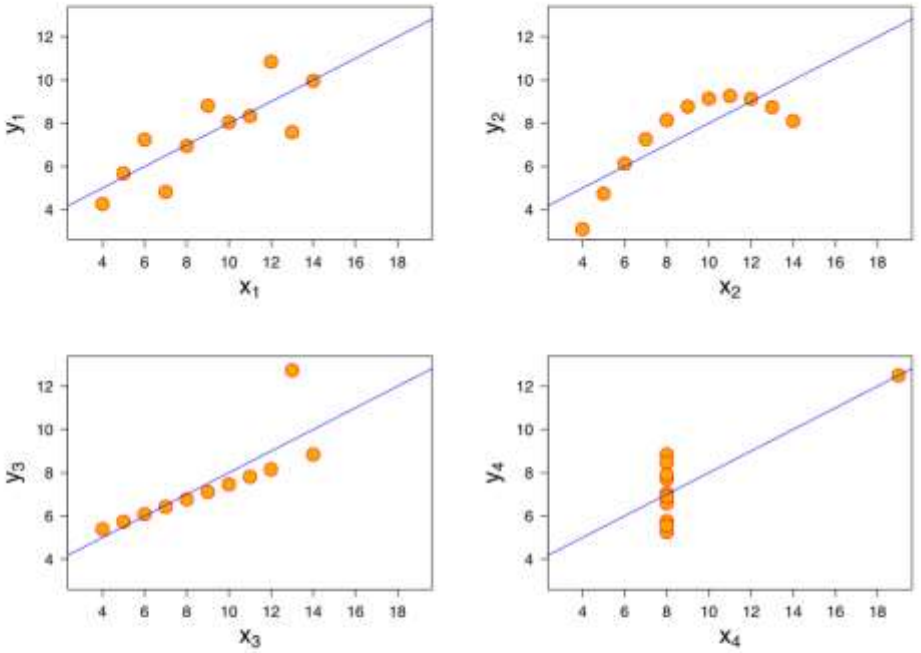
Evaluating Residuals

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...

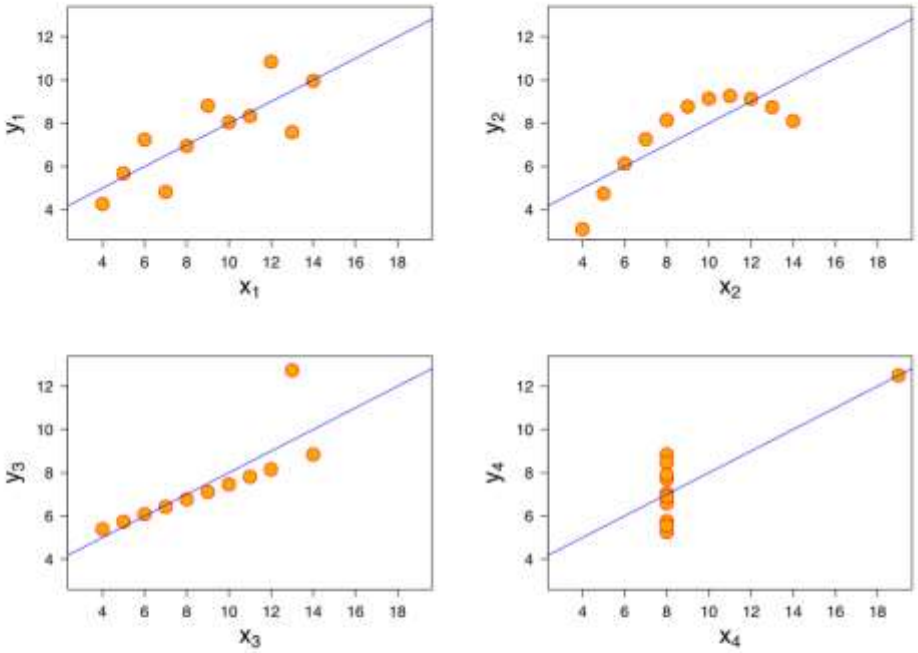
Linear Regression

- Recall Anscombe's quartet:



Linear Regression

- Clearly Linear Regression is not suitable!

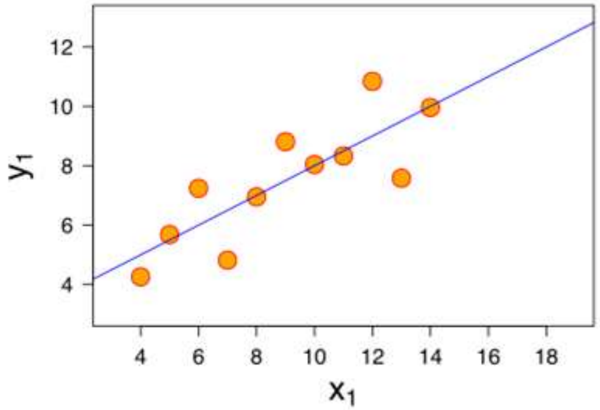


Linear Regression

- 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!

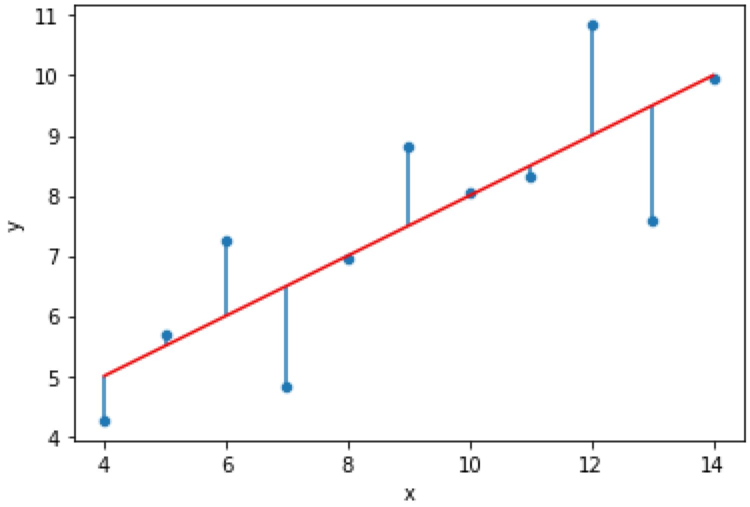
Linear Regression

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



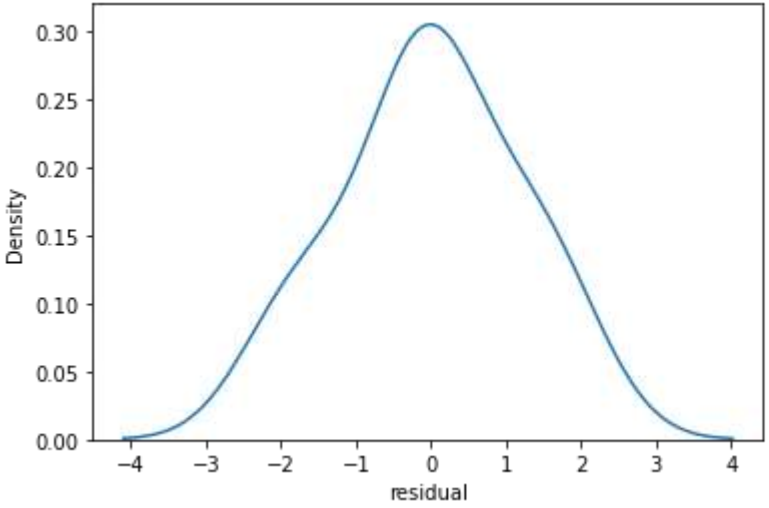
Linear Regression

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



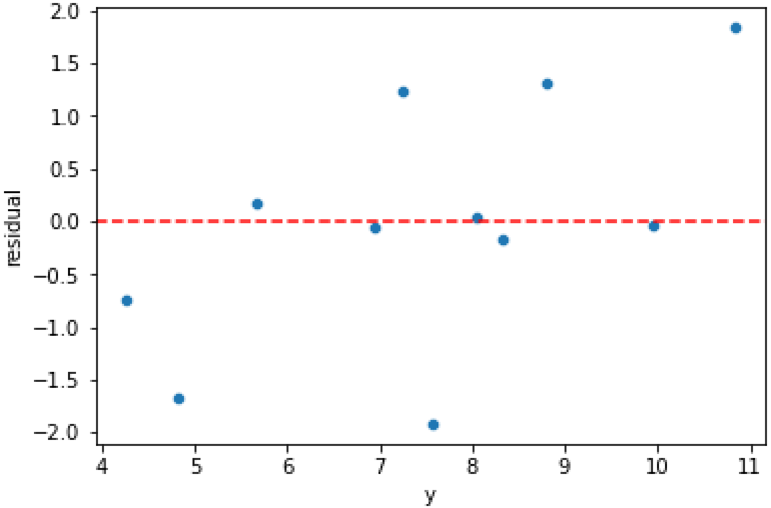
Linear Regression

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



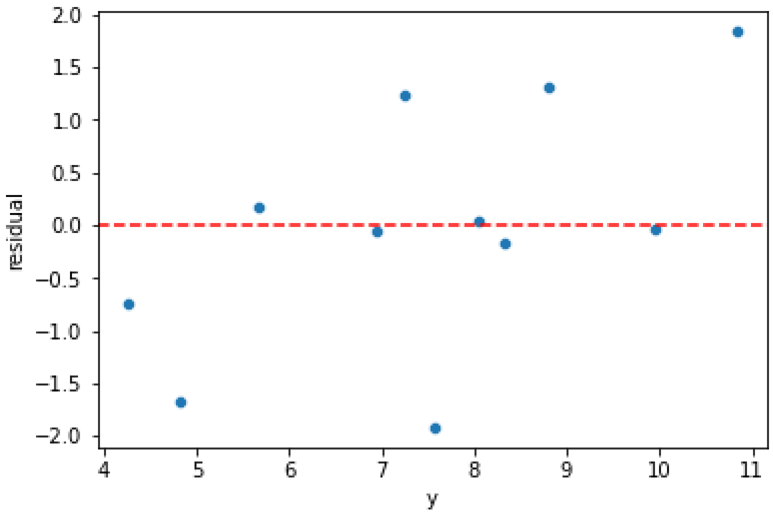
Linear Regression

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



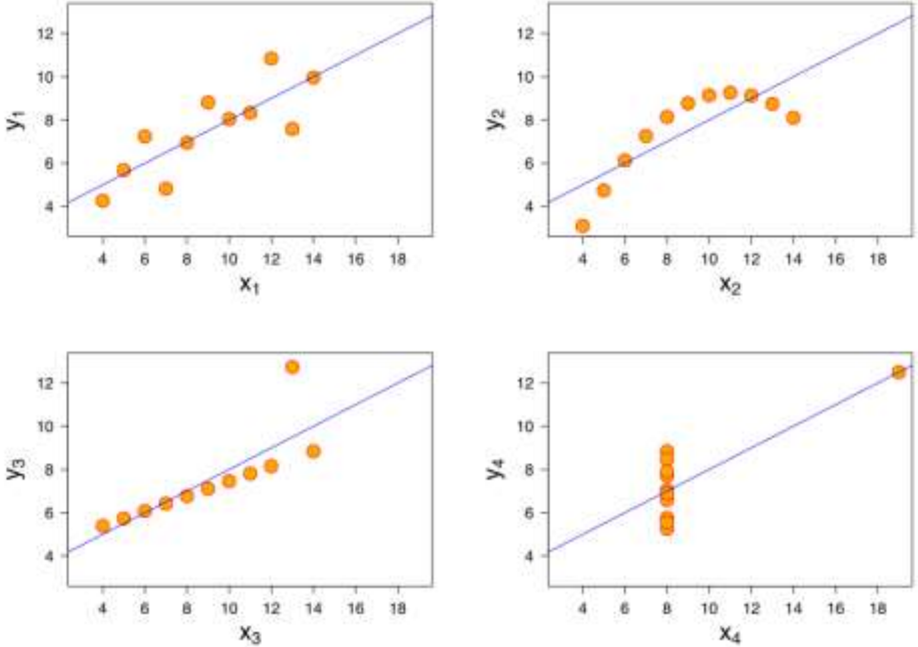
Linear Regression

- There should be no clear line or curve.



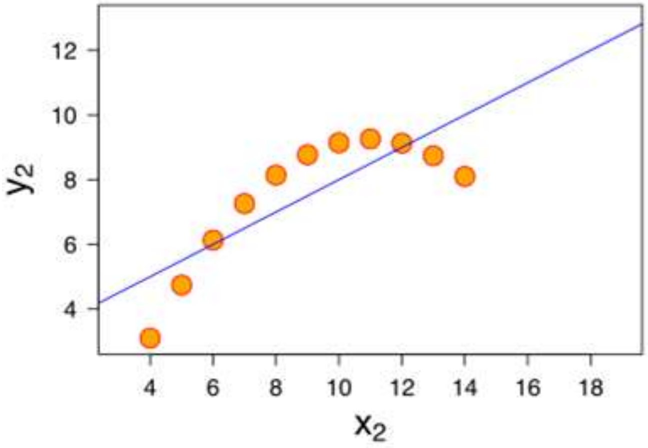
Linear Regression

- What about non valid datasets?



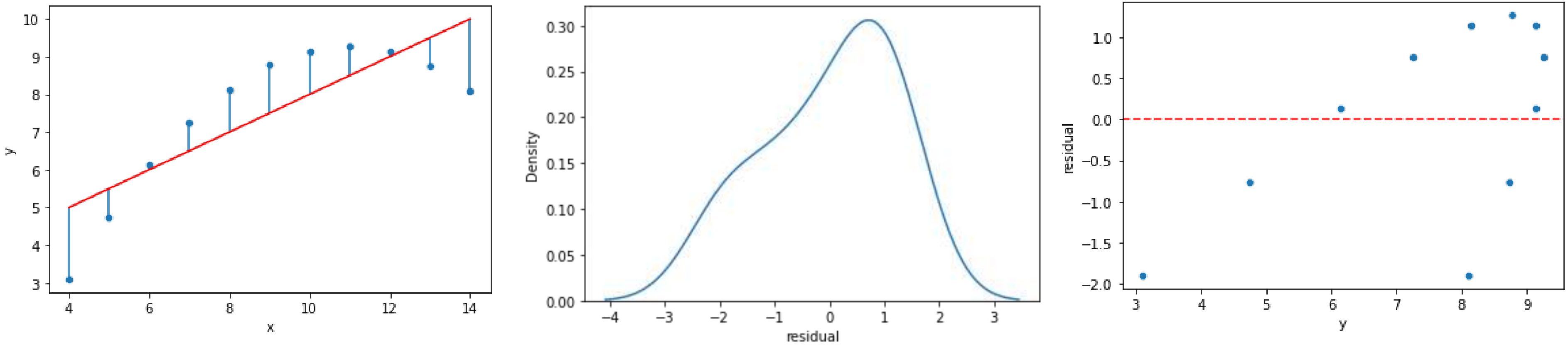
Linear Regression

- What about non valid datasets?



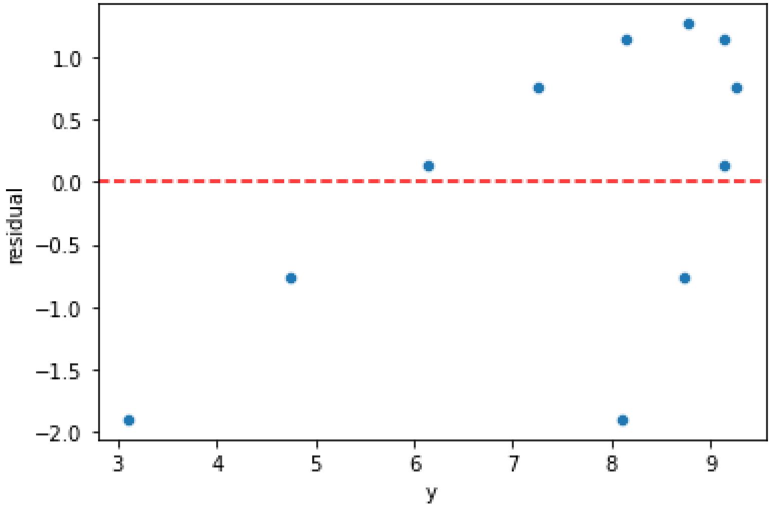
Linear Regression

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



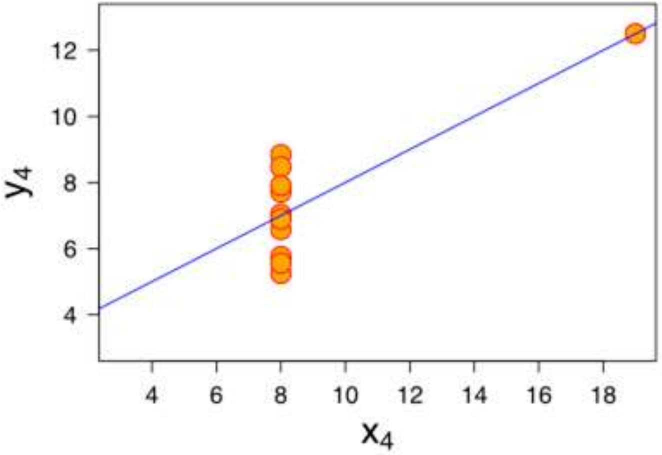
Linear Regression

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



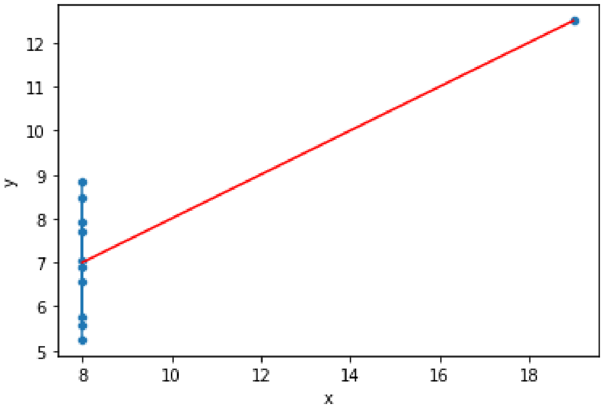
Linear Regression

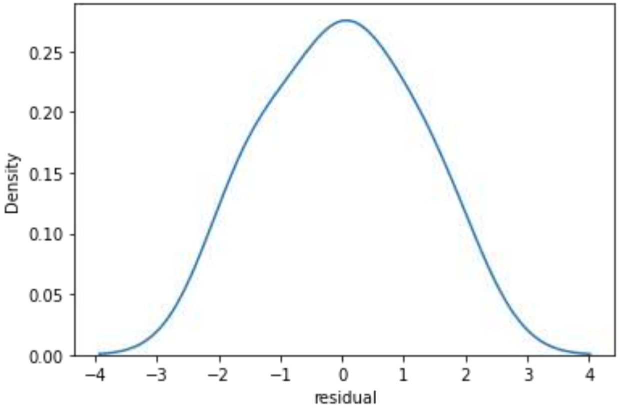
- What about non valid datasets?

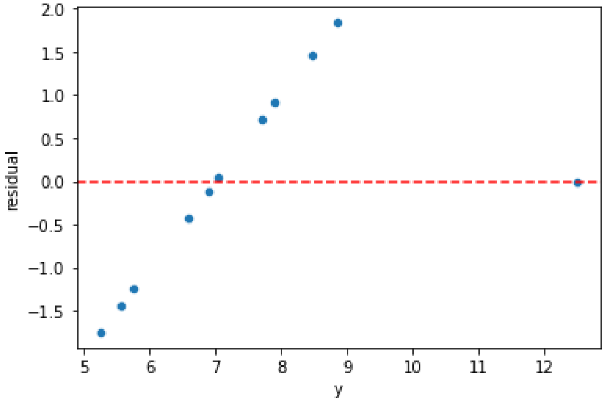


Linear Regression

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

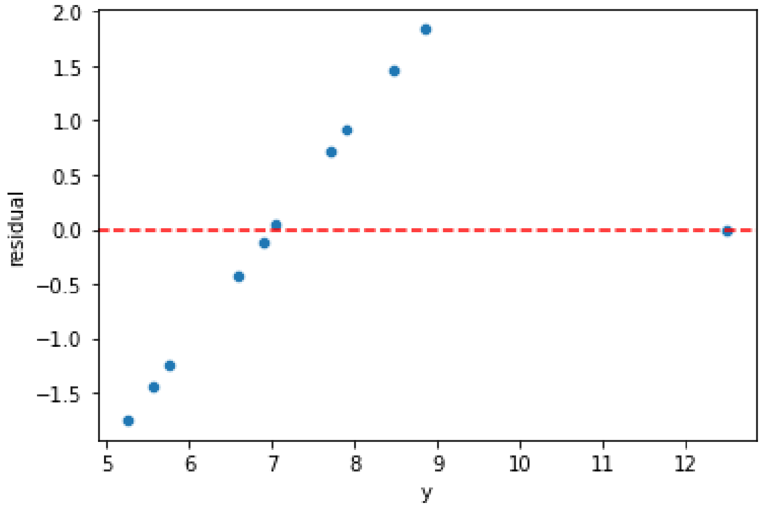






Linear Regression

- 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!

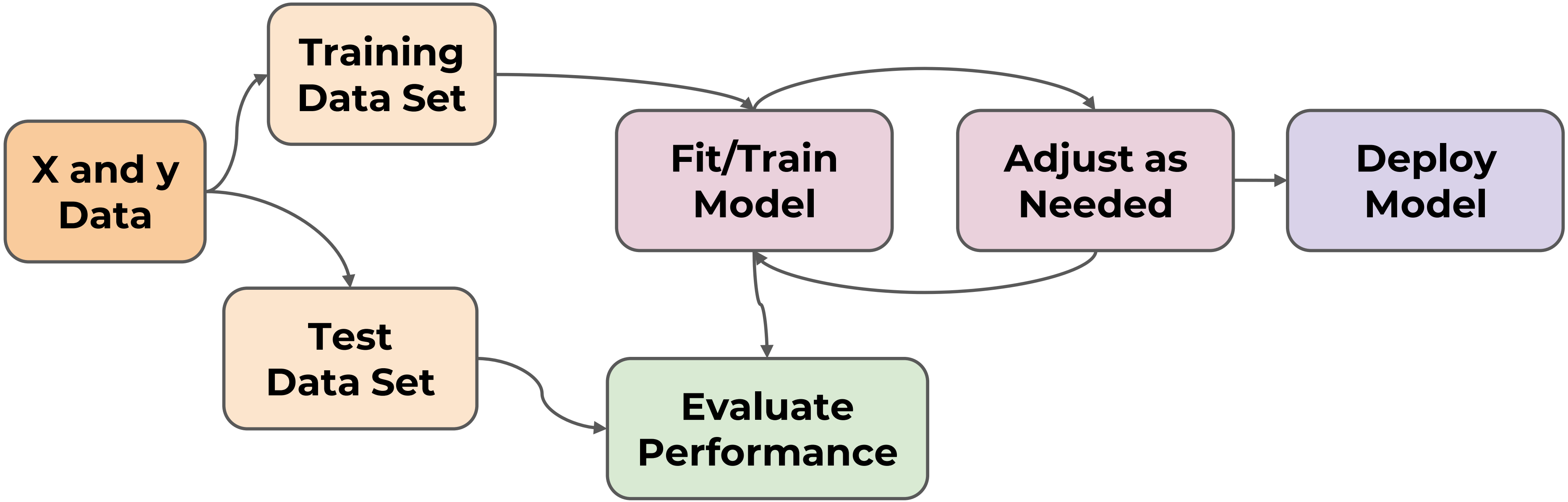
Model Deployment

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

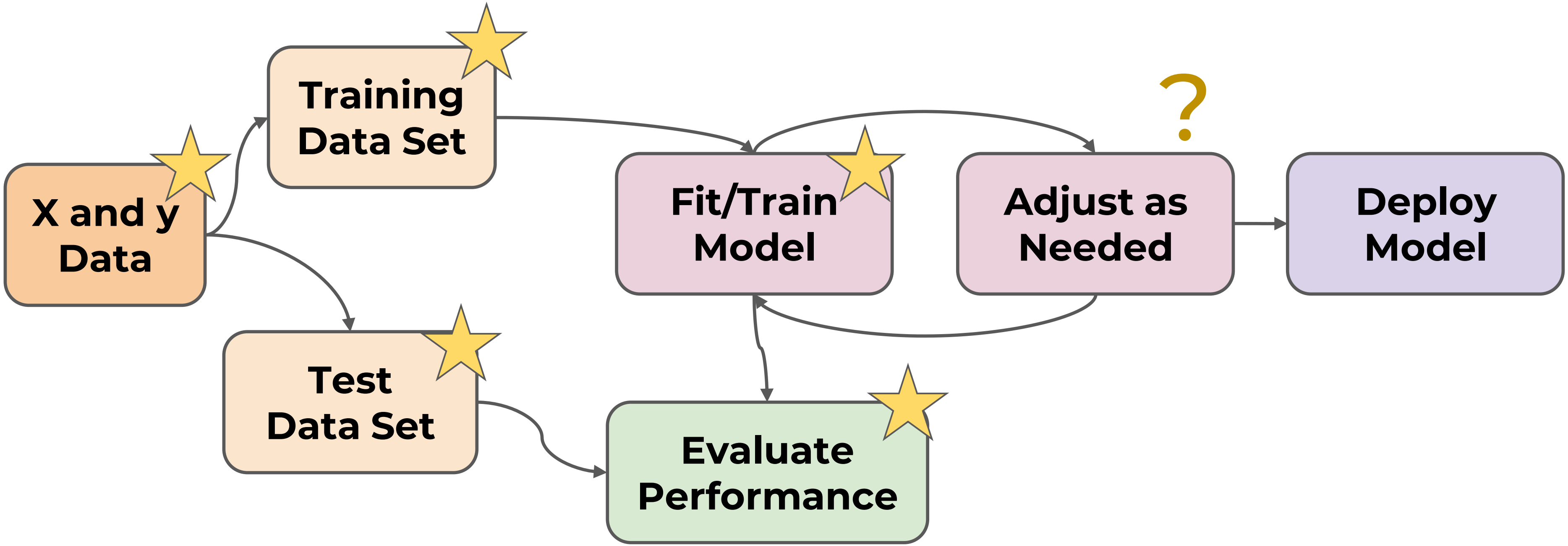
- Recall the Supervised ML Process



PIERIAN DATA

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

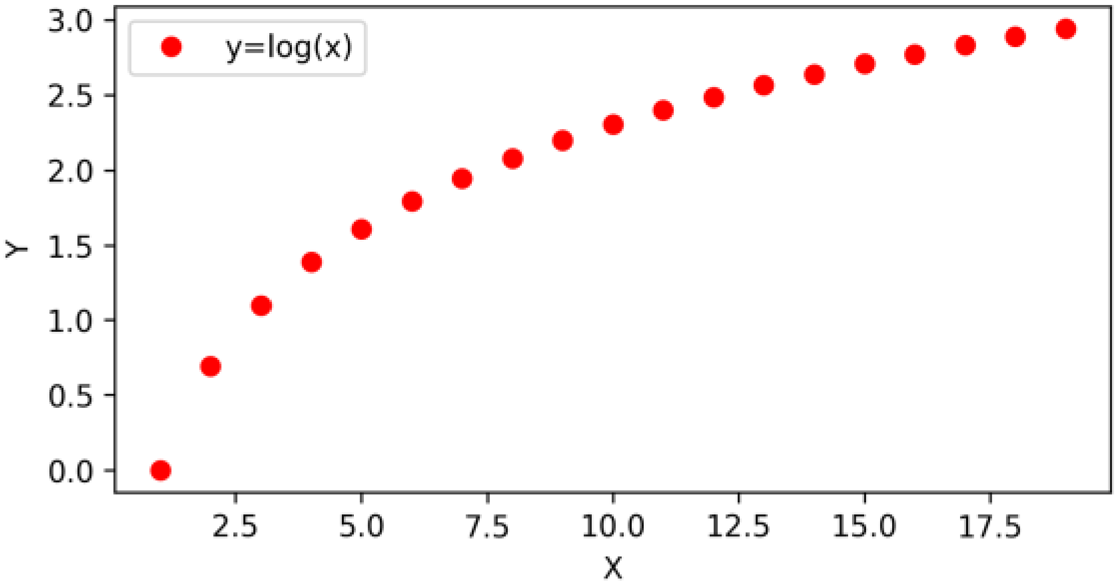
- We just completed a Linear Regression task, allowing us to predict future label values given a set of features!
- How can we now improve on a Linear Regression model?
- One approach is to consider **higher order relationships** on the features.

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.

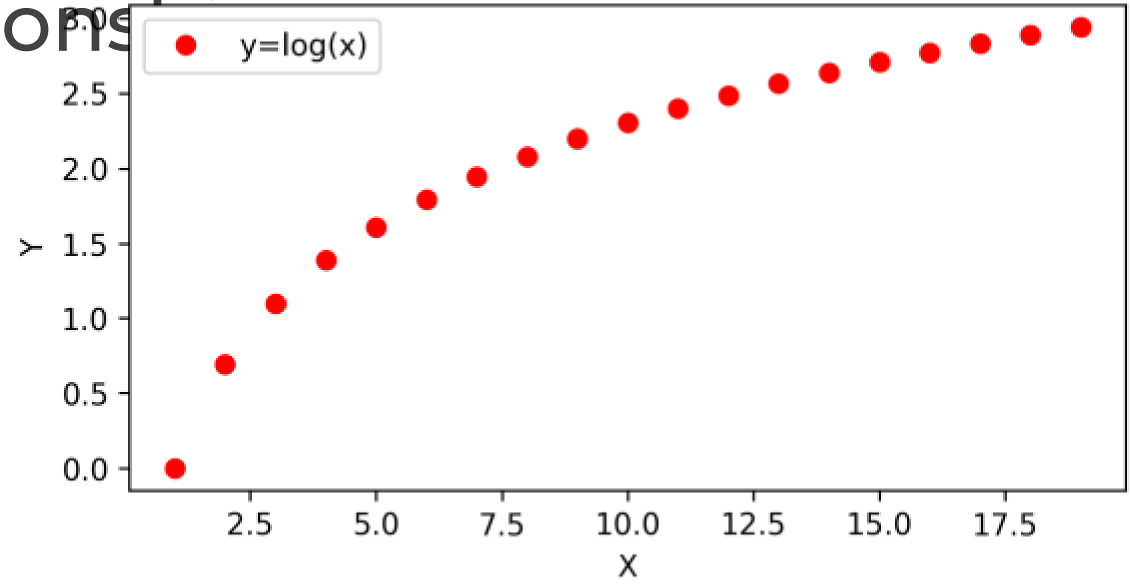
Polynomial Regression

- Imagine a feature that is not linear:



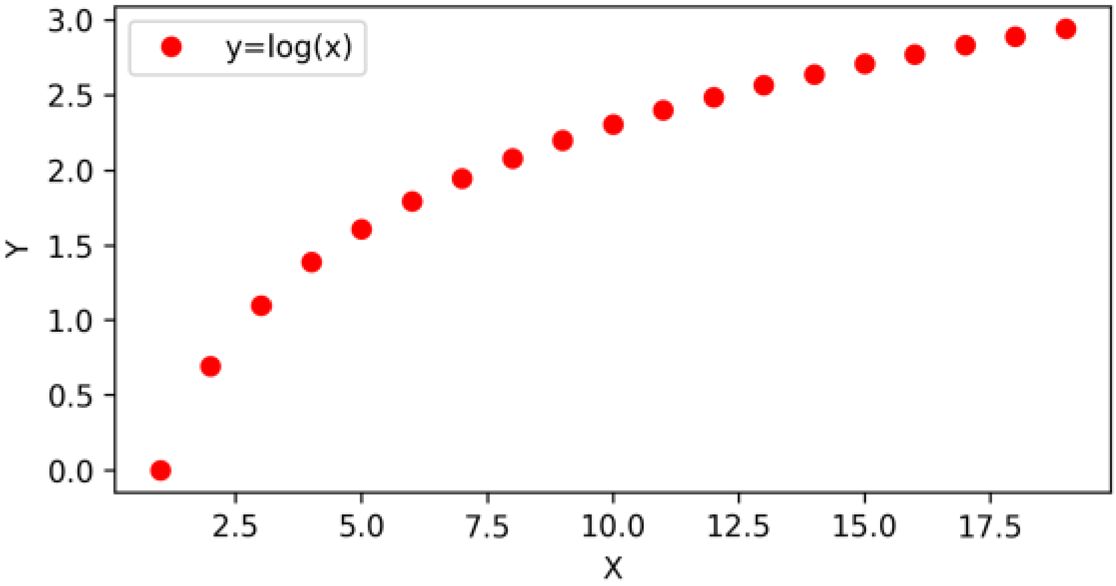
Polynomial Regression

- We know log(x) is not a linear relations



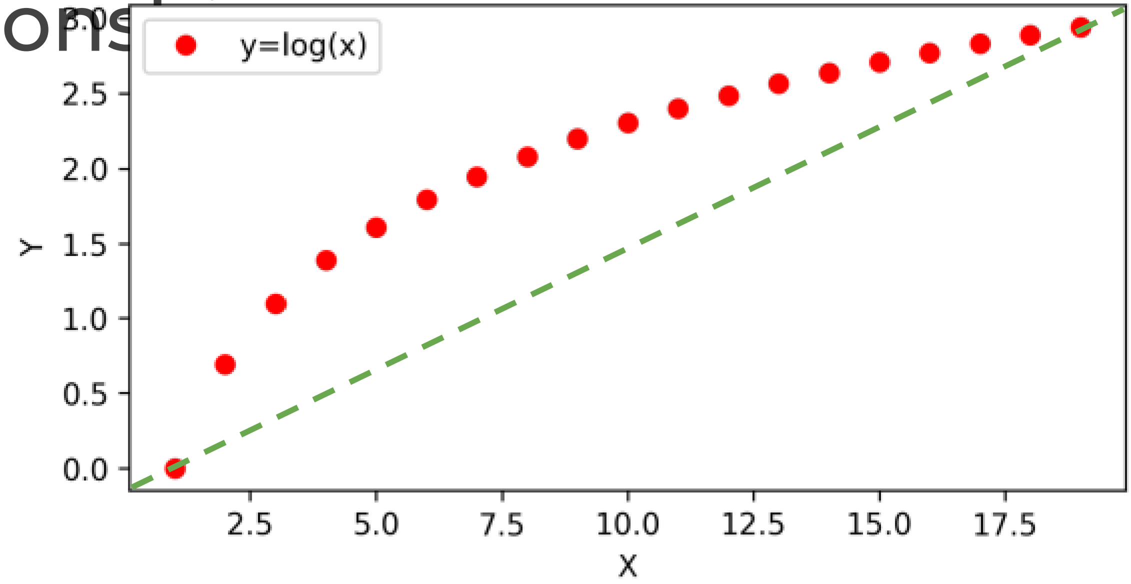
Polynomial Regression

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



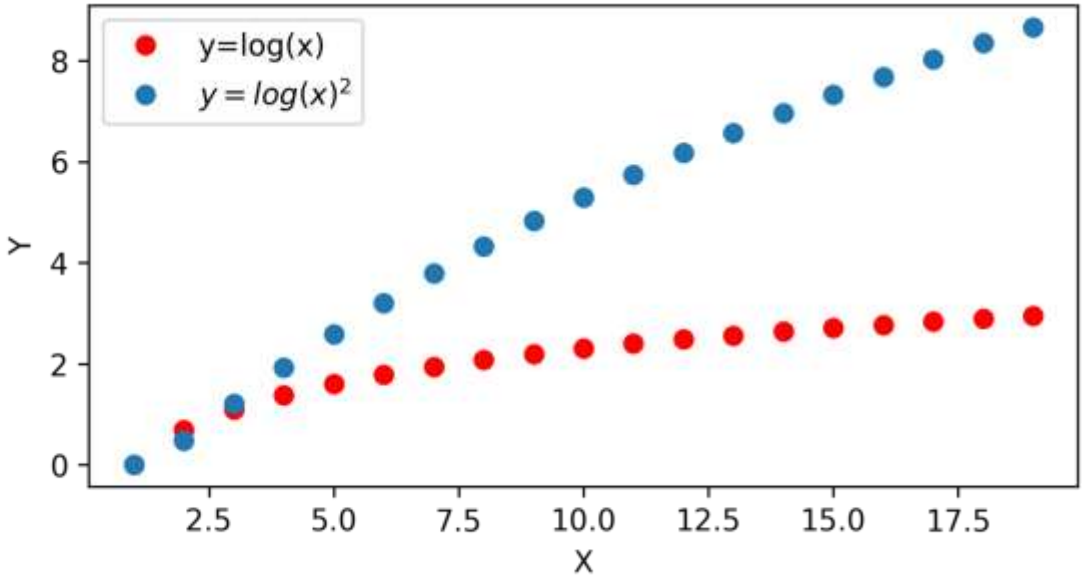
Polynomial Regression

- Will be difficult to find a linear relationship



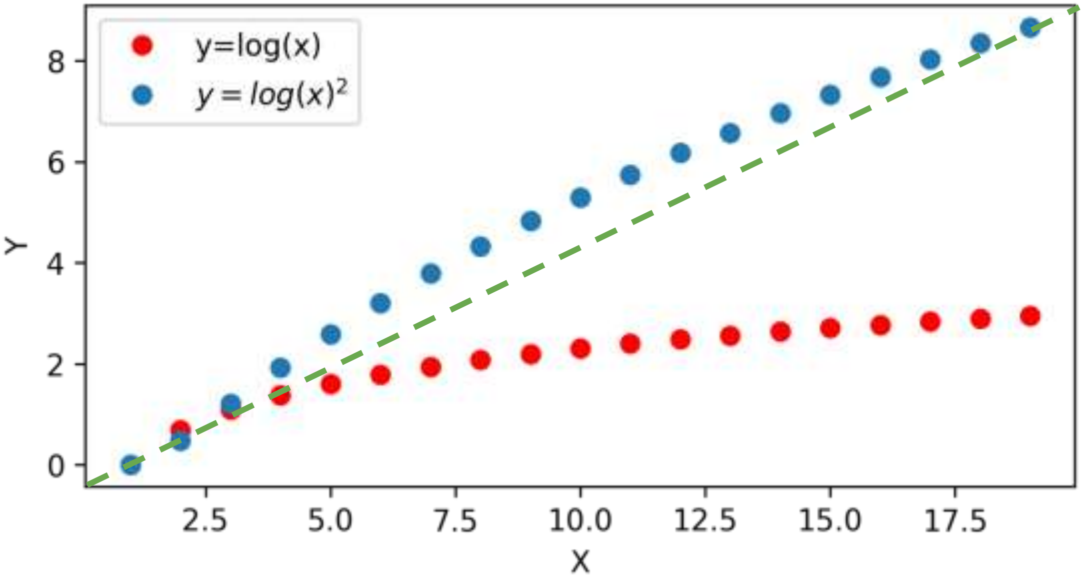
Polynomial Regression

- What about the square of this feature?



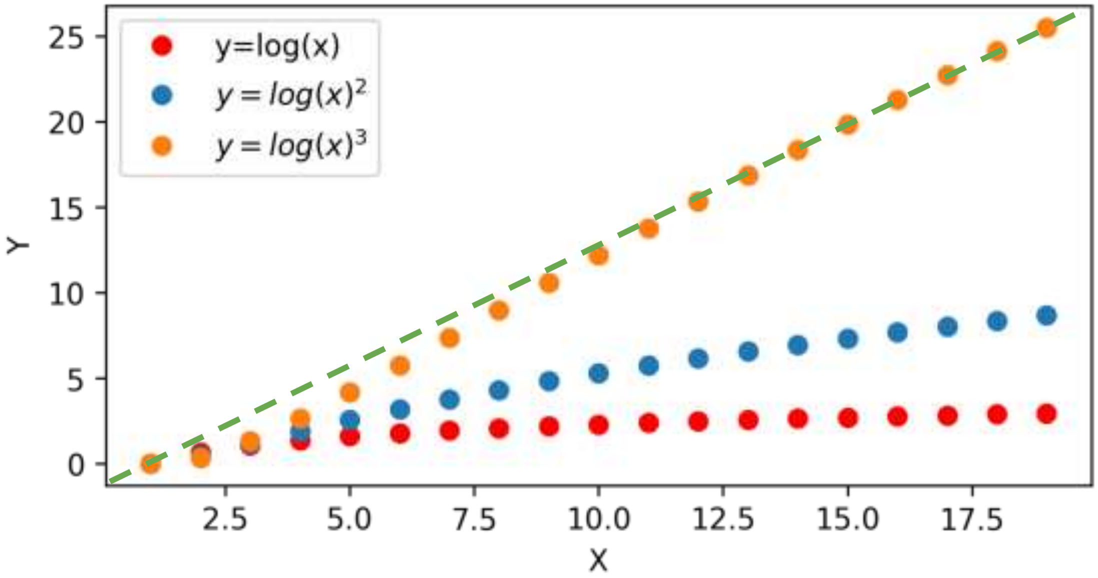
Polynomial Regression

- 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, ...)
  - Interactions between all pairs of features (e.g. x1 * x2, x1 * x3, ...)

Polynomial Regression

- Converting Two Features **A** and **B**
    - **1, A, B, A^2, AB, B^2**

Polynomial Regression

- Converting Two Features **A** and **B**
  - **1, A, B, A^2, AB, B^2**
- Generalized terms of features X_1 and X_2
  - **1, X_1, X_2, X_1^2, X_1X_2, X_2^2**

Polynomial Regression

- Converting Two Features **A** and **B**
  - **1, A, B, A^2, AB, B^2**
- Generalized terms of features X_1 and X_2
  - **1, X_1, X_2, X_1^2, X_1X_2, X_2^2**
- Example if row was $X_1=2$ and $X_2=3$
  - **1, 2, 3, 4, 6, 9**

Polynomial Regression

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

Polynomial Regression

Creating Polynomial Features

Polynomial Regression

Training and Evaluating Model

Bias-Variance Trade Off

Overfitting versus Underfitting

Overfitting and Underfitting

- We have seen that a higher order polynomial model performs significantly better than a standard linear regression model.
- But how can we choose the optimal degree for the polynomial?
- What trade-offs are we to consider as we increase model complexity?

Overfitting and Underfitting

- In general, increasing model complexity in search for better performance leads to a **Bias-Variance trade-off**.
- We want to have a model that can generalize well to new unseen data, but can also account for variance and patterns in the known data.

Overfitting and Underfitting

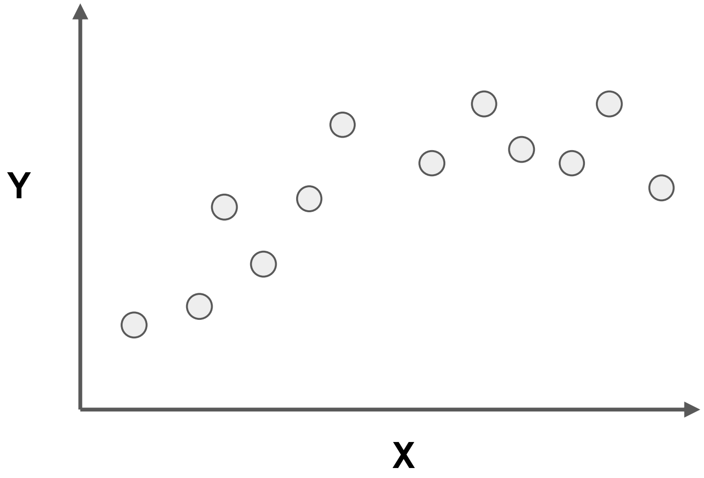
- Extreme bias or extreme variance both lead to bad models.
- We can visualize this effect by considering a model that underfits (high bias) or a model that overfits (high variance).
- Let’s start with a model that overfits to a dataset...

Overfitting and Underfitting

- **Overfitting**
  - The model fits too much to the noise from the data.
  - This often results in **low error on training sets but high error on test/validation sets.**

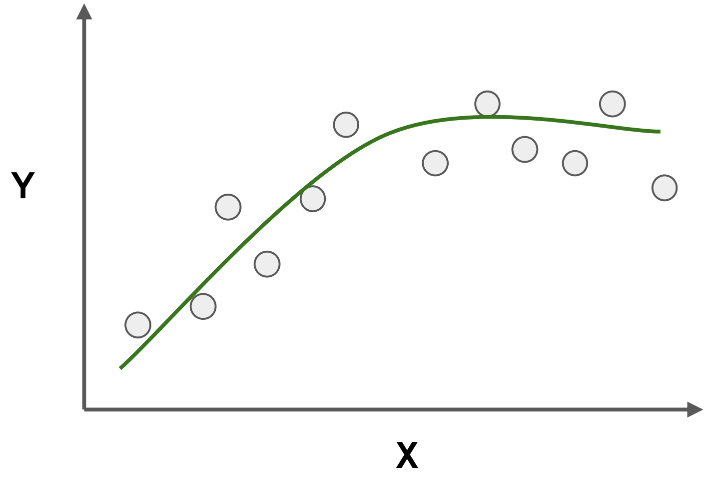
Overfitting and Underfitting

Data



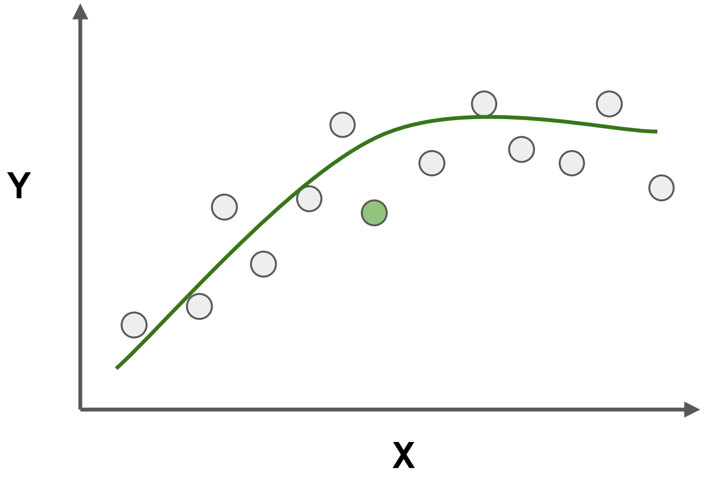
Overfitting and Underfitting

Good Model



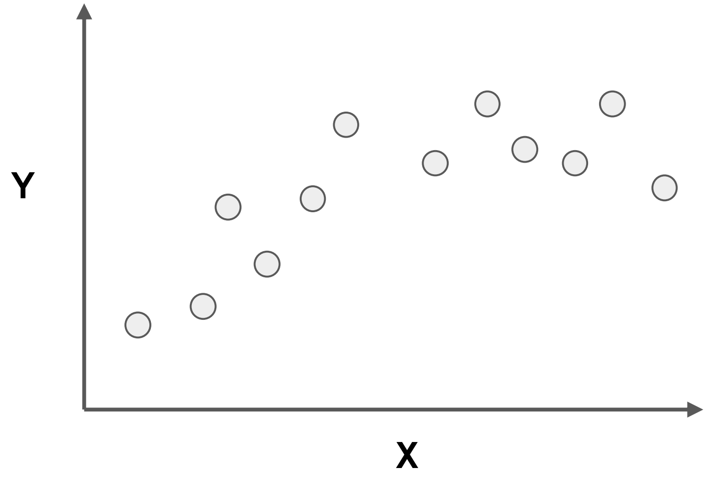
Overfitting and Underfitting

Good Model



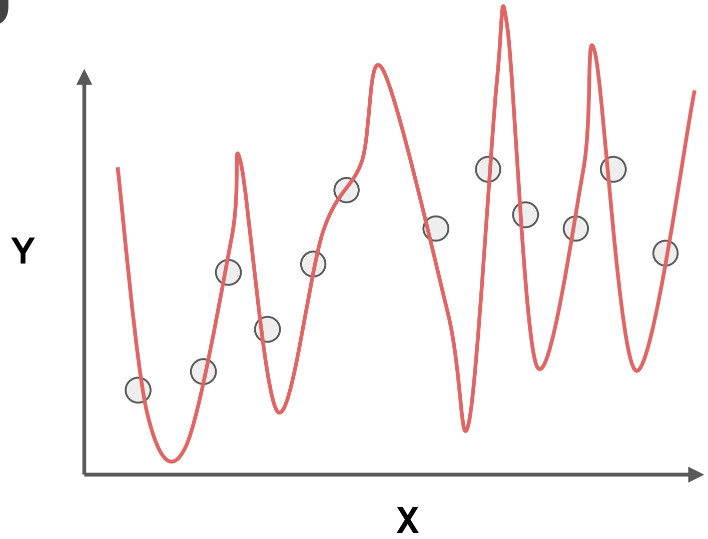
Overfitting and Underfitting

- **Overfitting**



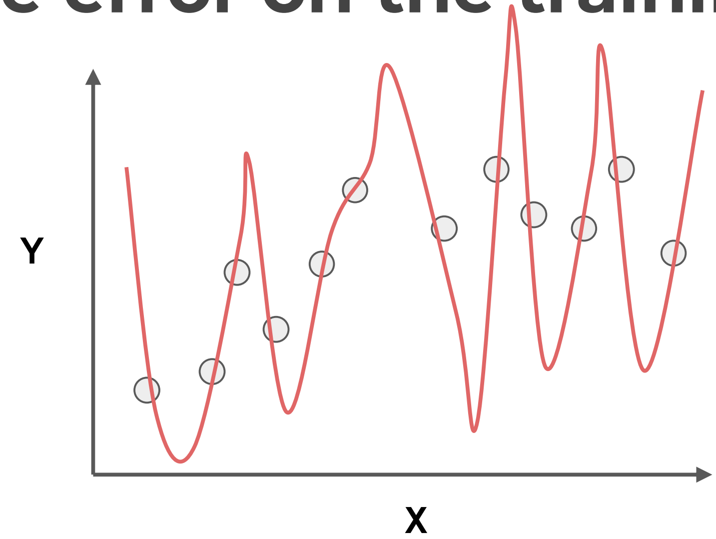
Overfitting and Underfitting

- **Overfitting**



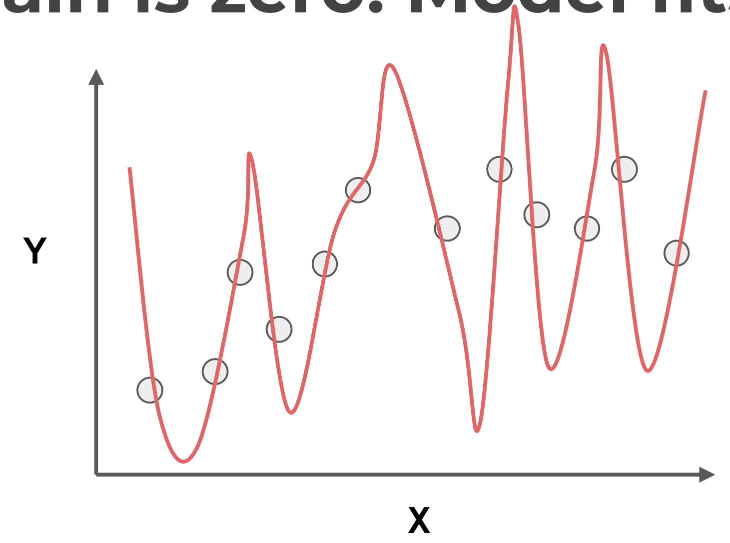
Overfitting and Underfitting

- **What is the error on the training data here?**



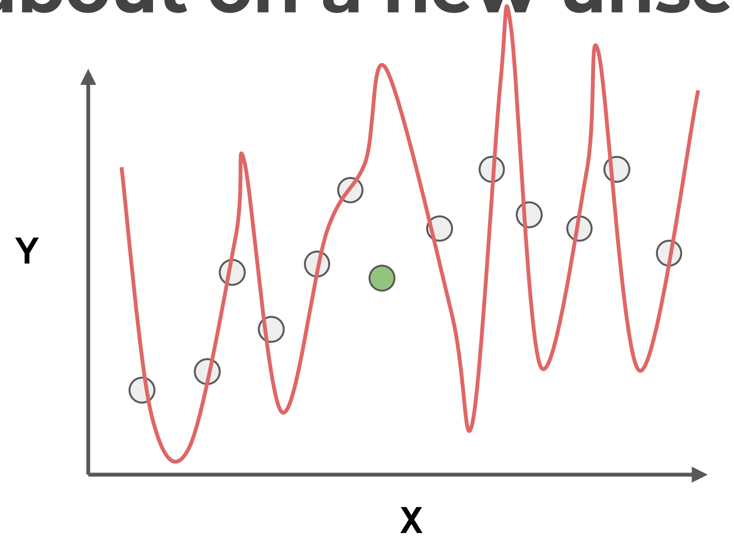
Overfitting and Underfitting

* **Error on train is zero! Model fits perfectly!**



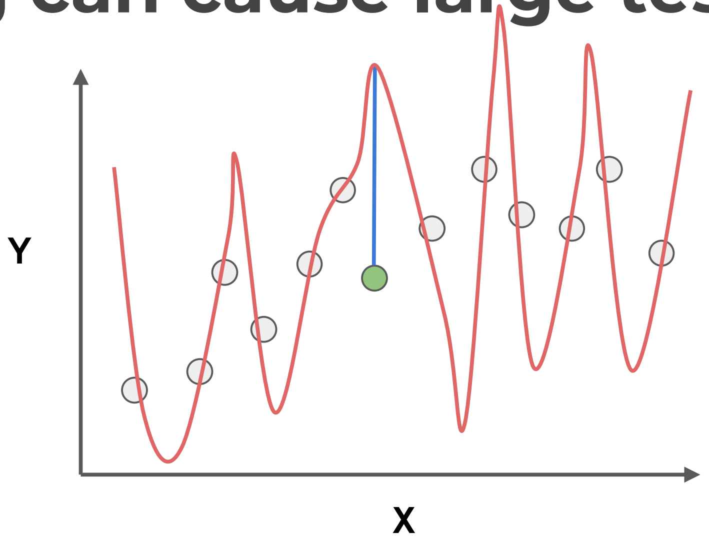
Overfitting and Underfitting

- **But what about on a new unseen data point?**



Overfitting and Underfitting

* **Overfitting can cause large test errors!**



Overfitting and Underfitting

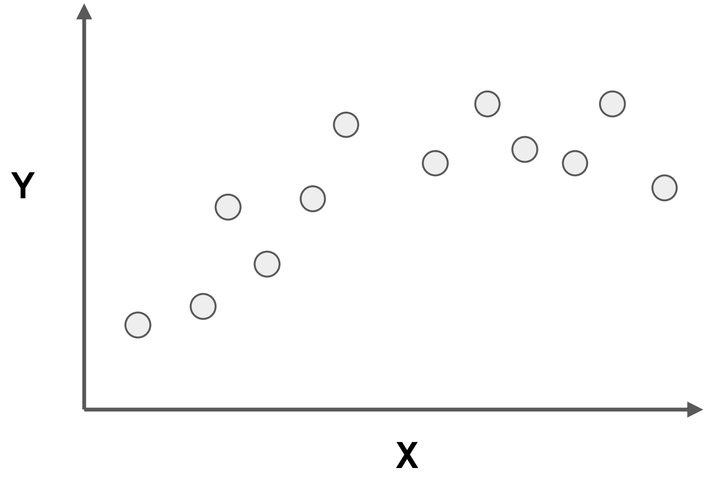
- **Overfitting**
  - Model is fitting too much to noise and variance in the training data.
  - Model will perform very well on training data, but have poor performance on new unseen data.

Overfitting and Underfitting

- **Underfitting**
  - Model does not capture the underlying trend of the data and does not fit the data well enough.
  - Low variance but high bias.
  - Underfitting is often a result of an excessively simple model.

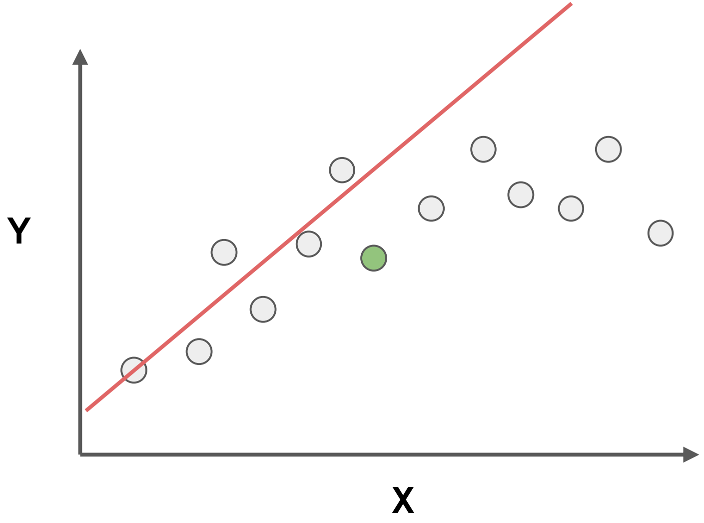
Overfitting and Underfitting

Data



Overfitting and Underfitting

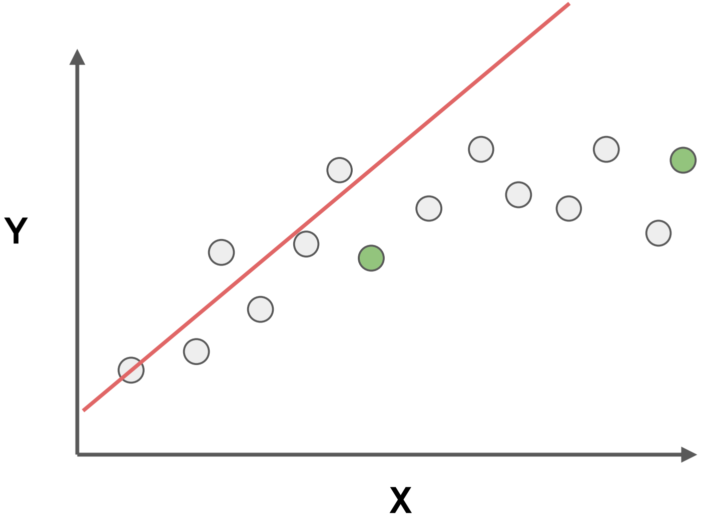
Underfitting



PIERIAN DATA

Overfitting and Underfitting

Underfitting



Overfitting and Underfitting

- **Underfitting**
  - Model has high bias and is generalizing too much.
  - Underfitting can lead to poor performance in both training and testing data sets.

Overfitting and Underfitting

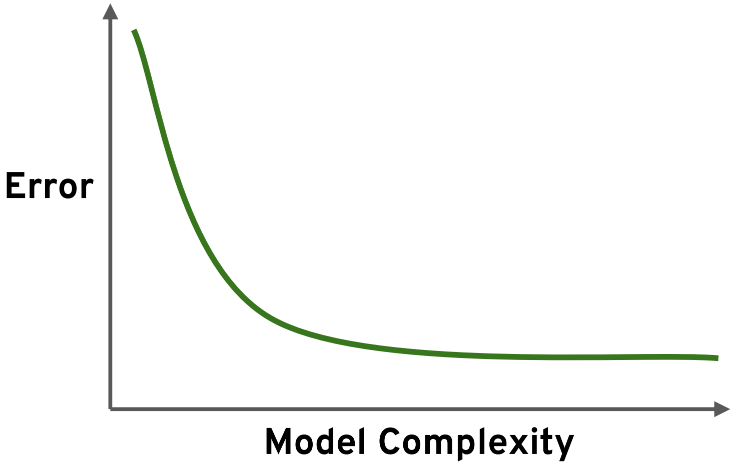
- **Overfitting versus Underfitting**
  - Overfitting can be harder to detect, since good performance on training data could lead to a model that appears to be performing well.

Overfitting and Underfitting

- This data was easy to visualize, but how can we see underfitting and overfitting when dealing with multi dimensional data sets?
- First let's imagine we trained a model and then measured its error versus model complexity (e.g. higher order polynomials).

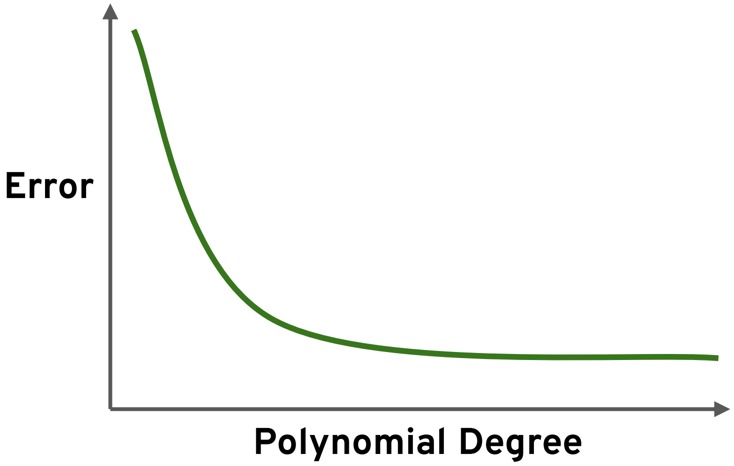
Overfitting and Underfitting

- Good Model



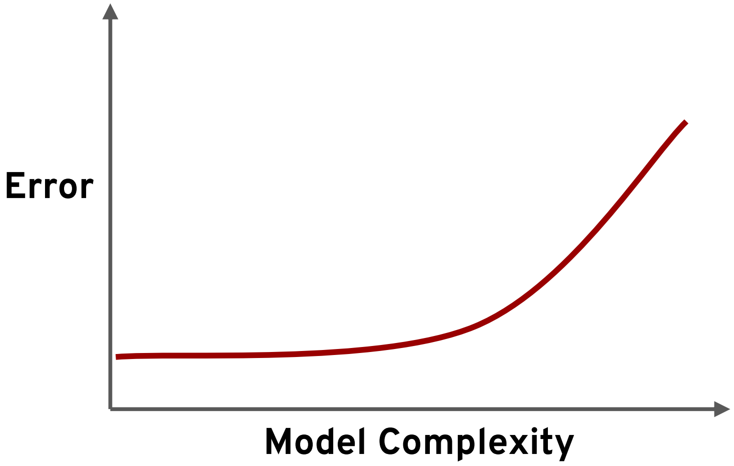
Overfitting and Underfitting

- Good Model



Overfitting and Underfitting

- Bad Model



Overfitting and Underfitting

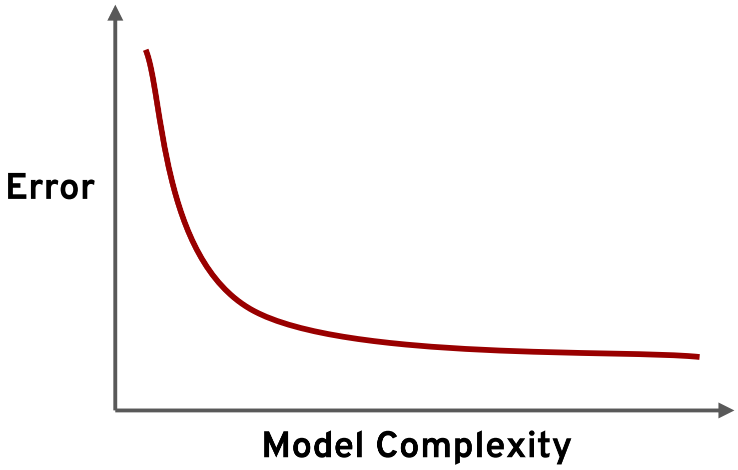
- When thinking about **overfitting** and **underfitting** we want to keep in mind the relationship of model performance on the training set versus the test/validation set.

Overfitting and Underfitting

- Let's imagine we split our data into a **training set** and a **test set**

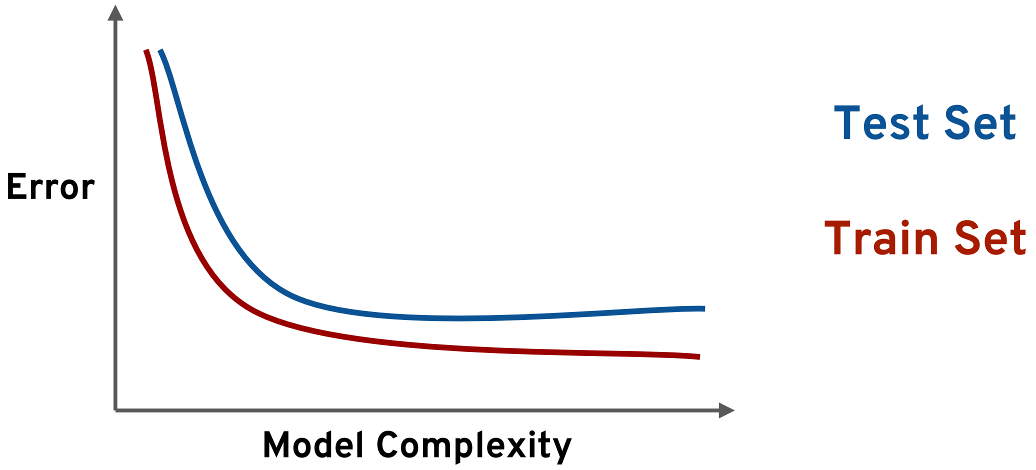
Overfitting and Underfitting

- We first see performance on the **training set**



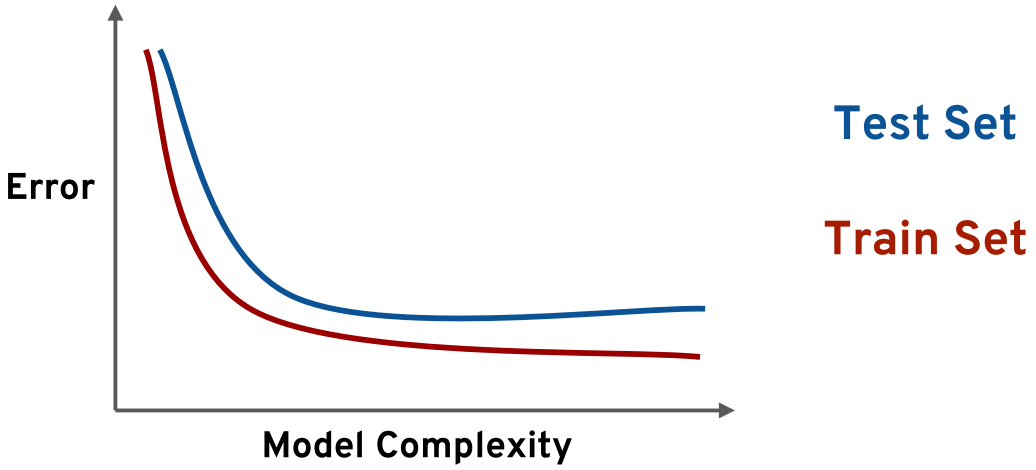
Overfitting and Underfitting

- Next we check performance on the **test set**



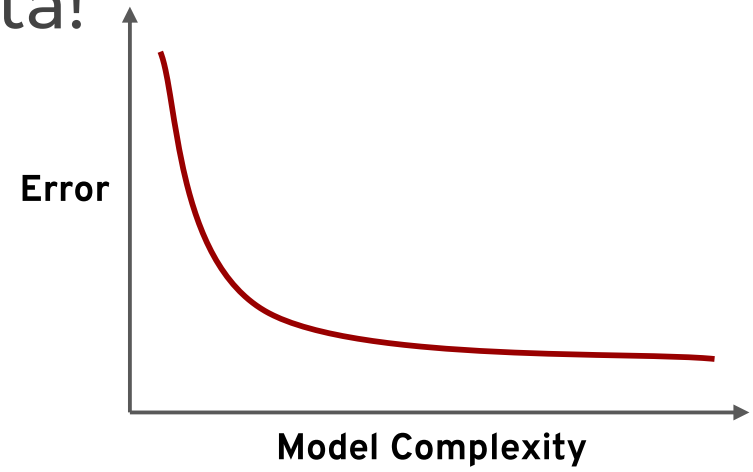
Overfitting and Underfitting

- Ideally the model would perform well on both, with similar behavior.



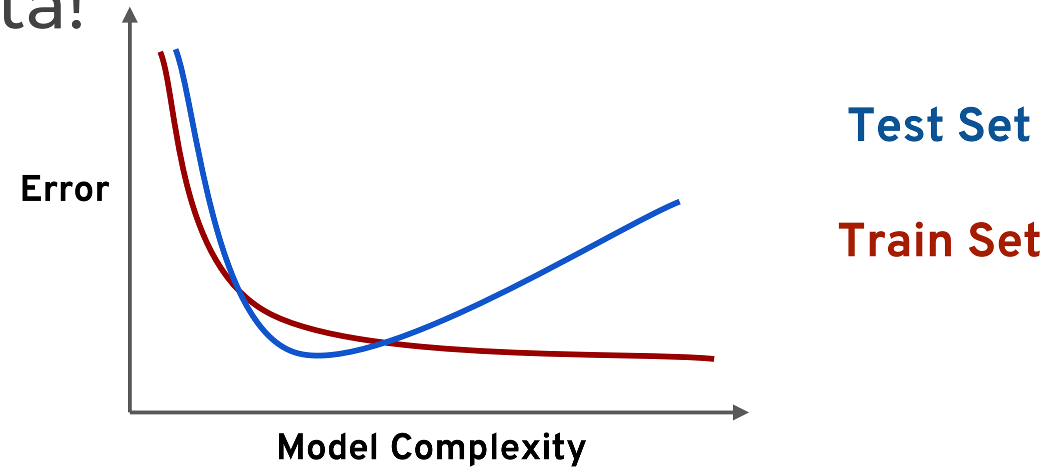
Overfitting and Underfitting

- But what happens if we overfit on the training data? That means we would perform poorly on new test data!



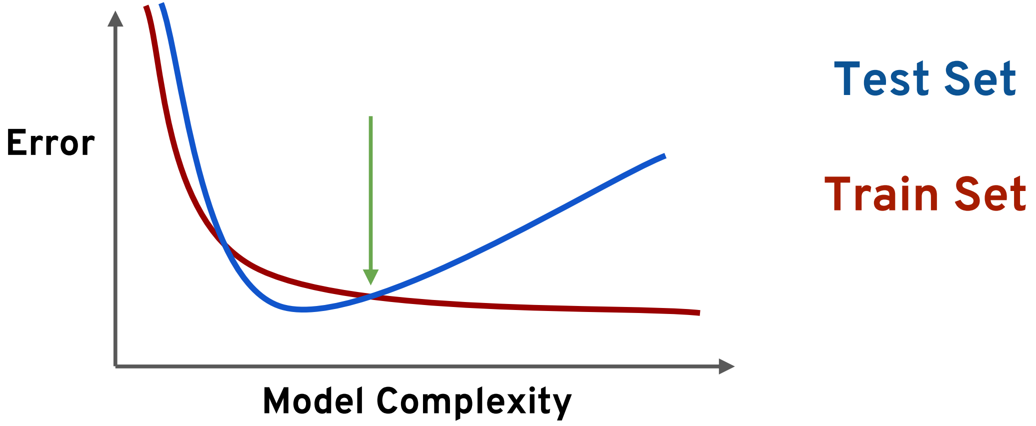
Overfitting and Underfitting

- But what happens if we overfit on the training data? That means we would perform poorly on new test data!



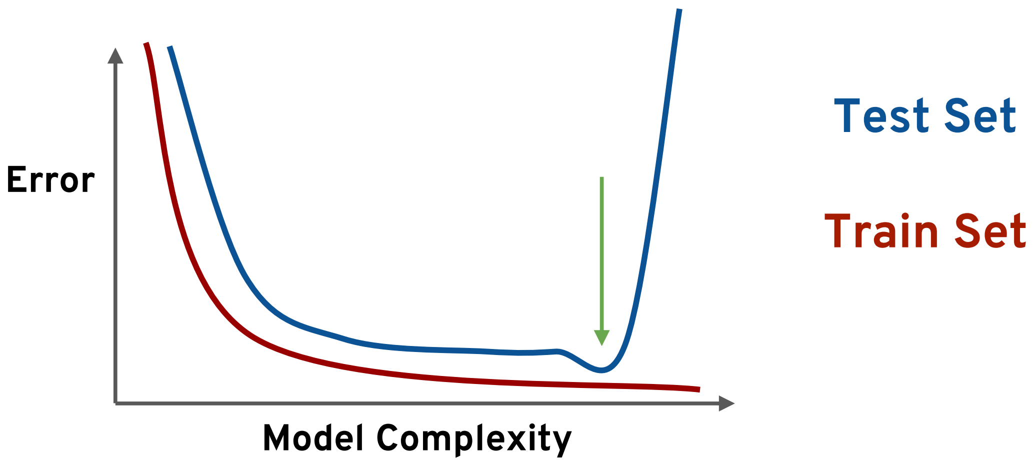
Overfitting and Underfitting

- This is a good indication too much complexity, you should look for the point to determine appropriate values!



Overfitting and Underfitting

- For certain algorithms this test error jump can be sudden instead of gradual.



Overfitting and Underfitting

- This means when deciding optimal model complexity **and** wanting to fairly evaluate our model’s performance, we can consider both the train error and test error to select an ideal complexity.

Overfitting and Underfitting

- In the case of Polynomial Regression, complexity directly relates to degree of the polynomial, but many machine learning algorithms have their own hyperparameters that can increase complexity.

Overfitting and Underfitting

- Let’s explore this further in the next lecture!

Polynomial Regression

Adjusting Model Parameters

Polynomial Regression

- Let's explore choosing the optimal model complexity (order of polynomial).
- As we previously discussed, we will need to understand error for both training and test data to look out for potential overfitting.

Polynomial Regression

Model Deployment

Regularization

Theory

PIERIAN DATA

Linear Regression

- Section

Regularization

Coding Implementations

Linear Regression

- Section

Regularization