Machine Learning : 06048203

Linear Regression

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

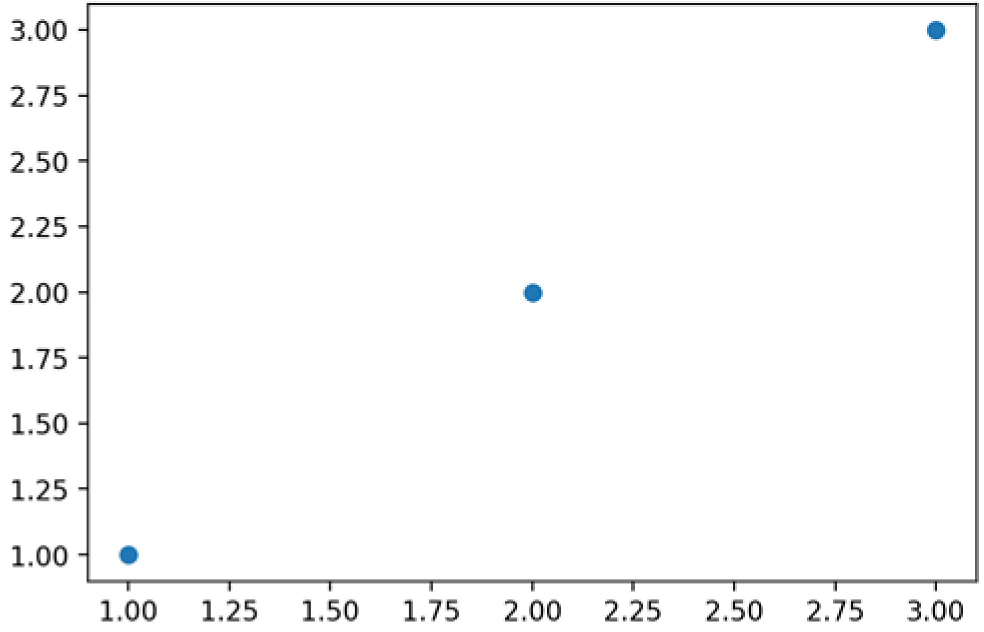
- This will include understanding:
  - Brief History
  - Linear Relationships
  - Ordinary Least Squares
  - Cost Functions
  - Gradient Descent
  - Vectorization

Linear Regression

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

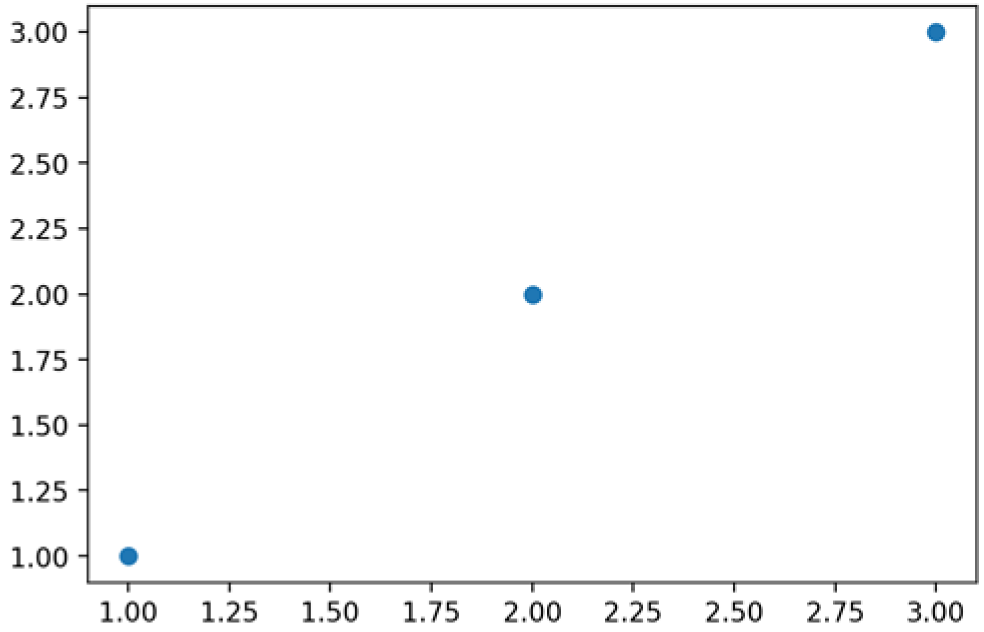
Linear Regression

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



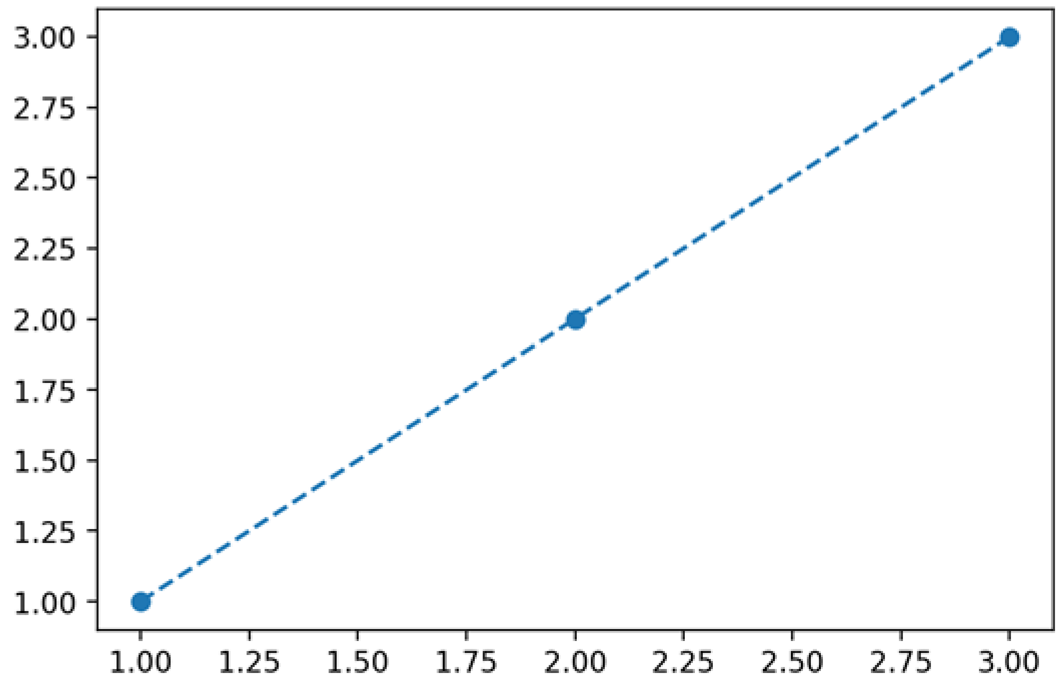
Linear Regression

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



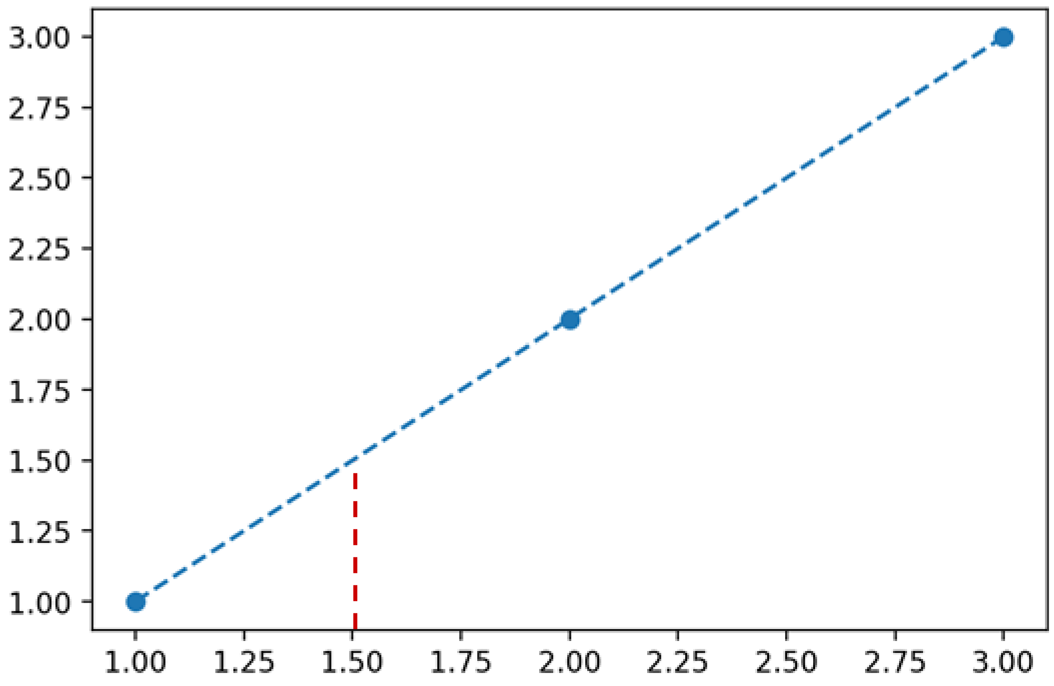
Linear Regression

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



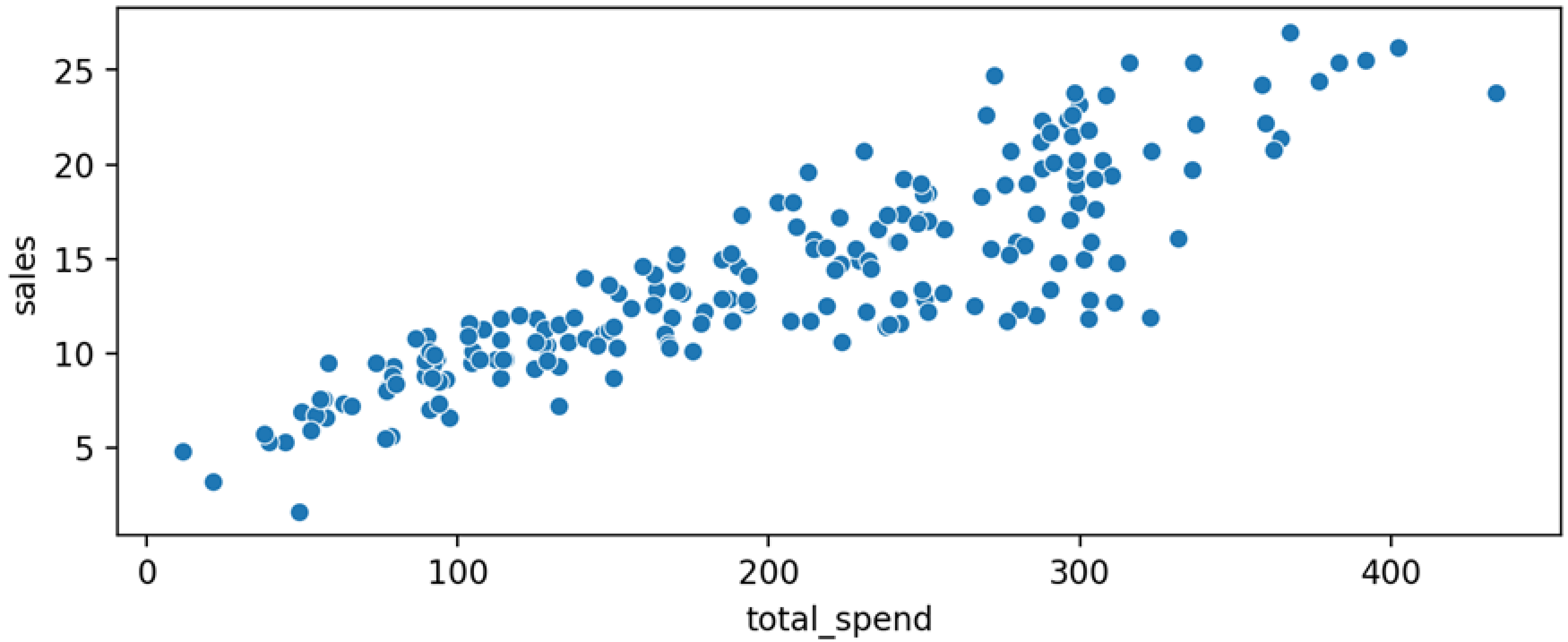
Linear Regression

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



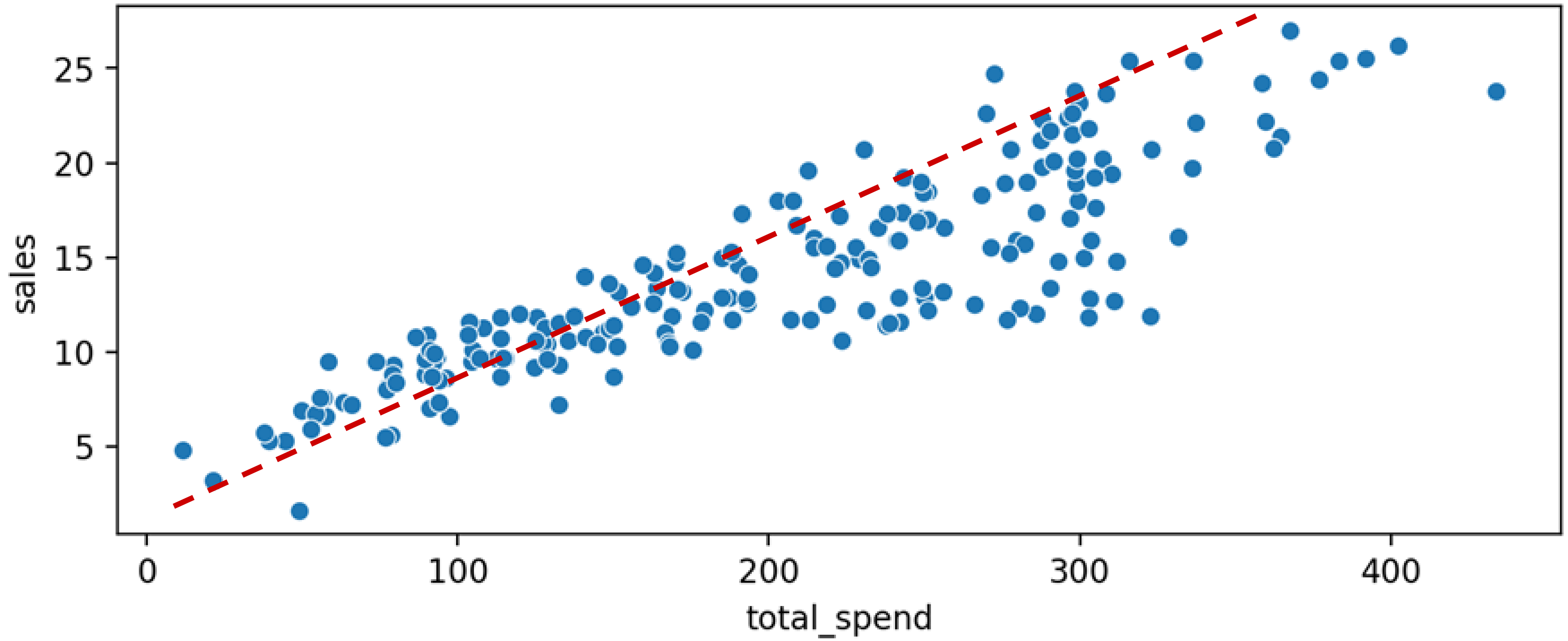
Linear Regression

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



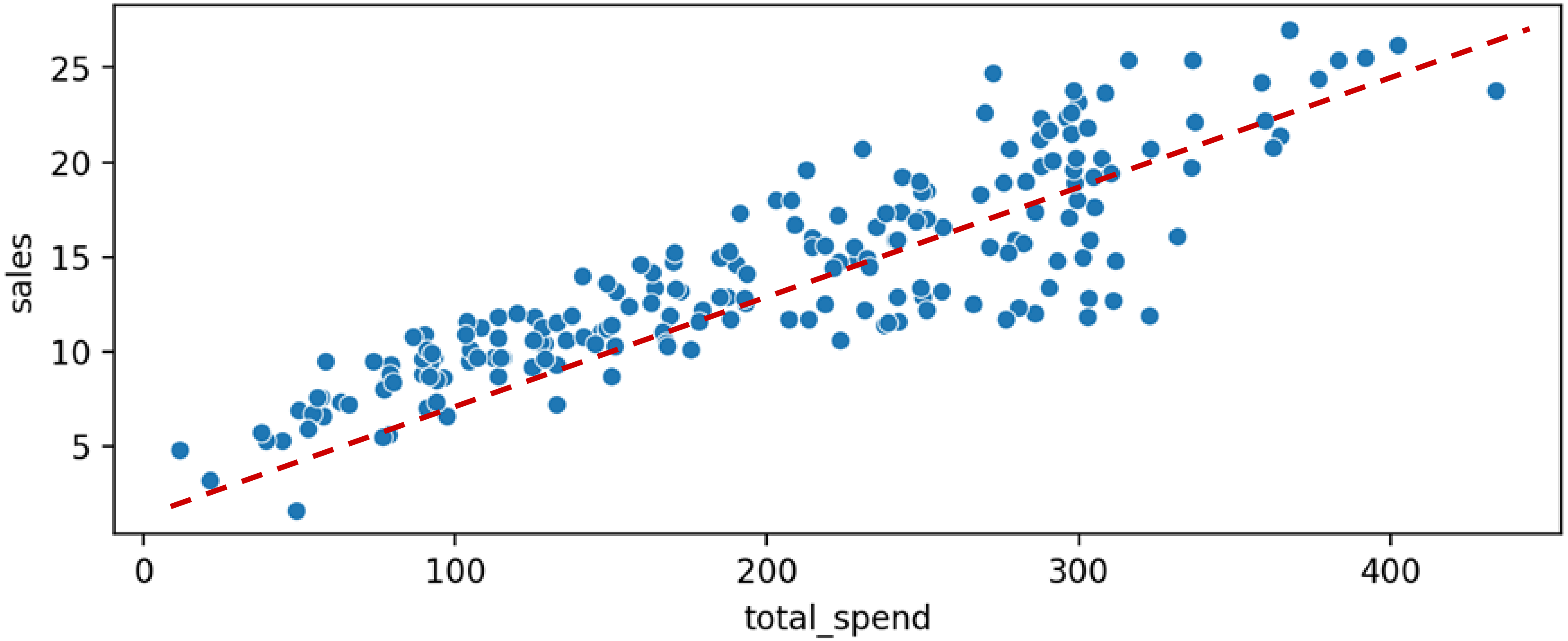
Linear Regression

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



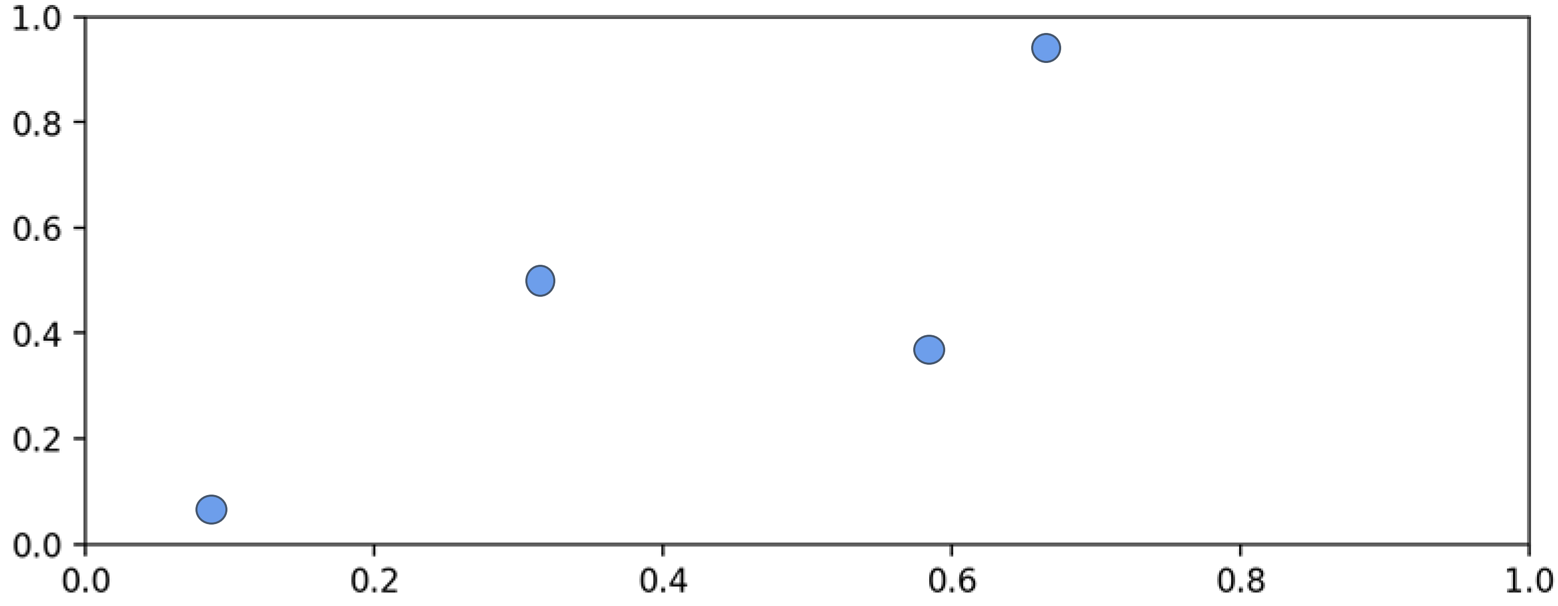
Linear Regression

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



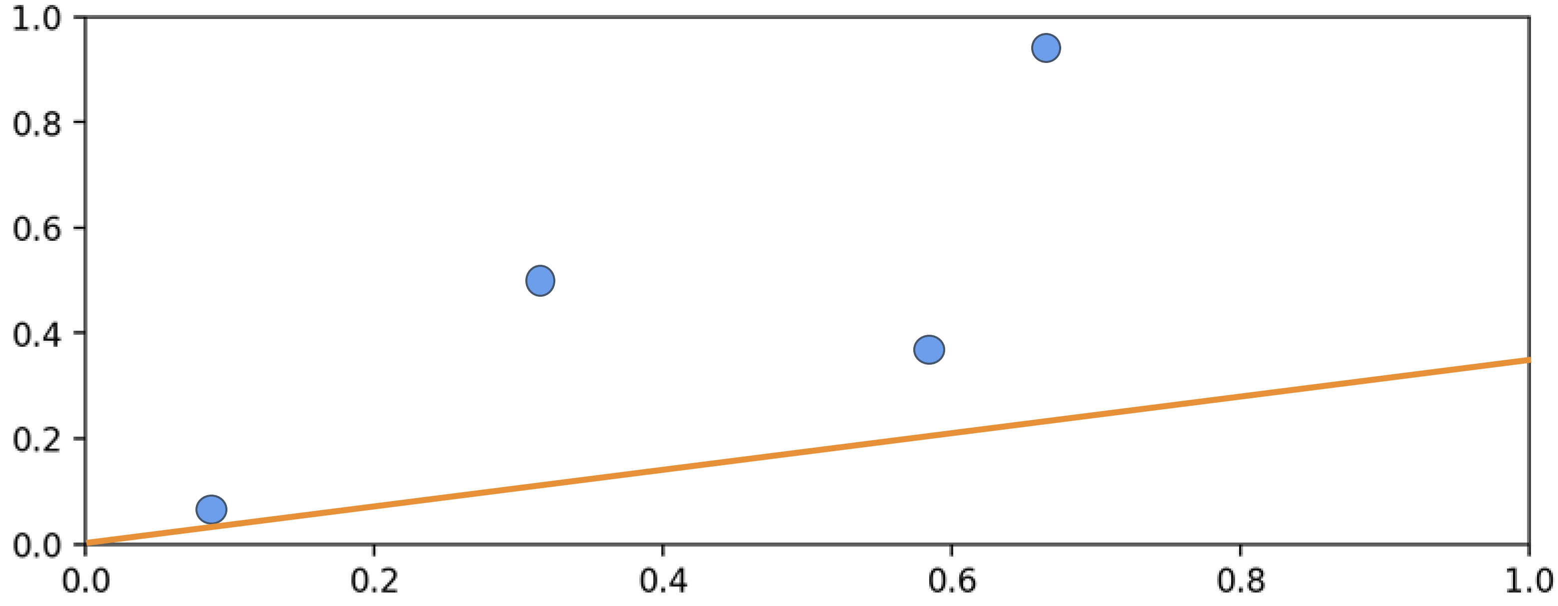
Linear Regression

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



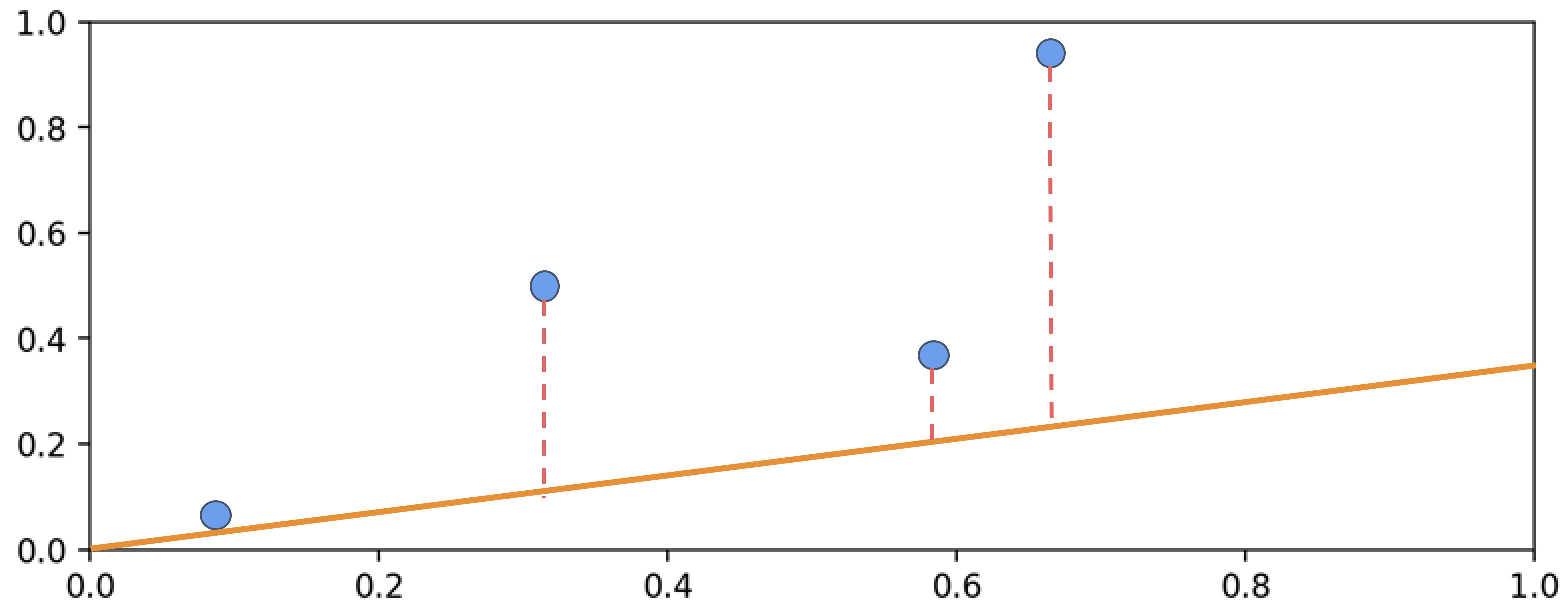
Linear Regression

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



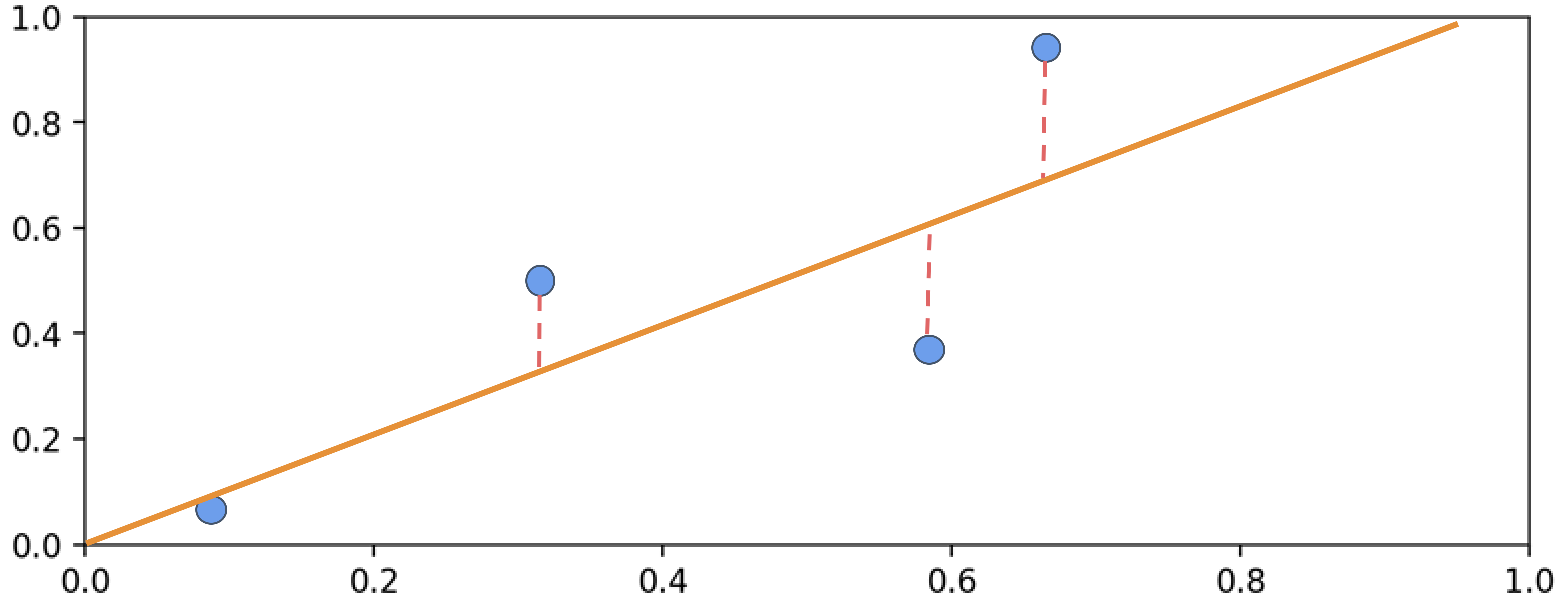
Linear Regression

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



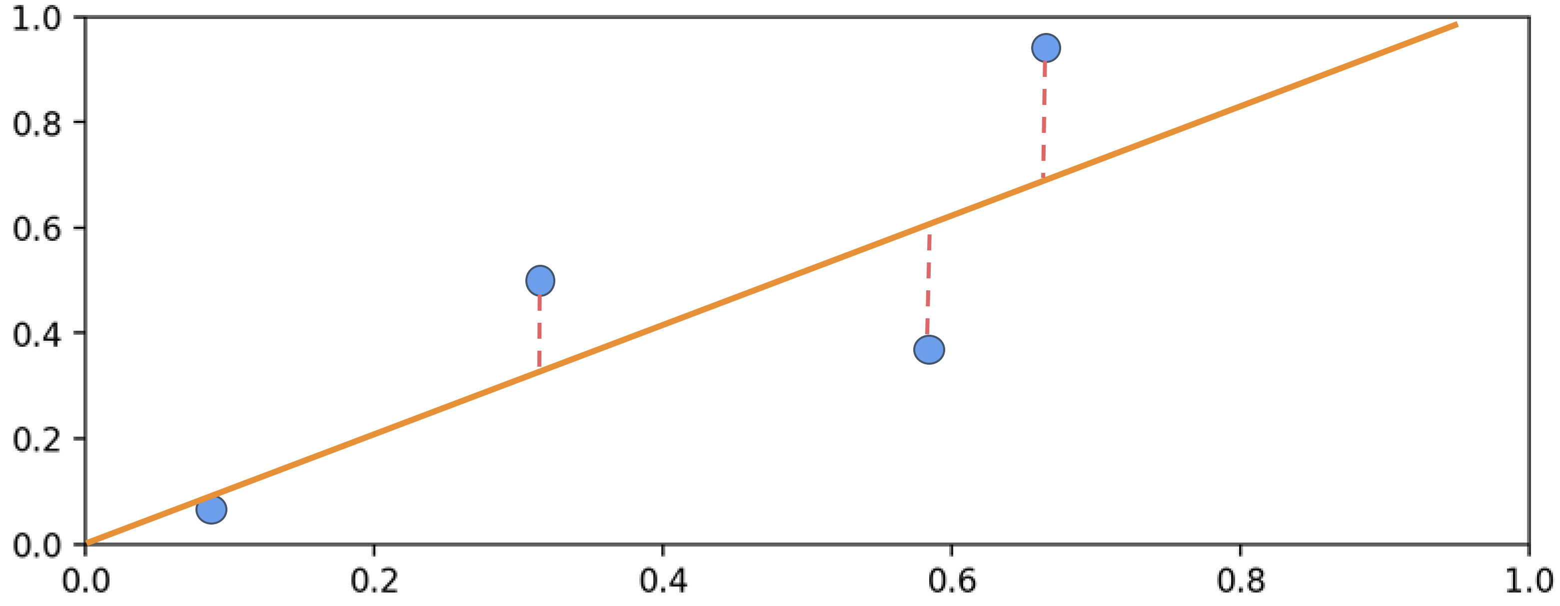
Linear Regression

- Some lines will clearly be better fits than others.



Linear Regression

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

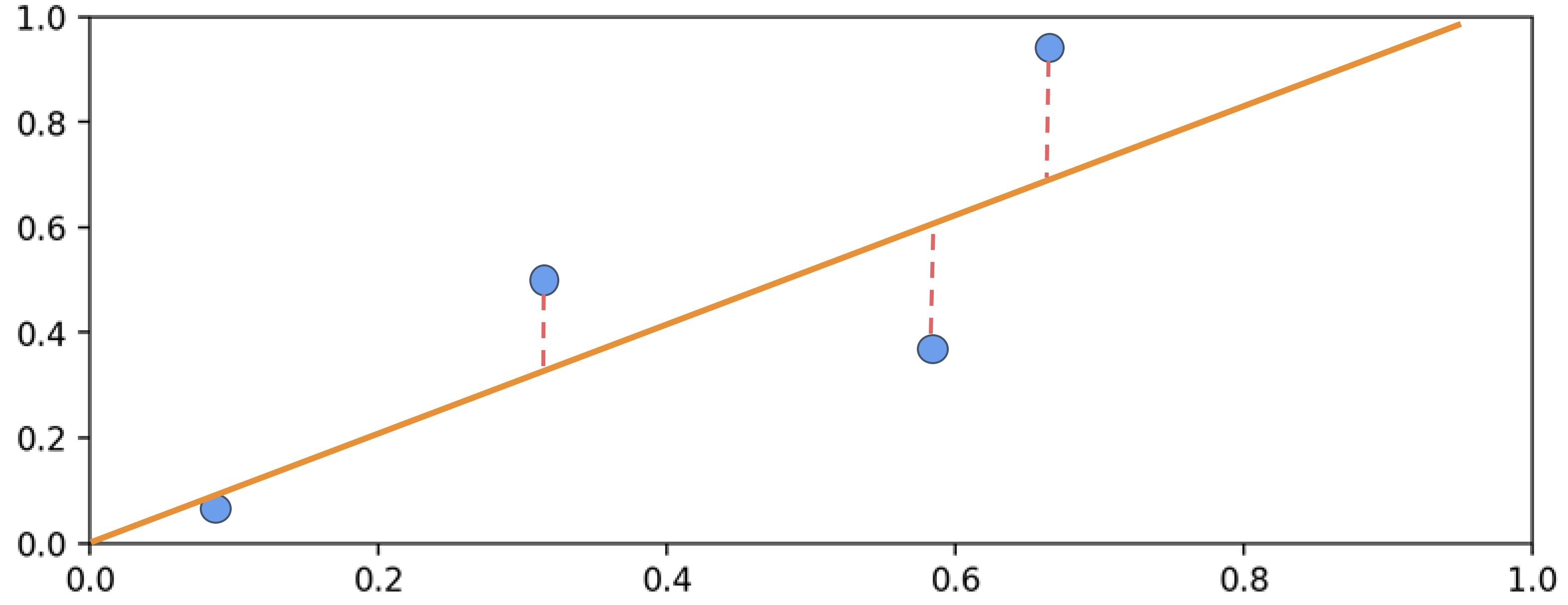


Linear Regression

- **Ordinary Least Squares** (OLS)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.

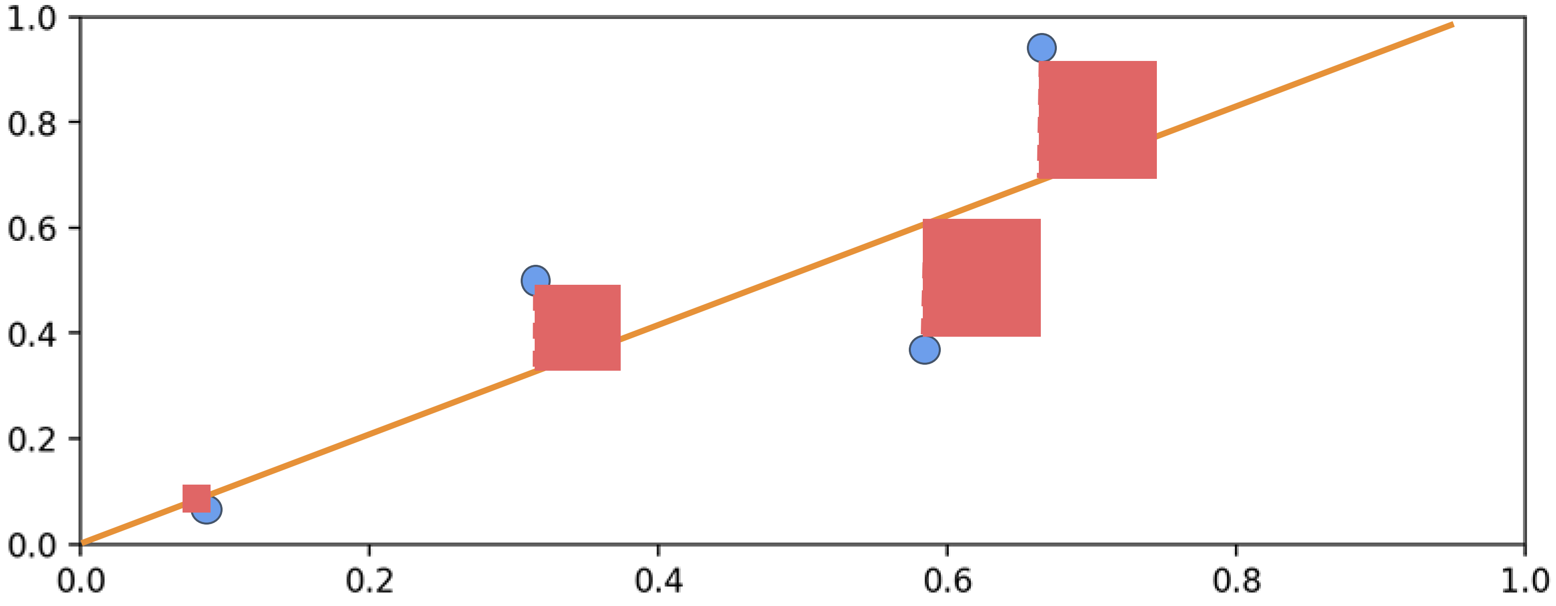
Linear Regression

- 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^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**

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

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

	X		**y**
$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	**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**	
$\mathbf{y}$	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$

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	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$

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

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	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$

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

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

Linear Regression

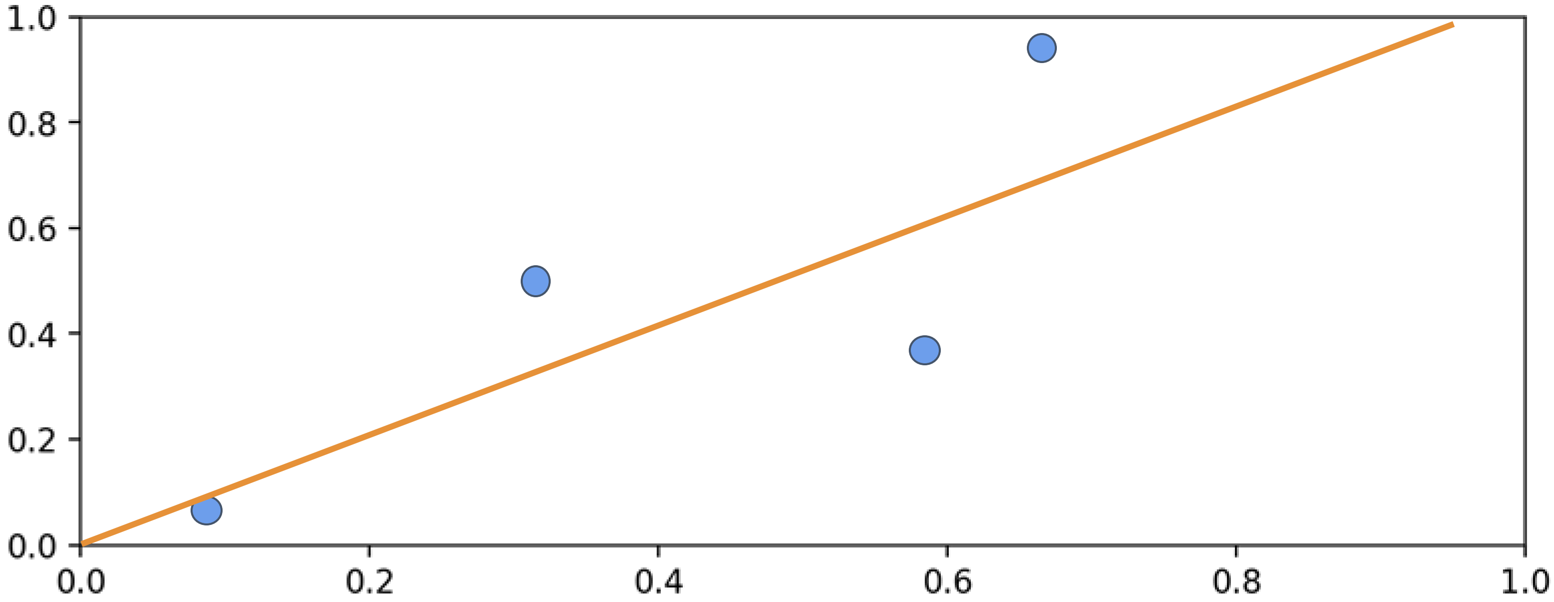
- Note the y hat symbol displays a prediction. There is usually no set of Betas to create a perfect fit to y!

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

Linear Regression

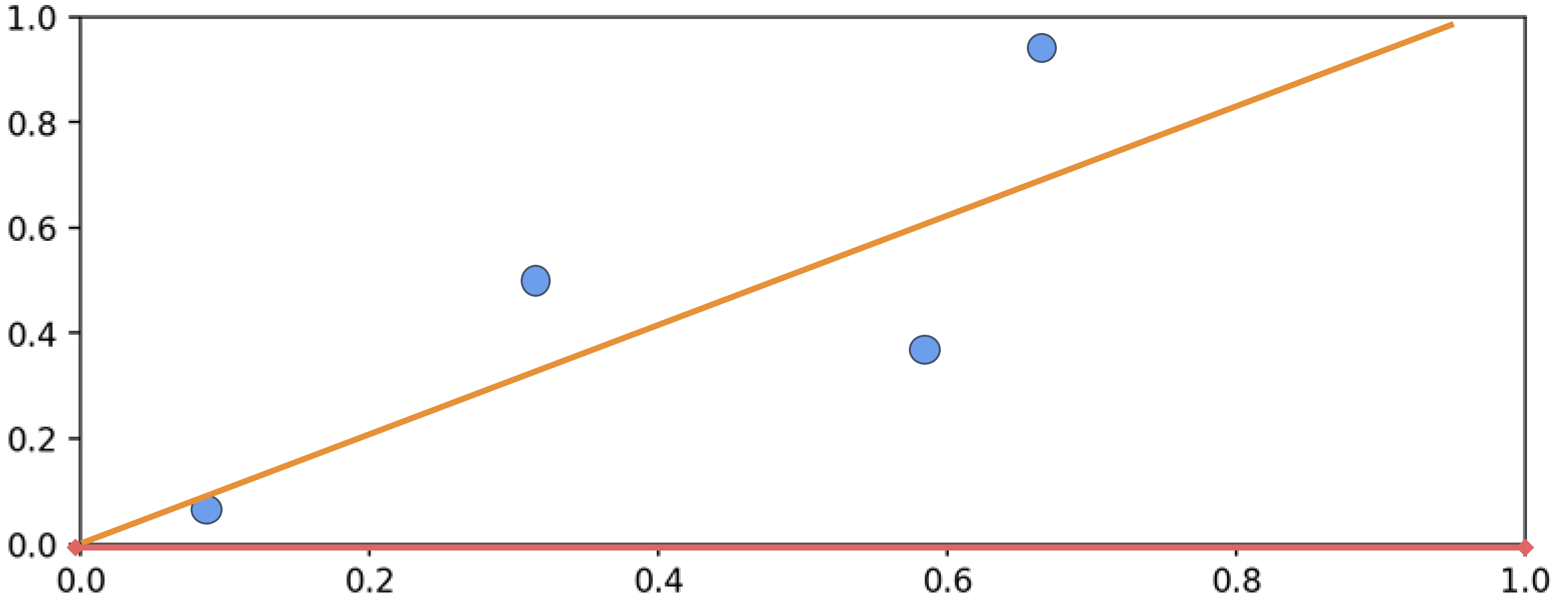
- Line equation:

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



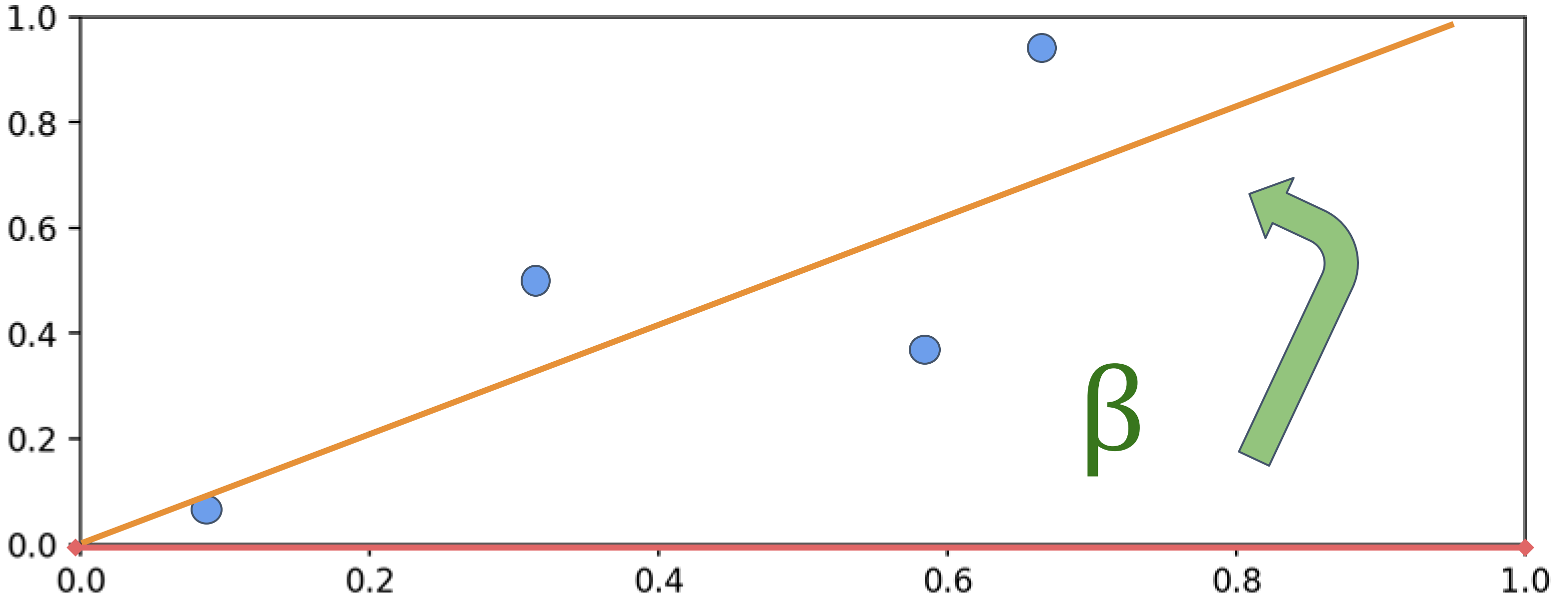
Linear Regression

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



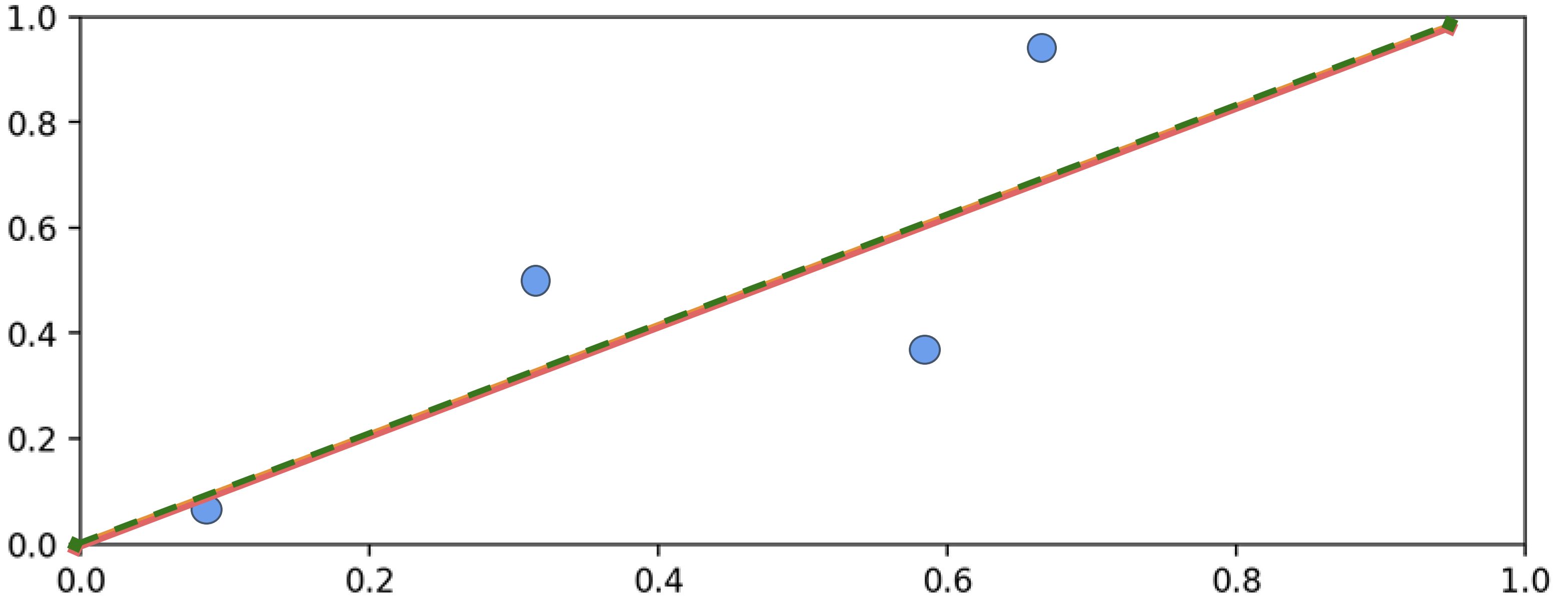
Linear Regression

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



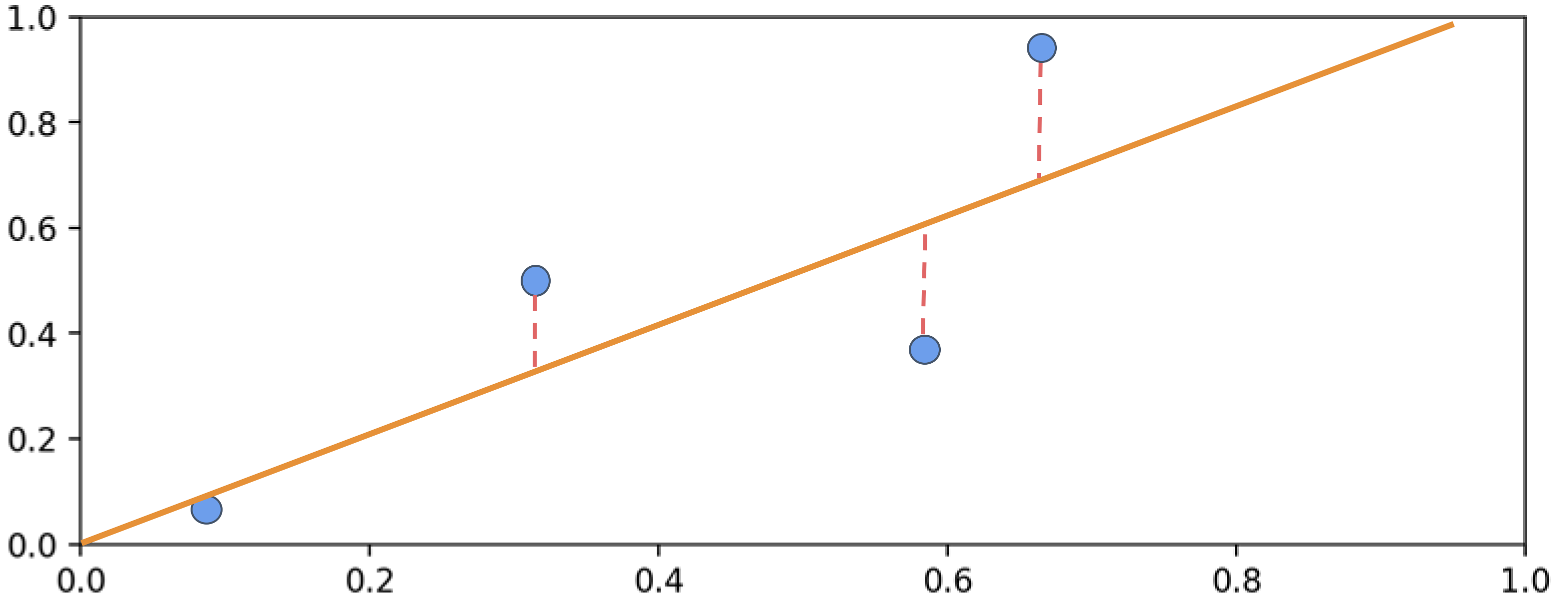
Linear Regression

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



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!

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

Introduction to Linear Regression

Cost Function

Linear Regression

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

Linear Regression

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

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

Linear Regression

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

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

Linear Regression

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

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

Linear Regression

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

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

Linear Regression

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

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

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

- 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

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

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

Linear Regression

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

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

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

Linear Regression

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

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

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

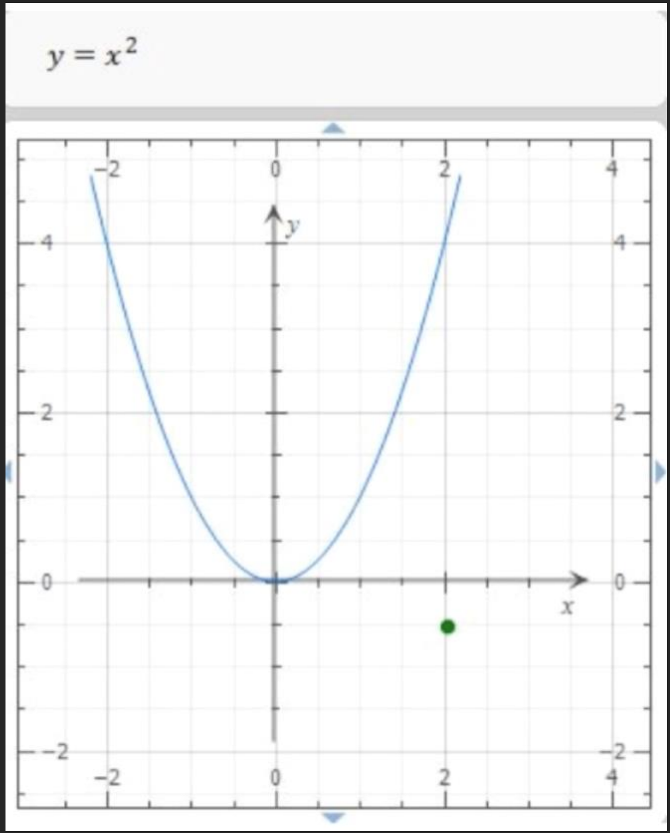
Linear Regression

- 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

Gradient Descent

Gradient Descent



Minimize $f(x) = x^2$

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

Gradient descent Linear Regression

- **Title**: Linear Regression with Gradient Descent
- **Content**:
  - Linear regression aims to fit a line to a set of data points.
  - Equation: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$
  - 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:
    $h_\beta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$
  - 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:
    - α is the learning rate.
    - The partial derivative represents the gradient of the cost function with respect to the parameter β_j.

Linear Regression

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

Linear Regression

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

Linear Regression

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

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

Linear Regression

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

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

Linear Regression

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

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

Linear Regression

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

Linear Regression

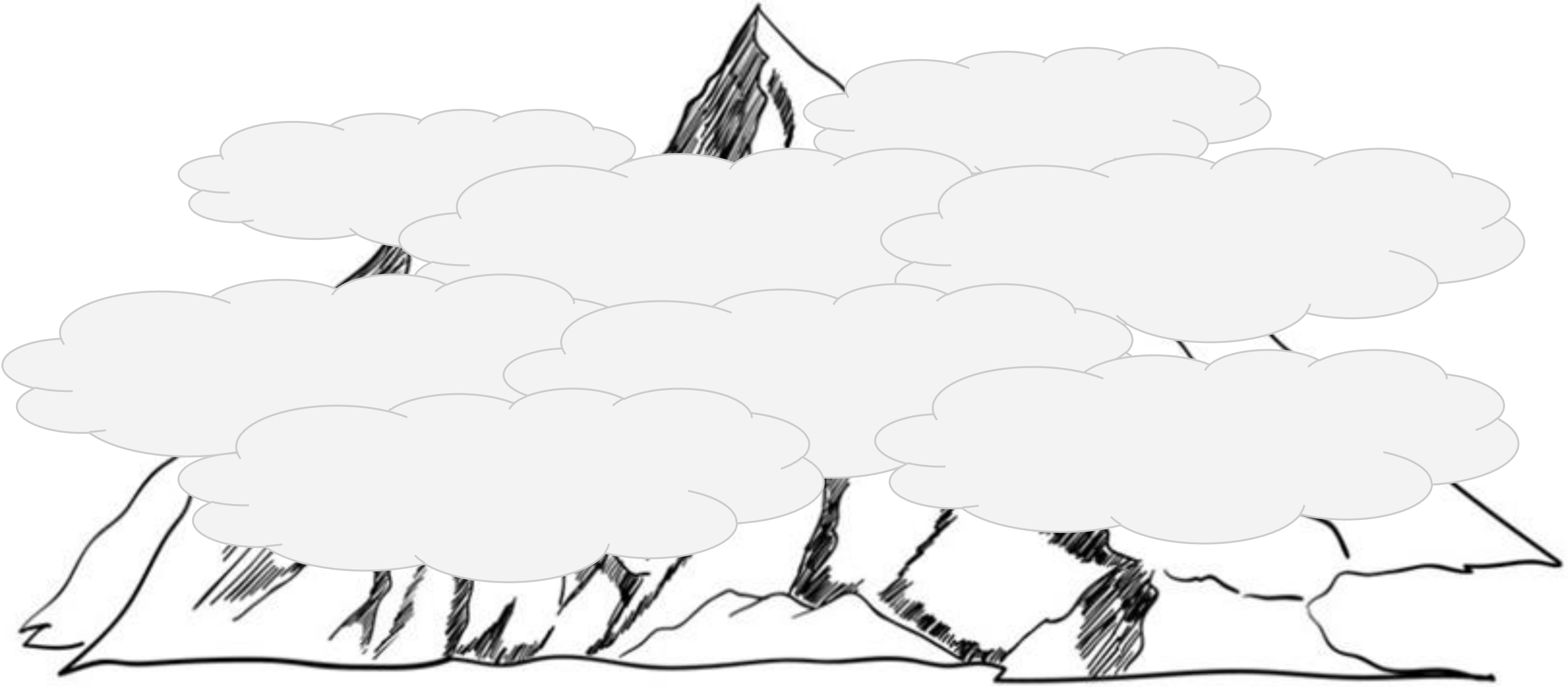
- Common mountain analogy

Linear Regression

- Common mountain analogy

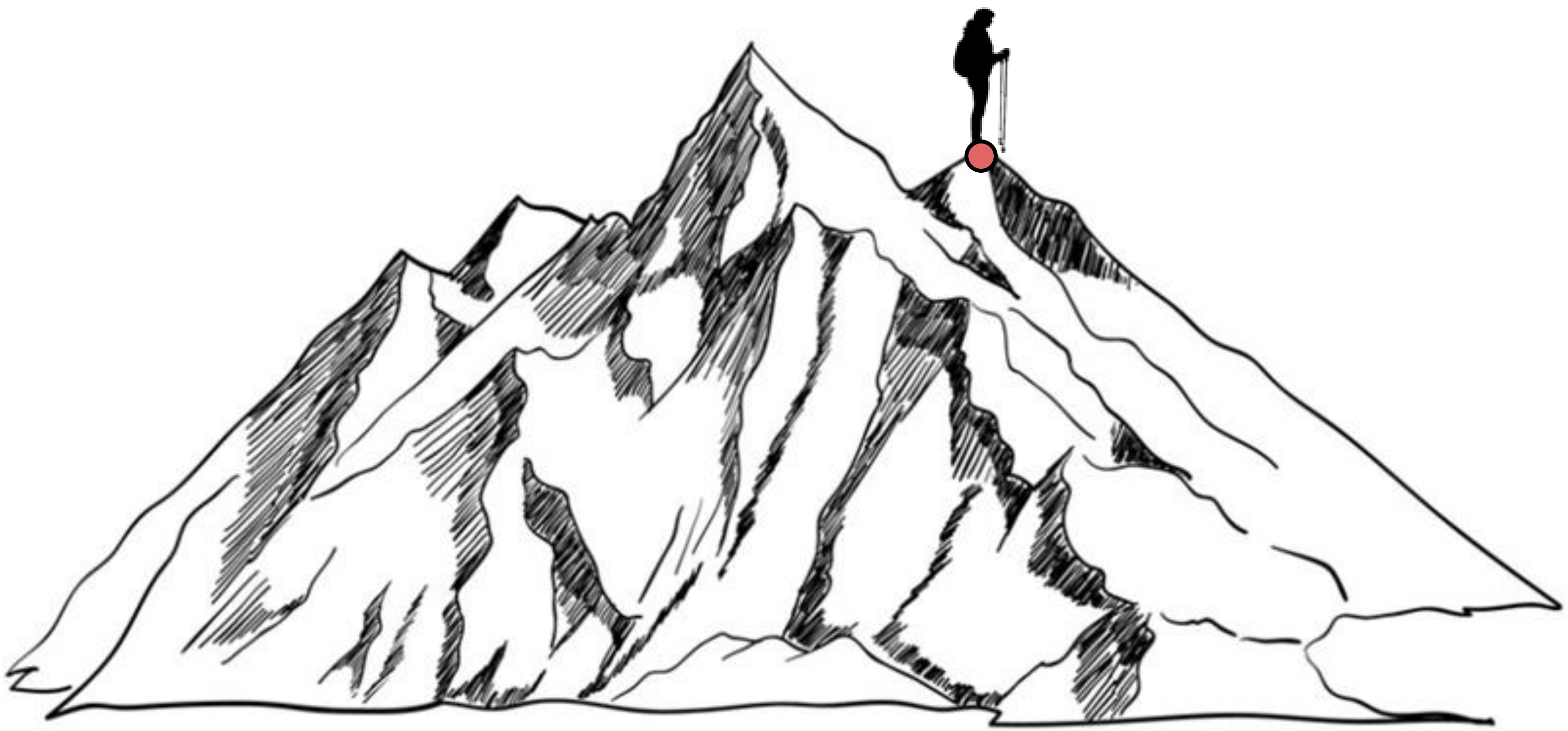
Linear Regression

- Common mountain analogy



Linear Regression

- Common mountain analogy

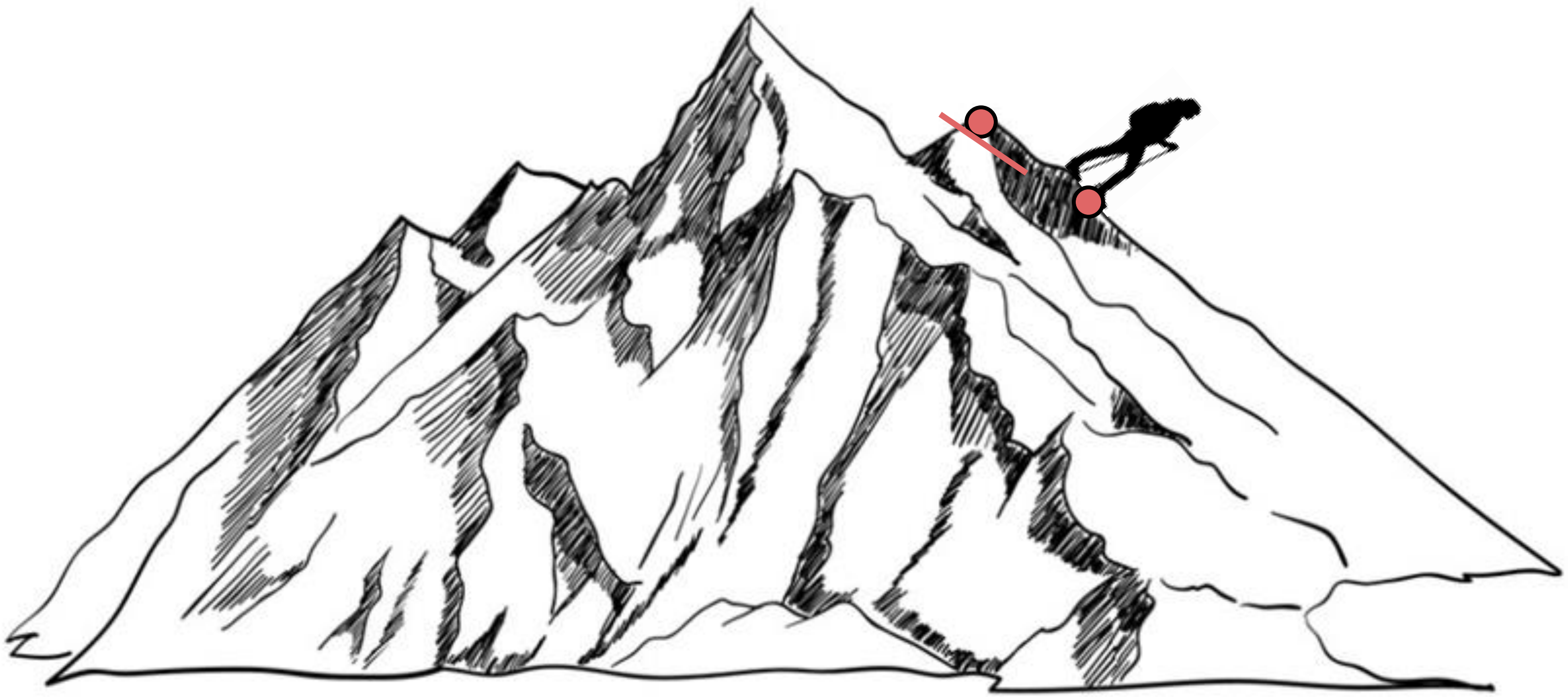


Linear Regression

- Common mountain analogy

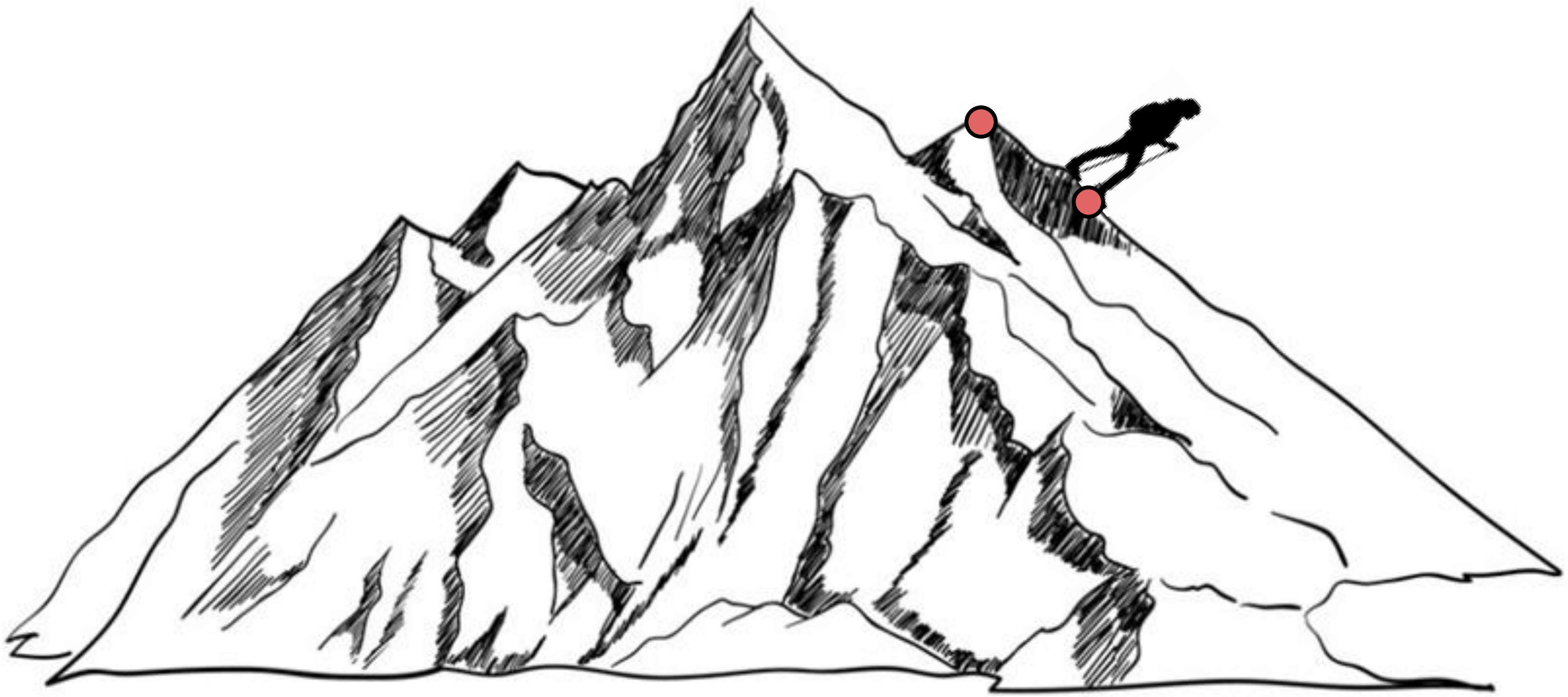
Linear Regression

- Common mountain analogy



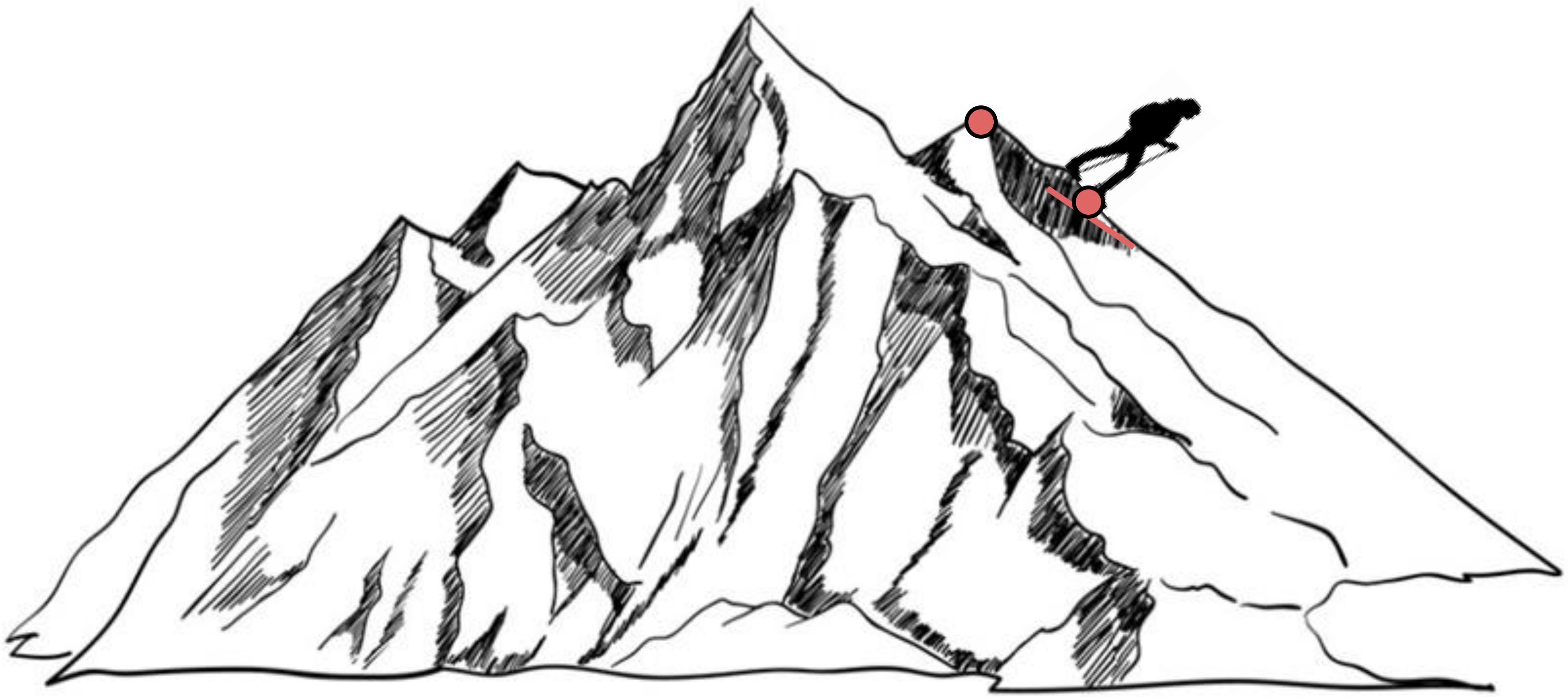
Linear Regression

- Common mountain analogy



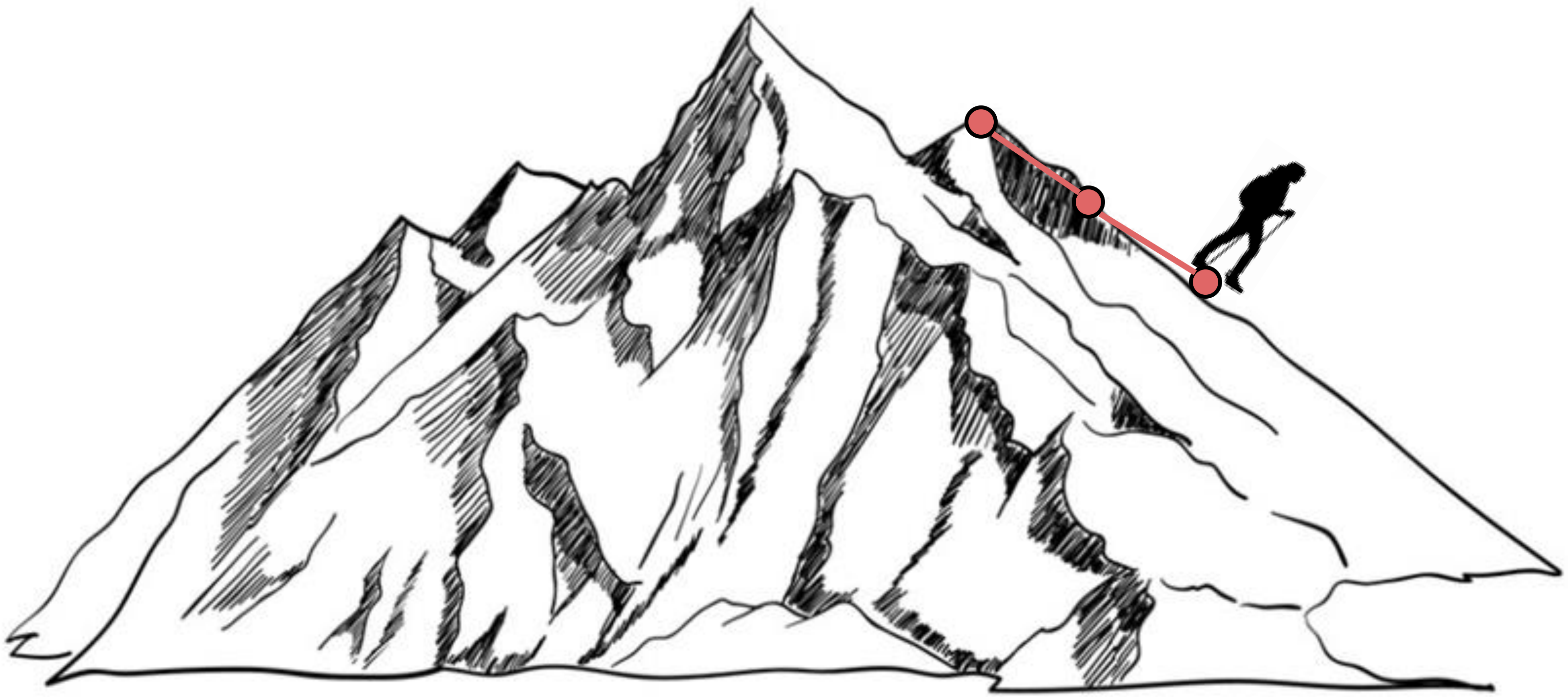
Linear Regression

- Common mountain analogy



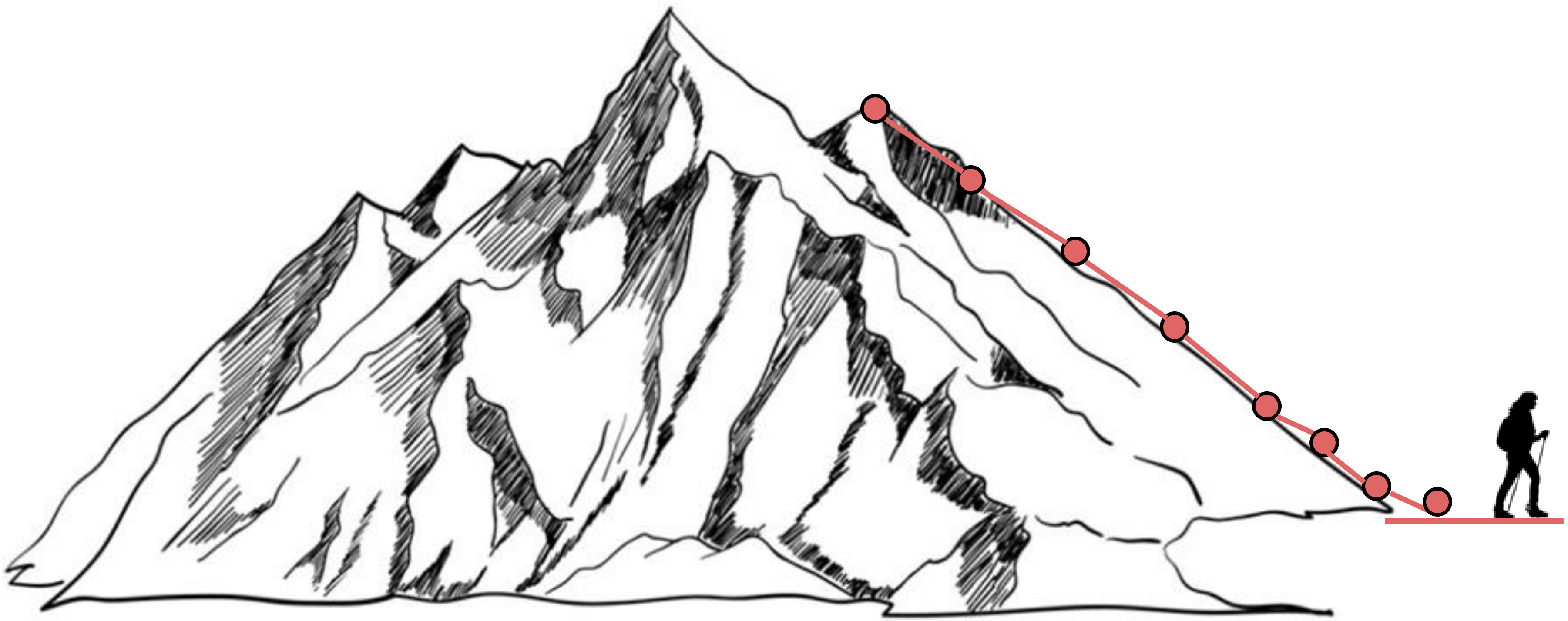
Linear Regression

- Common mountain analogy



Linear Regression

- Common mountain analogy

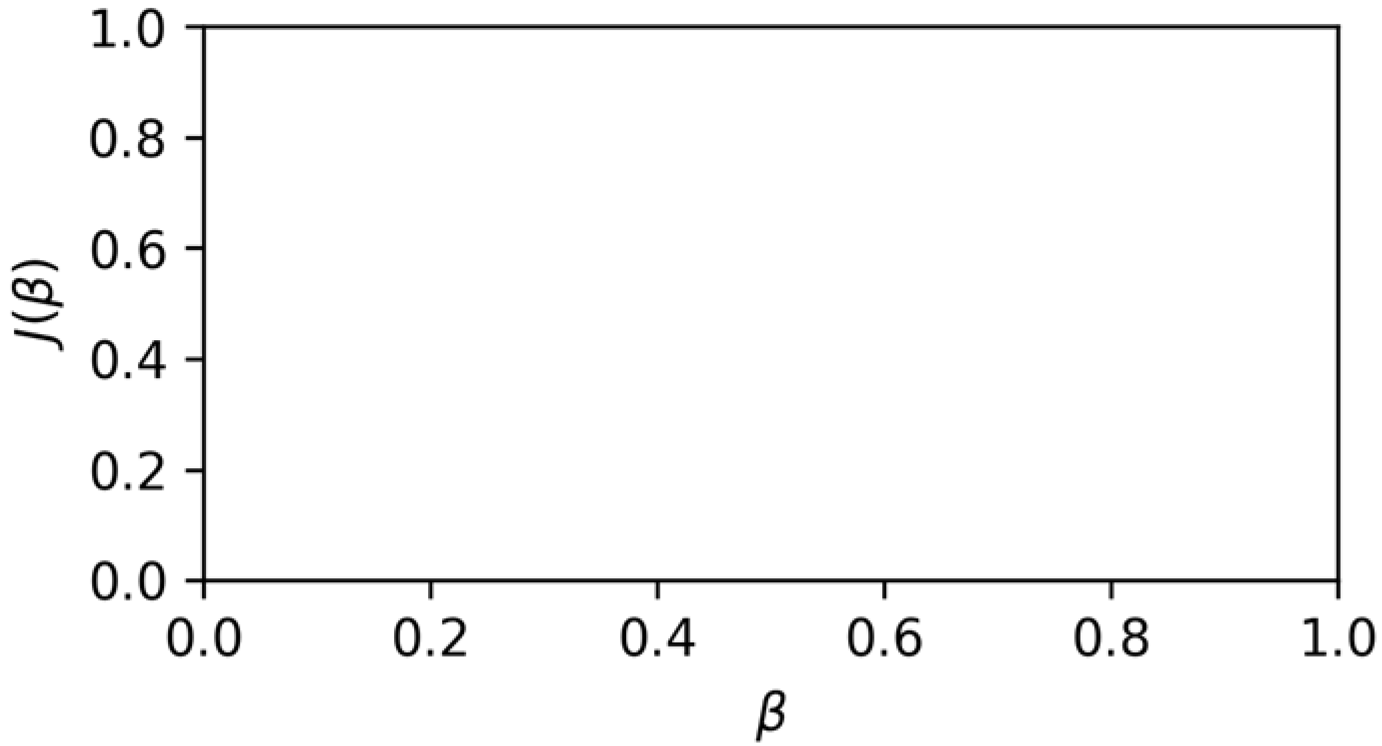


Linear Regression

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

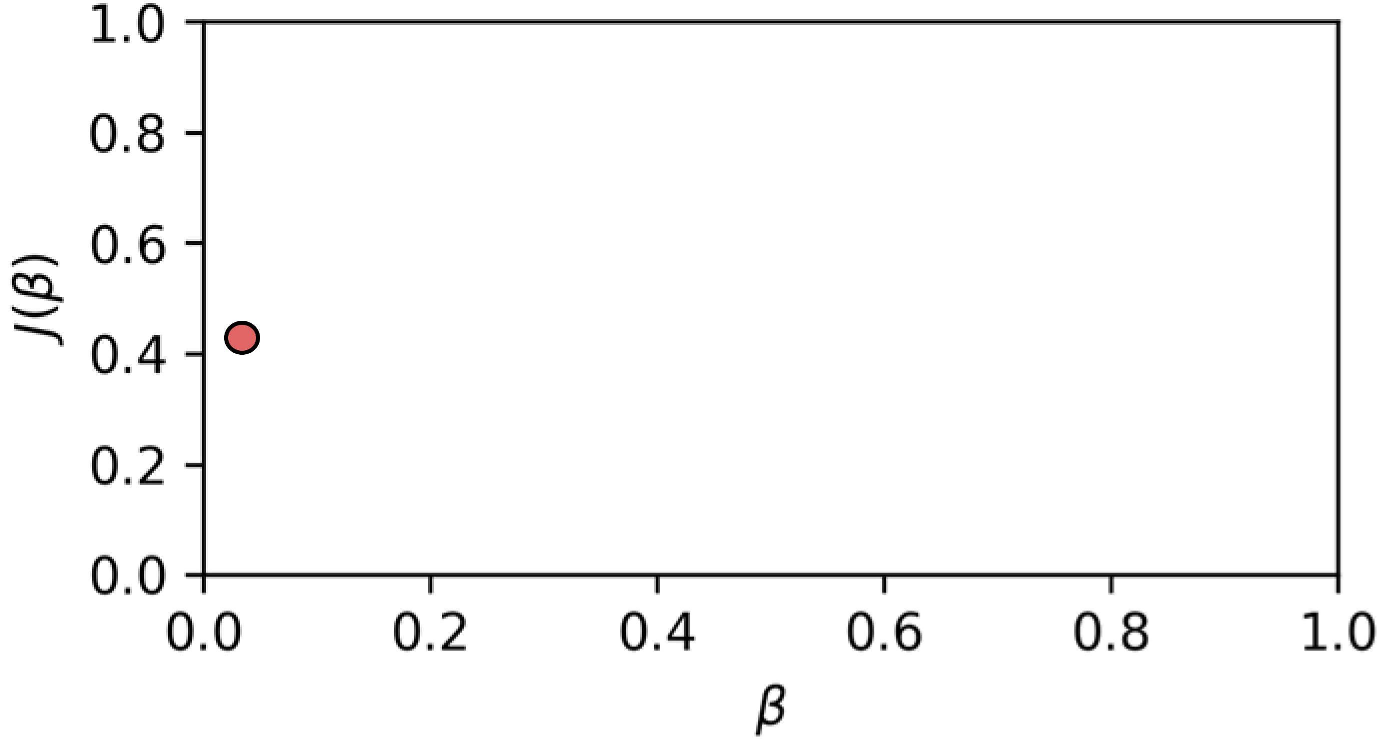
Linear Regression

- 1 dimensional cost function (single Beta)



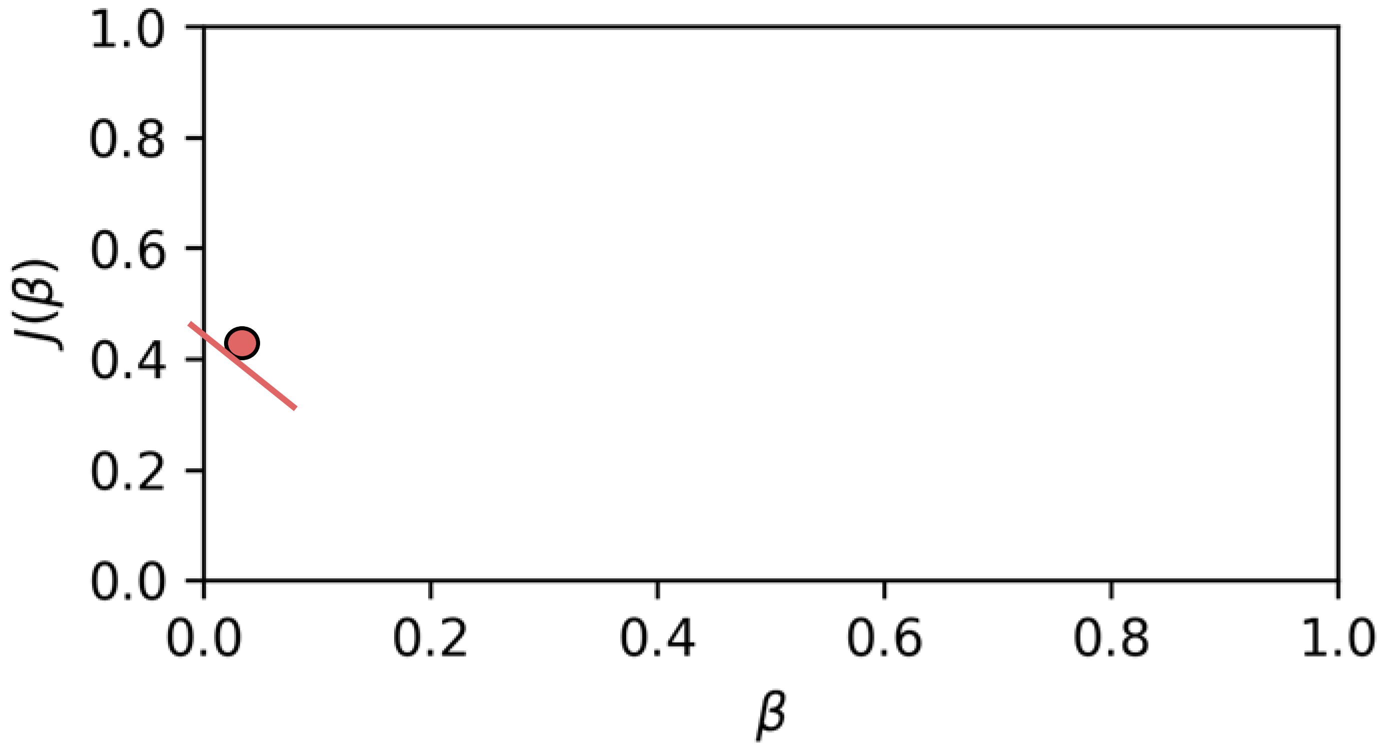
Linear Regression

- Choose a starting point



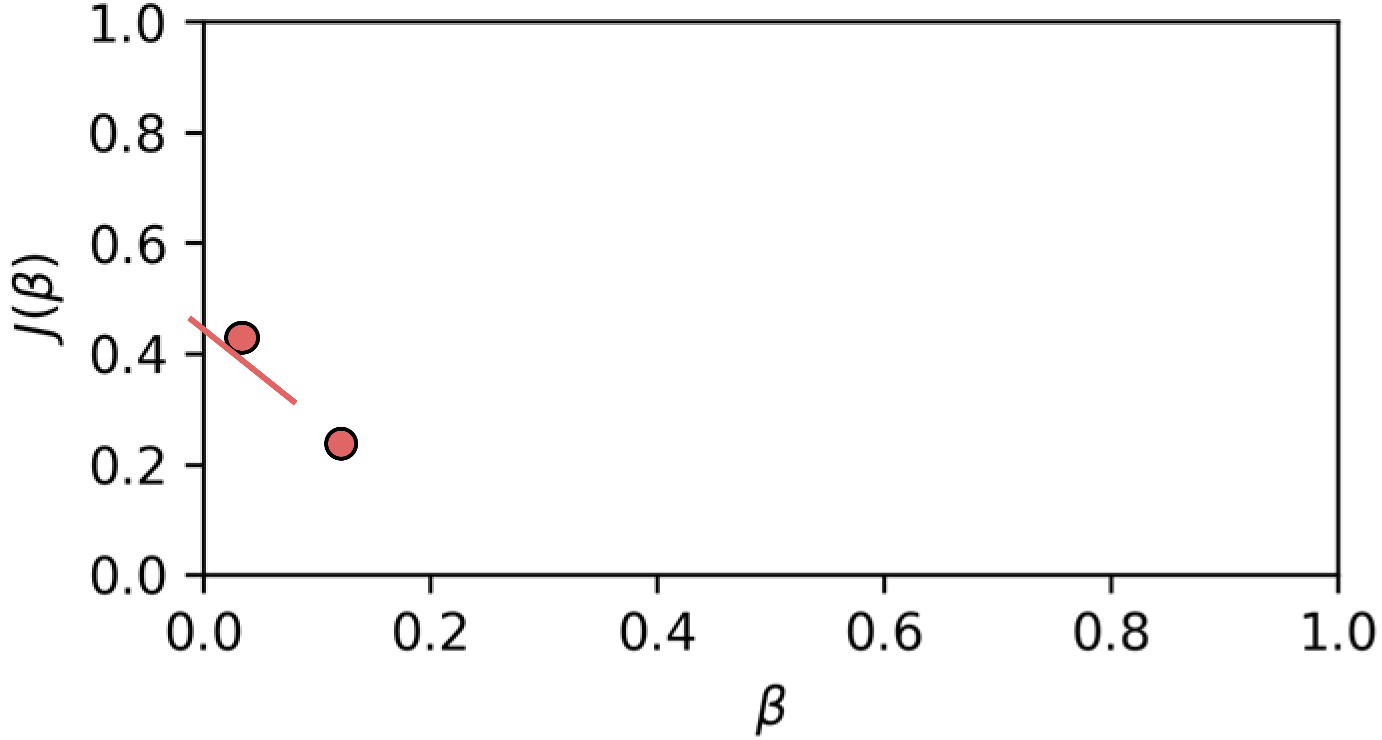
Linear Regression

- Calculate gradient at that point



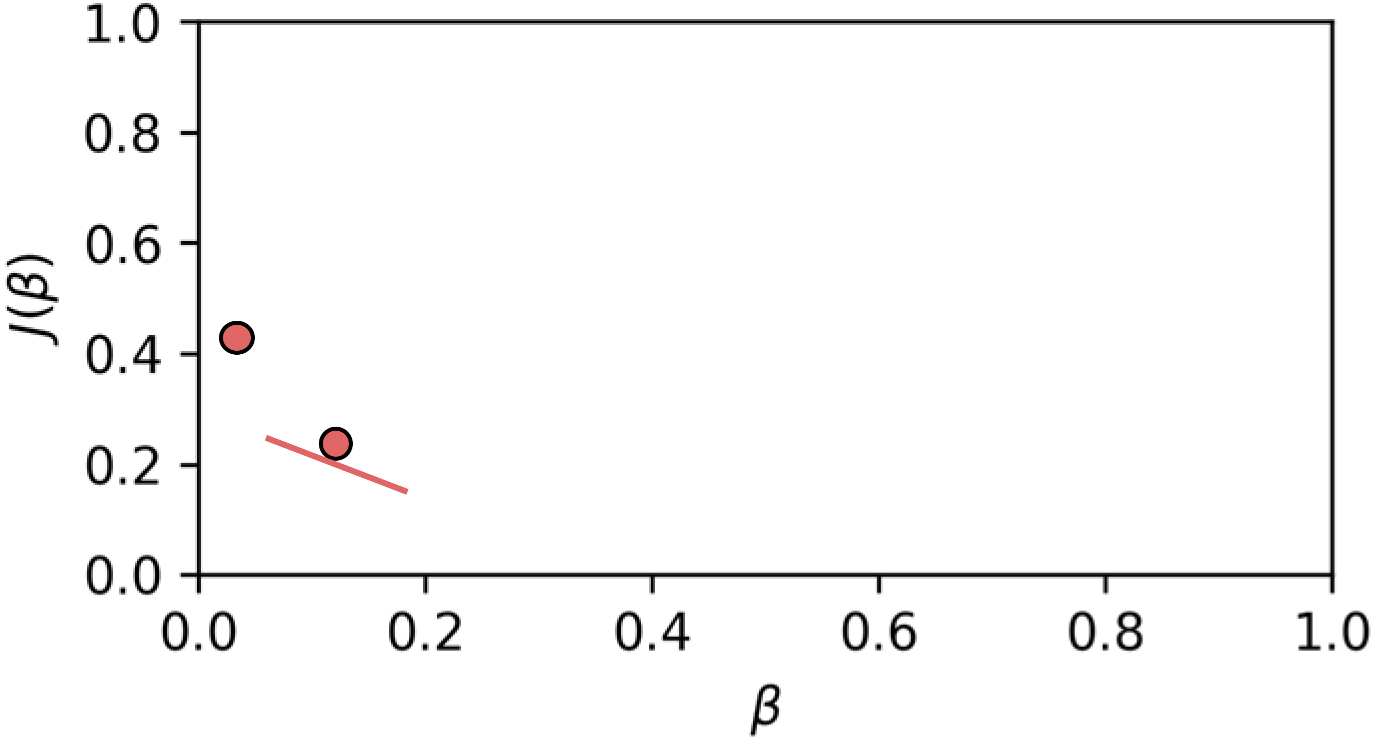
Linear Regression

- Step forward proportional to **negative** gradient



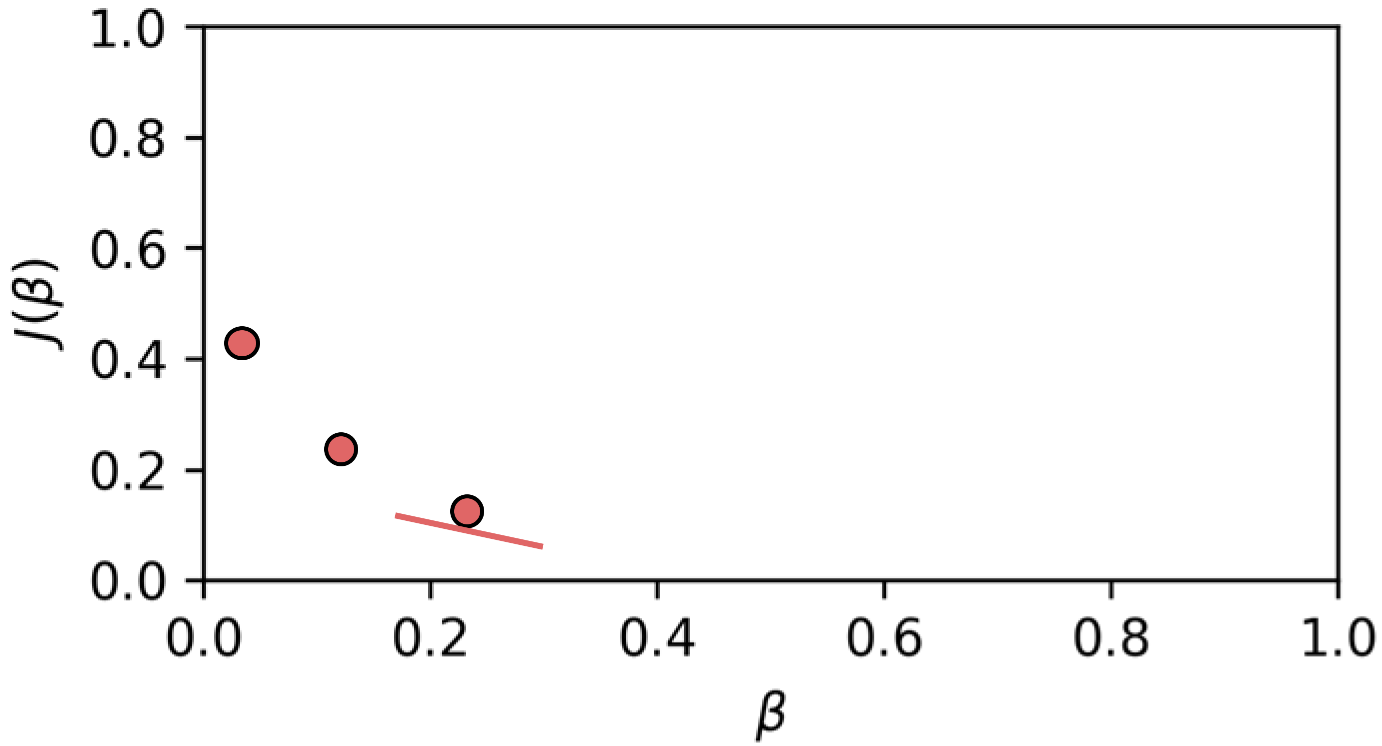
Linear Regression

- Repeat the steps



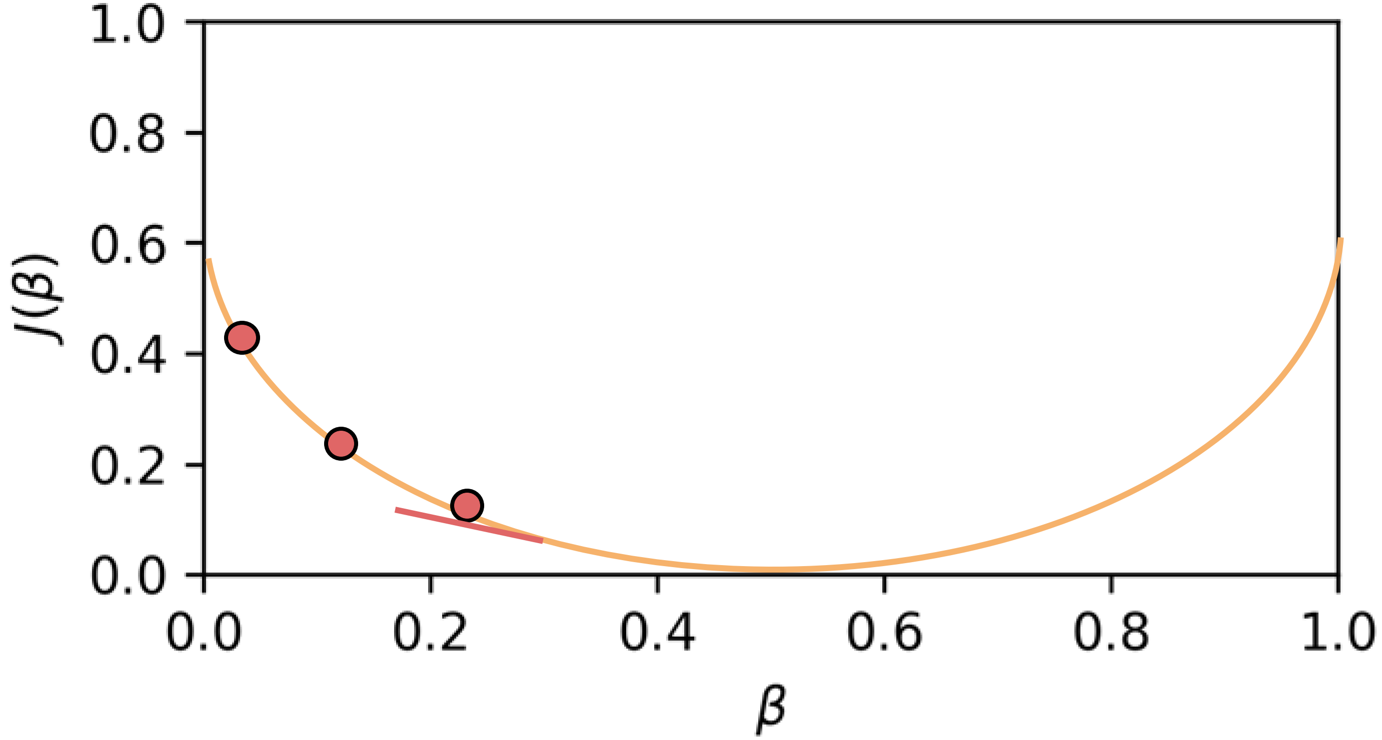
Linear Regression

- Repeat the steps



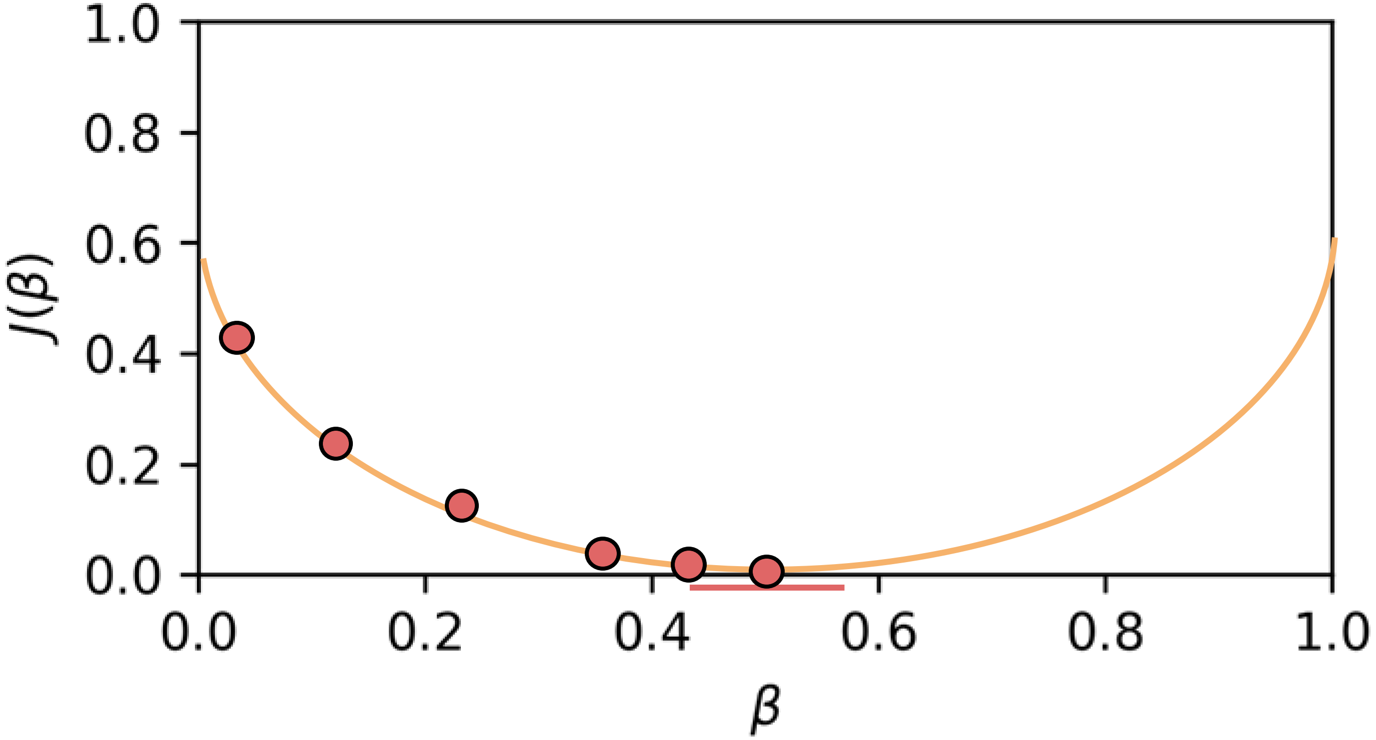
Linear Regression

- Note how we are essentially mapping the gradient!



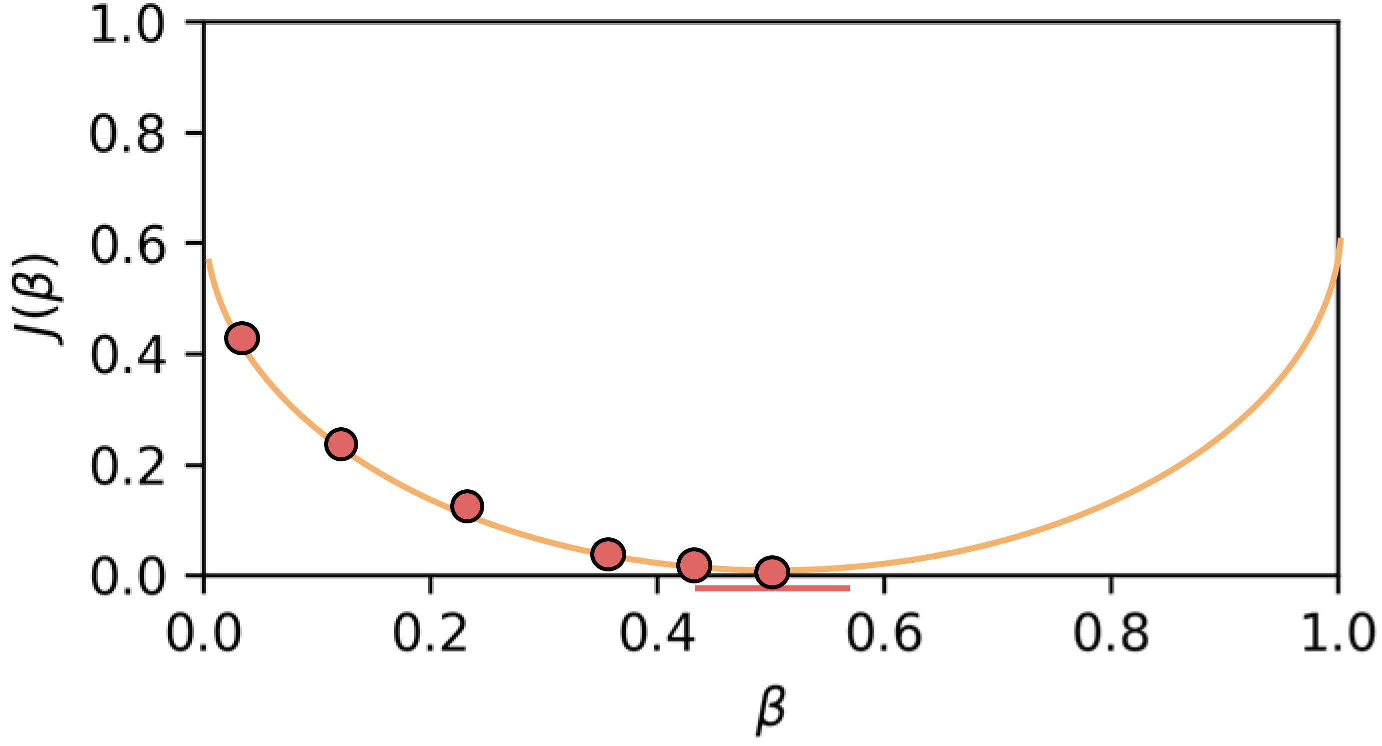
Linear Regression

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



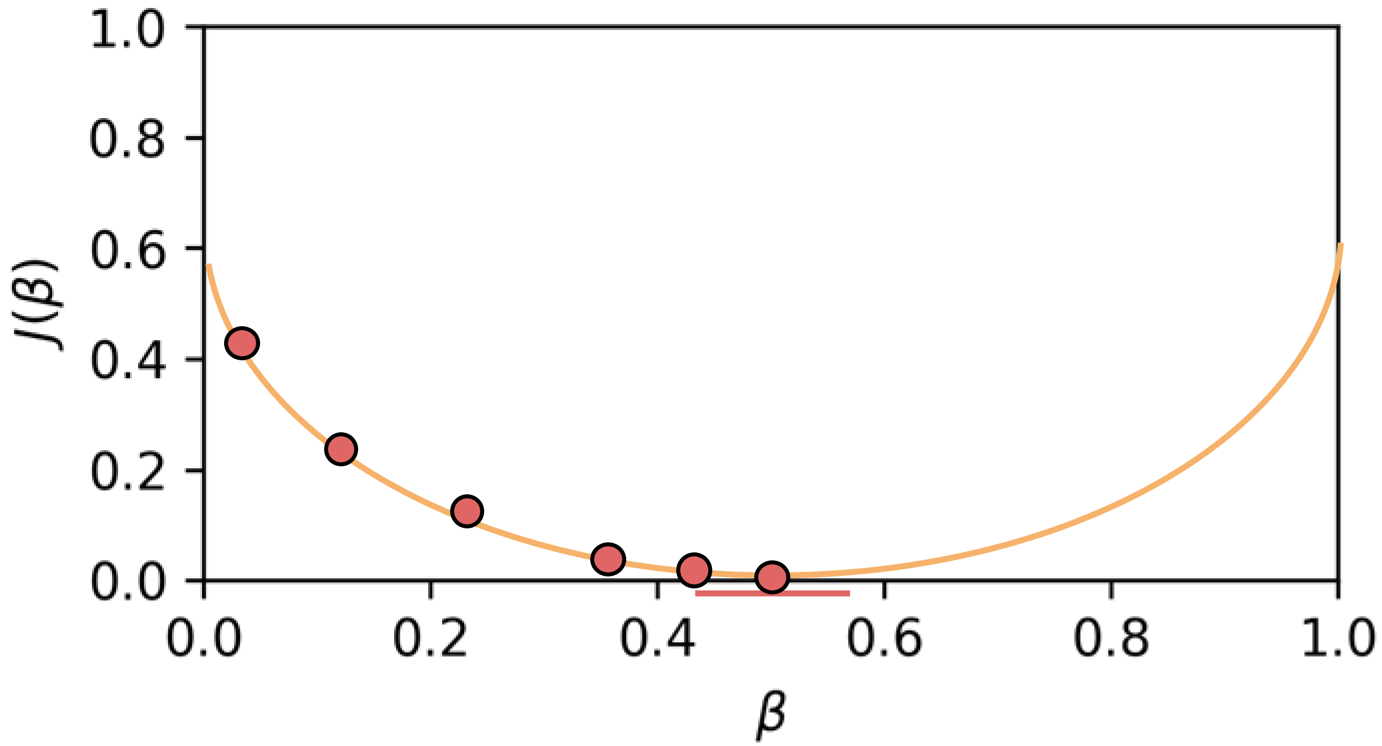
Linear Regression

- Steps are proportional to negative gradient!



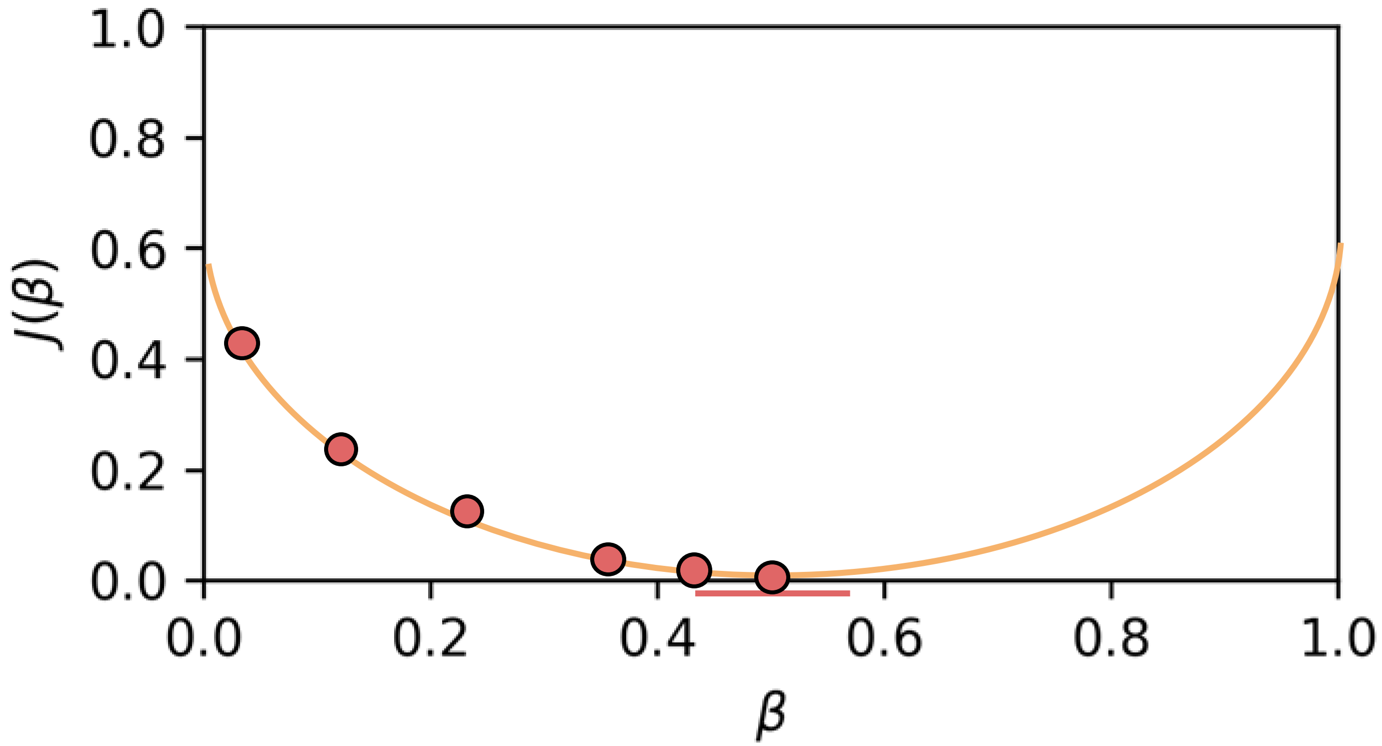
Linear Regression

- Steeper gradient at start gives larger steps.



Linear Regression

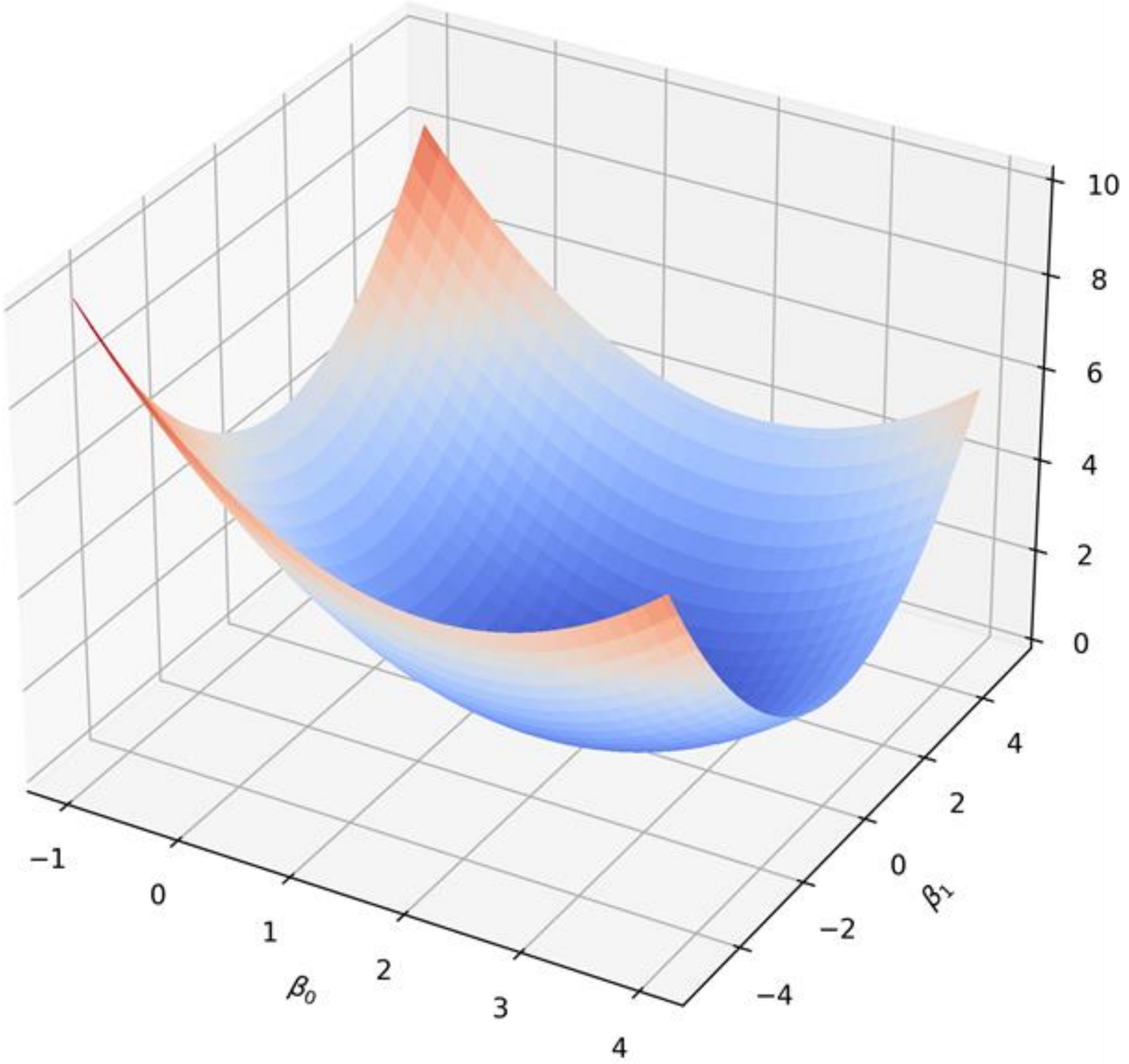
- Smaller gradient at end gives smaller steps.



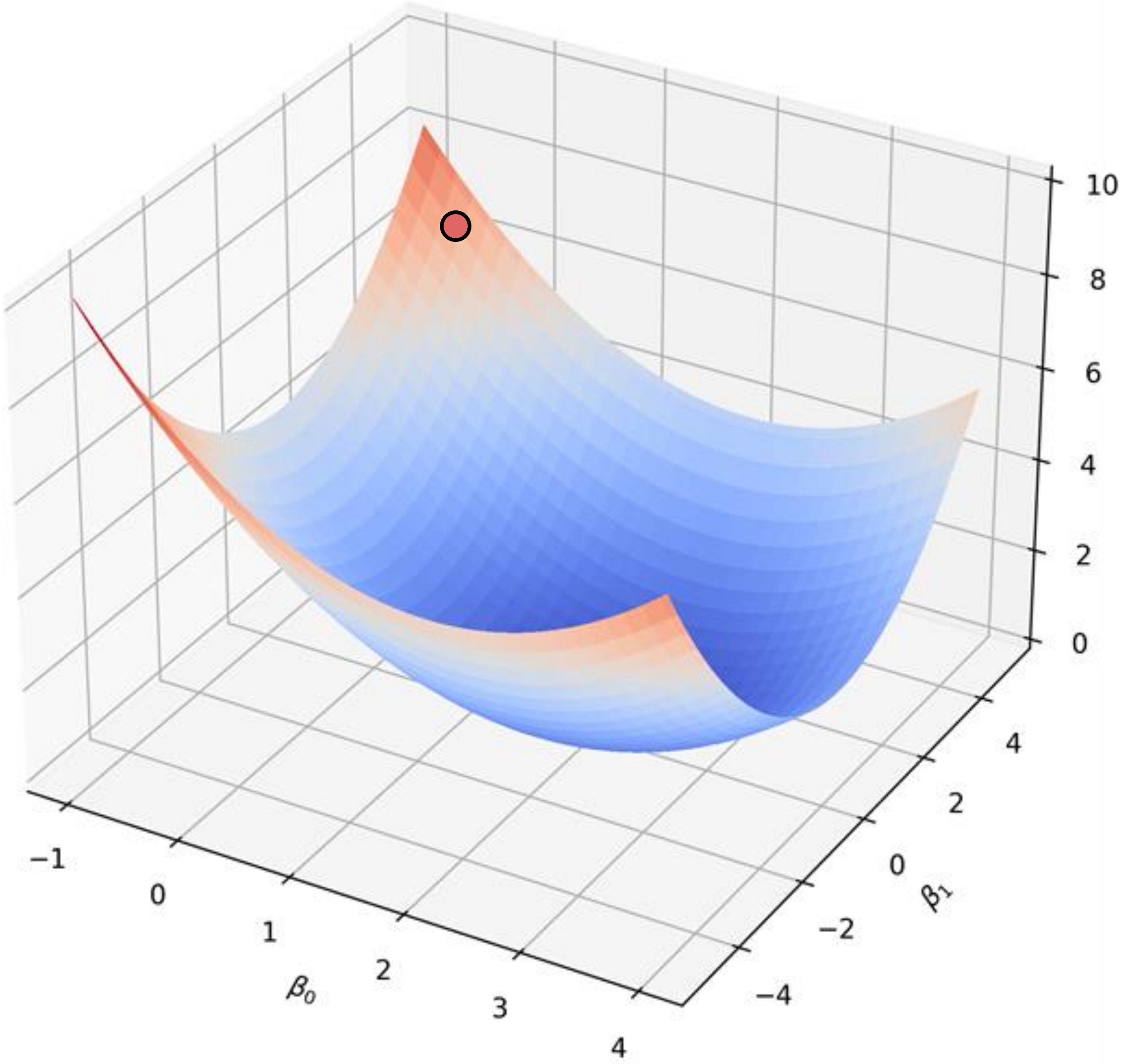
Linear Regression

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

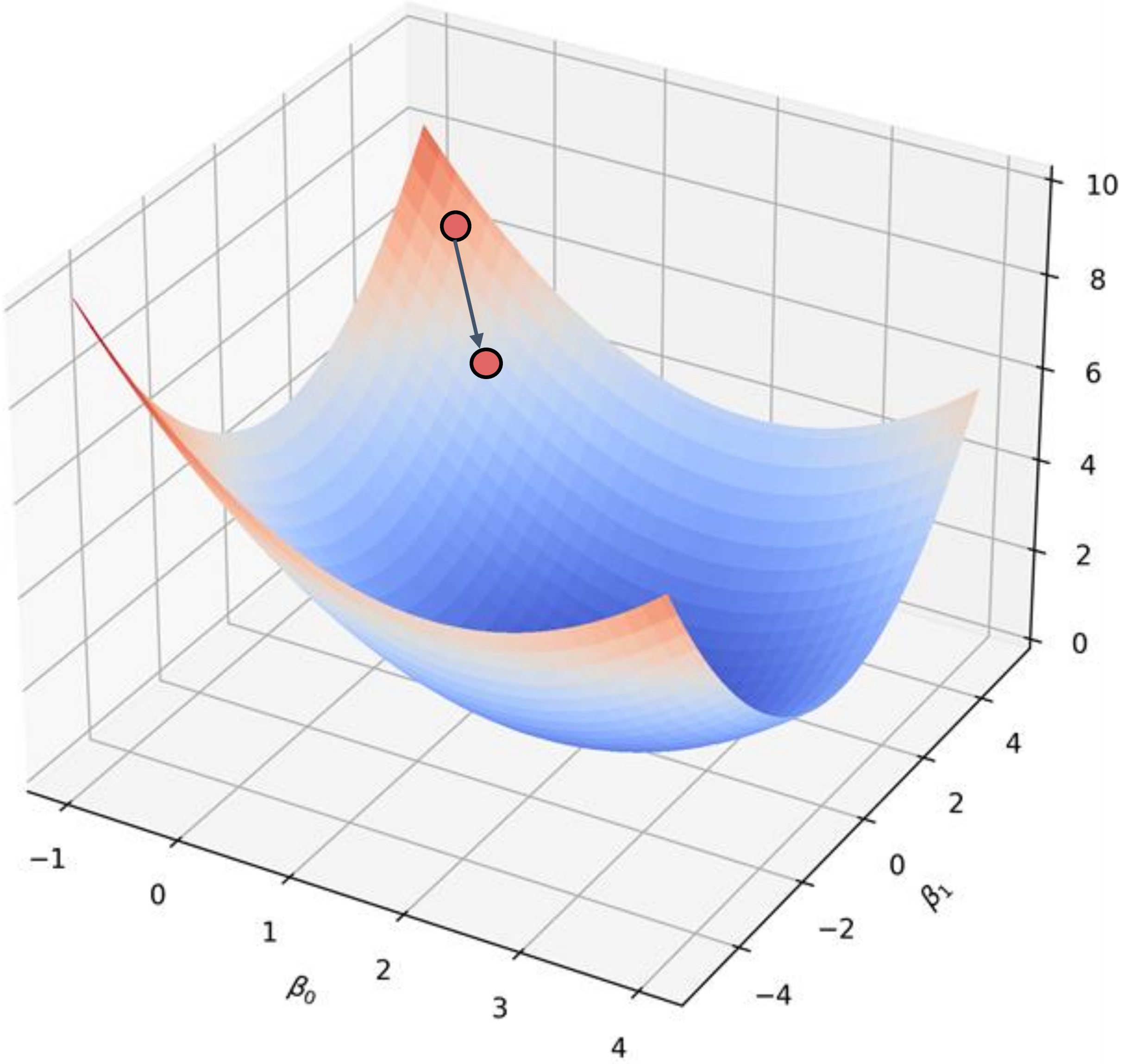
Linear Regression



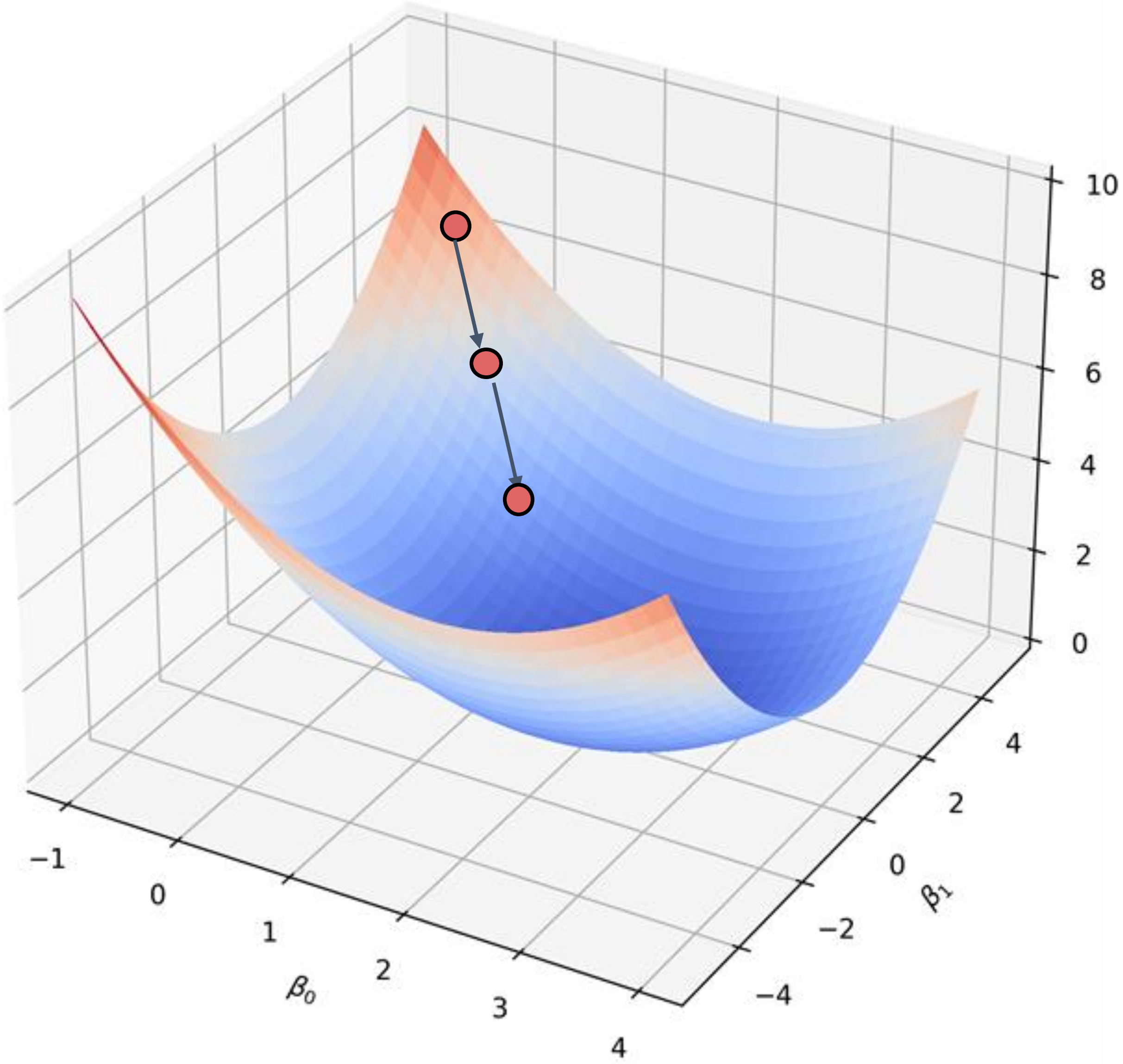
Linear Regression



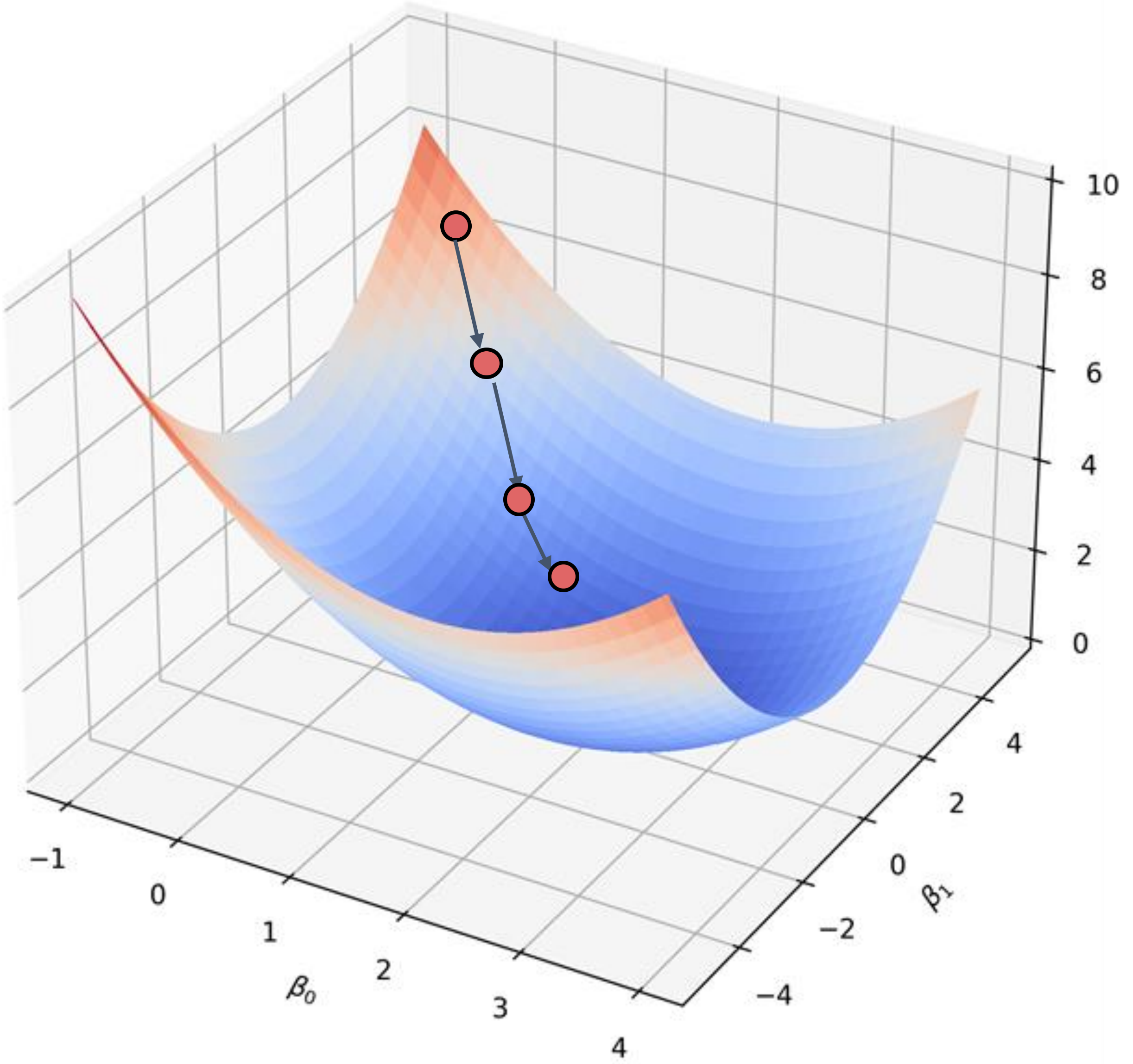
Linear Regression



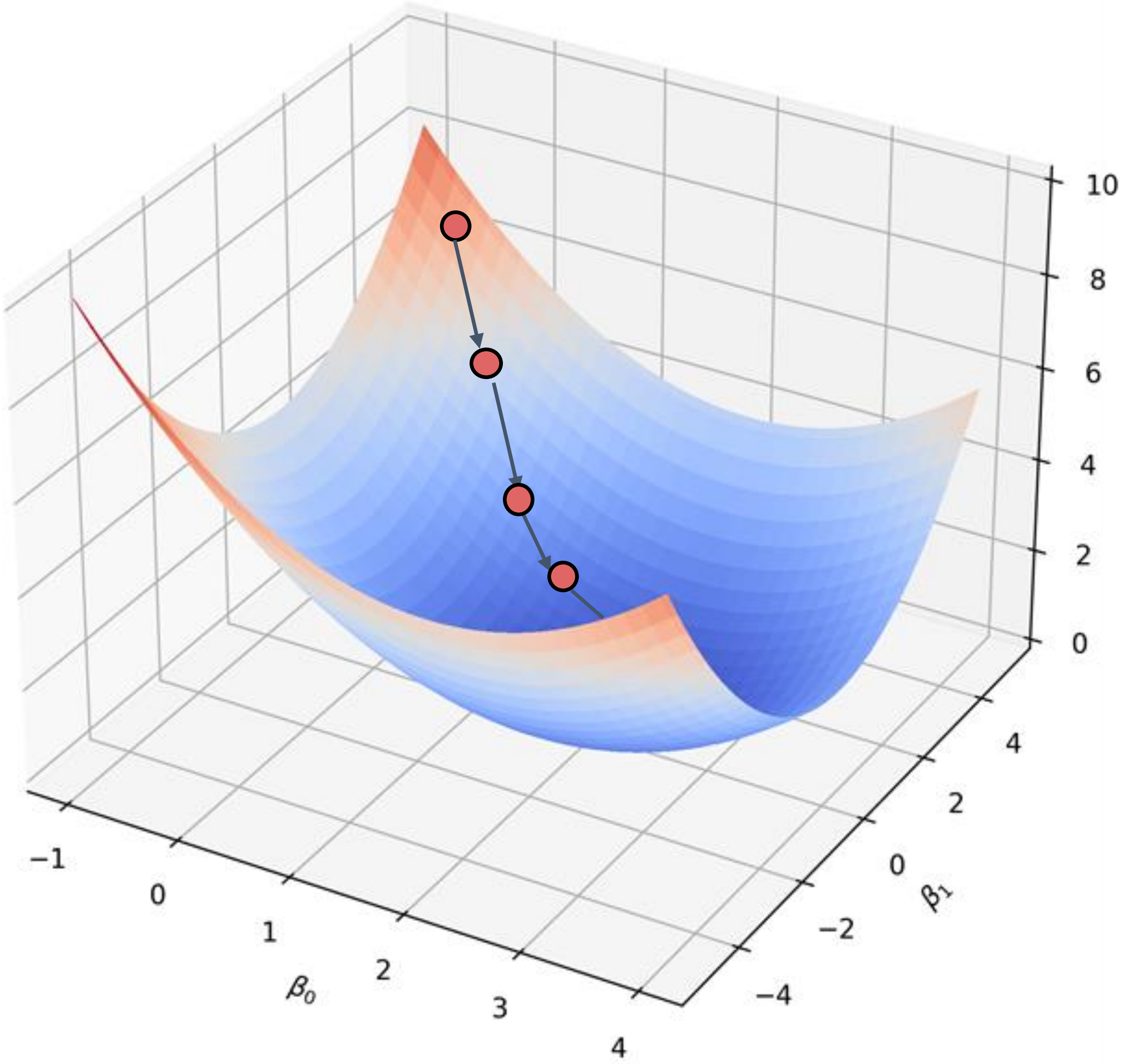
Linear Regression



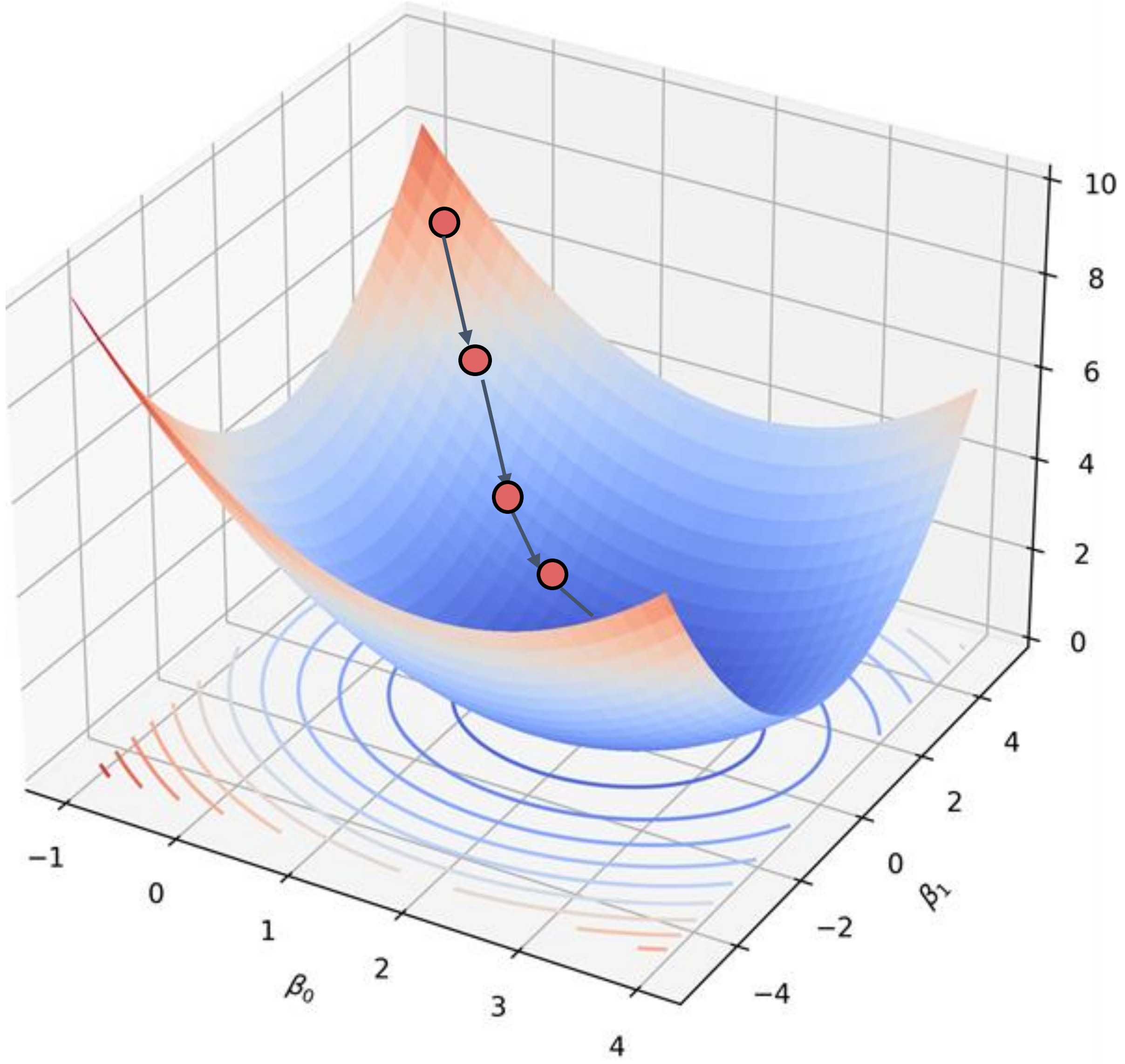
Linear Regression



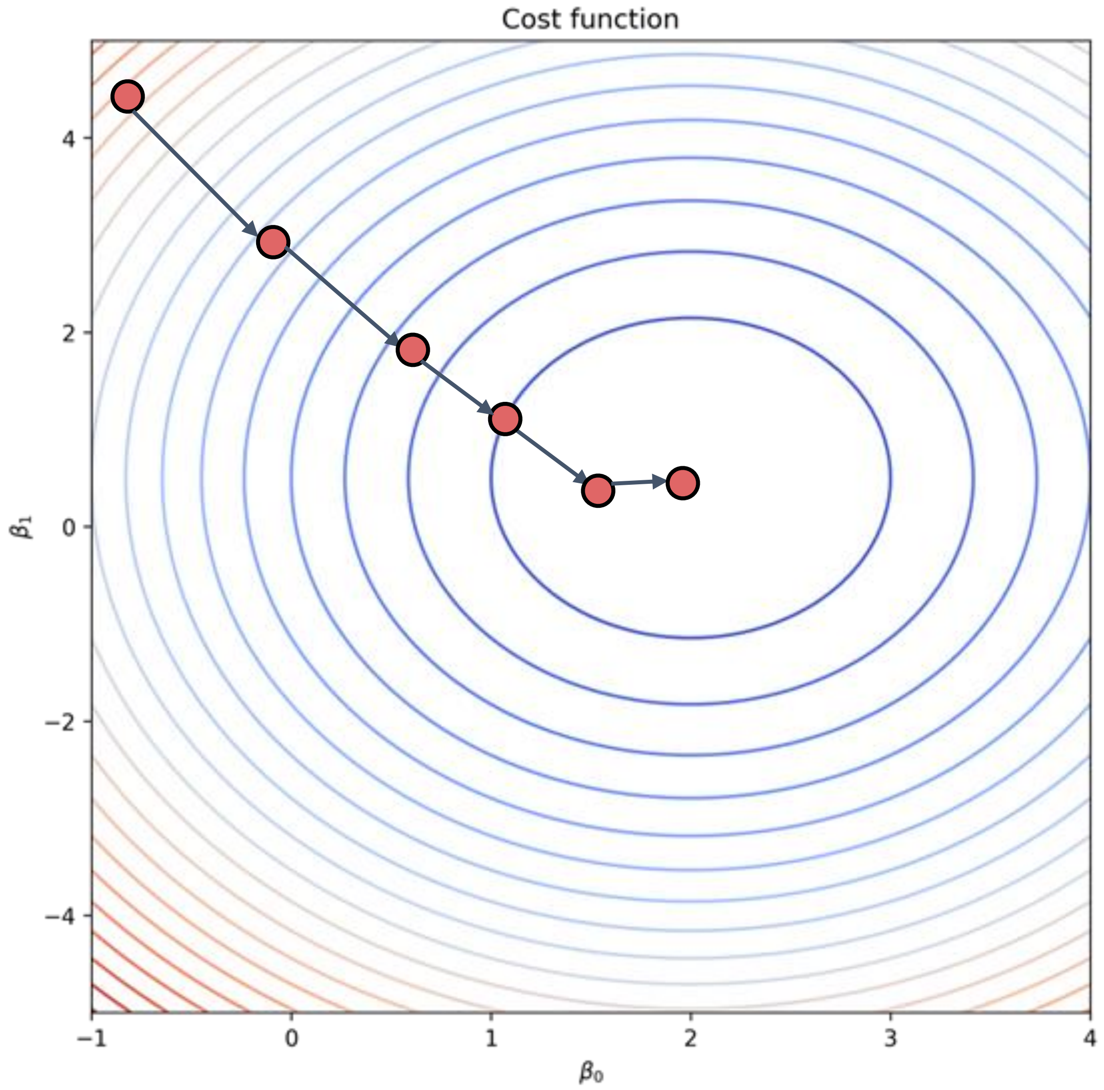
Linear Regression



Linear Regression



Linear Regression



Linear Regression

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

Scikit-Learn Overview

Supervised Machine Learning Process

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

TRAIN

TEST

Area m^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

Supervised Machine Learning Process

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

TRAIN

TEST

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

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.

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)
```

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

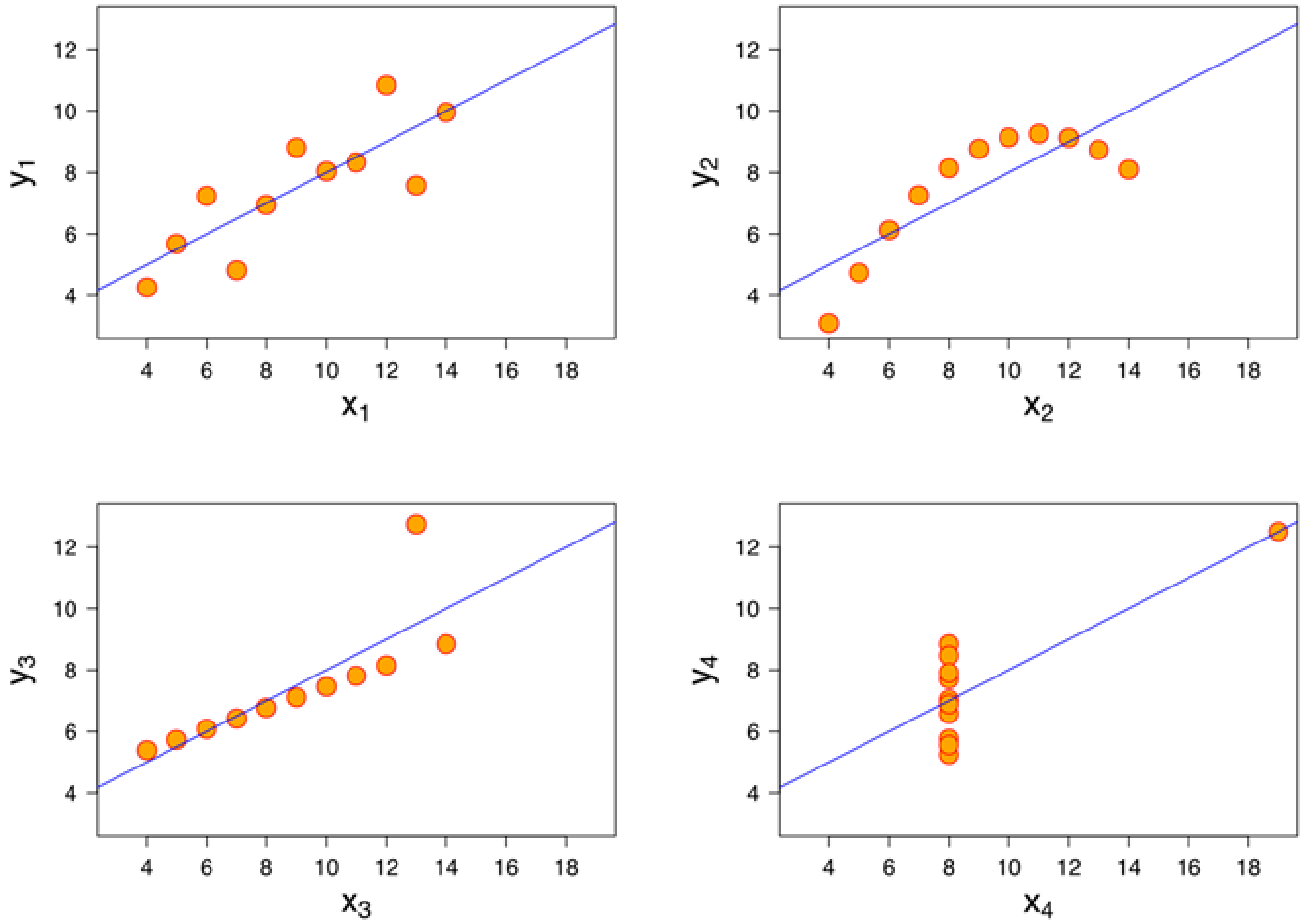
Evaluating Regression

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

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

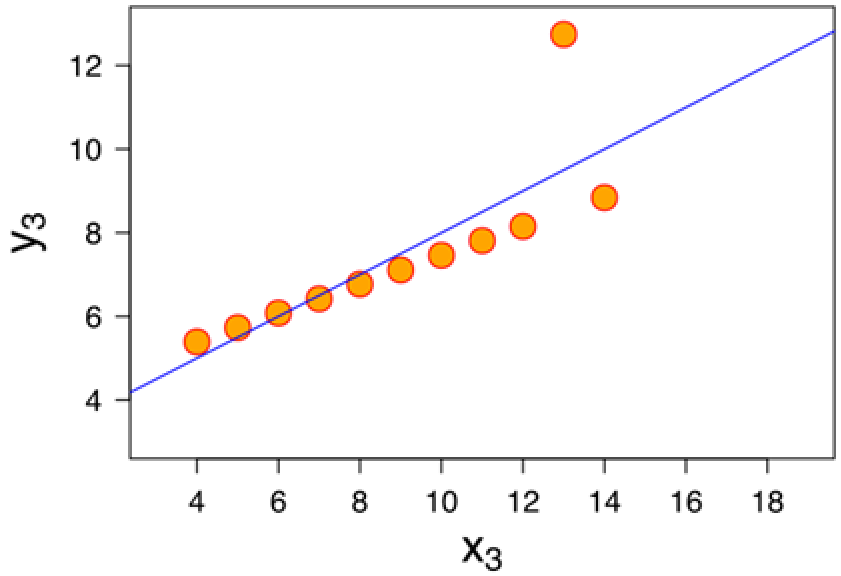
Evaluating Regression

- MAE won't punish large errors however.



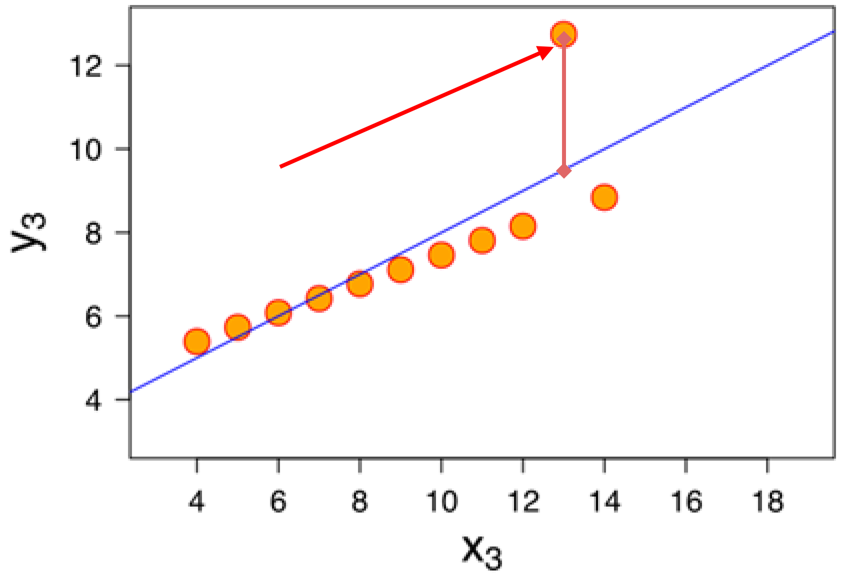
Evaluating Regression

- 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)

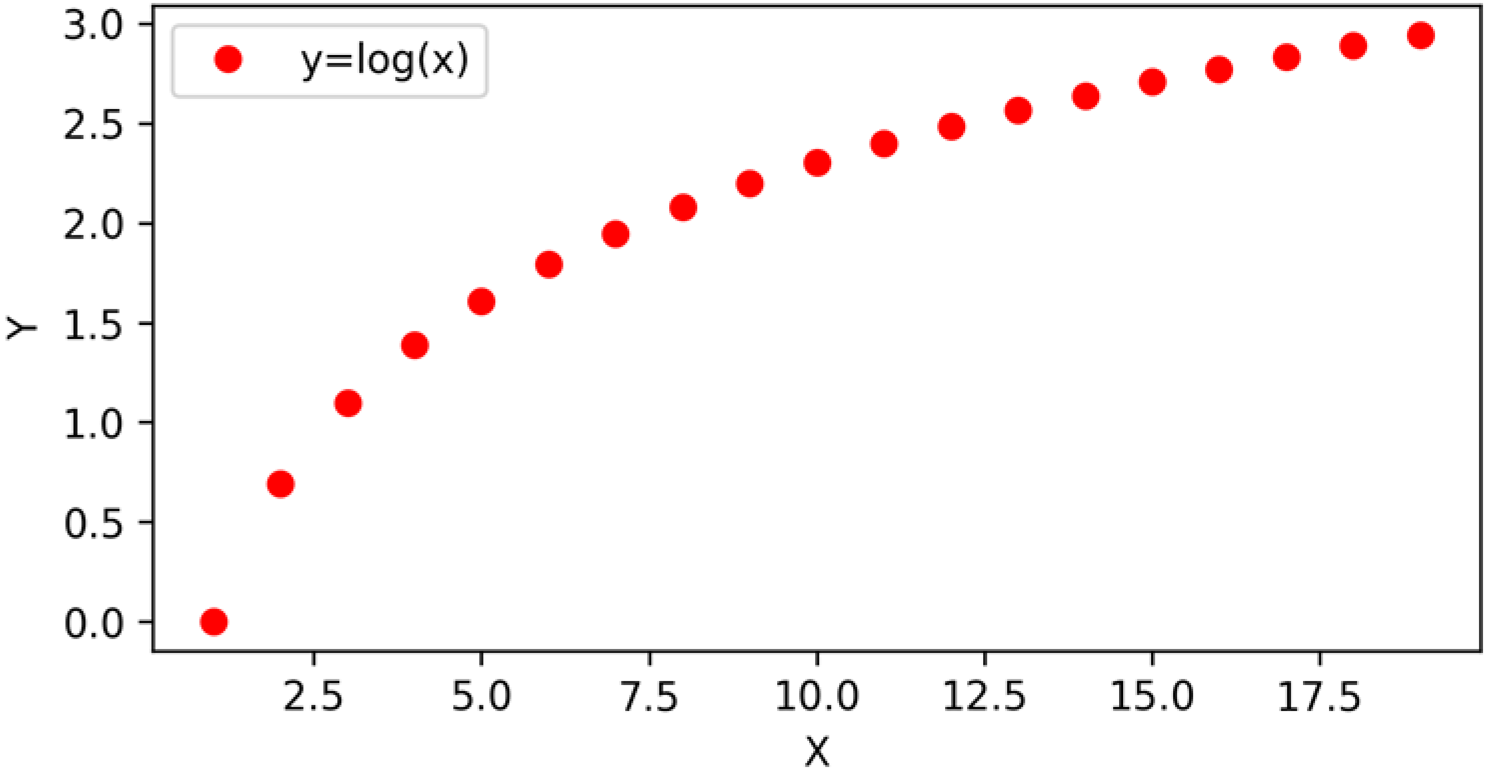
$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Polynomial Regression

Theory and Motivation

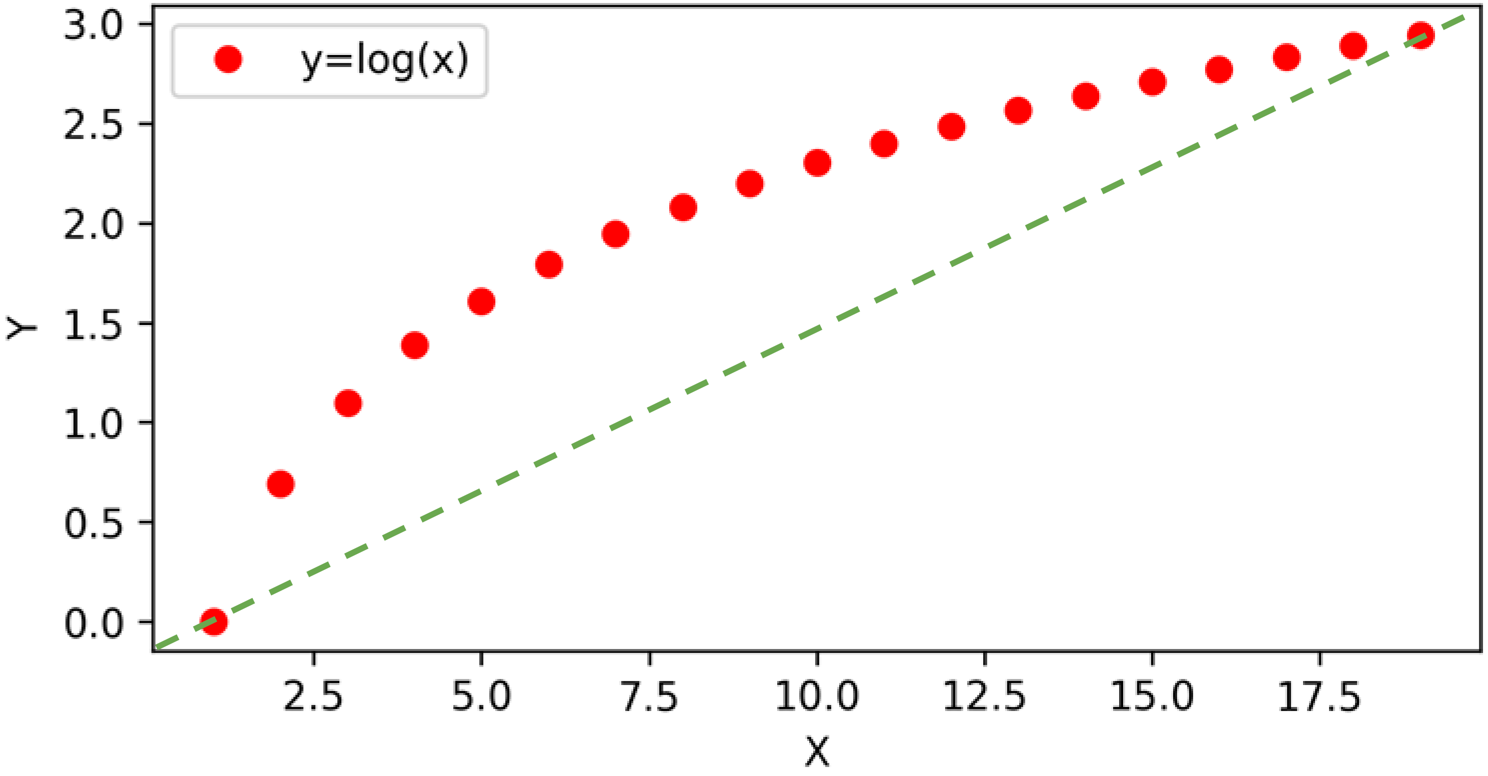
Polynomial Regression

- Imagine a feature that is not linear:



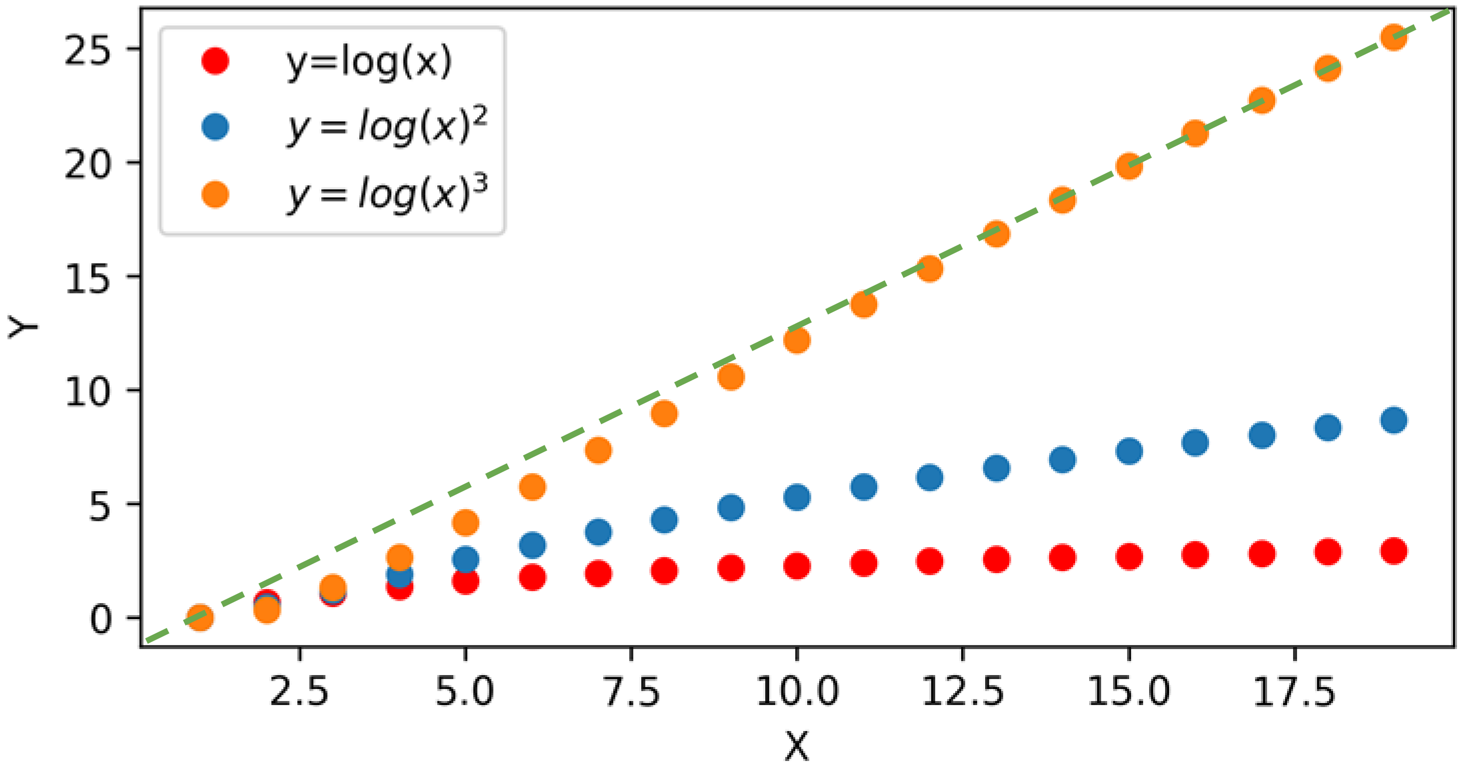
Polynomial Regression

- Will be difficult to find a linear relationship



Polynomial Regression

- Even more so for higher orders!



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**

Regularization

Regularization

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

Regularization

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

Regularization

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

Regularization

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

Regularization

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

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

Regularization

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

Regularization

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

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

Regularization

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

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

Feature Scaling

Feature Scaling

- Standardization:
  - Rescales data to have a mean (**μ**) of 0 and standard deviation (**σ**) of 1 (unit variance).

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

Feature Scaling

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

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

Feature Scaling

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

Cross Validation

Cross Validation

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

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

Cross Validation

- Convert to generalized form

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

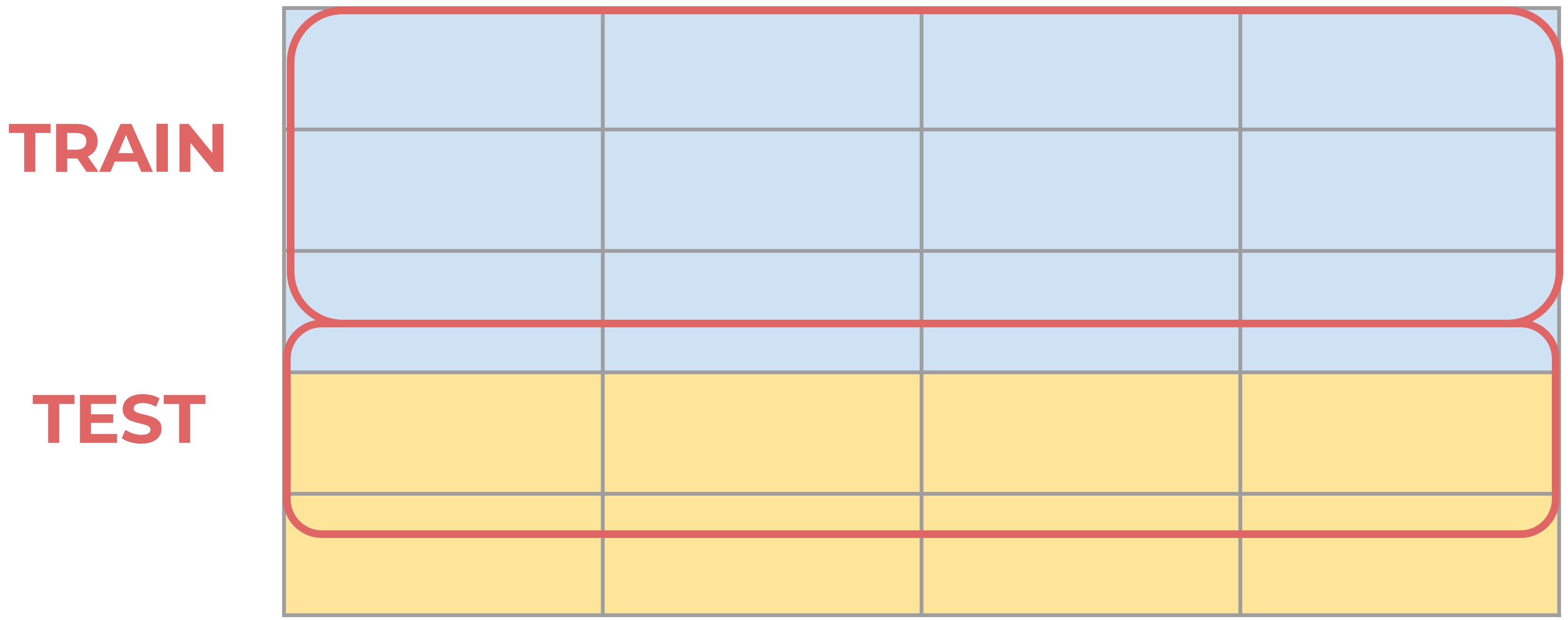
Cross Validation

- Color based off train vs. test set.

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

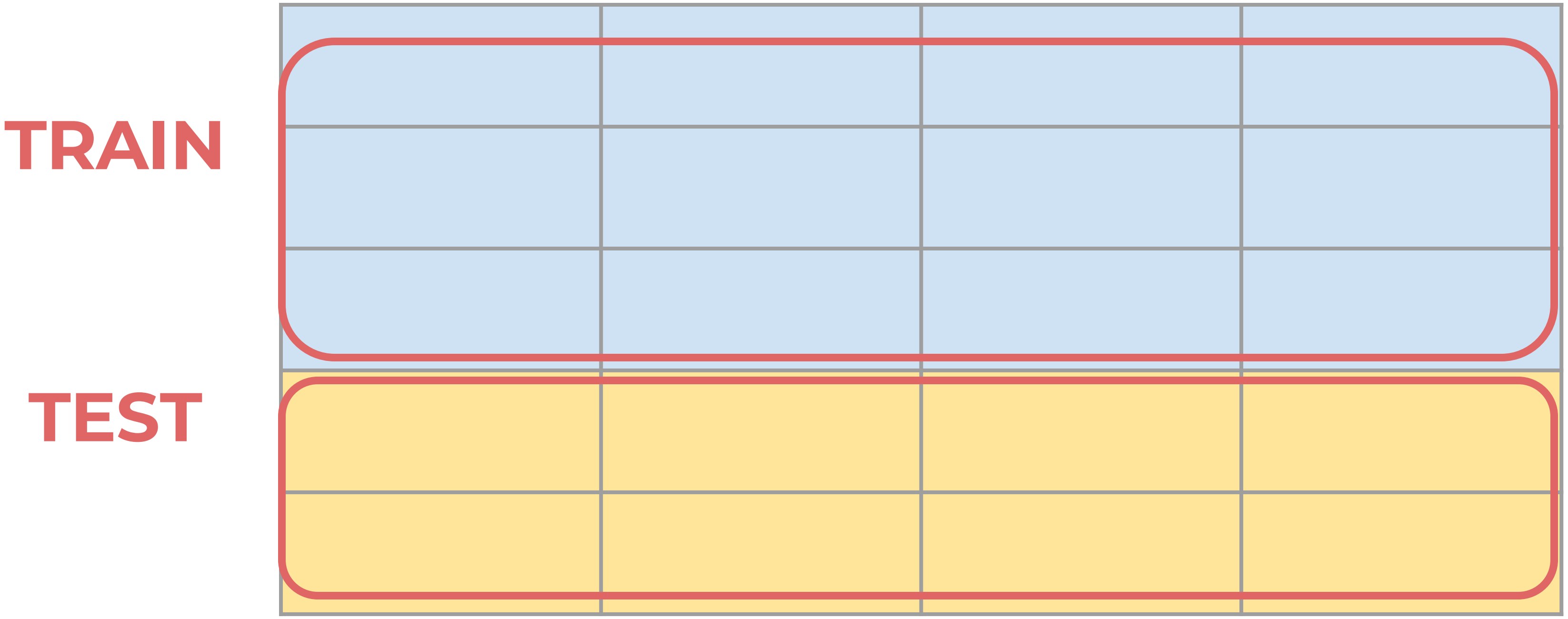
Cross Validation

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



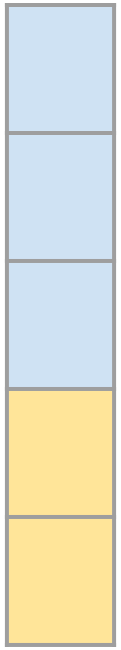
Cross Validation

- Rotate and resize:



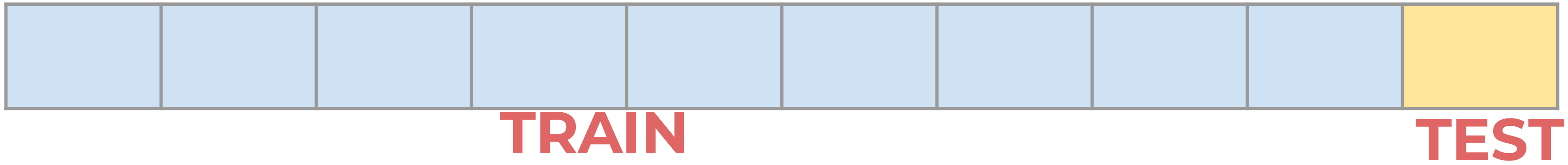
Cross Validation

- Rotate and resize:



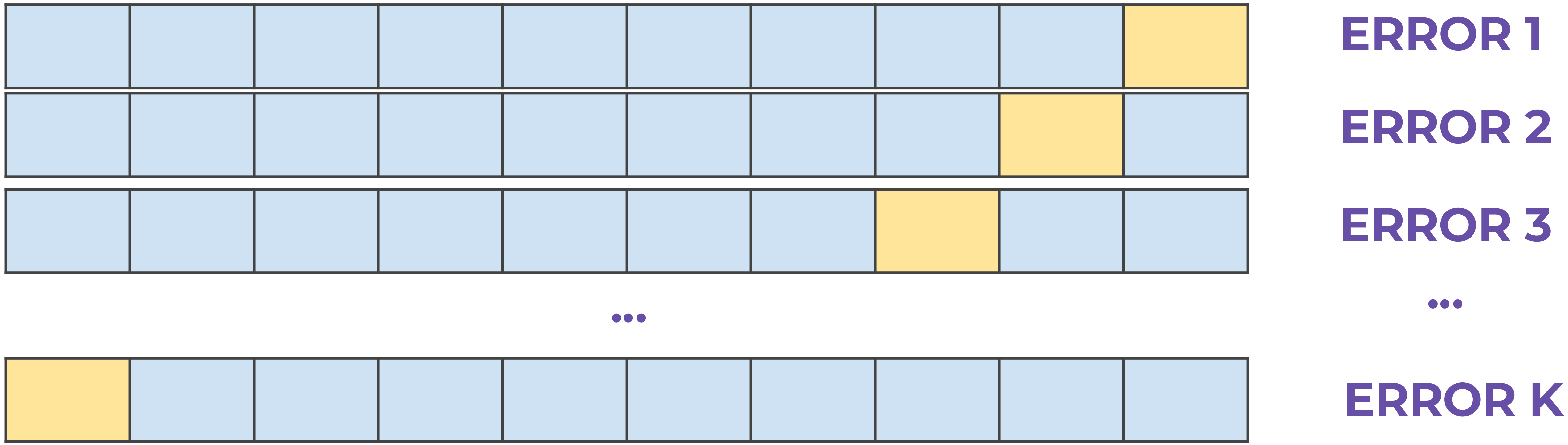
Cross Validation

- Now we can represent full data and splits:



Cross Validation

- Keep repeating for all possible K splits



Cross Validation

- Get average error