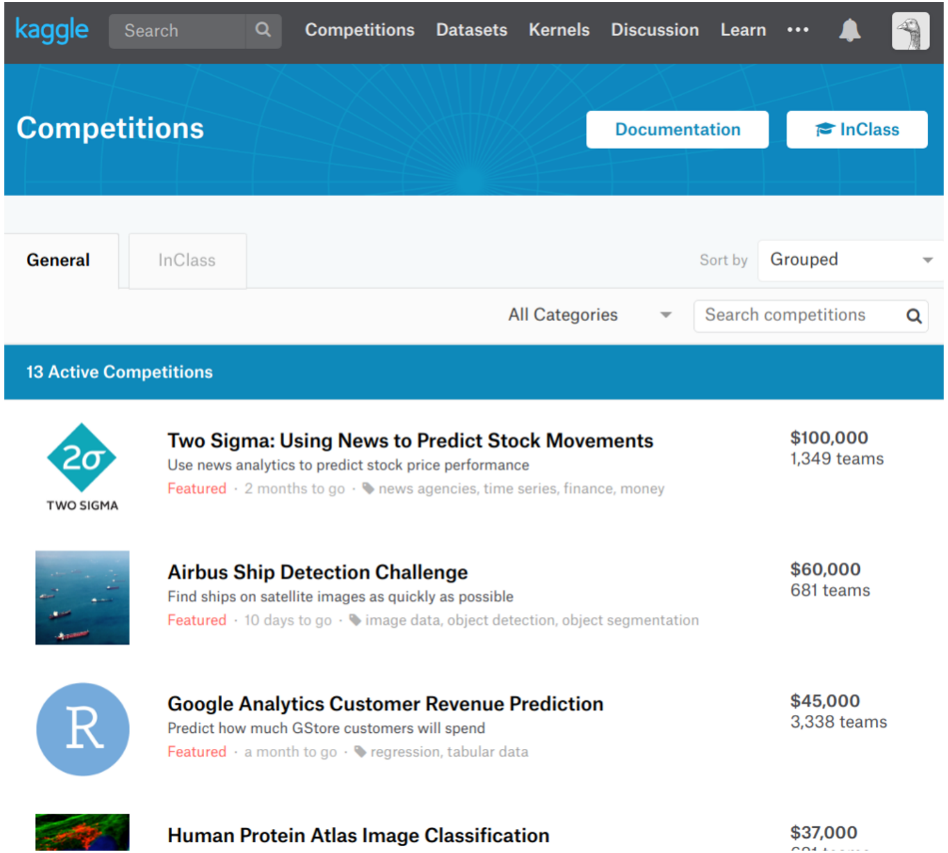
Data Science: Principles and Practice

Lecture 2: Linear Regression

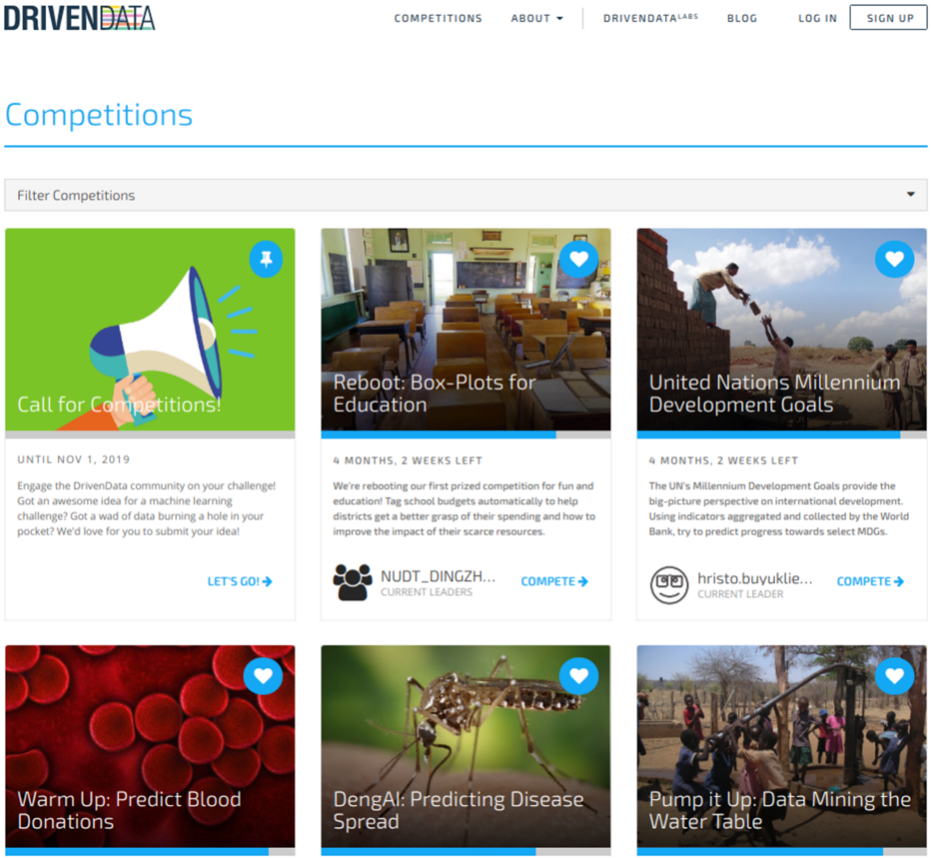
Ekaterina Kochmar[1]

UNIVERSITY OF CAMBRIDGE

[1] Based on slides from Marek Rei



kaggle.com



drivendata.org

Practical Data Science

- Kaggle datasets (https://www.kaggle.com/datasets)
- Data Science competitions (https://www.drivendata.org)
- UC Irvine Machine Learning Repository (https://archive.ics.uci.edu/ml/)
- Registry of Open Data on AWS (https://registry.opendata.aws)
- A Comprehensive List of Open Data Portals from Around the World (http://dataportals.org)
- Financial and economic datasets (https://www.quandl.com)
- Wikipedia's list of Machine Learning datasets (https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research)
- Datasets subreddit (https://www.reddit.com/r/datasets/)

Finally, your own data and projects

Data Science: Principles and Practice

01 Linear Regression

02 Optimization with Gradient Descent

03 Multiple Linear Regression and Polynomial Features

04 Overfitting

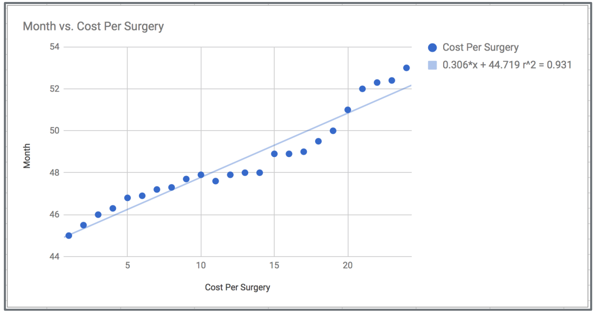
05 The First Practical

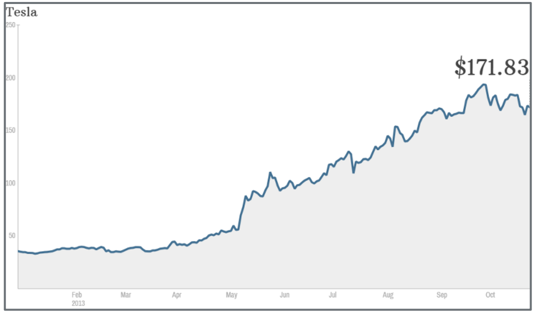
Linear regression

- **Linear regression** helps modelling how changes in one or more input variables (independent variables) affect the output (dependent variable)

- **Widely used algorithm** in machine learning and data science. **Application areas**: healthcare, social sciences, economics, environmental science, prediction of planetary movements

- Linear regression is an example of **supervised learning algorithms**





https://towardsdatascience.com/examples-of-applied-data-science-in-healthcare-and-e-commerce-e3b4a77ed306

Supervised Learning

Dataset: $\{< x_1, y_1 >, < x_2, y_2 >, < x_3, y_3 >, ..., < x_n, y_n >\}$

Input instances: $x_1, x_2, x_3, x_4, ..., x_n$

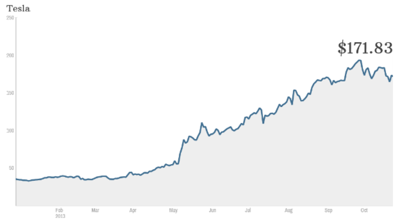
Known (desired) outputs: $y_1, y_2, y_3, y_4, ..., y_n$

Our goal: Learn the mapping $f : X \rightarrow Y$

such that $y_i = f(x_i)$ for all $i = 1, 2, 3, ..., n$

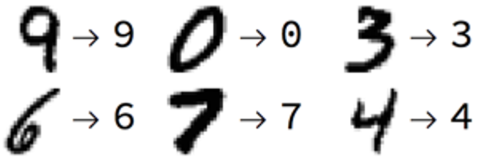
Continuous vs Discrete Problems

Regression: the desired labels are continuous



Company earnings, revenue → company stock price
House size and age → price

Classification: the desired labels are discrete



Handwritten digits → digit label
User tweets → detect positive/negative sentiment

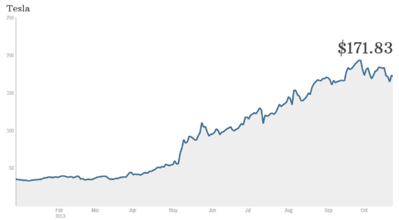
Regression or classification?

Model the salary of baseball players based on their game statistics

Continuous vs Discrete Problems

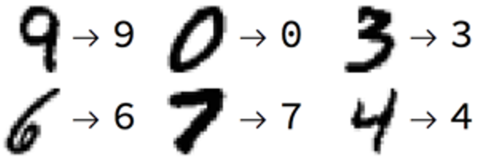
Regression: the desired labels are continuous

Company earnings, revenue → company stock price
House size and age → price



Classification: the desired labels are discrete

Handwritten digits → digit label
User tweets → detect positive/negative sentiment



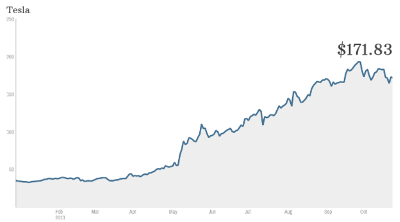
Regression or classification?

Model the salary of baseball players based on their game statistics → **regression**
Find what object is on a photo

Continuous vs Discrete Problems

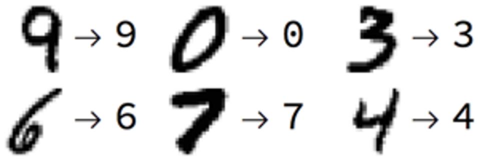
Regression: the desired labels are continuous

Company earnings, revenue → company stock price
House size and age → price



Classification: the desired labels are discrete

Handwritten digits → digit label
User tweets → detect positive/negative sentiment



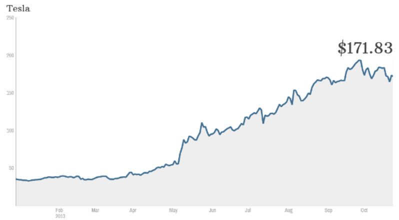
Regression or classification?

Model the salary of baseball players based on their game statistics → **regression**
Identify what object is on a photo → **classification**
Predict election results

Continuous vs Discrete Problems

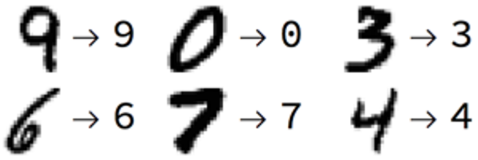
Regression: the desired labels are continuous

Company earnings, revenue → company stock price
House size and age → price



Classification: the desired labels are discrete

Handwritten digits → digit label
User tweets → detect positive/negative sentiment



Regression or classification?

Model the salary of baseball players based on their game statistics → **regression**
Identify what object is on a photo → **classification**
Predict election results → **regression** (%) / **classification** (winner)

Simplest Possible Linear Model

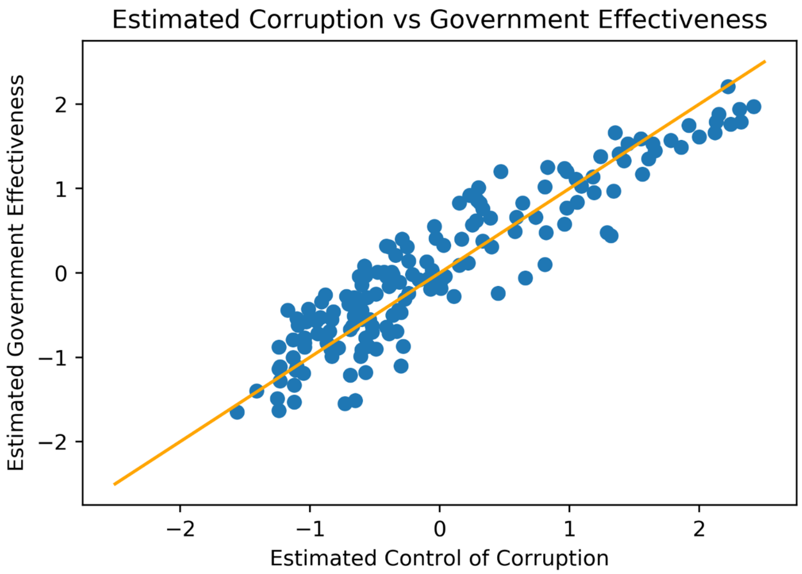
What is the simplest possible model for $f : X \rightarrow Y$?

$$y = x$$

Simplest Possible Linear Model

What is the simplest possible model for $f : X \to Y$?

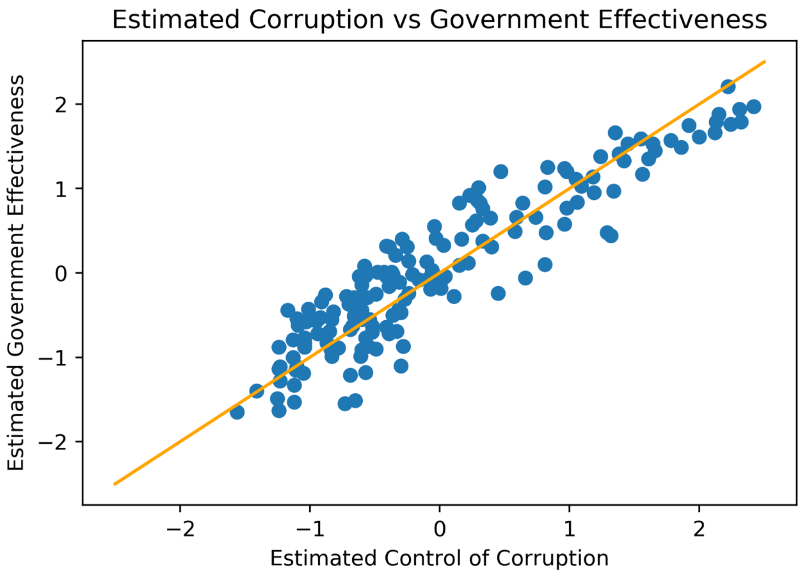
$$y = x$$

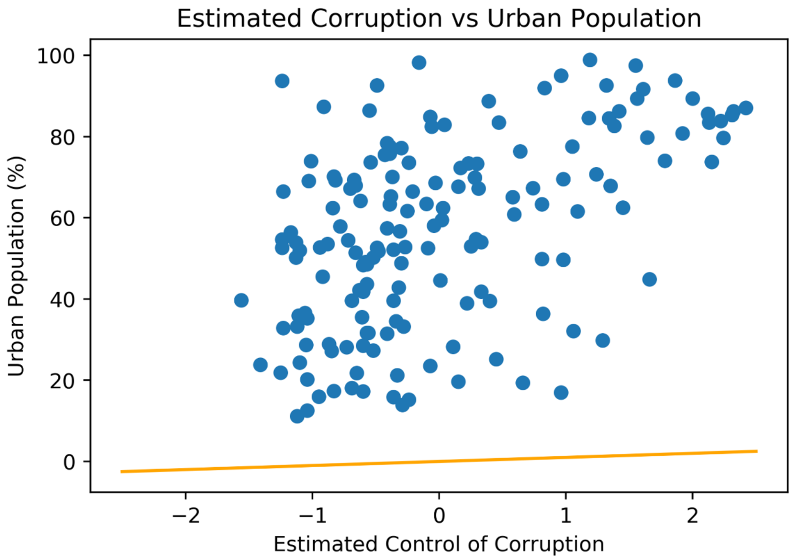


Simplest Possible Linear Model

What is the simplest possible model for $f : X \rightarrow Y$?

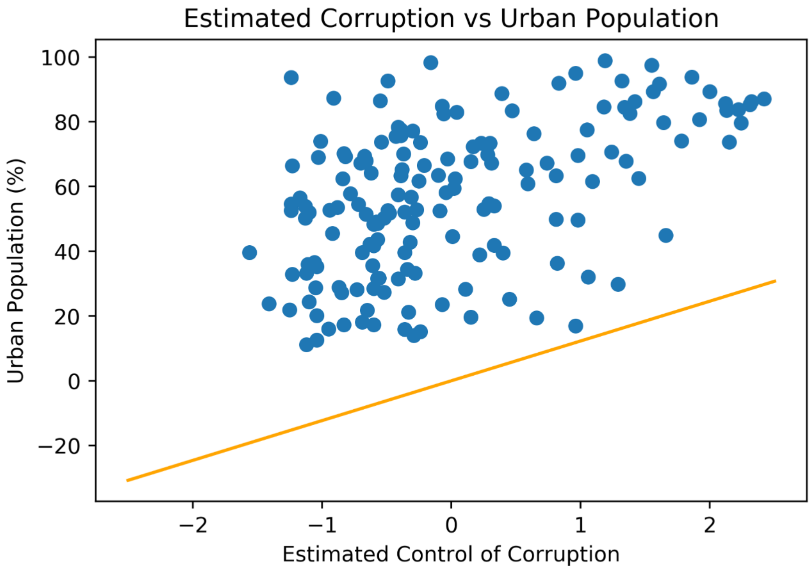
$$y = x$$



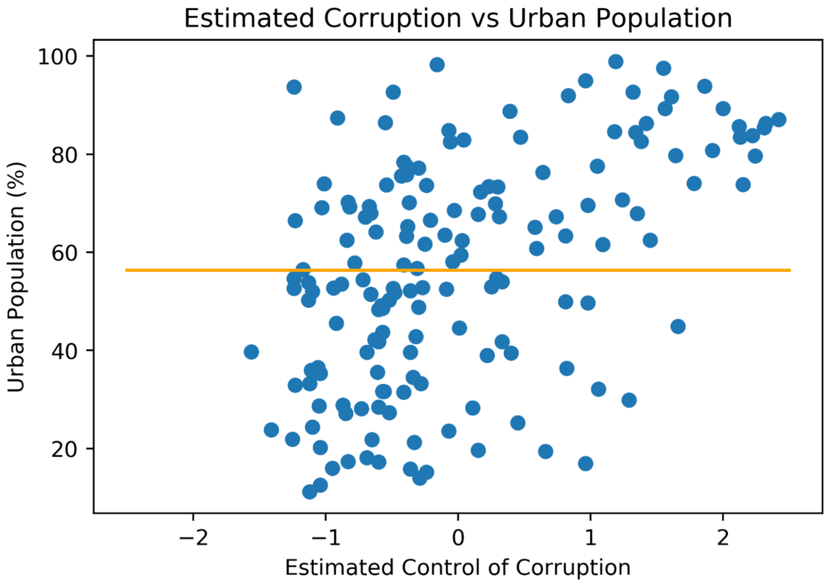


(Still Too Simple) Linear Models

$$y = ax$$

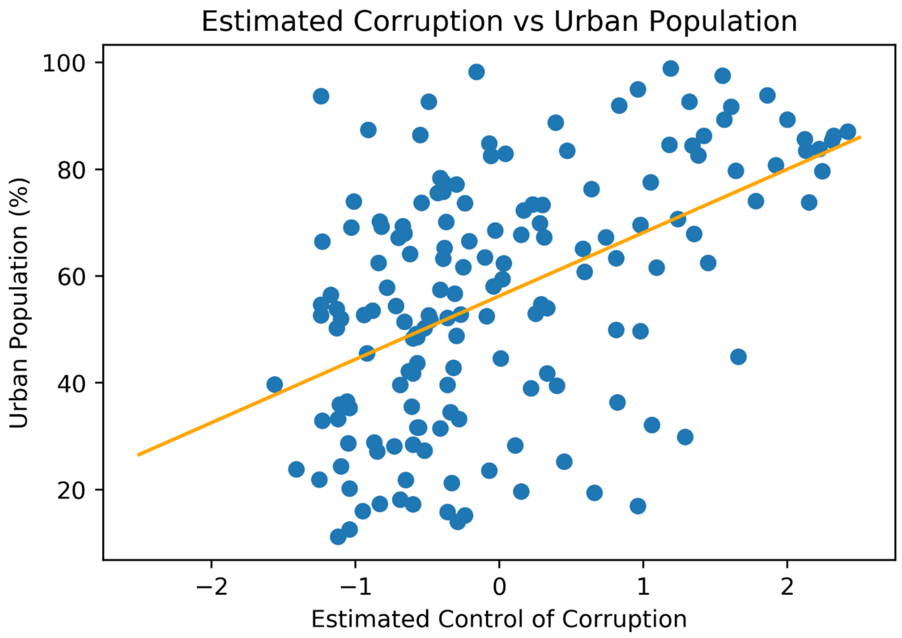


$$y = b$$



Linear Regression

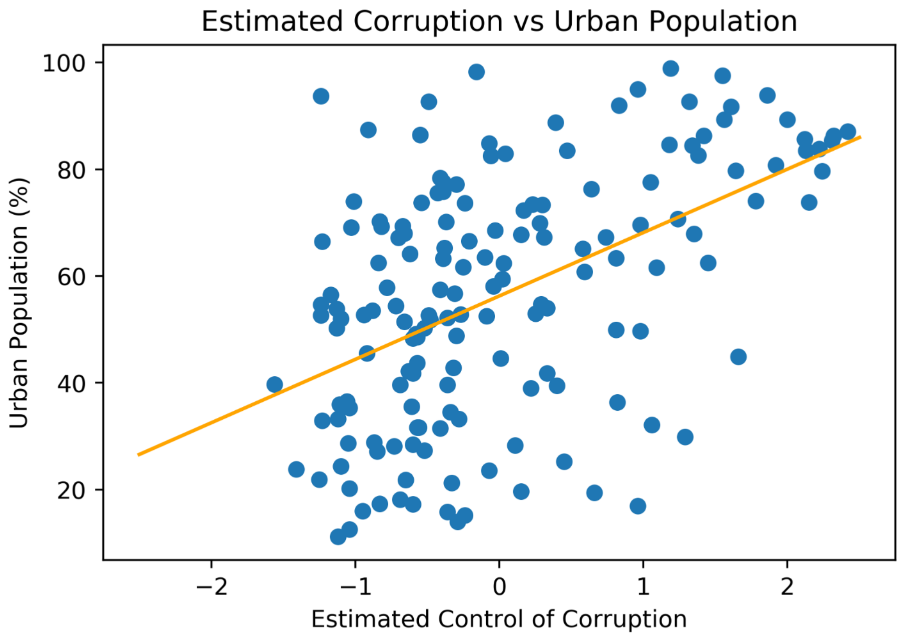
A better linear model: $y = ax + b$



$$y = ax + b$$

Linear Regression

A better linear model: $y = ax + b$



$y = ax + b$

Controls the angle

Controls the intercept

Linear Regression

$$x : \text{GDP per Capita}$$

$$y : \text{Enrolment Rate}$$

$$\hat{y} = ax + b$$

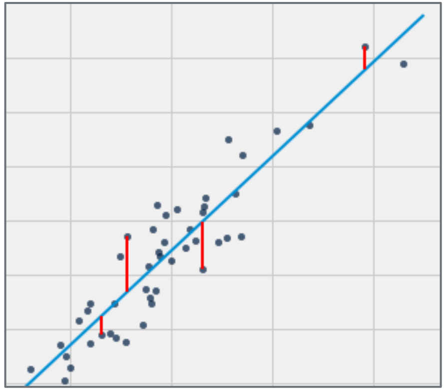
How do we find the best values for **a** and **b**?

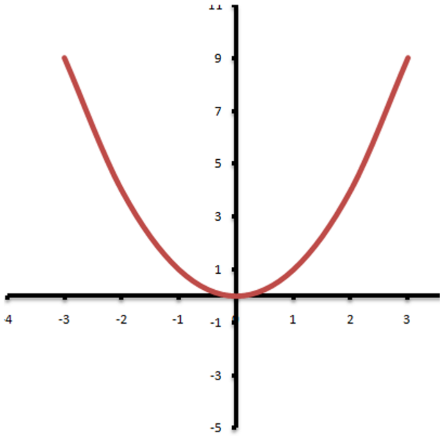
	Country Name	GDP per Capita (PPP USD)	Enrolment Rate, Tertiary (%)
0	Afghanistan	1560.67	3.33
1	Albania	9403.43	54.85
2	Algeria	8515.35	31.46
3	Antigua and Barbuda	19640.35	14.37
4	Argentina	12016.20	74.83
5	Armenia	8416.82	48.94
6	Australia	44597.83	83.24
7	Austria	43661.15	71.00
8	Azerbaijan	10125.23	19.65
9	Bahrain	24590.49	33.46
10	Bangladesh	1883.05	13.15
11	Barbados	26487.77	60.84
12	Belgium	39751.48	69.26

Loss Function

First, let's define what "best" actually means for us.

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



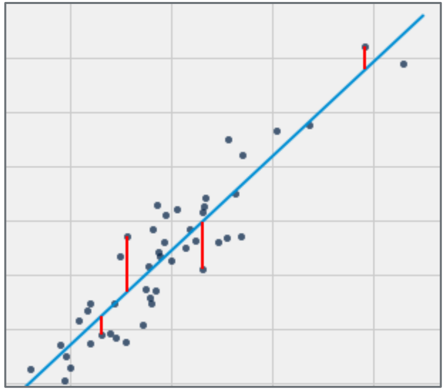


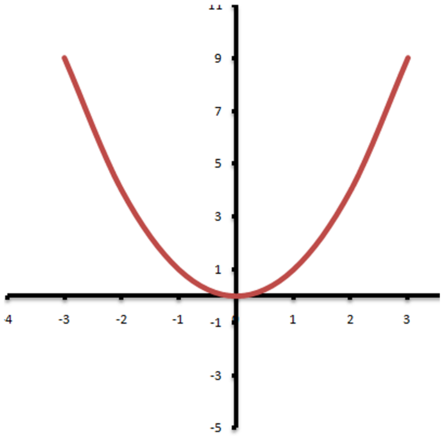
Loss Function

First, let's define what "best" actually means for us.

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

$$E = \frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$



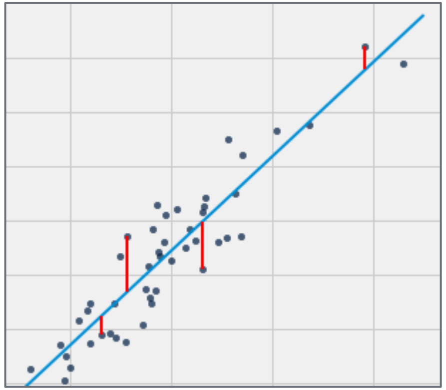


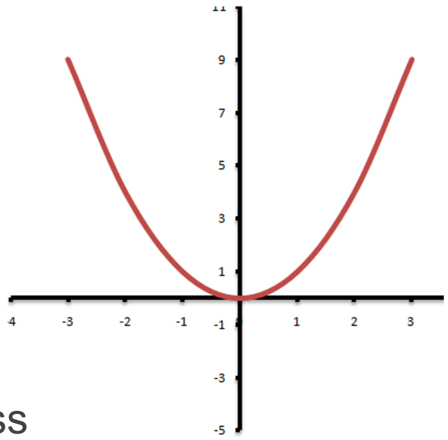
Loss Function

First, let's define what "best" actually means for us.

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

$$E = \frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$





- Smaller value of *E* means our predictions are close to the real values
- Individual large errors incur a large exponential penalty
- Many small errors are acceptable and get a very small loss
- Easily differentiable function

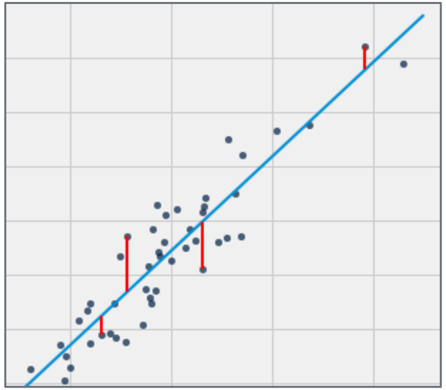
Loss Function

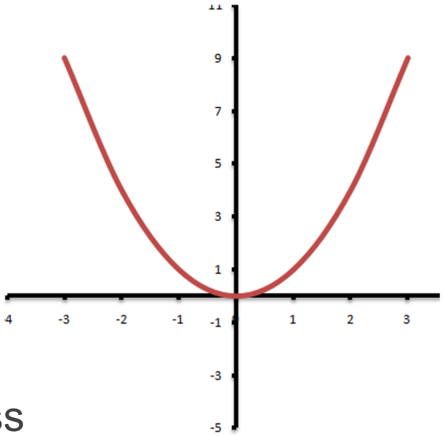
First, let's define what "best" actually means for us.

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

$$E = \frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$

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





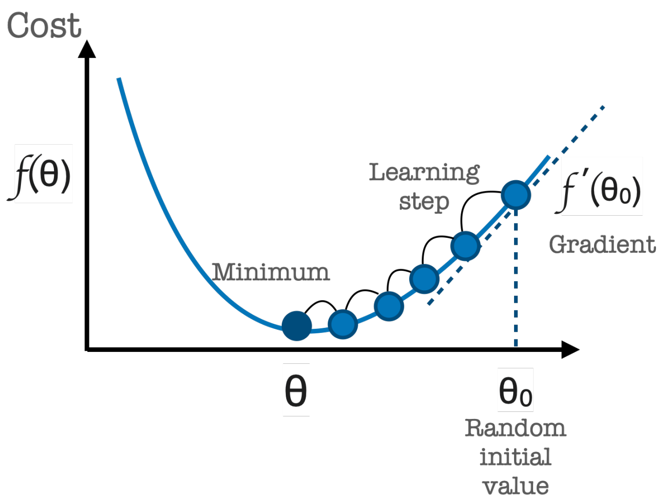
- Smaller value of E means our predictions are close to the real values
- Individual large errors incur a large exponential penalty
- Many small errors are acceptable and get a very small loss
- Easily differentiable function

Gradient Descent

$$\frac{\partial E}{\partial a} = \frac{\partial}{\partial a}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$

We can update **a** and **b** using the training data and the loss function.

The partial derivative of a function shows the direction of the slope.



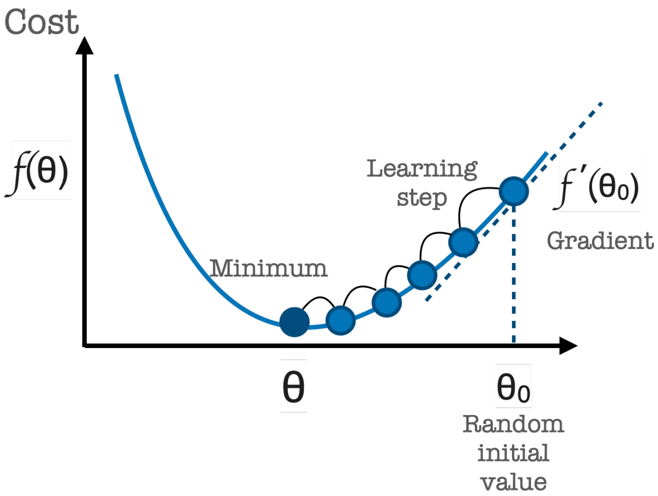
Gradient Descent

We can update **a** and **b** using the training data and the loss function.

The partial derivative of a function shows the direction of the slope.

$$\frac{\partial E}{\partial a} = \frac{\partial}{\partial a}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$

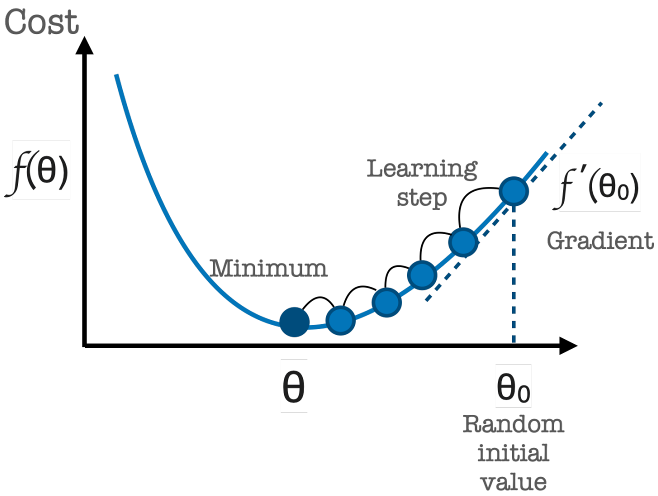
$$= \frac{1}{2}\sum_{i=1}^{M}\frac{\partial}{\partial a}(ax_i + b - y_i)^2$$



Gradient Descent

We can update **a** and **b** using the training data and the loss function.

The partial derivative of a function shows the direction of the slope.



$$\frac{\partial E}{\partial a} = \frac{\partial}{\partial a}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$

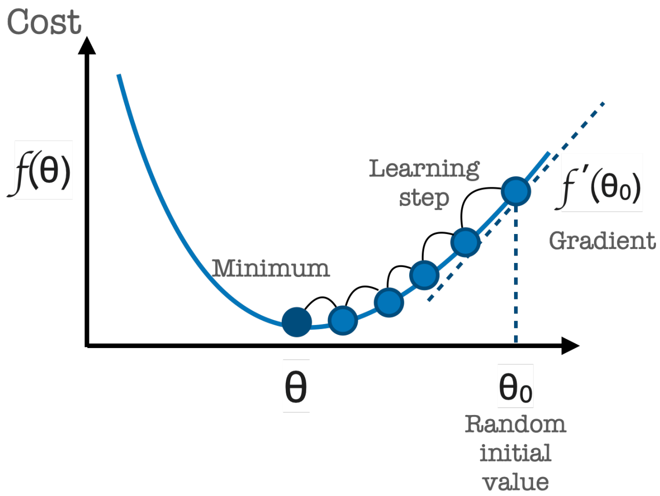
$$= \frac{1}{2}\sum_{i=1}^{M}\frac{\partial}{\partial a}(ax_i + b - y_i)^2$$

$$= \sum_{i=1}^{M}(ax_i + b - y_i)x_i = \sum_{i=1}^{M}(\hat{y}_i - y_i)x_i$$

Gradient Descent

We can update **a** and **b** using the training data and the loss function.

The partial derivative of a function shows the direction of the slope.



$$\frac{\partial E}{\partial a} = \frac{\partial}{\partial a}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{M}\frac{\partial}{\partial a}(ax_i + b - y_i)^2$$

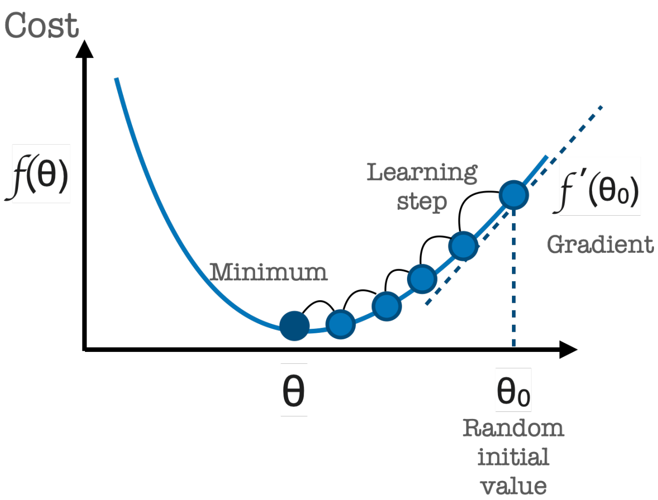
$$= \sum_{i=1}^{M}(ax_i + b - y_i)x_i = \sum_{i=1}^{M}(\hat{y}_i - y_i)x_i$$

$$\frac{\partial E}{\partial b} = \frac{\partial}{\partial b}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$

Gradient Descent

We can update **a** and **b** using the training data and the loss function.

The partial derivative of a function shows the direction of the slope.



$$\frac{\partial E}{\partial a} = \frac{\partial}{\partial a}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{M}\frac{\partial}{\partial a}(ax_i + b - y_i)^2$$

$$= \sum_{i=1}^{M}(ax_i + b - y_i)x_i = \sum_{i=1}^{M}(\hat{y}_i - y_i)x_i$$

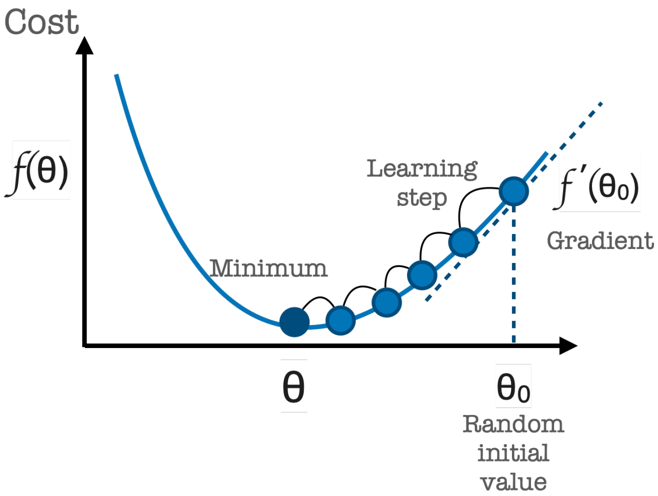
$$\frac{\partial E}{\partial b} = \frac{\partial}{\partial b}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$

$$= \sum_{i=1}^{M}(ax_i + b - y_i)$$

Gradient Descent

We can update **a** and **b** using the training data and the loss function.

The partial derivative of a function shows the direction of the slope.



$$\frac{\partial E}{\partial a} = \frac{\partial}{\partial a}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$
$$= \frac{1}{2}\sum_{i=1}^{M}\frac{\partial}{\partial a}(ax_i + b - y_i)^2$$
$$= \sum_{i=1}^{M}(ax_i + b - y_i)x_i = \sum_{i=1}^{M}(\hat{y}_i - y_i)x_i$$

$$\frac{\partial E}{\partial b} = \frac{\partial}{\partial b}\frac{1}{2}\sum_{i=1}^{M}(ax_i + b - y_i)^2$$
$$= \sum_{i=1}^{M}(ax_i + b - y_i)$$
$$= \sum_{i=1}^{M}(\hat{y}_i - y_i)$$

Gradient Descent

Gradient descent: Repeatedly update parameters **a** and **b** by taking small steps in the direction of the partial derivative.

$$a := a - \alpha \frac{\partial E}{\partial a} \qquad b := b - \alpha \frac{\partial E}{\partial b}$$

α : learning rate / step size

Gradient Descent

Gradient descent: Repeatedly update parameters **a** and **b** by taking small steps in the direction of the partial derivative.

$$a := a - \alpha \frac{\partial E}{\partial a} \qquad b := b - \alpha \frac{\partial E}{\partial b}$$

α : learning rate / step size

$$a := a - \alpha \sum_{i=1}^{M} (a x_i + b - y_i) x_i$$

$$b := b - \alpha \sum_{i=1}^{M} (a x_i + b - y_i)$$

Gradient Descent

Gradient descent: Repeatedly update parameters **a** and **b** by taking small steps in the direction of the partial derivative.

$$a := a - \alpha \frac{\partial E}{\partial a} \qquad b := b - \alpha \frac{\partial E}{\partial b}$$

α : learning rate / step size

$$a := a - \alpha \sum_{i=1}^{M} (ax_i + b - y_i) x_i$$

$$b := b - \alpha \sum_{i=1}^{M} (ax_i + b - y_i)$$

This same algorithm drives nearly all of the modern neural network models.

Gradient Descent

Implementing gradient descent by hand:

In [8]:
```
X = data["GDP per Capita (PPP USD)"].values
Y = data["Enrolment Rate, Tertiary (%)"].values

a = 0.0
b = 0.0
learning_rate = 1e-11


for epoch in range(10):
    update_a = 0.0
    update_b = 0.0
    error = 0.0
    for i in range(len(Y)):
        y_predicted = a * X[i] + b
        update_a += (y_predicted - Y[i])*X[i]
        update_b += (y_predicted - Y[i])
        error += np.square(y_predicted - Y[i])
    a = a - learning_rate * update_a
    b = b - learning_rate * update_b
    rmse = np.sqrt(error / len(Y))
    print("RMSE: " + str(rmse))

plot_simple_linear_regression(X, Y, a, b)
```

```
RMSE: 43.60705215347086
RMSE: 27.764974739091667
RMSE: 27.121980238646962
RMSE: 27.101664900304836
RMSE: 27.101030544822766
RMSE: 27.101010737719825
RMSE: 27.101010112785858
RMSE: 27.101010086586154
RMSE: 27.101010079074783
RMSE: 27.10101007214674
```

See lecture2.ipynb at https://github.com/ekochmar/cl-datasci-pnp-2021

Gradient Descent

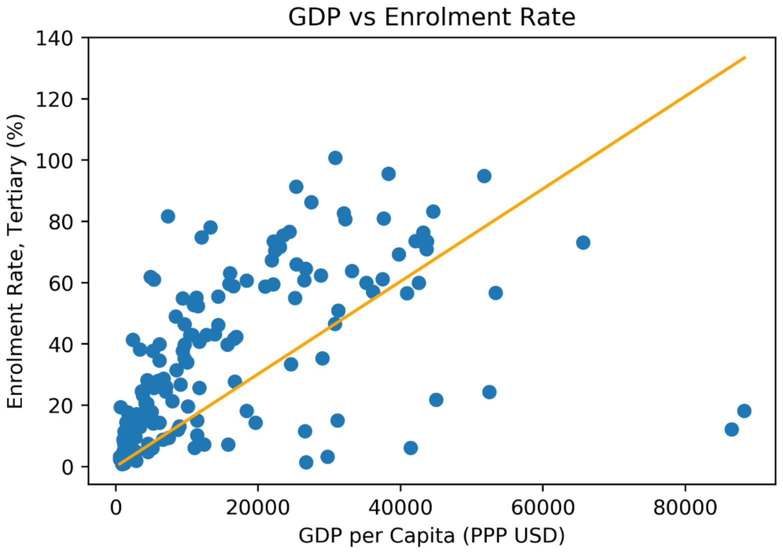
Implementing gradient descent by hand:

In [8]:
```
X = data["GDP per Capita (PPP USD)"].values
Y = data["Enrolment Rate, Tertiary (%)"].values

a = 0.0
b = 0.0
learning_rate = 1e-11


for epoch in range(10):
    update_a = 0.0
    update_b = 0.0
    error = 0.0
    for i in range(len(Y)):
        y_predicted = a * X[i] + b
        update_a += (y_predicted - Y[i])*X[i]
        update_b += (y_predicted - Y[i])
        error += np.square(y_predicted - Y[i])
    a = a - learning_rate * update_a
    b = b - learning_rate * update_b
    rmse = np.sqrt(error / len(Y))
    print("RMSE: " + str(rmse))

plot_simple_linear_regression(X, Y, a, b)
```



Gradient Descent

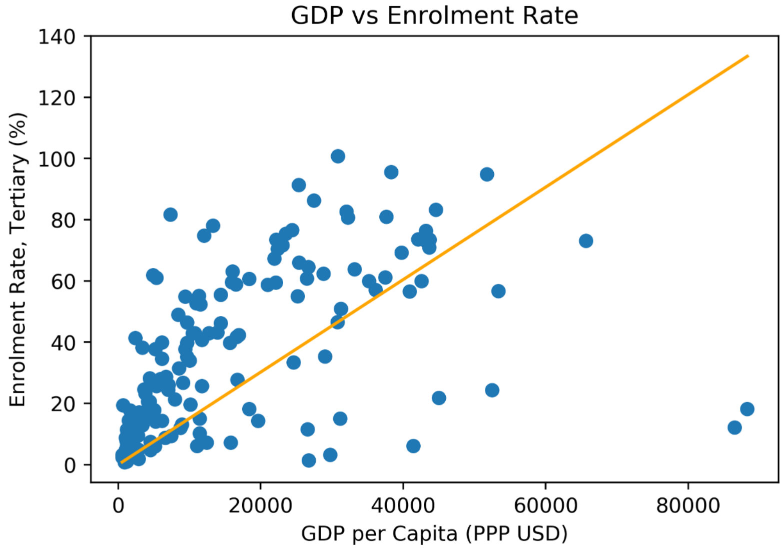
A more compact version, operating over all the datapoints at once:

```
In [9]: X = data["GDP per Capita (PPP USD)"].values
        Y = data["Enrolment Rate, Tertiary (%)"].values

        a = 0.0
        b = 0.0
        learning_rate = 1e-11

        for epoch in range(10):
            y_predicted = a * X + b
            a = a - learning_rate * ((y_predicted - Y)*X).sum()
            b = b - learning_rate * (y_predicted - Y).sum()
            rmse = np.sqrt(np.square(y_predicted - Y).mean())
            print("RMSE: " + str(rmse))

        plot_simple_linear_regression(X, Y, a, b)
```



The Gradient

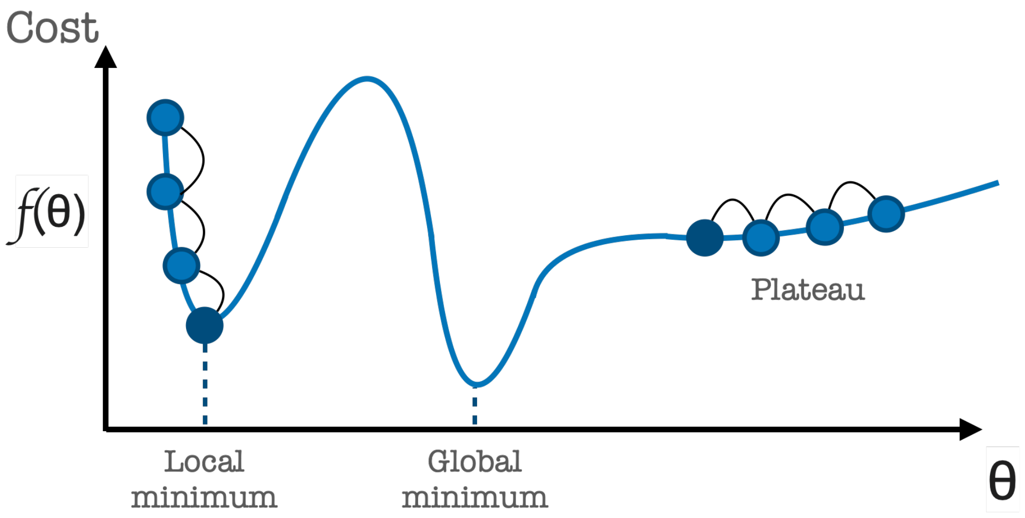
It may be more convenient to work with vector notation.

The gradient is a vector of all partial derivatives.

For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the gradient is

$$\nabla_\theta f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \frac{\partial f(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_n} \end{bmatrix}$$

Gradient Descent Pitfalls



- Starting with a “bad” random initialization point may cause the algorithm being stuck in a *local* (rather than *global*) *minimum*, or stop too early on a *plateau*
- Luckily, cost function for Linear Regression is *convex*

The Analytical Solution

Solving the single-variable linear regression with the analytical solution (normal equation)

$$X = \begin{bmatrix} x_1 & 1.0 \\ x_2 & 1.0 \\ \vdots & \vdots \\ x_M & 1.0 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} \qquad \theta = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$\nabla_\theta E(\theta) = X^T(X\theta - y) = 0$$

$$\implies \theta^* = (X^T X)^{-1} X^T y$$

The Analytical Solution

Solving the single-variable linear regression with the analytical solution (normal equation)

$$X = \begin{bmatrix} x_1 & 1.0 \\ x_2 & 1.0 \\ \vdots & \vdots \\ x_M & 1.0 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} \qquad \theta = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$\nabla_\theta E(\theta) = X^T(X\theta - y) = 0$$

$$\implies \theta^* = (X^T X)^{-1} X^T y$$

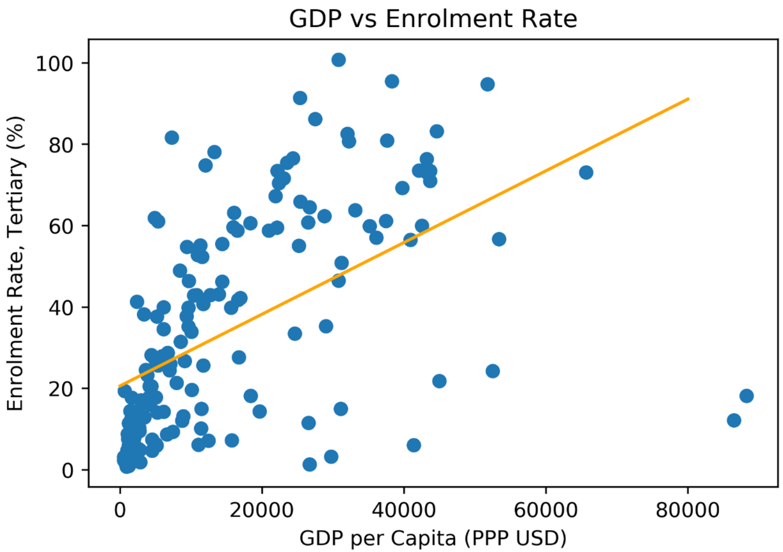
Great for directly finding the optimal parameter values.
Not so great for large problems: matrix inversion has cubic complexity $O(n^3)$.

Analytical Solution with Scikit-Learn

```python
from sklearn.linear_model import LinearRegression

model = LinearRegression(fit_intercept=True)
X = data["GDP per Capita (PPP USD)"].values.reshape(-1,1)
Y = data["Enrolment Rate, Tertiary (%)"]
model.fit(X, Y)

mse = np.square(Y - model.predict(X)).mean()
print("RMSE: " + str(np.sqrt(mse)))
```



RMSE: 22.630490998345973

Multiple Linear Regression

We normally use more than 1 input feature in our model

Output label

Input features

	GDP per Capita (PPP USD)	Population Density (persons per sq km)	Population Growth Rate (%)	Urban Population (%)	Life Expectancy at Birth (avg years)	Fertility Rate (births per woman)	Infant Mortality (deaths per 1000 births)	Unemployment, Total (%)	Estimated Control of Corruption (scale -2.5 to 2.5)	Estimated Government Effectiveness (scale -2.5 to 2.5)	Internet Users (%)	Enrolment Rate, Tertiary (%)
0	1560.67	44.62	2.44	23.86	60.07	5.39	71.0	8.5	-1.41	-1.40	5.45	3.33
1	9403.43	115.11	0.26	54.45	77.16	1.75	15.0	14.2	-0.72	-0.28	54.66	54.85
2	8515.35	15.86	1.89	73.71	70.75	2.83	25.6	10.0	-0.54	-0.55	15.23	31.46
3	19640.35	200.35	1.03	29.87	75.50	2.12	9.2	8.4	1.29	0.48	83.79	14.37
4	12016.20	14.88	0.88	92.64	75.84	2.20	12.7	7.2	-0.49	-0.25	55.80	74.83

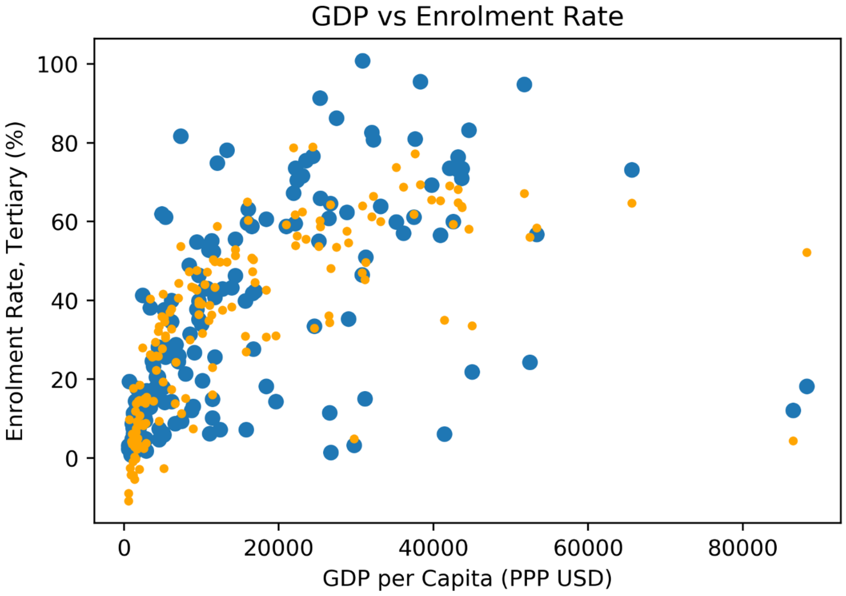
$$y^{(i)} = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_3 x_3^{(i)} + \cdots + \theta_N x_N^{(i)} + \theta_{N+1}$$

Multiple Linear Regression

```
model = LinearRegression(fit_intercept=True)
X = data.copy().drop(["Country Name",
                      "Enrolment Rate, Tertiary (%)"],
                     axis=1)
Y = data["Enrolment Rate, Tertiary (%)"]

model.fit(X, Y)

mse = np.square(Y - model.predict(X)).mean()
print("RMSE: " + str(np.sqrt(mse)))
```



RMSE: 14.40196

Exploring the Parameters

model.coef_ now contains optimized coefficients for each of the input features

model.intercept_ contains the intercept

```
headers=list(X)
coefficients = []
for i in range(len(headers)):
    coefficients.append({"Property": headers[i],
                         "coefficient": model.coef_[i]})
pd.DataFrame(coefficients)
```

	Property	coefficient
0	GDP per Capita (PPP USD)	0.000236
1	Population Density (persons per sq km)	-0.012085
2	Population Growth Rate (%)	-12.605788
3	Urban Population (%)	0.361150
4	Life Expectancy at Birth (avg years)	0.584344
5	Fertility Rate (births per woman)	5.795337
6	Infant Mortality (deaths per 1000 births)	-0.092305
7	Unemployment, Total (%)	-0.312737
8	Estimated Control of Corruption (scale -2.5 to...	-5.153427
9	Estimated Government Effectiveness (scale -2.5...	4.035069
10	Internet Users (%)	0.149982

Exploring the Parameters

The coefficients are only comparable if we standardize the input features first.

```
Z = pd.DataFrame(data, columns=["GDP per Capita (PPP USD)"])
Z_scaled = preprocessing.scale(Z)
```

	Z	Z_scaled
0	1560.67	-0.859361
1	9403.43	-0.379854
2	8515.35	-0.434152
3	19640.35	0.246031
4	12016.20	-0.220110

	Property	coefficient
0	GDP per Capita (PPP USD)	3.865747
1	Population Density (persons per sq km)	-2.748875
2	Population Growth Rate (%)	-14.487085
3	Urban Population (%)	8.359783
4	Life Expectancy at Birth (avg years)	5.126343
5	Fertility Rate (births per woman)	8.122616
6	Infant Mortality (deaths per 1000 births)	-2.126688
7	Unemployment, Total (%)	-2.385280
8	Estimated Control of Corruption (scale -2.5 to...	-5.023631
9	Estimated Government Effectiveness (scale -2.5...	3.714866
10	Internet Users (%)	4.329112

Polynomial Features

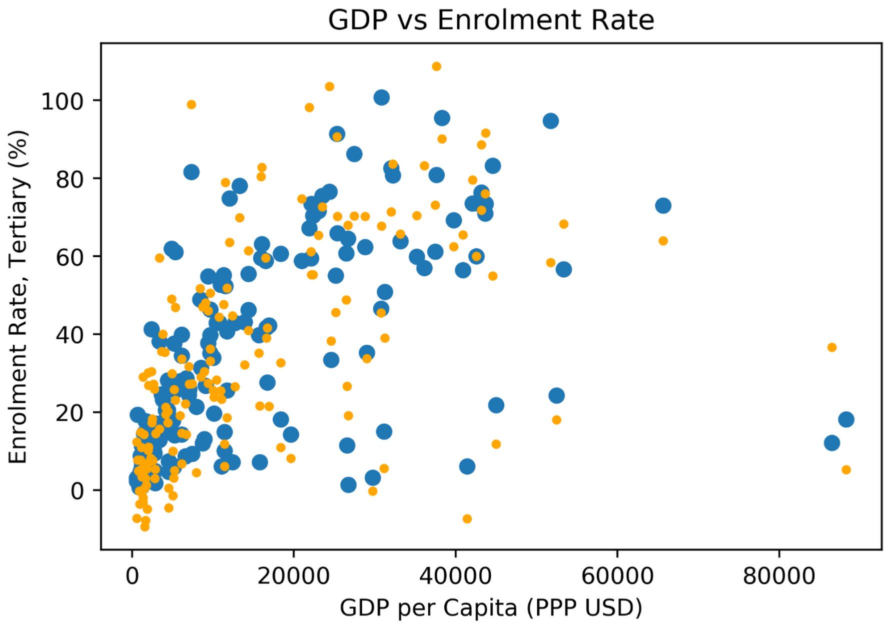
Polynomial combinations of the features.

With degree 2, features $[z_1, z_2]$

would become $[1, z_1, z_2, z_1^2, z_1 z_2, z_2^2]$

```python
from sklearn.preprocessing import PolynomialFeatures

model = LinearRegression(fit_intercept=True)
X = data.copy().drop(["Country Name",
                      "Enrolment Rate, Tertiary (%)"],
                     axis=1)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
Y = data["Enrolment Rate, Tertiary (%)"]
model.fit(X_poly, Y)
```



RMSE: 13.6692

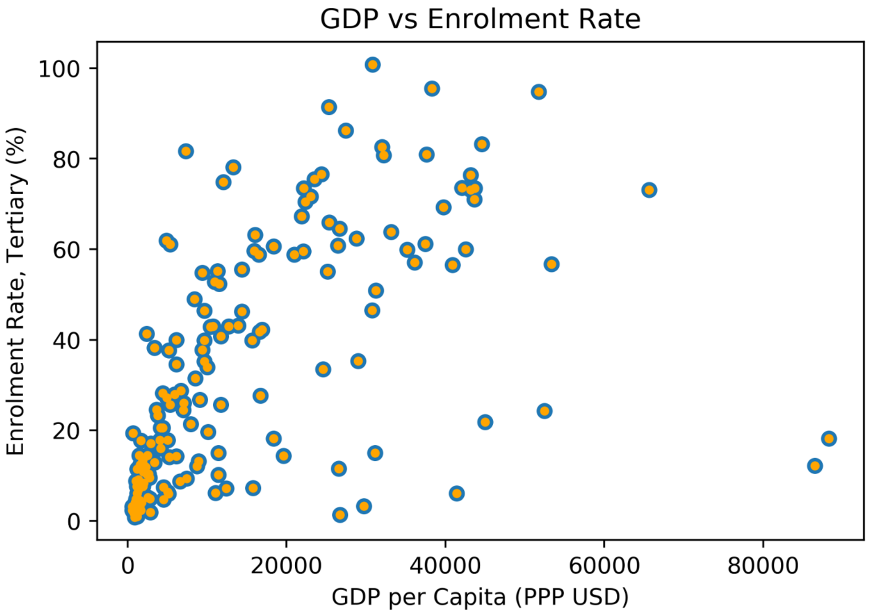
Polynomial Features

With 3rd degree polynomial features, the linear regression model now has 364 input features:

$$\frac{(n+d)!}{n!d!} = \frac{(11+3)!}{11!3!} = 364$$

```python
model = LinearRegression(fit_intercept=True)
X = data.copy().drop(["Country Name",
                      "Enrolment Rate, Tertiary (%)"],
                     axis=1)
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)
Y = data["Enrolment Rate, Tertiary (%)"]
model.fit(X_poly, Y)

mse = np.square(Y - model.predict(X_poly)).mean()
print("RMSE: " + str(np.sqrt(mse)))
```

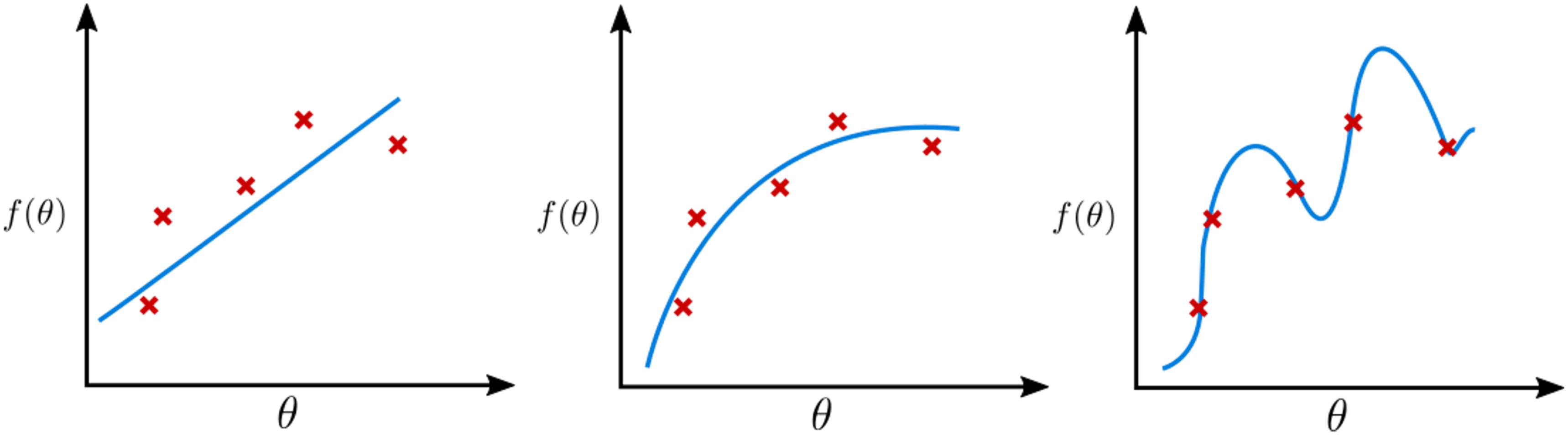


RMSE: 0.00018

Overfitting

There are twice as many features/parameters as there are datapoints in the whole dataset.

This can easily lead to **overfitting**:



Dataset Splits

Training Set

For training your models, fitting the parameters

Development Set

For continuous evaluation and hyperparameter selection

Test Set

For realistic evaluation once the training and tuning is done

Stratified Sampling

Making sure the proportion of classes is kept the same in the splits

Training Set

For training your models, fitting the parameters

Development Set

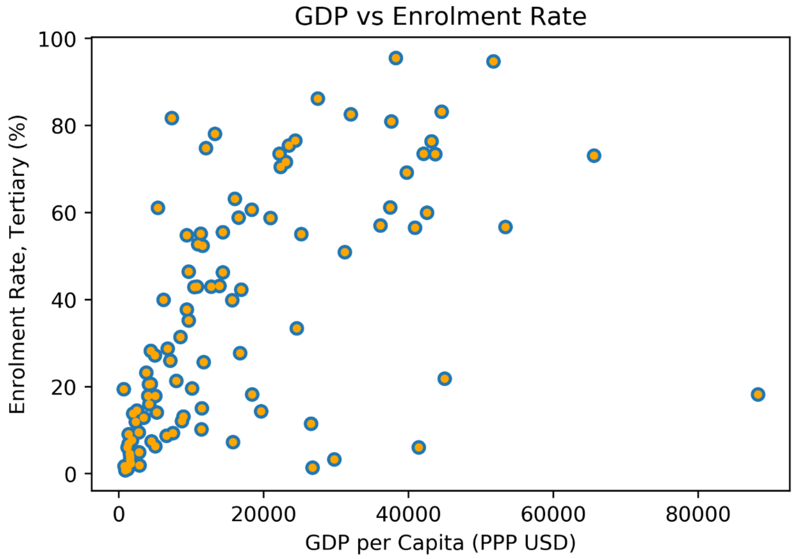
For continuous evaluation and hyperparameter selection

Test Set

For realistic evaluation once the training and tuning is done

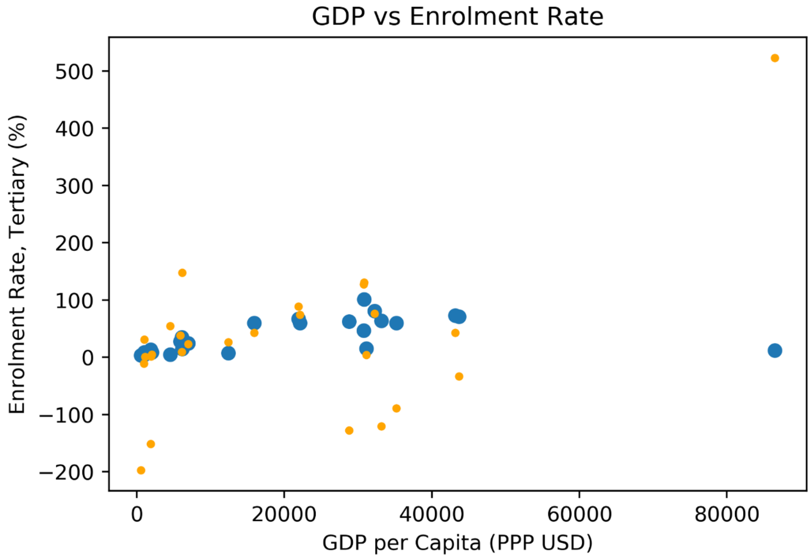
Overfitting

Training set
3rd degree polynomial features



RMSE: 1.1422e-07

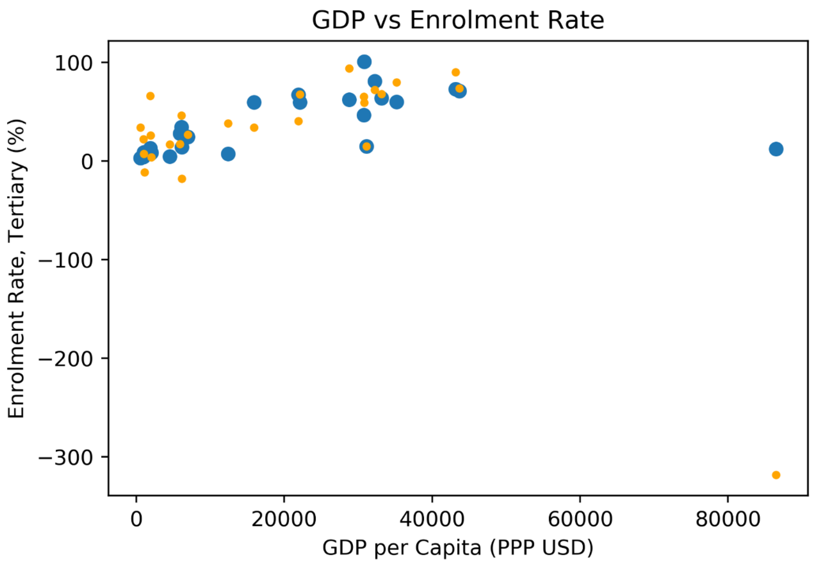
Development / Validation set
3rd degree polynomial features



RMSE: 133.4137

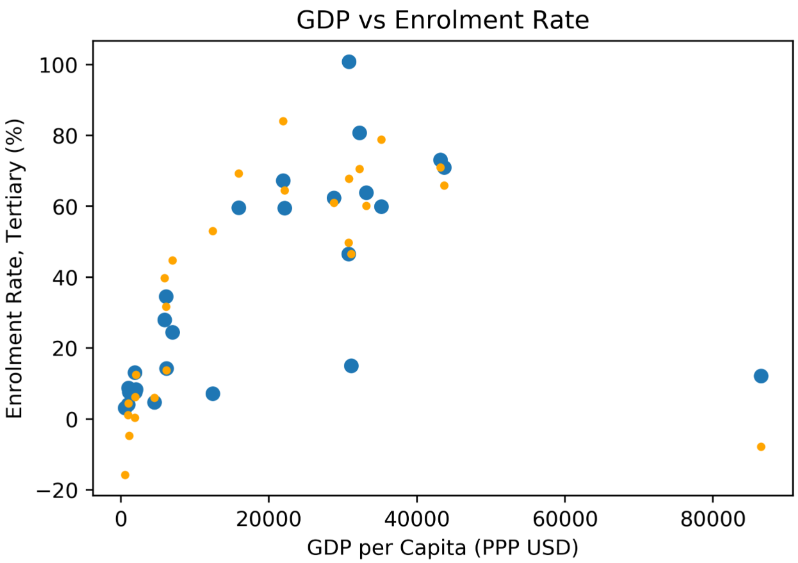
Overfitting

Development set
2nd degree polynomial features



RMSE: 68.4123

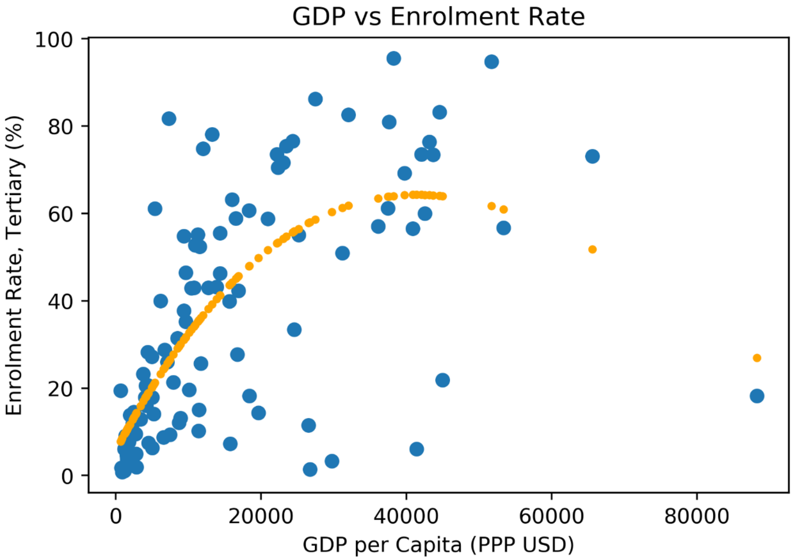
Development set
1st degree polynomial features



RMSE: 16.1414

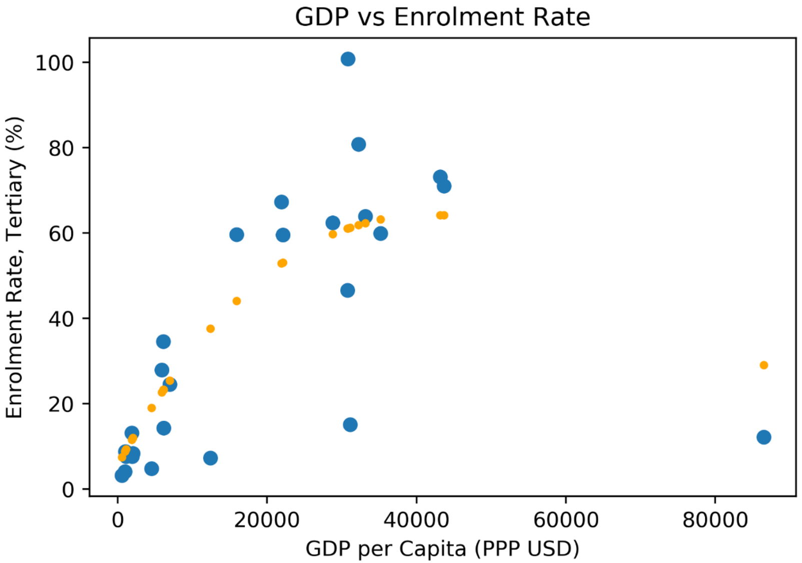
Overfitting

<u>Training set</u>
1 input feature (GDP)
3rd degree polynomial features



RMSE: 19.8130

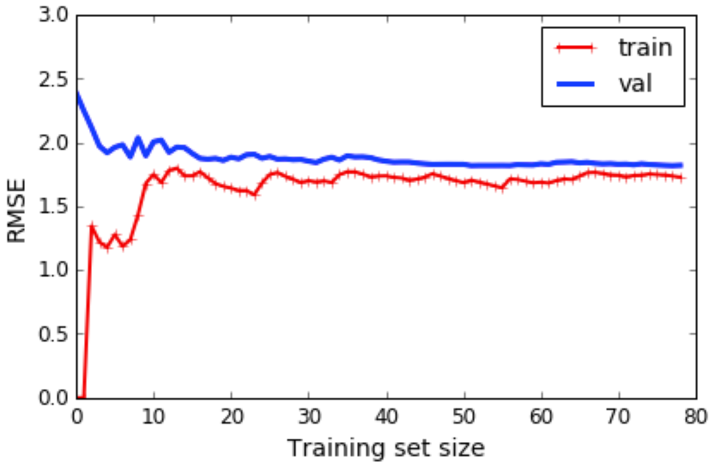
<u>Development set</u>
1 input feature (GDP)
3rd degree polynomial features



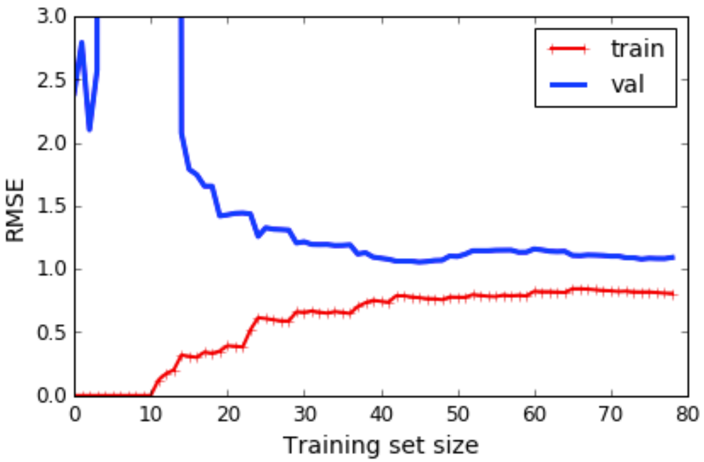
RMSE: 15.9834

How to Spot Overfitting

Learning curves with 1st degree polynomial features



Learning curves with 10th degree polynomial features



https://github.com/ageron/handson-ml/

Ways to Prevent Overfitting

Regularize (constrain) the model, so that it has fewer degrees of freedom.
E.g. reduce the number of polynomial degrees, or *constrain the weights by adding a regularization term to the cost function:*

- **Ridge Regression** cost function: $J(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$
  - i.e., adding l_2-norm of the weight vector as the regularization term
  - *alpha* controls the amount of regularization: *alpha*=0 → Linear Regression

- **Lasso Regression** cost function: $J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$ i.e., l_1-norm

- **Elastic Net** cost function: $J(\theta) = MSE(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} \theta_i^2$ i.e. a mix of Ridge and Lasso controlled by ratio *r*

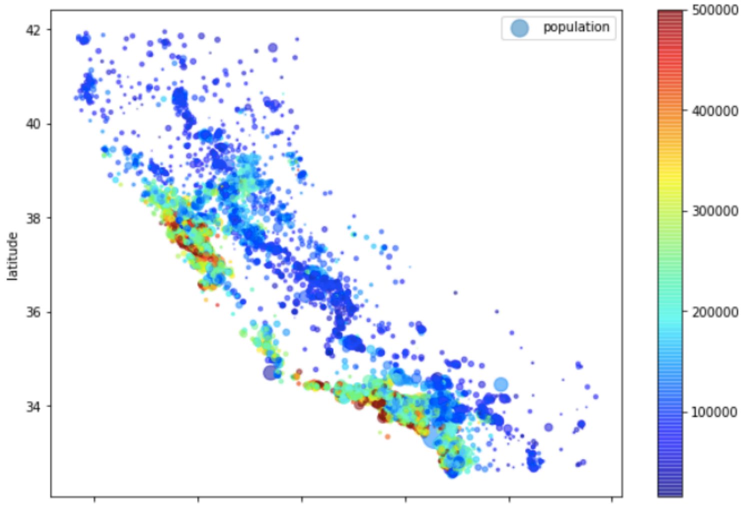
Practical 1

Data

- **California House Prices Dataset** containing information on a number of independent variables about the block groups in California from 1990 Census

- **Dependent variable:** house price

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.385070
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000



Your task: Learning objectives

- Load the dataset
- Understand the data, the attributes and their correlations
- Split the data into training and test sets
- Apply normalisation, scaling and other transformations to the attributes if needed
- Build a machine learning model
- Evaluate the model and investigate the errors
- Tune your model to improve performance

Practical 1 Logistics

- Data and code for Practical 1 can be found on: Github (https://github.com/ekochmar/cl-datasci-pnp-2021/tree/master/DSPNP_practical1)

- Practical session is on Tuesday 10 November, 3-4pm, over Zoom

- At the practical, be prepared to discuss the task and answer the questions about the code to get a ‘pass’

- After the practical, upload your solutions (Jupyter notebook or Python code) to Moodle