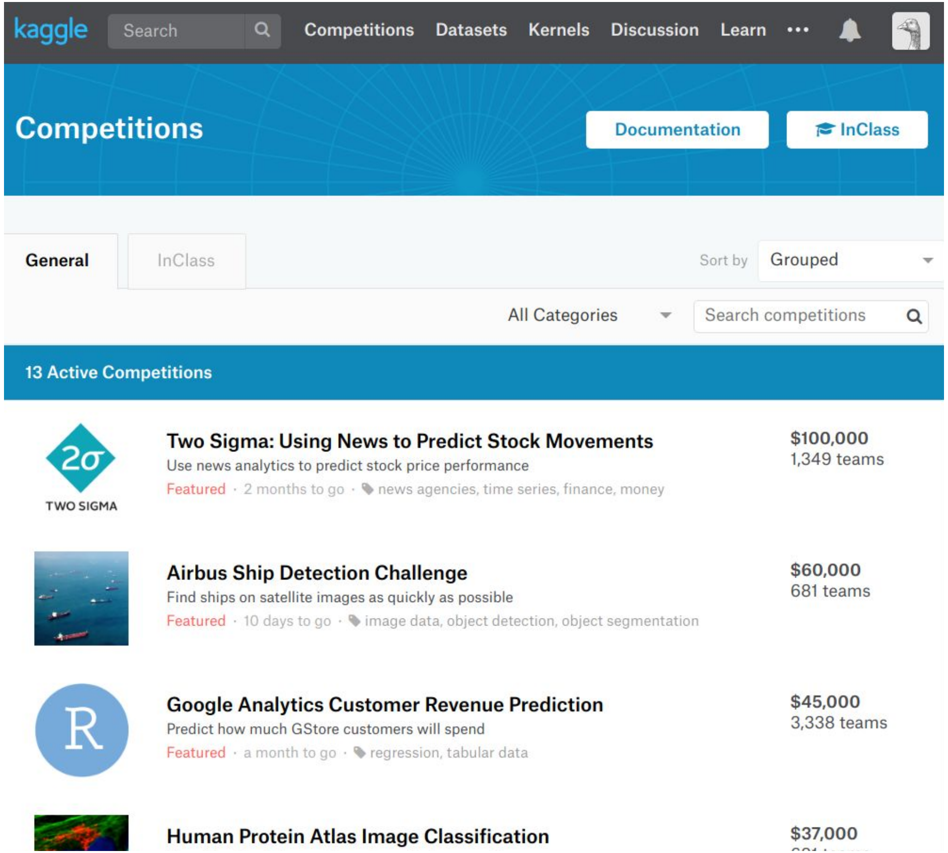
Data Science: Principles and Practice

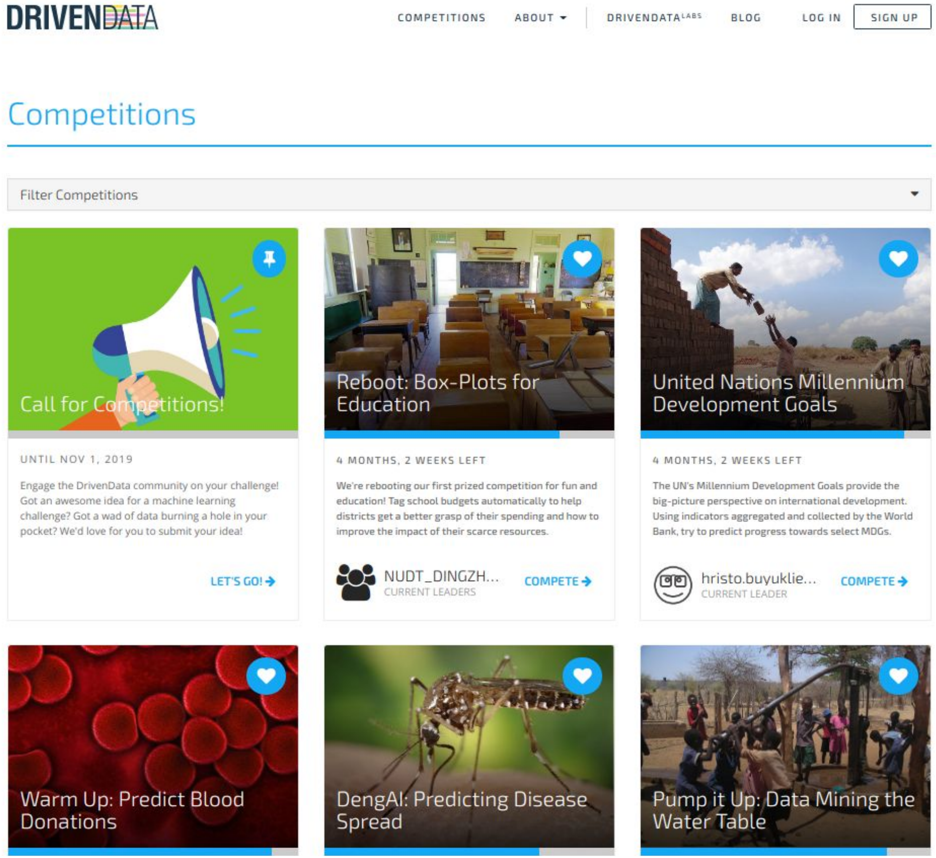
Lecture 2: Linear Regression

Marek Rei

UNIVERSITY OF CAMBRIDGE



kaggle.com



drivendata.org

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

Supervised Learning

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

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

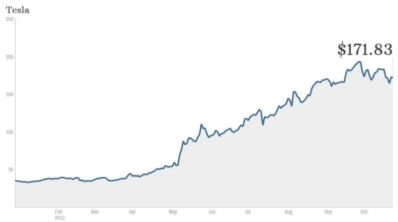
Known (desired) outputs: $y_1, y_2, y_3, y_4, ..., y_n$

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

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

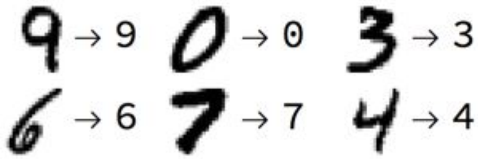
Continuous vs Discrete Problems

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

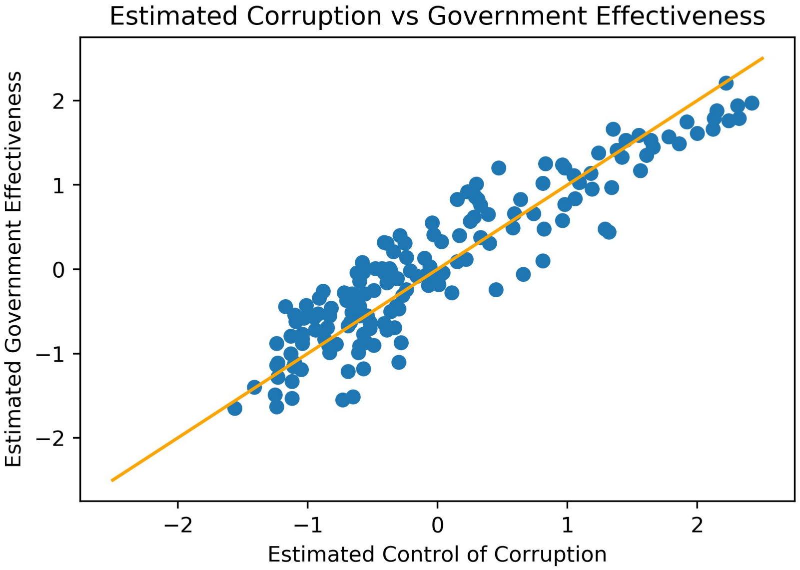
Regression or classification?

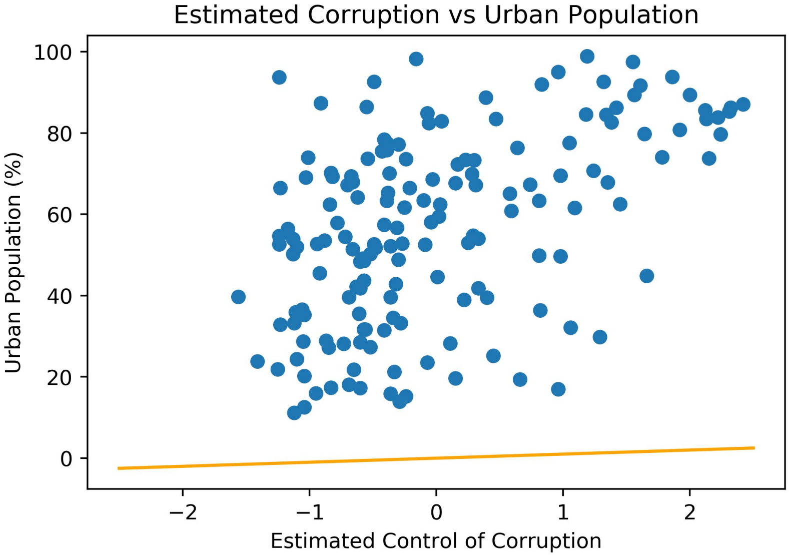
Model the salary of baseball players based on their game statistics
Find what object is on a photo
Predicting election results

Simplest Possible Linear Model

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

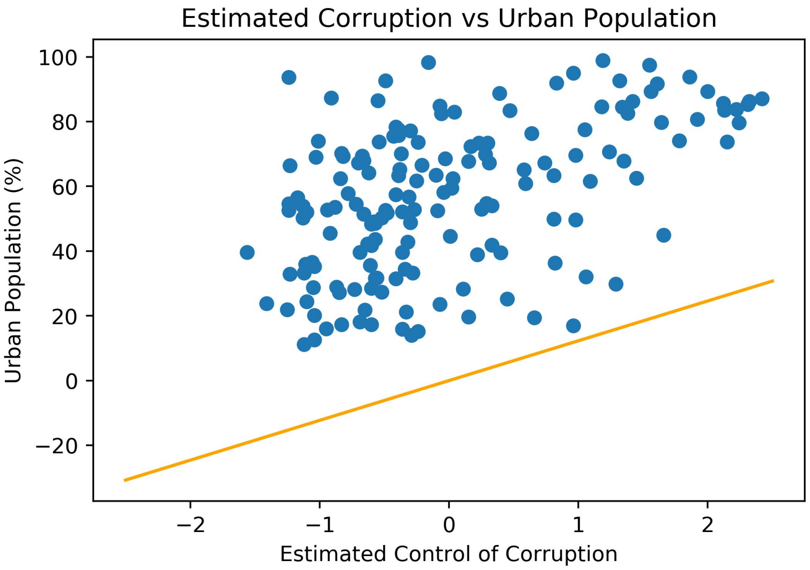
$$y = x$$



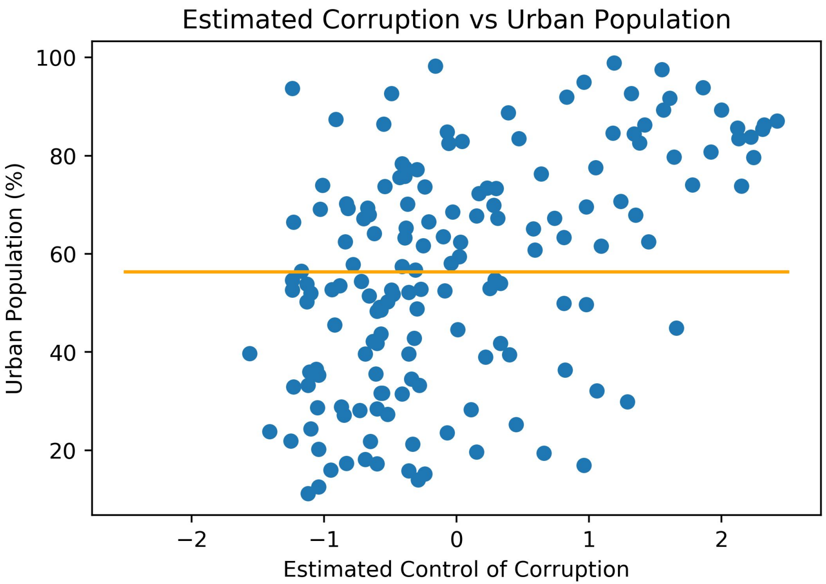


(Still Too Simple) Linear Models

$$y = ax$$

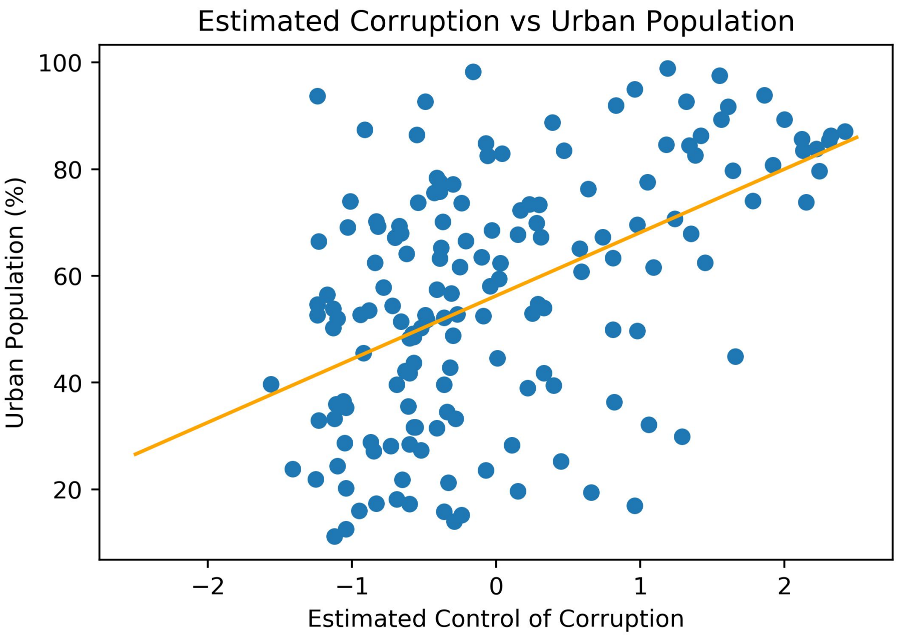


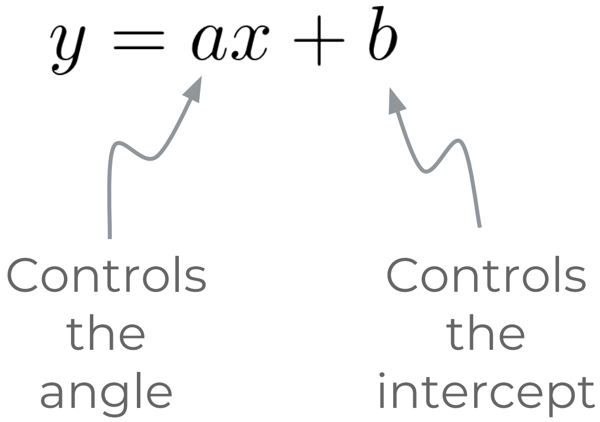
$$y = b$$



Linear Regression

Better linear model: $$y = ax + b$$





Linear Regression

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

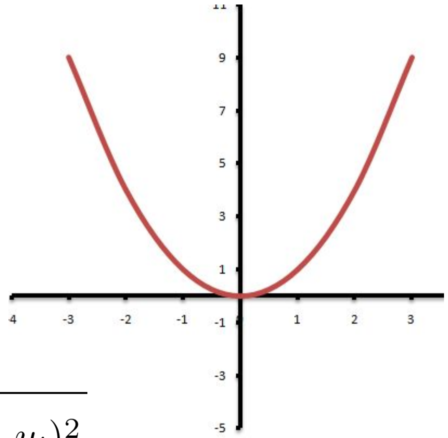
$$y : \text{Enrolment Rate}$$

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

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.

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

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

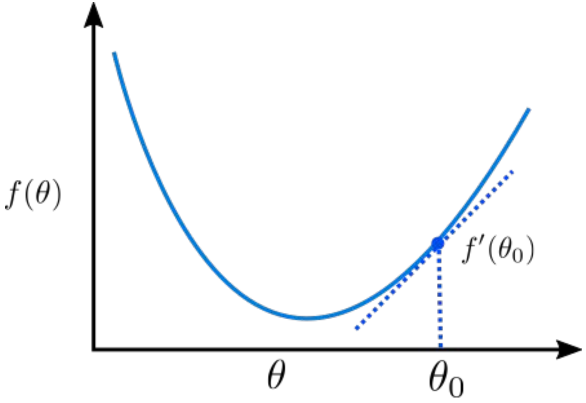
$$RMSE = \sqrt{\frac{\sum_{i=1}^{M}(\hat{y}_i - y_i)^2}{M}}$$

- 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

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

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

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

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

Gradient Descent

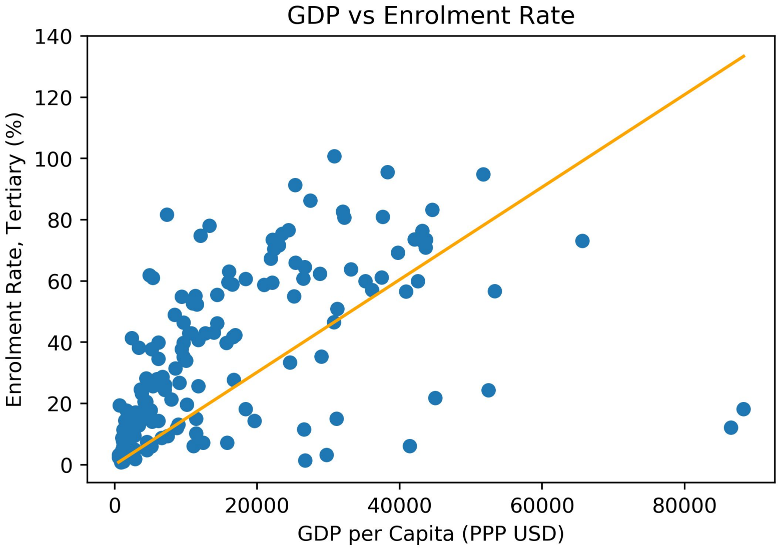
Implementing gradient descent by hand

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)
```



Gradient Descent

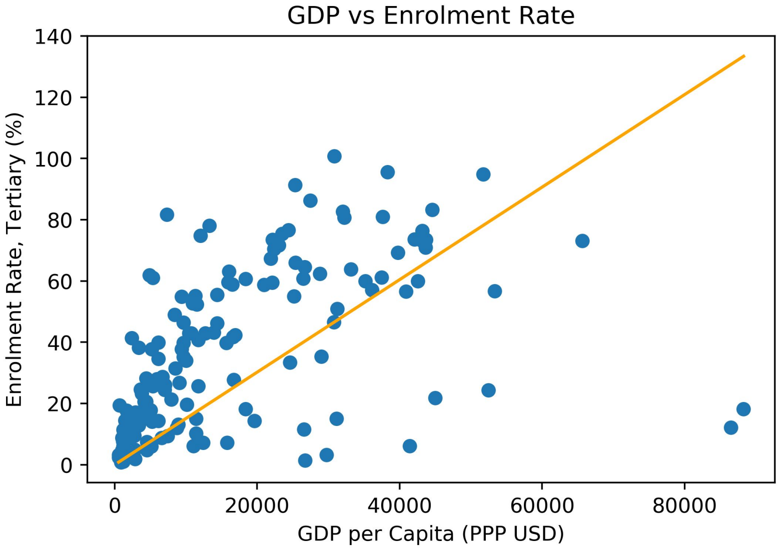
A more compact version, operating over all the datapoints at once.

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 can be more convenient to work with vector notation.

The gradient is a vector of all partial derivatives.

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

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

The Analytical Solution

Solving the single-variable linear regression with the analytical solution

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

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

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

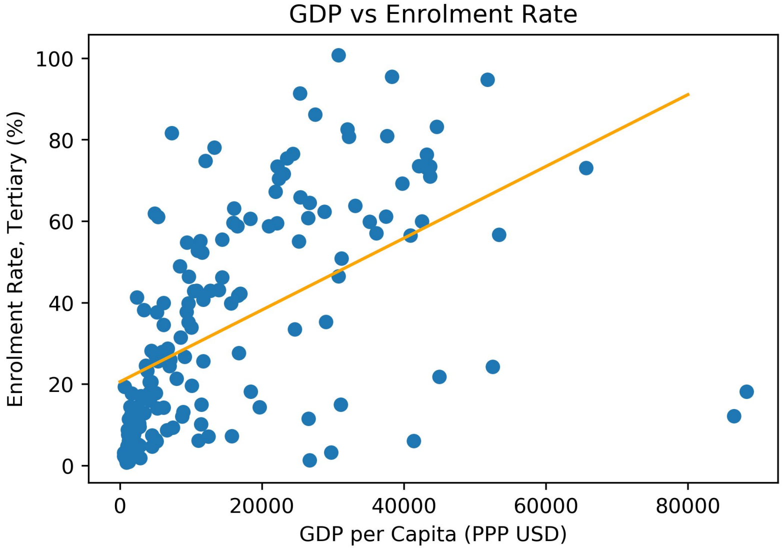
Great for directly finding the optimal parameter values.
Not so great for large problems: matrix inversion has cubic complexity.

Analytical Solution with Scikit-Learn

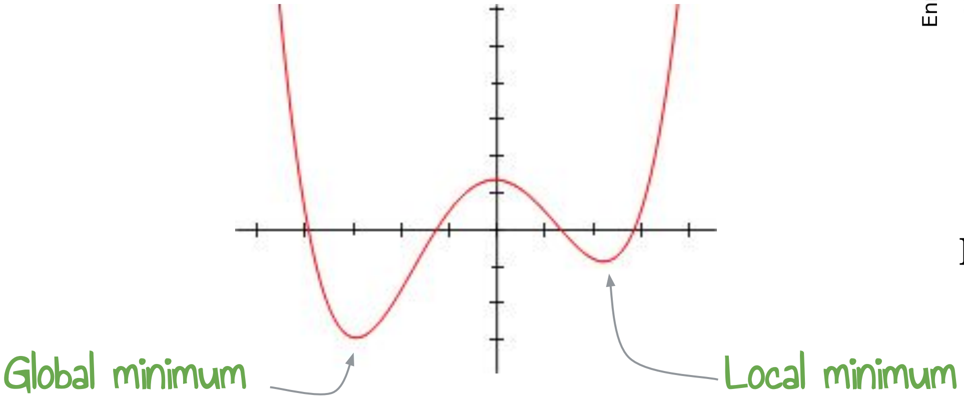
```
from sklearn.linear_model import LinearRegression

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

Input features

Output label

	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

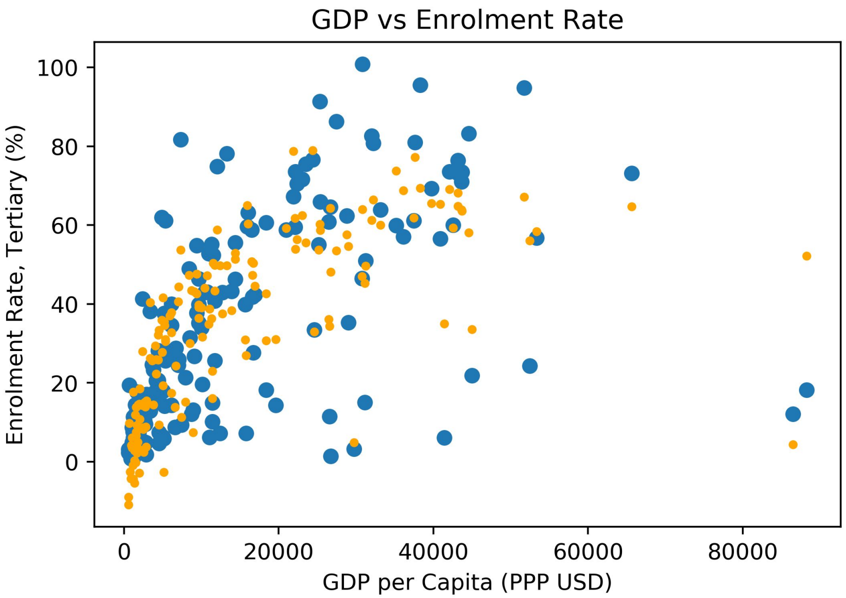
$$y^{(i)} = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_3 x_3^{(i)} + \cdots + \theta_N x_N^{(i)} + \theta_{N+1}$$

Multiple Linear Regression

```python
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

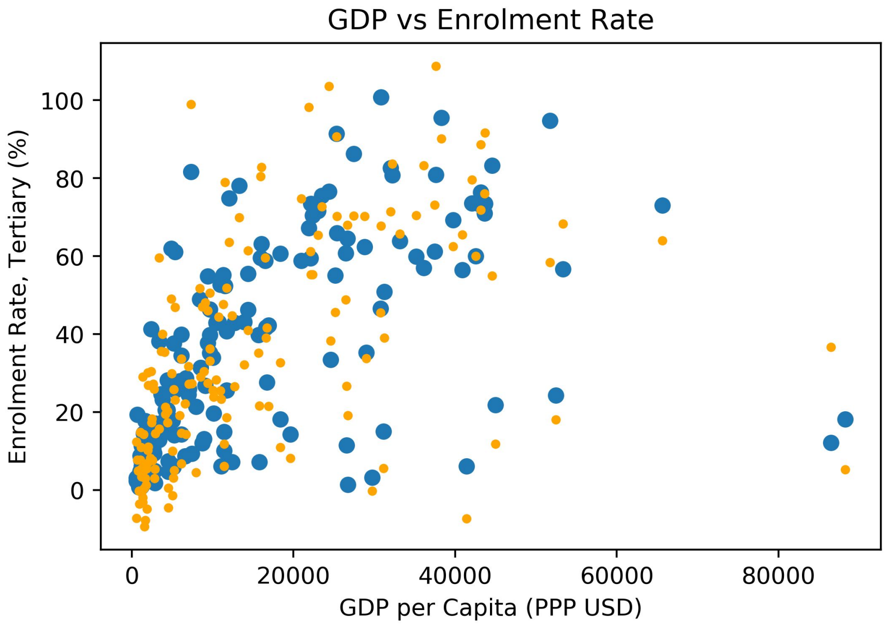
Polynomial combinations of the features.

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

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

```
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)
```



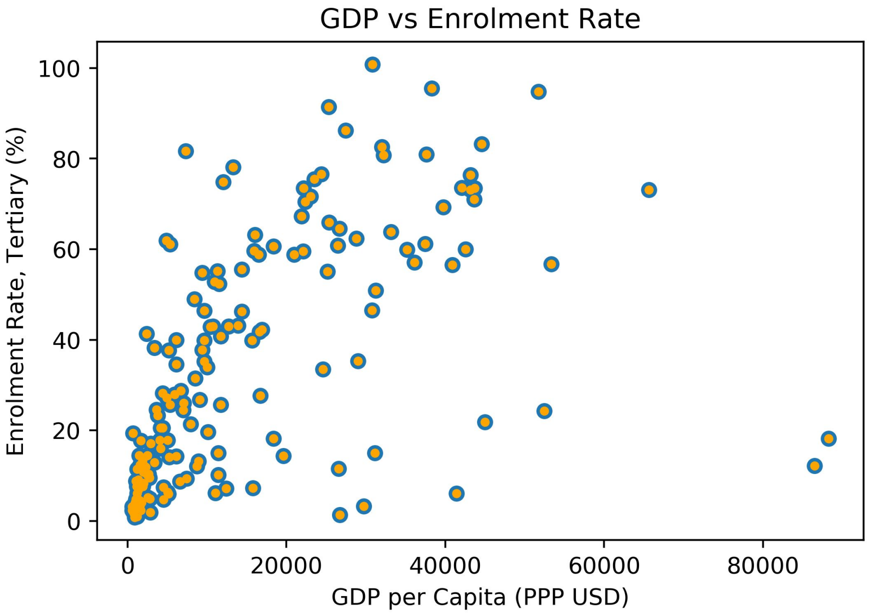
RMSE: 13.6692

Polynomial Features

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

```
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)))
```

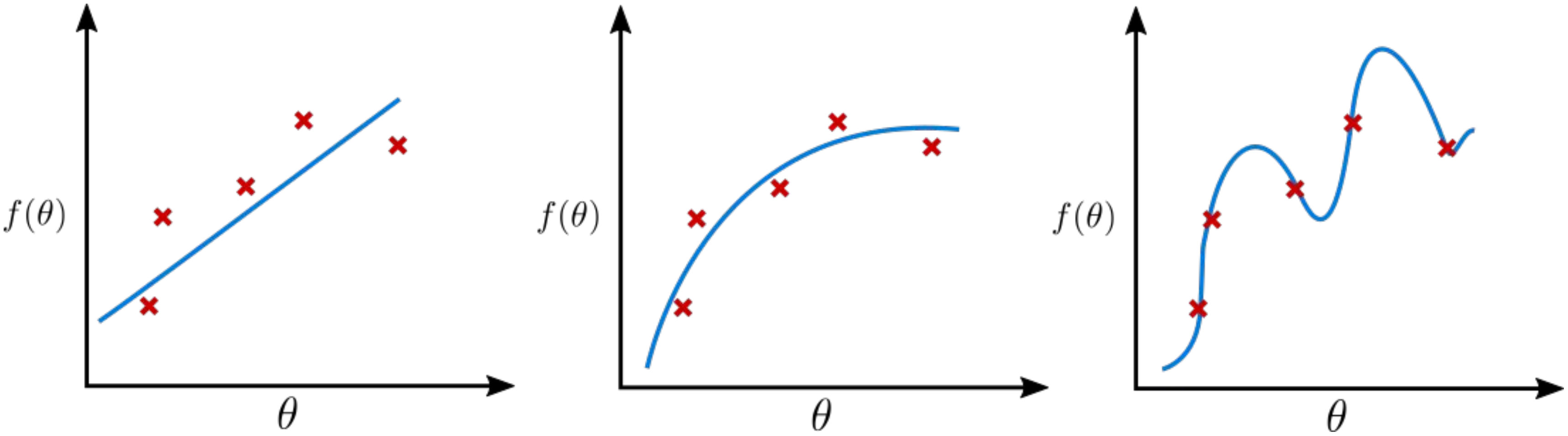


RMSE: 0.00018

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

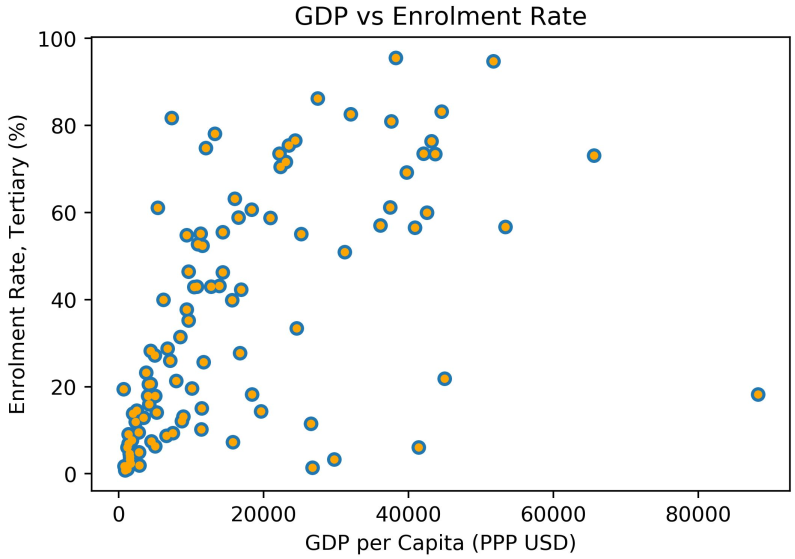
For continuous evaluation and hyperparameter selection

Test Set

For realistic evaluation once the training and tuning is done

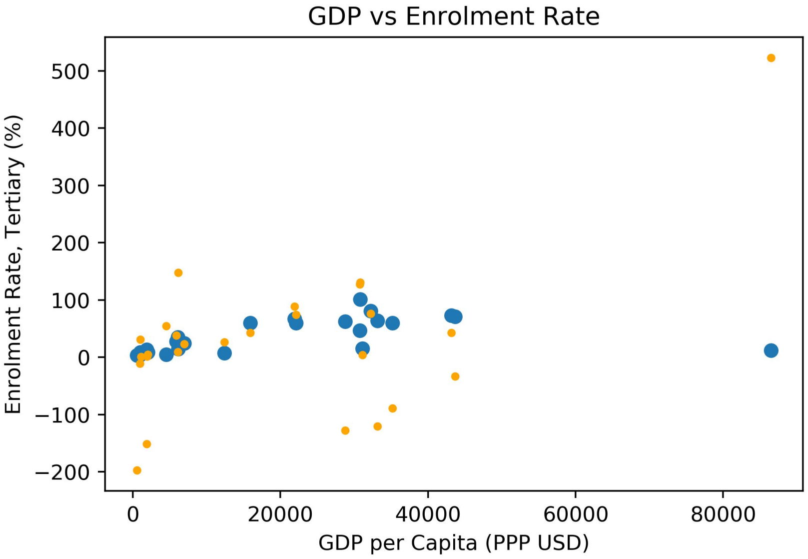
Overfitting

Training set
3rd degree polynomial features



RMSE: 1.1422e-07

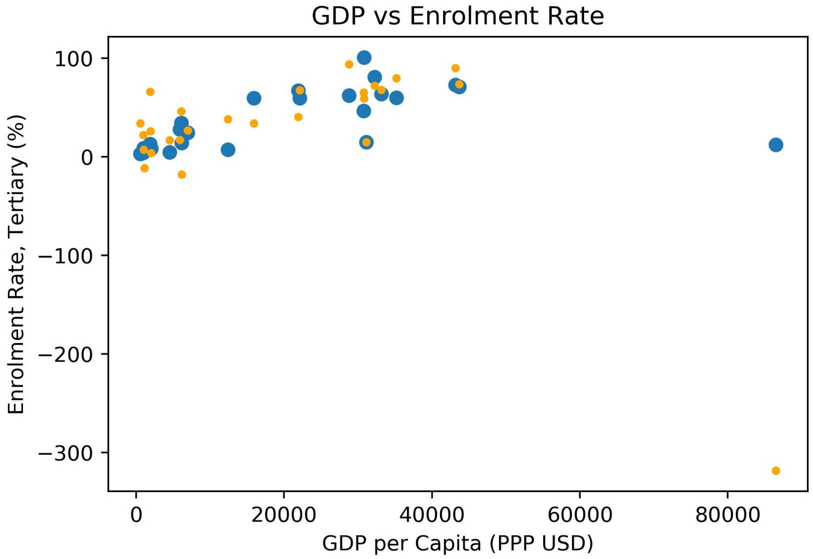
Development set
3rd degree polynomial features



RMSE: 133.4137

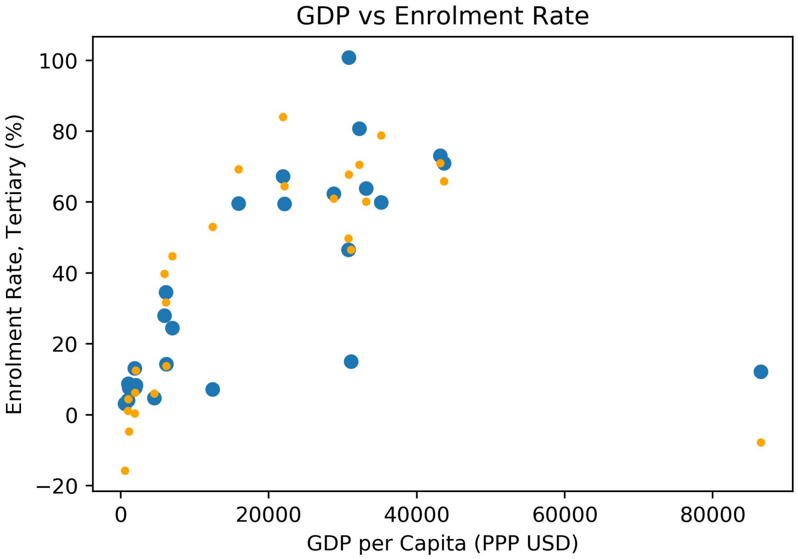
Overfitting

Development set
2nd degree polynomial features



RMSE: 68.4123

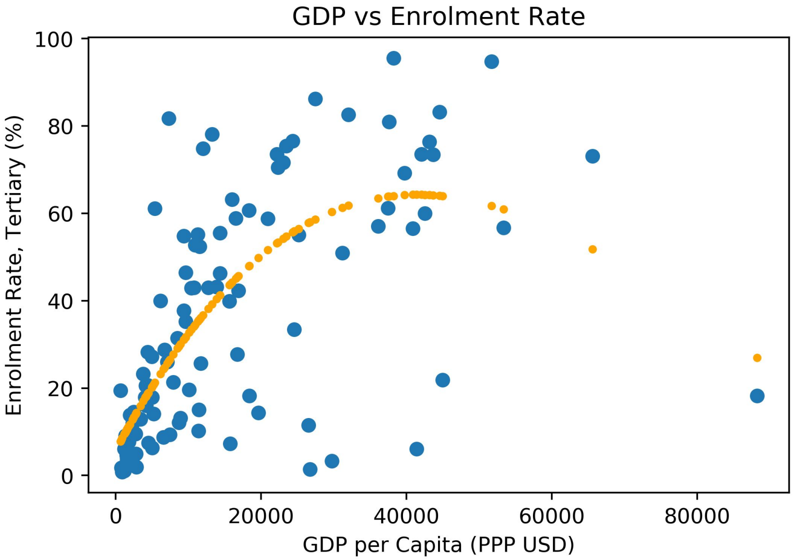
Development set
1st degree polynomial features



RMSE: 16.1414

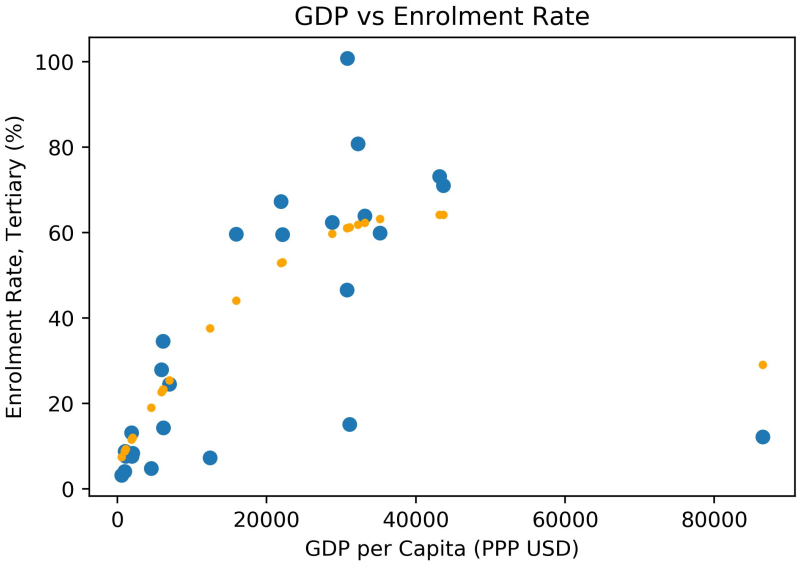
Overfitting

Training set
1 input feature (GDP)
3rd degree polynomial features



RMSE: 19.8130

Development set
1 input feature (GDP)
3rd degree polynomial features



RMSE: 15.9834

GDP vs Enrolment Rate