Member-only story

# Normalization vs Standardization, which one is better

In this tutorial let us see which one is the best feature engineering technique of them all.

Tanu N Prabhu · Follow

Published in Towards Data Science

5 min read · Apr 22, 2020



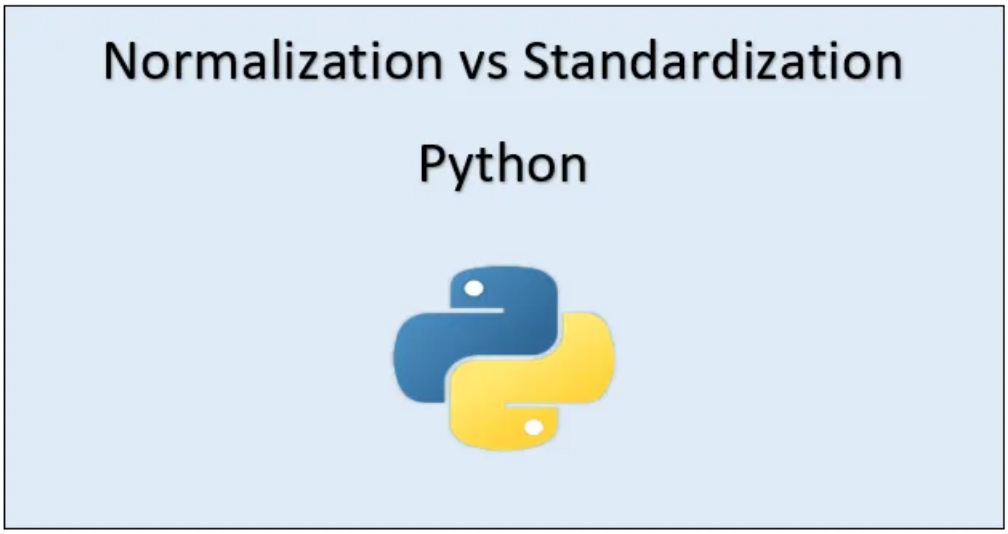



Image credits to Author (Tanu Nanda Prabhu)

As we all know **feature engineering is a problem of transforming raw data into a dataset.** There are various feature engineering techniques available out there. The two most widely used and commonly confused feature engineering techniques are:

- **Normalization**
- **Standardization**

Today on this beautiful day or night we will explore both of these techniques and see some of the common assumptions made by data analysts while solving a data science problem. Also, the whole code for this tutorial can be found on my GitHub Repository below

**Tanu-N-Prabhu/Python**

Permalink Dismiss GitHub is home to over 40 million developers working together to host and review code, manage...

github.com

## Normalization

### Theory

Normalization is the process of converting a numerical feature into a standard range of values. The range of values might be either [-1, 1] or [0, 1]. For example, think that we have a data set comprising two features named "**Age**" and the "**Weight**" as shown below:

| | Age | Weight |
|---|---|---|
| **0** | 5 | 5 |
| **1** | 10 | 8 |
| **2** | 15 | 13 |
| **3** | 20 | 17 |
| **4** | 25 | 27 |
| **5** | 30 | 33 |
| **6** | 35 | 36 |
| **7** | 40 | 40 |
| **8** | 45 | 50 |
| **9** | 50 | 70 |
| **10** | 55 | 78 |
| **11** | 60 | 80 |
| **12** | 65 | 100 |
| **13** | 70 | 103 |
| **14** | 75 | 108 |
| **15** | 80 | 109 |
| **16** | 85 | 113 |
| **17** | 90 | 120 |
| **18** | 95 | 123 |
| **19** | 100 | 130 |

Image Credits to Author (Tanu Nanda Prabhu)

Suppose the actual range of a feature named "**Age**" is **5** to **100.** We can normalize these values into a range of **[0, 1]** by subtracting **5** from every value of the "**Age**" column and then dividing the result by **95** (100–5). To make things clear in your brain we can write the above as a formula.

$$\bar{x}^{(j)} = \frac{x^{(j)} - min^{(j)}}{max^{(j)} - min^{(j)}},$$

Image credits to The Hundred-Page Machine Learning Book by Andriy Burkov

where **min^(j)** and **max^(j)** are the **minimum** and the **maximum** values of the feature j in the dataset.

#### Implementation

Now that you know the theory behind it let's now see how to put it into production. As normal there are two ways to implement this: **Traditional Old school manual method** and the other using **`sklearn preprocessing`** library. Today let's take the help of `sklearn` library to perform normalization.

**Using sklearn preprocessing — Normalizer**

Before feeding the "**Age**" and the "**Weight**" values directly to the method we need to convert these data frames into a `numpy` array. To do this we can use the `to_numpy()` method as shown below:

```
# Storing the columns Age values into X and Weight as Y

X = df['Age']
y = df['Weight']
X = X.to_numpy()
y = y.to_numpy()
```

The above step is very important because of both the `fit()` and the `transform()` method works only on an array.

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer().fit([X])
normalizer.transform([X])
```

```
array([[0.01866633, 0.03733267, 0.055999  , 0.07466534, 0.09333167,
        0.11199801, 0.13066434, 0.14933068, 0.16799701, 0.18666335,
        0.20532968, 0.22399602, 0.24266235, 0.26132869, 0.27999502,
        0.29866136, 0.31732769, 0.33599403, 0.35466036, 0.3733267 ]])
```

Image credits to Author (Tanu Nanda Prabhu)

```
normalizer = Normalizer().fit([y])
normalizer.transform([y])
```

```
array([[0.01394837, 0.02231739, 0.03626577, 0.04742446, 0.07532121,
        0.09205925, 0.10042828, 0.11158697, 0.13948372, 0.1952772 ,
        0.2175946 , 0.22317395, 0.27896743, 0.28733646, 0.30128483,
        0.3040745 , 0.3152332 , 0.33476092, 0.34312994, 0.36265766]])
```

Image credits to Author (Tanu Nanda Prabhu)

As seen above both the arrays have the values in the range **[0, 1]**. More details about the library can be found below:

**6.3. Preprocessing data - scikit-learn 0.22.2 documentation**

The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw...

scikit-learn.org

#### When should we actually normalize the data?

Although normalization is not mandatory or a requirement (must-do thing). There are two ways it can help you which is

- Normalizing the data will **increase the speed of learning.** It will increase the speed both during building (training) and testing the data. Give it a try!!
- It will avoid **numeric overflow**. What is really means is that normalization will ensure that our inputs are roughly in a small relatively small range. This will avoid problems because computers usually have problems dealing with very small or very large numbers.

## Standardization

#### Theory

Standardization or **z-score normalization** or **min-max scaling** is a technique of rescaling the values of a dataset such that they have the properties of a standard normal distribution with $\mu = 0$ (mean — average values of the feature) and $\sigma = 1$ (standard deviation from the mean). This can be written as:

$$\hat{x}^{(j)} = \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}}.$$

Image credits to The Hundred-Page Machine Learning Book by Andriy Burkov

#### Implementation

Now there are plenty of ways to implement standardization, just as normalization, we can use `sklearn` library and use `StandardScalar` method as shown below:

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit_transform([X])
sc.transform([X])

sc.fit_transform([y])
sc.transform([y])
```

You can read more about the library from below:



#### Z-Score Normalization

Similarly, we can use the pandas `mean` and `std` to do the needful

```
# Calculating the mean and standard deviation

df = (df - df.mean())/df.std()
print(df)
```

| | Age | Weight |
|---|---|---|
| 0 | -1.605793 | -1.458724 |
| 1 | -1.436762 | -1.389426 |
| 2 | -1.267731 | -1.273929 |
| 3 | -1.098701 | -1.181531 |
| 4 | -0.929670 | -0.950538 |
| 5 | -0.760639 | -0.811942 |
| 6 | -0.591608 | -0.742644 |
| 7 | -0.422577 | -0.650247 |
| 8 | -0.253546 | -0.419253 |
| 9 | -0.084515 | 0.042734 |
| 10 | 0.084515 | 0.227529 |
| 11 | 0.253546 | 0.273727 |
| 12 | 0.422577 | 0.735714 |
| 13 | 0.591608 | 0.805012 |
| 14 | 0.760639 | 0.920509 |
| 15 | 0.929670 | 0.943608 |
| 16 | 1.098701 | 1.036006 |
| 17 | 1.267731 | 1.197701 |
| 18 | 1.436762 | 1.266999 |
| 19 | 1.605793 | 1.428694 |

Image credits to Author (Tanu Nanda Prabhu)

#### Min-Max scaling

Here we can use pandas `min` and `max` to do the needful

```
# Calculating the minimum and the maximum

df = (df-df.min())/(df.max()-df.min())
print(df)
```

| | Age | Weight |
|---|---|---|
| 0 | 0.000000 | 0.000 |
| 1 | 0.052632 | 0.024 |
| 2 | 0.105263 | 0.064 |
| 3 | 0.157895 | 0.096 |
| 4 | 0.210526 | 0.176 |
| 5 | 0.263158 | 0.224 |
| 6 | 0.315789 | 0.248 |
| 7 | 0.368421 | 0.280 |
| 8 | 0.421053 | 0.360 |
| 9 | 0.473684 | 0.520 |
| 10 | 0.526316 | 0.584 |
| 11 | 0.578947 | 0.600 |
| 12 | 0.631579 | 0.760 |
| 13 | 0.684211 | 0.784 |
| 14 | 0.736842 | 0.824 |
| 15 | 0.789474 | 0.832 |
| 16 | 0.842105 | 0.864 |
| 17 | 0.894737 | 0.920 |
| 18 | 0.947368 | 0.944 |
| 19 | 1.000000 | 1.000 |

Image credits to Author (Tanu Nanda Prabhu)

Usually, the **Z-score normalization** is preferred because **min-max scaling** is prone to **overfitting**.

#### When to actually use Standardization and Normalization?

There is no one answer to the above question. If you have a **small dataset** and have **sufficient time** then you can experiment with both of the above techniques and choose the best one. Below is the **rule of thumb** that you can follow:

- You can use **standardization** on **unsupervised learning algorithms**. In this case, standardization is **more beneficial** than normalization.
- If you see a **bell-curve** in your data then **standardization** is more preferable. For this, you will have to plot your data.
- If your dataset has **extremely high** or **low values** (**outliers**) then **standardization** is more preferred because usually, normalization will **compress** these values into a **small range**.

In any other cases apart from the above-given one's **normalization** holds good. Again if you have enough time experiment with both of the feature engineering techniques.

Alright, you guys have reached the end of the tutorial. I hope you guys learned a thing or two today. I used the textbook named "**The Hundred-Page Machine Learning Book by Andriy Burkov**" as a reference (Chapter 5) to write this tutorial. You can have a look at it. If you guys have any doubt regarding this tutorial you can use the comment section down below. I will try to answer it as soon as possible. Until then Stay Safe, Good Bye. See you next time. For more updates on Datafied to read and write more python notebooks.

**Datafied**

Create a portfolio * Upload Jupyter Notebooks * Tell stories with your data * Create a free portfolio Explore notebooks...

www.datafied.world

Data Science | Programming | Normalization | Standardization | Python

Follow

## Written by Tanu N Prabhu

1.5K Followers · Writer for Towards Data Science

MSc in Computer Science | He/Him | Tech and pet enthusiast | Don't believe me, read a couple of my writings | Writing since June 19, 2019 |

**More from Tanu N Prabhu and Towards Data Science**

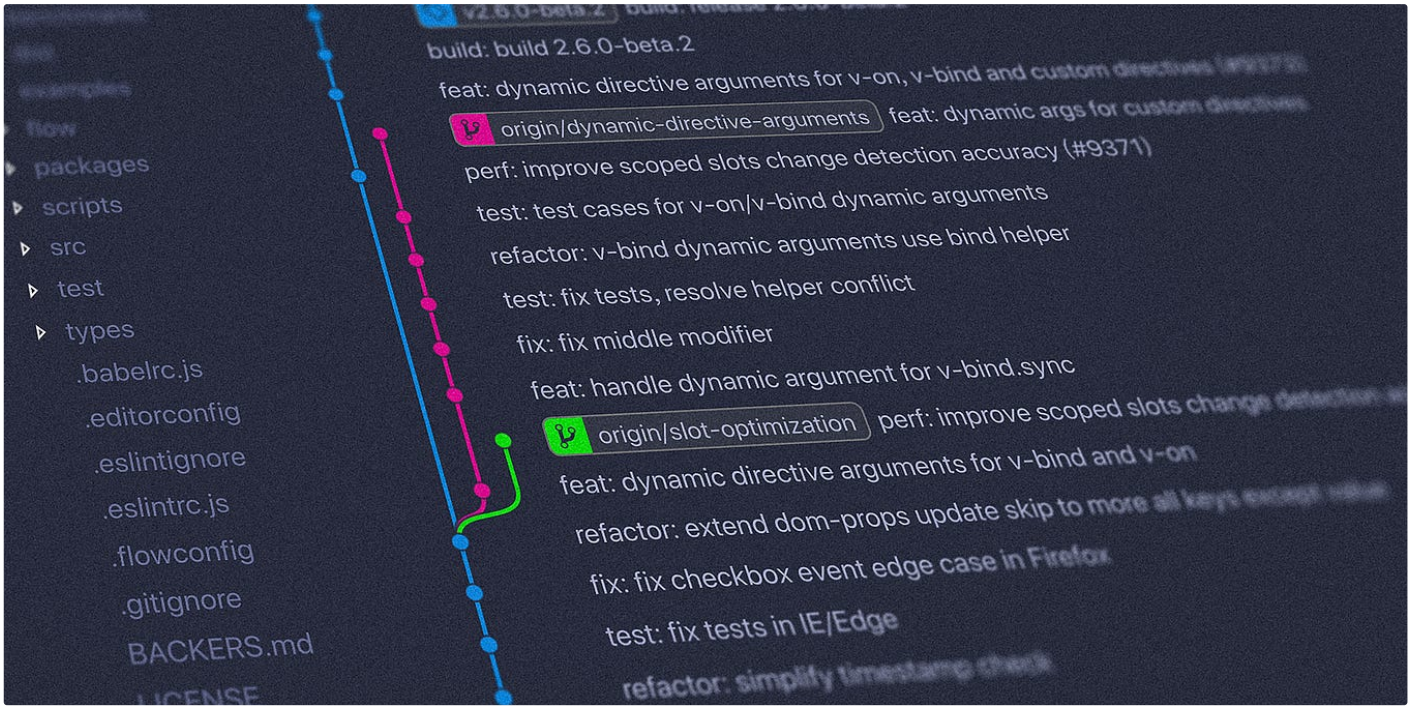

Tanu N Prabhu in Towards Data Science

## Google Trends API for Python

In this tutorial, I will demonstrate how to use the Google Trends API for getting the current trending topics on the internet.

· 5 min read · Feb 29, 2020





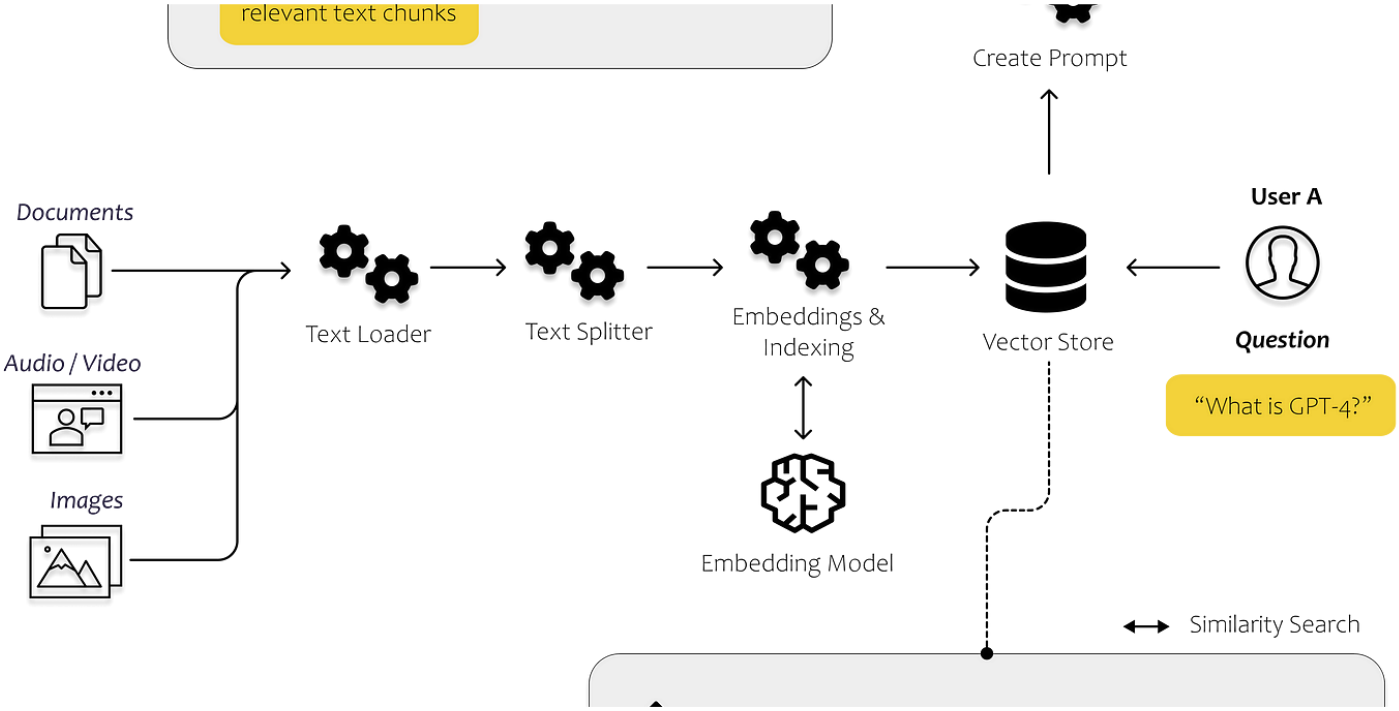

Miriam Santos in Towards Data Science

## Pandas 2.0: A Game-Changer for Data Scientists?

The Top 5 Features for Efficient Data Manipulation

7 min read · Jun 27

1.6K 18




Dominik Polzer in Towards Data Science

## All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

· 26 min read · Jun 21

2.1K 21

Tanu N Prabhu in Towards Data Science

## Cheat-sheet for Google Colab

In this tutorial, you will learn how to make the most out of Google Colab.

· 11 min read · May 1, 2020

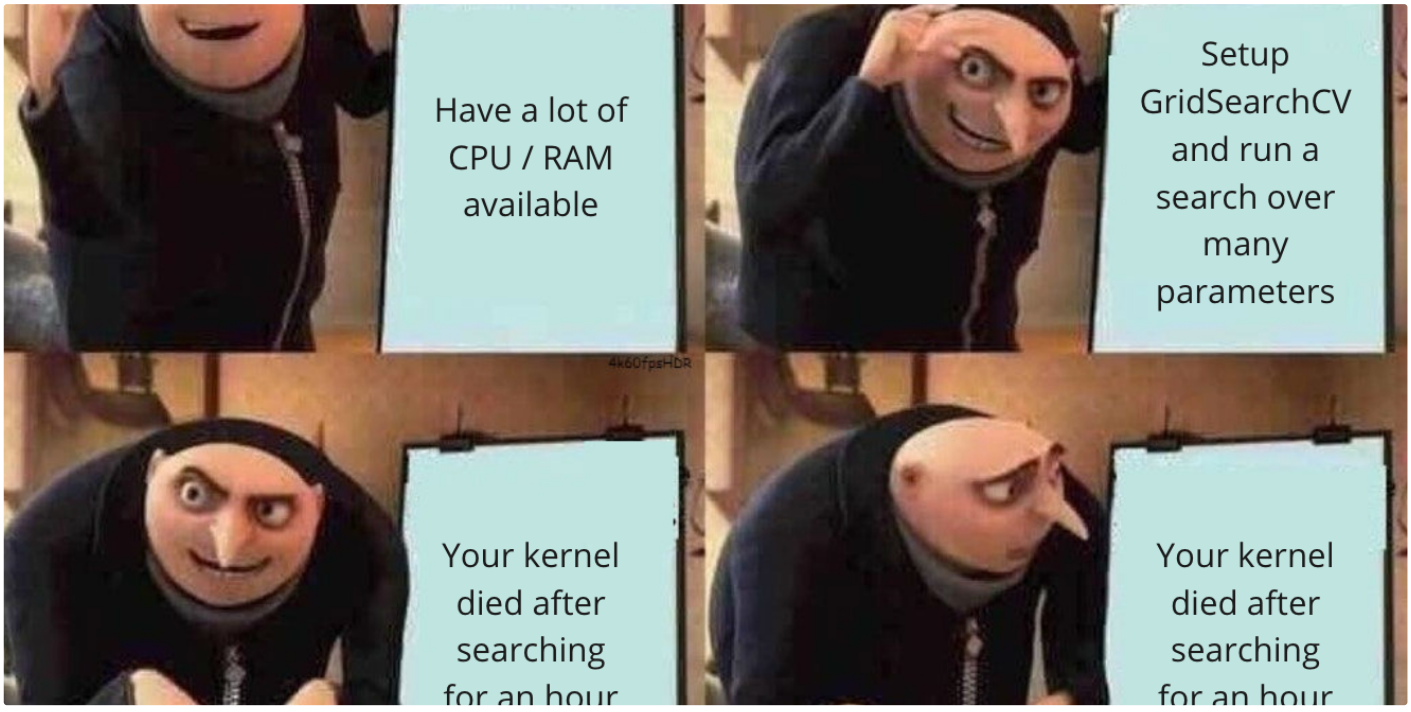
Matt Chapman in Towards Data Science

## The Portfolio that Got Me a Data Scientist Job

Spoiler alert: It was surprisingly easy (and free) to make

· 10 min read · Mar 24

3.6K 58




Ali Soleymani

## Grid search and random search are outdated. This approach outperforms both.

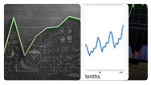
If you're a data scientist, there is a good chance you have used "Grid Search" to fine-tune the hyperparameters of your model. This is a...

✦ · 7 min read · Feb 8

360 5

### Lists

**Coding & Development**
11 stories · 48 saves



**Predictive Modeling w/ Python**
18 stories · 113 saves

**New_Reading_List**
174 stories · 19 saves

**General Coding Knowledge**
20 stories · 68 saves


Youssef Hosni in Level Up Coding

## 13 SQL Statements for 90% of Your Data Science Tasks

Structured Query Language (SQL) is a programming language designed for managing and manipulating relational databases. It is widely used by...

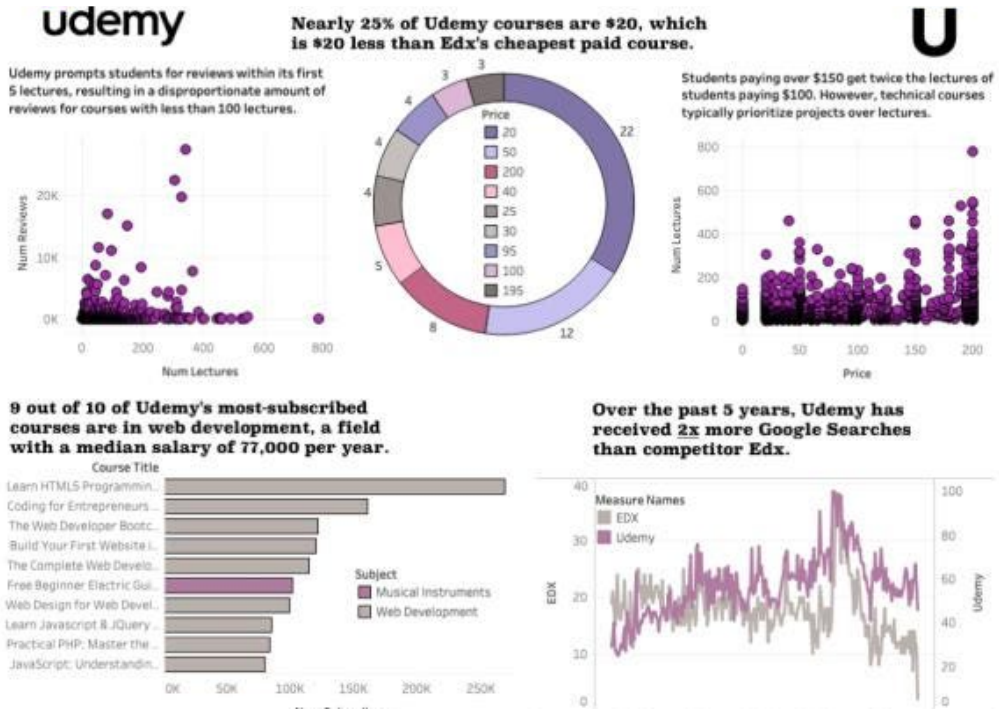
✦ · 15 min read · Feb 26

Rukshan Pramoditha in Towards Data Science

# Addressing Overfitting 2023 Guide—13 Methods

Your one-stop place to learn 13 effective methods to prevent overfitting in machine learning and deep learning models

· 14 min read · Nov 22, 2022



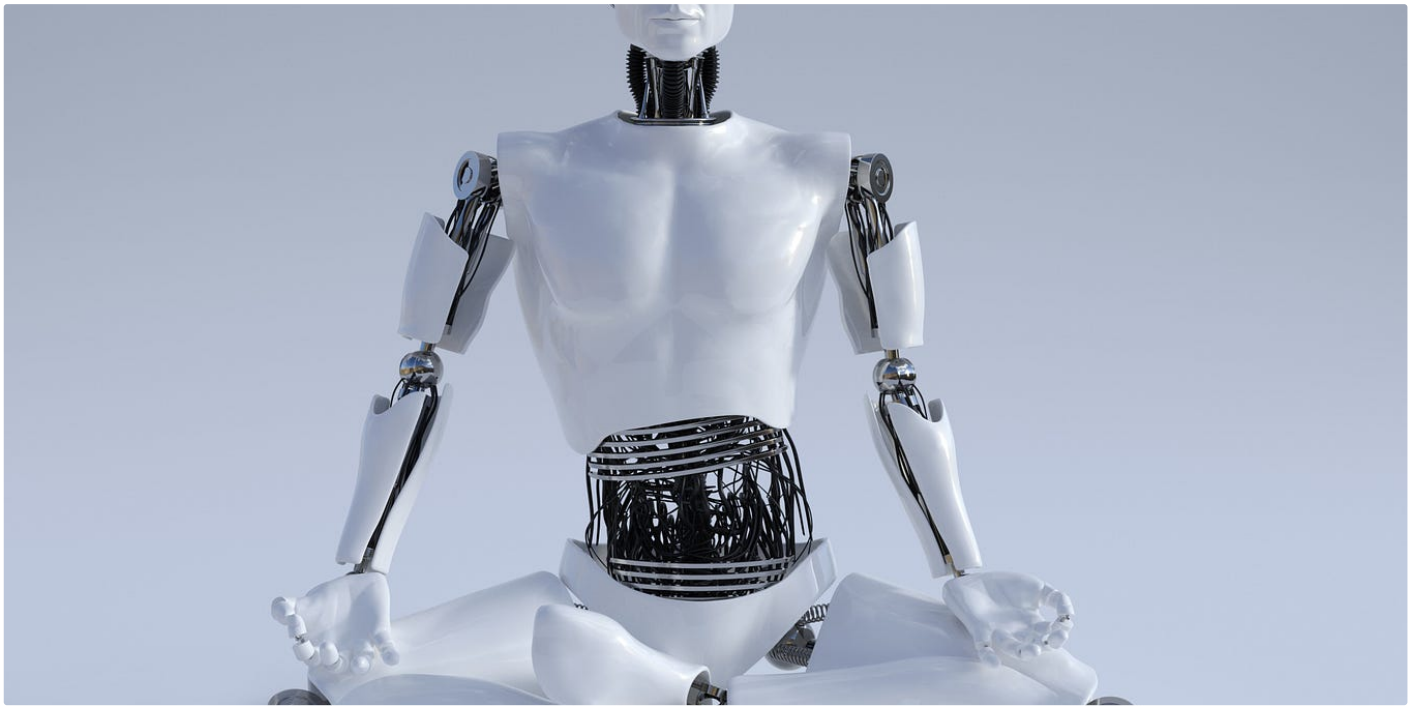

Zach Quinn in Pipeline: Your Data Engineering Resource

## Creating The Dashboard That Got Me A Data Analyst Job Offer

A walkthrough of the Udemy dashboard that got me a job offer from one of the biggest names in academic publishing.

· 9 min read · Dec 5, 2022

1.2K 20



The PyCoach in Artificial Corner

# You’re Using ChatGPT Wrong! Here’s How to Be Ahead of 99% of ChatGPT Users

Master ChatGPT by learning prompt engineering.

· 7 min read · Mar 17

27K 495

See more recommendations