

Follow

516K Followers · Editors' Picks · Features · Explore · Contribute · About

You have 2 free member-only stories left this month. [Sign up for Medium and get an extra one](#)

How to deal with imbalanced data in Python

The pythonic solution to balance the imbalanced

 Jack Tan · Nov 11 · 8 min read •

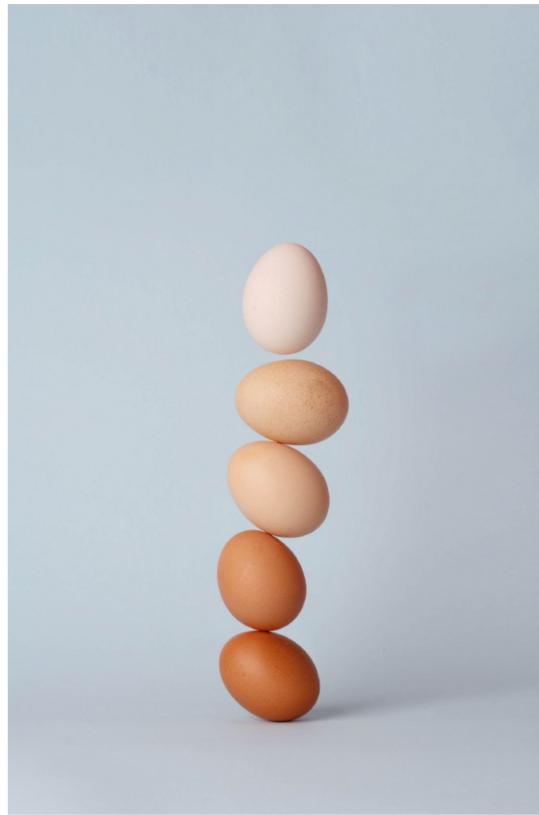


Photo by [吉昌](#) on [Unsplash](#)

Imbalanced data is a very common occurrence in real-world domains, especially when the subject of interest for a decision-making system is a rare but important case. This can be a problem when a future decision is to be made based on insights from historical data. Inadequate data from the minority case can hamper the robustness of the new decision being made.

The case of imbalanced data exists almost to any real-life applications. For instance, the [average customer churn rate](#) for wireless carriers in the US is somewhere between 1–6%. Likewise in marketing, the [average conversion rate](#) also lingers in the single-digit range.

So, what can the problem be when using imbalanced data in Machine Learning (ML)? How can you deal with the imbalanced data? This article will walk you through all these questions and show you some practical examples in Python.

As a surprise, I will include a “one more thing” bonus at the end of this article on how to choose the best sampling ratio using a combination of oversampling and undersampling method.

What are imbalanced data precisely?

Imbalanced data sets are a special case for classification problem where the class distribution is not uniform among the classes. Typically, they are composed by two classes: The majority (negative) class and the minority

(positive) class [1].

Normally the minority class is what we hope the ML model would be able to correctly predict and thus known as the positive class. For instance, if a model is trying to train on a set of 10,000 customer data with 3% conversion rate, it might not have enough samples to be trained well enough to predict the probability of positive conversion in the future.

So, how do we define the threshold for imbalanced data exactly?

According to [Google Developers](#), the answer can be divided into 3 degrees of imbalance — mild (20–40%), moderate (1–20%), and extreme (<1%).

Why does it matter?

The impacts of imbalanced data are implicit, i.e. it does not raise an immediate error when you build and run your model, but the results can be delusive. If the degree of class imbalance for the majority class is extreme, then a machine trained classifier may yield high overall prediction accuracy since the model is most likely to predict most samples belonging to the majority class. For example, in a two-class problem with a class distribution of 90:10, the performance of the classifier on majority-class examples will count nine times as much as the performance on minority-class examples.

How to deal with imbalanced data?

Several solutions have been suggested in the literature to address this problem, amongst which are:

- **Data-level techniques** — At the data level, solutions work by applying resampling techniques to balance the dataset. These can be done by oversampling the minority class, which is to synthetically create new instances from existing ones; or undersampling the majority class, which eliminates some instances in the majority class. However, both techniques can have their drawbacks. Oversampling new data can cause the classifier to overfit; whereas undersampling can discard essential information. A combination of both techniques with a heuristic approach can be found in specialized literature with excellent results.
- **Algorithmic-level techniques** — Algorithmic level solutions can be done by adjusting weighted costs accordingly to the number of training instances in each class. In parametric classifier like [Support Vector Machine](#), grid search and cross-validation can be applied to optimise the C and gamma values. For non-parametric classifier like the [decision tree](#), adjusting the probabilistic estimate at the tree leaf can improve the performance.
- **A combination of both** — A hybrid approach is also constantly being explored in various literature, including [AdaOUBoost](#) (adaptive oversampling and undersampling boost) proposed by Peng and Yao and [Learning By Recognition](#), using the concept of auto association-based classification approach proposed by Japkowicz.

Dealing with imbalanced data in Python

One of the most popular libraries for sampling methods in Python is none other than the [imbalanced-learn](#) package. It provides several methods for both over- and undersampling, as well as some combinational methods. For this tutorial, we will explore one example for each of these 3 methods:

- Random undersampling with [RandomUnderSampler](#)
- Oversampling with [SMOTE](#) (Synthetic Minority Over-sampling Technique)
- A combination of both random undersampling and oversampling using pipeline

The dataset used in this tutorial is based on the bank marketing data from the [UCI repo](#). It is a classification problem where the goal is to predict if the client will subscribe to a term deposit. All the categorical (text) feature columns are [encoded](#) into machine-friendly numerical forms. The encoded dataset is hosted on [Github](#). In order to test how good the data perform after resampling, we train the *Support Vector Machine* model on the resampled data to check for the model performance. The complete Python codes can also be found in the same [Github](#) repository.

The reason why this dataset is chosen because it reflects the common imbalanced dataset experienced in daily applications. As expected, the data is highly imbalanced and only around 13% [$5,289/(5,289+39,922)$] of the contacted clients actually subscribed to a term deposit. In Python, the

number of each predicted class can be printed using the method

```
value_counts() .
```

```
[7] print(df['y'].value_counts())
```

```
0    39922  
1     5289  
Name: y, dtype: int64
```

Distribution of the targeted classes. 0 is the majority negative class aka clients who did not convert whereas 1 represents the minority positive class aka clients who subscribed to a term deposit.

1. Random Undersampling



Photo by [Patrick Fore on Unsplash](#)

As the name suggests, random undersampling reduces the number of majority class randomly down to the desired ratio against the minority class. This is probably the easiest way to undersample and can actually yield good results if there are a lot of the majority class instances that are close to each other. For a more statistical way of reducing majority class, you can refer to [Condensed nearest neighbour](#) (aka. CNN, based on the Hart algorithm), [Tomek Links](#) (an improvement over the original CNN technique), [NearMiss](#), and so forth just to name a few. Fortunately, all the mentioned techniques are available in the same [imbalanced-learn](#) package which you can easily import to try them out.

The code snippet is split into two parts: PART 1 shows you the codes to random undersample the majority class; PART 2 trains the resampled data with *Support Vector Machine* and output the ROC AUC score.

```
1 #PART 1  
2 # import random undersampling and other necessary libraries  
3 from collections import Counter  
4 from imblearn.under_sampling import RandomUnderSampler  
5 from sklearn.model_selection import train_test_split  
6 import pandas as pd  
7 import numpy as np  
8 import warnings  
9 warnings.simplefilter(action='ignore', category=FutureWarning)  
10  
11 #import data  
12 url = "https://raw.githubusercontent.com/jackty9/Hanlding_Imbalanced_Data_in_Python/master/bank-  
13 df = pd.read_csv(url)  
14  
15 # Separating the independent variables from dependent variables  
16 X = df.iloc[:, :-1]  
17 y = df.iloc[:, -1]  
18  
19 #Split train-test data  
20 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30)  
21  
22 # summarize class distribution  
23 print("Before undersampling: ", Counter(y_train))  
24  
25 # define undersampling strategy  
26 undersample = RandomUnderSampler(sampling_strategy='majority')  
27  
28 # fit and apply the transform  
29 X_train_under, y_train_under = undersample.fit_resample(X_train, y_train)  
30  
31 # summarize class distribution  
32 print("After undersampling: ", Counter(y_train_under))  
33  
34 #PART 2  
35 # import SVM libraries  
36 from sklearn.svm import SVC  
37 from sklearn.metrics import classification_report, roc_auc_score  
38  
39 model=SVC()  
40 clf_under = model.fit(X_train_under, y_train_under)  
41 pred_under = clf_under.predict(X_test)  
42  
43 print("ROC AUC score for undersampled data: ", roc_auc_score(y_test, pred_under))
```

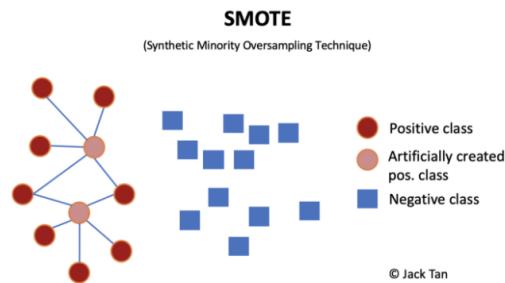
random.undersampling.imblearn.py hosted with ❤ by GitHub

view raw

```
Before undersampling: Counter({0: 27962, 1: 3685})
After undersampling: Counter({0: 3685, 1: 3685})
ROC AUC score for undersampled data: 0.7388684434398952
```

The output of the code snippet: 1st line shows the original data distribution; 2nd line shows the undersampled distribution; 3rd line shows the model performance using the undersampled data

2. Oversampling with SMOTE



An illustration of oversampling with SMOTE using 5 as k nearest neighbours. Self-illustrated by the author.

For over-sampling techniques, [SMOTE](#) (Synthetic Minority Oversampling Technique) is considered as one of the most popular and influential data sampling algorithms in ML and data mining. With SMOTE, the minority class is over-sampled by creating “synthetic” examples rather than by oversampling with replacement [2]. These introduced synthetic examples are based along the line segments joining a defined number of k minority class nearest neighbours, which is in the `imblearn` package is set at [five_by_default](#).

The code snippet is split into two parts: PART 1 shows you the codes to oversample the minority class with SMOTE; PART 2 trains the resampled data with *Support Vector Machine* and output the ROC AUC score.

```
1 #PART 1
2 # import SMOTE oversampling and other necessary libraries
3 from collections import Counter
4 from imblearn.over_sampling import SMOTE
5 from sklearn.model_selection import train_test_split
6 import pandas as pd
7 import numpy as np
8 import warnings
9 warnings.simplefilter(action='ignore', category=FutureWarning)
10
11 #import data
12 url = "https://raw.githubusercontent.com/jackty9/Handling_Imbalanced_Data_in_Python/master/bank-
13 df = pd.read_csv(url)
14
15 # Separating the independent variables from dependent variables
16 X = df.iloc[:, :-1]
17 y = df.iloc[:, -1]
18
19 #Split train-test data
20 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
21
22 # summarize class distribution
23 print("Before oversampling: ", Counter(y_train))
24
25 # define oversampling strategy
26 smote = SMOTE()
27
28 # fit and apply the transform
29 X_train_SMOTE, y_train_SMOTE = smote.fit_resample(X_train, y_train)
30
31 # summarize class distribution
32 print("After oversampling: ", Counter(y_train_SMOTE))
33
34 #PART 2
35 # import SVM libraries
36 from sklearn.svm import SVC
37 from sklearn.metrics import classification_report, roc_auc_score
38
39 model=SVC()
40 clf_SMOTE = model.fit(X_train_SMOTE, y_train_SMOTE)
41 pred_SMOTE = clf_SMOTE.predict(X_test)
42
43 print("ROC AUC score for oversampled SMOTE data: ", roc_auc_score(y_test, pred_SMOTE))
```

SMOTE_oversampling_imblearn.py hosted with ❤ by GitHub [view raw](#)

```
Before oversampling: Counter({0: 27943, 1: 3704})
After oversampling: Counter({1: 27943, 0: 27943})
ROC AUC score for oversampled SMOTE data: 0.7493411314174148
```

The output of the code snippet: 1st line shows the original data distribution; 2nd line shows the oversampled distribution; 3rd line shows the model performance using the oversampled data

3. A combination of under- and oversampling method using pipeline



Photo by [The Creative Exchange](#) on [Unsplash](#)

The code snippet is split into two parts: PART 1 shows you the codes to create a pipeline to resample the data using both under- and oversampling method; PART 2 trains the resampled data with *Support Vector Machine* and output the mean ROC AUC score after cross-validation.

```
1 #PART 1
2 # import sampling and other necessary libraries
3 from collections import Counter
4 from imblearn.over_sampling import SMOTE
5 from sklearn.model_selection import train_test_split
6 import pandas as pd
7 import numpy as np
8 import warnings
9 warnings.simplefilter(action='ignore', category=FutureWarning)
10 from sklearn.svm import SVC
11 from imblearn.over_sampling import SMOTE
12 from imblearn.under_sampling import RandomUnderSampler
13 from imblearn.pipeline import Pipeline
14
15 #import data
16 url = "https://raw.githubusercontent.com/jackty9/Hanlding_Imbalanced_Data_in_Python/master/bank-
17 df = pd.read_csv(url)
18
19 # Separating the independent variables from dependent variables
20 X = df.iloc[:, :-1]
21 y = df.iloc[:, -1]
22
23 # define pipeline
24 model = SVC()
25 over = SMOTE(sampling_strategy=0.4)
26 under = RandomUnderSampler(sampling_strategy=0.5)
27 steps = [('o', over), ('u', under), ('model', model)]
28 pipeline = Pipeline(steps=steps)
29
30 #PART 2
31 # import libraries for evaluation
32 from sklearn.model_selection import cross_val_score
33 from sklearn.metrics import roc_auc_score
34 from numpy import mean
35
36 # evaluate pipeline
37 scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=5, n_jobs=-1)
38 score = mean(scores)
39 print('ROC AUC score for the combined sampling method: %.3f' % score)
```

A_combination_of_over_and_under_sampling_imblearn.py hosted with ❤ by GitHub [view raw](#)

ROC AUC score for the combined sampling method: 0.820

The output of the code snippet showing the ROC AUC scores using a combined sampling method.

Bonus — How do I know which sampling ratio is the best?

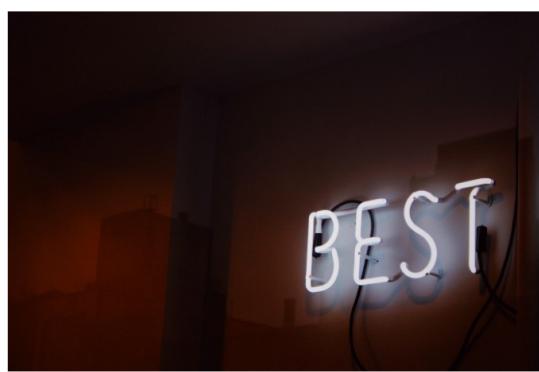


Photo by [Jon Tyson](#) on [Unsplash](#)

In the tutorial above, we either under- or/and oversample the data to have a balanced 50:50 distribution of both classes. But what if we want to find out the best sampling ratio based on the model performance? We can do this by

implementing a `for` loops in Python to iterate a possible list of sampling ratio and see which combination has the best performance based on the mean ROC AUC score (or other performance metrics of choice) after cross validation.

The code snippet shows you the codes to create a pipeline to resample the data using both under- and oversampling method. The list of possible ratios for oversampling is [0.3, 0.4, 0.5] while for undersampling [0.7, 0.6, 0.5]. These ratios are selected randomly for demonstration purposes and can be replaced with other values after some trial and error. The *Support Vector Machine* is then trained with the resampled data for every possible pair of the ratios combination before outputting the mean ROC AUC score after cross-validation.

```
1 # import necessary libraries
2 import numpy as np
3 from sklearn.svm import SVC
4 from imblearn.over_sampling import SMOTE
5 from imblearn.under_sampling import RandomUnderSampler
6 from imblearn.pipeline import Pipeline
7 from sklearn.model_selection import cross_val_score
8 from sklearn.metrics import roc_auc_score
9 from numpy import mean
10
11 # Import data
12 url = "https://raw.githubusercontent.com/jackty9/Handling_Imbalanced_Data_in_Python/master/bank"
13 df = pd.read_csv(url)
14
15 # Separating the independent variables from dependent variables
16 X = df.loc[:, :-1]
17 y = df.loc[:, -1]
18
19 # Values to evaluate
20 over_values = [0.3, 0.4, 0.5]
21 under_values = [0.7, 0.6, 0.5]
22 for o in over_values:
23     for u in under_values:
24         # Define pipeline
25         model = SVC()
26         over = SMOTE(sampling_strategy=o)
27         under = RandomUnderSampler(sampling_strategy=u)
28         steps = [('over', over), ('under', under), ('model', model)]
29         pipeline = Pipeline(steps=steps)
30
31         # Evaluate pipeline
32         scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=5, n_jobs=-1)
33         score = mean(scores)
34         print('SMOTE oversampling rate: {:.1f}, Random undersampling rate: {:.1f}, Mean ROC AUC: {:.3f}'
```

```

SMOTE oversampling rate=0.3, Random undersampling rate=0.7 , Mean ROC AUC: 0.820
SMOTE oversampling rate=0.3, Random undersampling rate=0.6 , Mean ROC AUC: 0.820
SMOTE oversampling rate=0.3, Random undersampling rate=0.5 , Mean ROC AUC: 0.820
SMOTE oversampling rate=0.4, Random undersampling rate=0.7 , Mean ROC AUC: 0.820
SMOTE oversampling rate=0.4, Random undersampling rate=0.6 , Mean ROC AUC: 0.820
SMOTE oversampling rate=0.4, Random undersampling rate=0.5 , Mean ROC AUC: 0.820
SMOTE oversampling rate=0.5, Random undersampling rate=0.7 , Mean ROC AUC: 0.822
SMOTE oversampling rate=0.5, Random undersampling rate=0.6 , Mean ROC AUC: 0.822
SMOTE oversampling rate=0.5, Random undersampling rate=0.5 , Mean ROC AUC: 0.818

```

As seen in the output snapshot, the best sampling ratio in this example is to have an oversampling ratio of 0.5, followed by an undersampling ratio of 0.7.

Conclusion

There is no short answer to the question if an undersampling or oversampling method is better for imbalanced data, even more so on when it comes to which exact sampling technique (random or SMOTE) to use. However, most researches have pointed out that undersampling in most cases does produce better results than oversampling. In industrial practice it has been reported that tech giants like Facebook and Microsoft also tend to use undersampling method when it comes to classifying their ad performance. The common argument is that undersampling is generally "cheaper" than oversampling and since the class of interest is the minority positive class, reducing some information of the majority negative class is acceptable. But there is no guarantee that the same can always be replicated in your case.

My advice is to decide based on the time and resources you have when handling imbalanced data. If you have a limited amount of time and resources to investigate which method works best for you, then you can opt for random undersampling from the get-go. However, if you have a bit of luxury to try out different sampling methods and techniques, I will highly recommend the latter. Once you have a well-balanced data, your model will potentially learn to classify different classes better for a more reliable prediction.

the best features for your model, for that you are welcome to check out my article in [Feature Selection](#).

*Disclaimer: Due to the stochastic nature of the train-test split and cross-validation, you might not get the exact output as shown.

Reference:

[1] [Imbalanced data set, Keel 2018](#)

[2] [SMOTE: Synthetic Minority Over-sampling Technique](#)

The image shows a newsletter sign-up form titled "Sign up for The Daily Pick" by Towards Data Science. It features a text input for "Your email" and a button labeled "Get this newsletter". Below the input field, there is a note about creating a Medium account and links to the Privacy Policy and terms of service. The background of the form is white with a thin blue border.



Data Science Programming Python Machine Learning Imbalanced Data

More from Towards Data Science

A Medium publication sharing concepts, ideas, and codes.

Follow

Papasot · Nov 11

The Age of Mourning Social Robots

Mourning social robots echoes their success



Photo by [Lenin Estrada on Unsplash](#)

Futuristic films always fascinated me, high tech gadgets, flying cars and how could we possibly forget the robot assistants. It seems that our imagination has always been the drive towards technological evolution, paving the way for our innovative abilities to follow. Yet, the thing with an abundant imagination is that it is also insatiable. We constantly yearn for more; a better phone, a smarter robot, a faster laptop, to the extent where innovations become outdated before they even reach the market. In fact, Matt Haig perfectly describes this phenomenon in his book 'Notes on a Nervous Planet',

We buy something and we enjoy it — we enjoy the newness of it — or a little while but then we get used to having it, we acclimatise, and so we need something else. We need to feel that sense of change, of variety. Something newer, something better, something upgraded. And the same thing happens again. ...

Read more · 7 min read



Rafael Bastos · Nov 11

How to get started on Kaggle Competitions

Step by step guide on how to structure your first Data Science projects on Kaggle



Photo by Jonathan Chng on Unsplash

If you are starting your journey in data science and machine learning, you may have heard of [Kaggle](#), the world's largest data science community. With the myriad of courses, books, and tutorials addressing the subject online, it's perfectly normal to feel overwhelmed with no clue where to start.

Although there isn't a unanimous agreement on the best approach to take when starting to learn a skill, getting started on Kaggle from the beginning of your data science path is solid advice.

It is an amazing place to learn and share your experience and data scientists of all levels can benefit from collaboration and interaction with other users. More experienced users can keep up to date with new trends and technologies, while beginners will find a great environment to get started in the field.

Read more · 8 min read

89 Q

• B

Sean Zhai · Nov 11 ★

QuickStart: spaCy on AWS Lambda Up and Running in 10 Minutes

The easy way to use spaCy for serverless computing



Photo by QHUNG TA on Unsplash

Let's keep our promise and get it done first. Further discussion follows the QuickStart section if you want to read further. I hope this arrangement could save you time.

...

QuickStart: Step-by-Step

I. Setup Lambda Layers for spaCy

- Log in to the AWS account. Create a new Lambda function. Choose Python 3.8, and use default settings.
- Go to Lambda [*Configuration*] [*Designer*], and click on *Layers*. Then click on [*Add a layer*] on the lower panel.

Read more · 4 min read

79 Q

• B

Gergely D. Németh · Nov 11 ★

Racial Bias in BERT

Understanding and visualizing unjust bias in BERT embedding vectors

Machine Learning fairness and bias is an increasingly discussed issue. With the development of pre-trained language models used in transfer learning applications, it is crucial to understand the unjust bias inherited with pre-trained models.

In this post, I'll focus on the English BERT [1], a language model trained on the Toronto Book Corpus [2] and Wikipedia. To detect bias in the model, I'll use the same set of black and white, female and male first names as used by Sweeney when she showed the racial bias in Google search [3]. Names in her work based on Bertran et al. [4] and Fryer and Levitt [5]. ...

[Read more · 7 min read](#)

80 1



Satyam Kumar · Nov 11 ★

7 Over Sampling techniques to handle Imbalanced Data

Deep dive analysis of various oversampling techniques

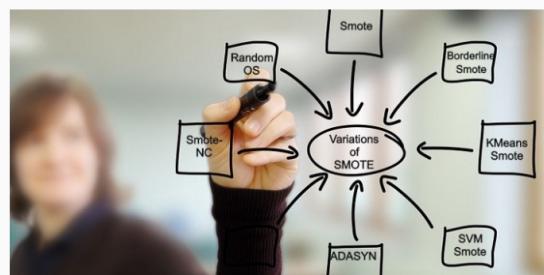


Image by LTD EHU from Pixabay

Modeling imbalanced data is the major challenge that we face when we train a model. For dealing with the classification problems the class balance of the target class label plays an important role in modeling. For imbalance class problems i.e presence of minority class in the dataset, the models try to learn only the majority class and result in biased prediction.

Some of the famous examples of imbalanced class problems are:

1. Credit Card Fraud Detection
2. Disease diagnosis
3. Spam detection, and many more

The imbalance of the dataset needs to be handled before training a model. There are various techniques to handle class balance, some of them being Oversampling, Undersampling, or a combination of both. ...

[Read more · 5 min read](#)

77



[Read more from Towards Data Science](#)

More From Medium

[How fast is C++ compared to Python?](#)
Naser Tamimi in Towards Data Science



[How To Create A Fully Automated AI Based Trading System With Python](#)
Rubén Romero in Towards Data Science



[Is Python Really a Bottleneck?](#)
Anna Anisienia in Towards Data Science



[Microservice Architecture and its 10 Most Important Design Patterns](#)
Md Kamruzzaman in Towards Data Science



[12 Data Science Projects for 12 Days of Christmas](#)
Terence Shin in Towards Data Science



[A Full-Length Machine Learning Course in Python for Free](#)
Rashida Nasrin Sucky in Towards Data Science



[Study Plan for Learning Data Science Over the Next 12 Months](#)
Daniel Morales in Towards Data Science



[How We, Two Beginners, Placed in Kaggle Competition Top 4%](#)
Andre Ye in Towards Data Science

