Yong Cui Follow
Nov 14, 2021 · 7 min read ★ · Listen

# Build Your First Machine Learning Model With Zero Configuration — Exploring Google Colab

It's just that easy to start your machine learning journey.

Machine learning (ML) is trending, and every company wants to leverage ML to help them better their products or services. Thus, we've been observing a growing demand for ML engineers, and such demand has drawn the attention of many people. However, ML may sound daunting to many, especially to those who have little coding or data-related work experience.

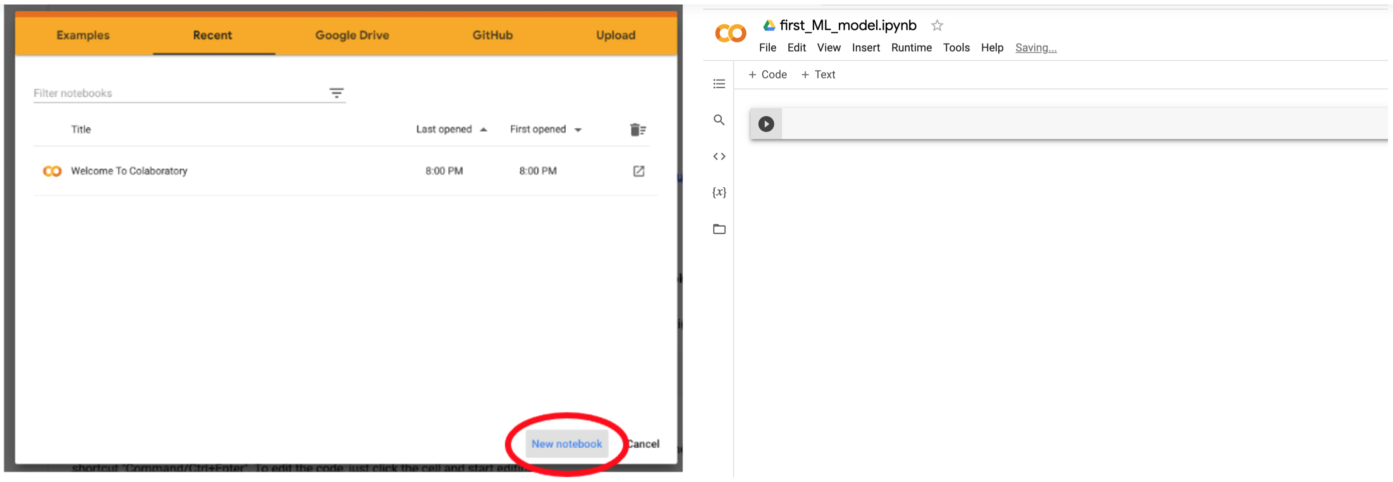
One probable reason is that it takes a good deal of efforts to set up the computer, allowing them to develop any ML models. In this article, I'd like to introduce Google Colab, a free tool (with paid upgrade options, though) for ML model learning and building. More importantly, as you shall find out, it has zero configuration for you — it's ready to use now — with the only requirement that you have a Google account. If you don't have one, please sign up such that you can follow along the tutorial.

I'm assuming that you don't know too much about ML, but are very enthusiastic about learning about ML. It doesn't really matter how much Python you know. I'll explain the major steps using laymen languages as much as possible.

Without further ado, let's get it started. If you want to see the source code, you can access the Notebook using the link.

**Step 1. Creating a Notebook in Colab**

In Colab, you work with Notebooks (*.ipynb), just like you work with documents (*.docx) in Microsoft Word. So, the first step to use Colab is to create a new Notebook by going to: https://colab.research.google.com/.



New Notebook in Colab (Image by Author)

After clicking the "New notebook" button, you'll see that Colab creates a new notebook with a default name of `Untitled1.ipynb` . For the sake of the current tutorial, let's call it `first_ml_model.ipynb` .

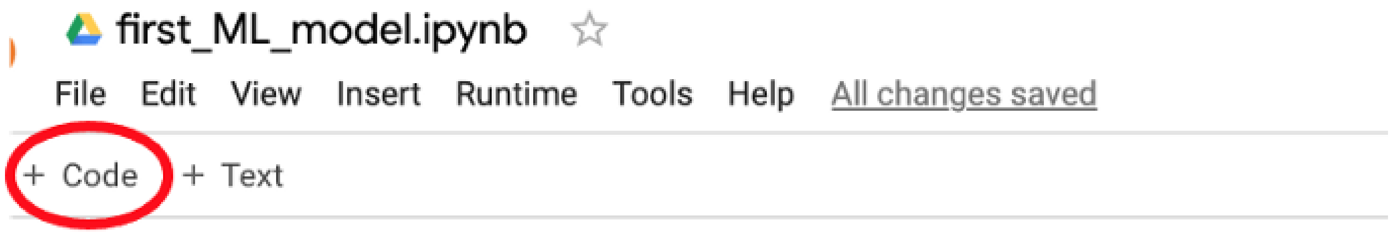
From now on, we'll work on this notebook.

### Step 2. Importing Dependencies

When we build our model, we need code libraries that experienced ML developers have developed. In essence, these libraries serve as toolsets by providing pre-defined functionalities for processing the data and building the model. In this tutorial, we'll be mainly using the following libraries.

- scikit-learn: a ML library that consists of a variety of data processing functions and ML algorithms (e.g., regression, classification, and clustering). This library is also known as sklearn, and we'll use sklearn for referencing purposes.
- pandas: a data science library that is primarily specialized in pre-processing spreadsheet-like data before building ML models.

simply click the `+ Code` on the top, as shown below. You can add your own code notes by clicking `+ Text`.



Adding Code Cell (Image by Author)

With the created code, by running the following cell, you can import the needed libraries for the present tutorial.

```
from sklearn import datasets, model_selection, metrics, ensemble
import pandas as pd
```

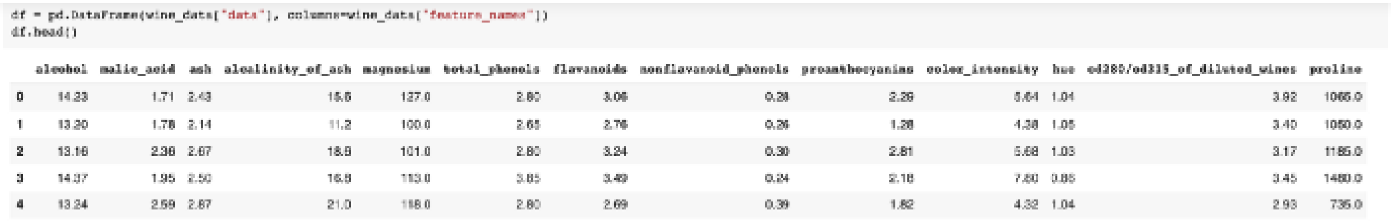
Importing Dependencies (Image by Author)

As a side note, if you're trying to configure a computer for ML, you need to install all these dependencies on top of configuring Python.

**Step 3. The Working Dataset**

For the current tutorial, we'll be using the red wine quality dataset. You can find more information about this dataset at kaggle.com, a popular data science and ML website that features a series competitions. You can also find the dataset's information on UCI, which is the leading ML data repository.

The wine dataset is often used as an example for showing ML models, and thus, it's



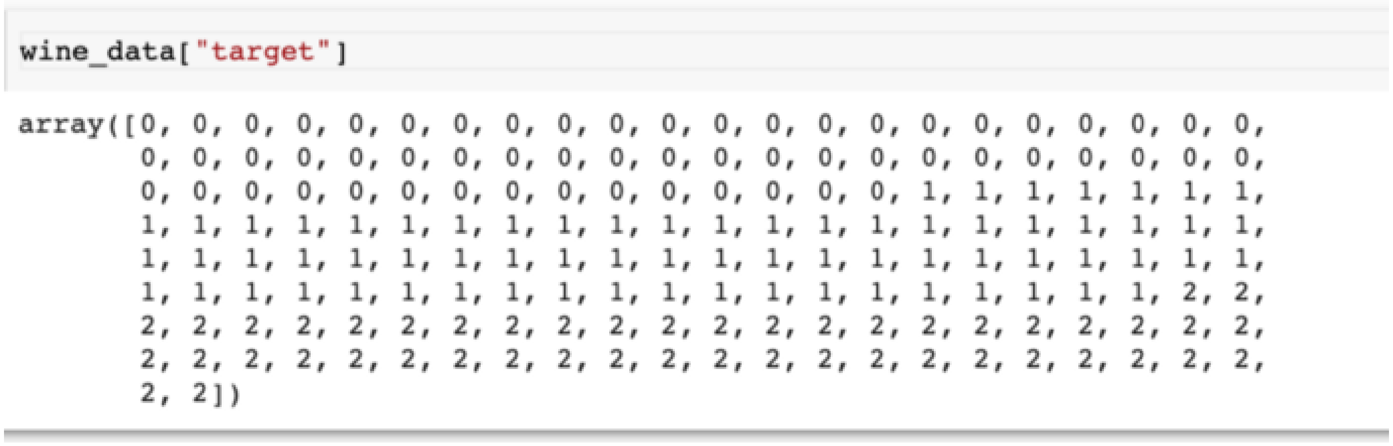
```
df = pd.DataFrame(wine_data["data"], columns=wine_data["feature_names"])
df.head()
```

| | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue | od280/od315_of_diluted_wines | proline |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 14.23 | 1.71 | 2.43 | 15.6 | 127.0 | 2.80 | 3.06 | 0.28 | 2.29 | 5.64 | 1.04 | 3.92 | 1065.0 |
| 1 | 13.20 | 1.78 | 2.14 | 11.2 | 100.0 | 2.65 | 2.76 | 0.26 | 1.28 | 4.38 | 1.05 | 3.40 | 1050.0 |
| 2 | 13.16 | 2.36 | 2.67 | 18.6 | 101.0 | 2.80 | 3.24 | 0.30 | 2.81 | 5.68 | 1.03 | 3.17 | 1185.0 |
| 3 | 14.37 | 1.95 | 2.50 | 16.8 | 113.0 | 3.85 | 3.49 | 0.24 | 2.18 | 7.80 | 0.86 | 3.45 | 1480.0 |
| 4 | 13.24 | 2.59 | 2.87 | 21.0 | 118.0 | 2.80 | 2.69 | 0.39 | 1.82 | 4.32 | 1.04 | 2.93 | 735.0 |

Wine DataFrame (Image by Author)

The above screenshot shows you the features of the data. In ML, we use "**features**" to study what factors may be important for the correct prediction. As you can see, there are 12 features available, and they're potentially important for the quality of the red wine, such as alcohol and malic acid.

One specific ML is concerned about classification. Each data record has a label showing its class, and the classes of all the records are known as "target" of the dataset. In the red wine data set, there are three classes for the labels, and we can check the labels, as shown below:



```
wine_data["target"]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2])
```

Red Wine Dataset — Target (Image by Author)

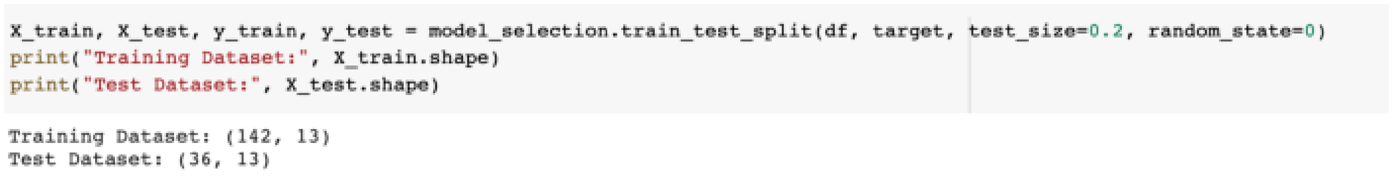
Please be noted that in a typical pipeline, we usually need to spend tons of time preparing the dataset. Some common preparations include outlier identification and removal/recoding, missing data handling, one-hot recoding (needed for certain models), dimensionality reduction, feature selections, scaling, and many others. Because the dataset has been cleaned up as a toy dataset in sklearn , we don't need to worry about

**Step 4. Training the Model**

The next step is train the ML model. You may be wondering what's the point of training an ML model. Well, for different use cases, there are different purposes. But in general, the purpose of training an ML model is more or less about making predictions on things that they've never seen. The model is about how to make good predictions. The way to create a model is called training — using existing data to identify a proper way to make predictions.

There are many different ways to build a model, such as K-nearest neighbors, SVC, random forest, and gradient boosting, just to name a few. For the purpose of the present tutorial showing you how to build an ML model using Google Colab, let's just use a model that's readily available in sklearn — the random forest classifier.

One thing to note is that because we have only one dataset. To test the model's performance, we'll split the dataset into two parts, one for training and the other for testing. We can simply use the `train_test_split` method, as shown below. The training dataset has 142 records, while the test dataset has 36 records, approximately in a ratio of 4:1 (note that `test_size=0.2` meaning 20% (with round-ups if needed) of the original dataset are used for testing).

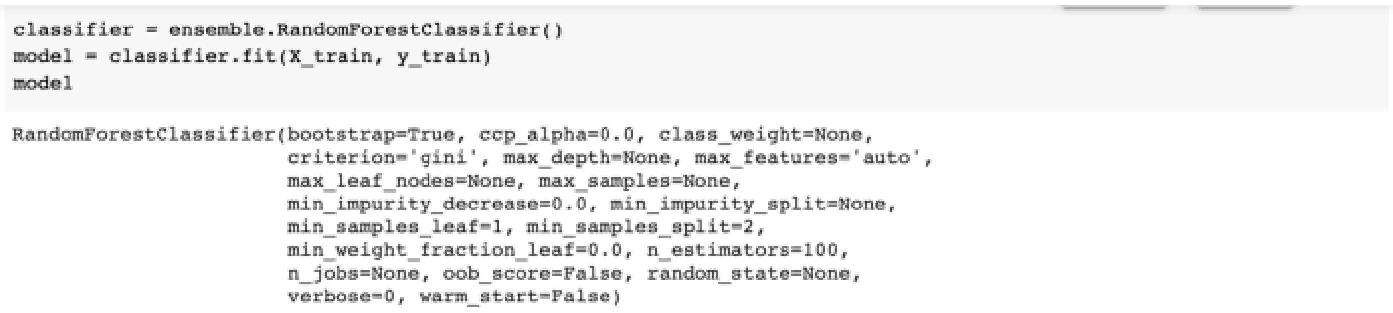


```
X_train, X_test, y_train, y_test = model_selection.train_test_split(df, target, test_size=0.2, random_state=0)
print("Training Dataset:", X_train.shape)
print("Test Dataset:", X_test.shape)
```

```
Training Dataset: (142, 13)
Test Dataset: (36, 13)
```

Split Dataset (Image by Author)

The nice thing about sklearn is that it does lots of heavy lifting for us by making many classifiers pre-configured such that we can use them with just a few lines of code. In the screenshot below, we first create a random forest classifier. In essence, it sets up the framework for us to put our data in to build the model.



```
classifier = ensemble.RandomForestClassifier()
model = classifier.fit(X_train, y_train)
model
```

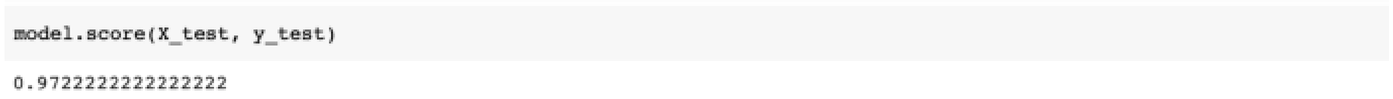
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Creating a Random Forest Classifier

Using classifier.fit, we're training the model to generate the parameters of the model, such that the model can be used for future predictions.

### Step 5. Making Prediction

With sklearn's trained model, we can test the model's performance on the test dataset that we created earlier. As shown below, we achieved a prediction of accuracy of 97.2%. Please note that achieving a high level like this in a toy dataset isn't untypical, but it is considered to be very high in real projects.



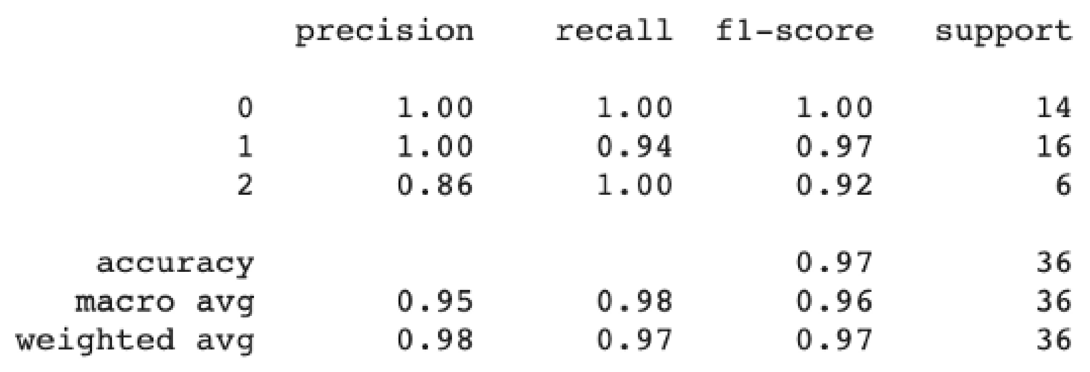
```
model.score(X_test, y_test)
```

```
0.9722222222222222
```

Prediction Accuracy (Image by Author)

If you want to take a closer look at the prediction of our model, you can run the following code, with which, we have a more complete report of the performance of our prediction model.

```
y_pred = model.predict(X_test)
report = metrics.classification_report(y_test, y_pred)
print(report)
```



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      0.94      0.97        16
           2       0.86      1.00      0.92         6

    accuracy                           0.97        36
   macro avg       0.95      0.98      0.96        36
weighted avg       0.98      0.97      0.97        36
```

Classification Report (Image by Author)

You may notice that there are several terms that you may be unfamiliar, such as precision and recall. You may find a related discussion on these terms and classification models.

**An Introduction to the ROC-AUC in Classification Tasks**

What does the curve mean?

towardsdatascience.com

**Conclusions**

In this article, I used Google Colab as the code editor to show you how to build an ML model to make predictions on a toy dataset. It's probably not even scratching the surface of all essential ML concepts. However, it does show you Google Colab is an easy to use tool that requires minimum configurations for you to start with your ML learning journey.

When you're comfortable with Google Colab, Python, and ML-related terminologies and

Thanks for reading this article. Stay connected by signing up my newsletter. Not a Medium member yet? Support my writing by using my membership link (at no extra cost for you, but a portion of your membership dues is redistributed to me by Medium as an incentive).