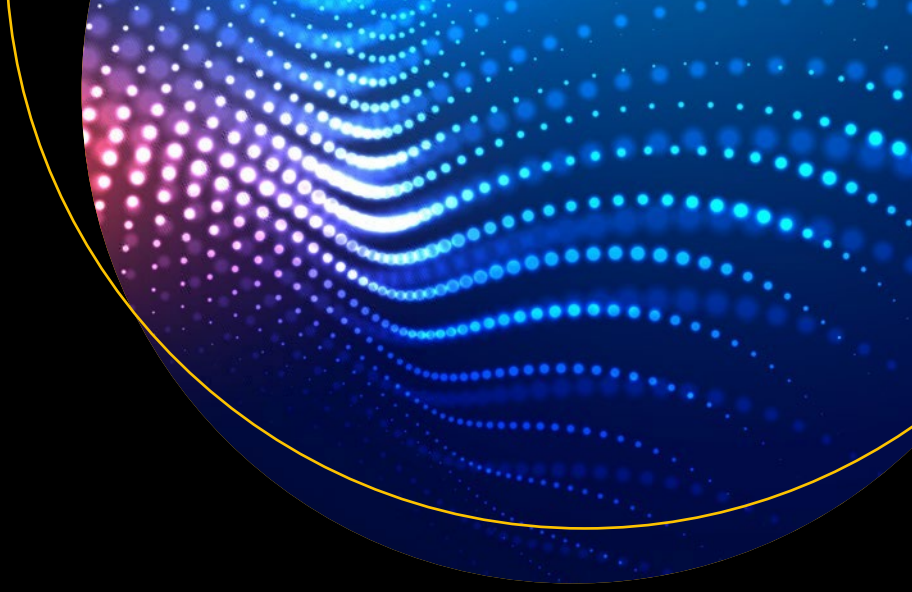


Beginning MLOps with MLFlow

Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure

Sridhar Alla
Suman Kalyan Adari

Apress®

Beginning MLOps with MLFlow

Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure

Sridhar Alla
Suman Kalyan Adari

Apress®

Beginning MLOps with MLFlow

Sridhar Alla
Delran, NJ, USA

Suman Kalyan Adari
Tampa, FL, USA

ISBN-13 (pbk): 978-1-4842-6548-2
ISBN-13 (electronic): 978-1-4842-6549-9
https://doi.org/10.1007/978-1-4842-6549-9

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: Laura Berendson
Coordinating Editor: Aditee Mirashi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

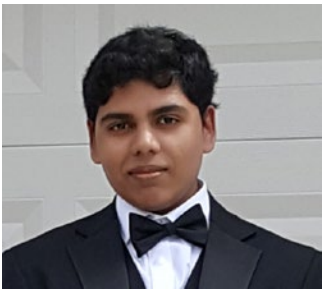
Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-6548-2. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Authors

Sridhar Alla is the founder and CTO of Bluewhale.one, the company behind the product Sas2Py (www.sas2py.com), which focuses on the automatic conversion of SAS code to Python. Bluewhale also focuses on using AI to solve key problems ranging from intelligent email conversation tracking to issues impacting the retail industry and more. He has deep expertise in building AI-driven big data analytical practices on both the public cloud and in-house infrastructures. He is a published author of books and an avid presenter at numerous Strata, Hadoop World, Spark Summit, and other conferences. He also has several patents filed with the US PTO on large-scale computing and distributed systems. He has extensive hands-on experience in most of the prevalent technologies, including Spark, Flink, Hadoop, AWS, Azure, TensorFlow, and others. He lives with his wife, Rosie, and daughters, Evelyn and Madelyn, in New Jersey, United States, and in his spare time loves to spend time training, coaching, and attending meetups. He can be reached at sid@bluewhale.one.



Suman Kalyan Adari is a current Senior and undergraduate researcher at the University of Florida specializing in deep learning and its practical use in various fields such as computer vision, adversarial machine learning, natural language processing (conversational AI) , anomaly detection, and more. He was a presenter at the IEEE Dependable Systems and Networks International Conference workshop on Dependable and Secure Machine Learning held in Portland, Oregon, United States in June 2019. He is also a published author, having worked on a book focusing on the uses of deep learning in anomaly detection. He can be reached at `sadari@ufl.edu`.

About the Technical Reviewer

Pramod Singh is a Manager, Data Science at Bain & Company. He has over 11 years of rich experience in the Data Science field working with multiple product- and service-based organizations. He has been part of numerous ML and AI large scale projects. He has published three books on large scale data processing and machine learning. He is also a regular speaker at major AI conferences such as the O'Reilly AI & Strata conference.

Acknowledgments

Sridhar Alla

I would like to thank my wonderful wife, Rosie Sarkaria, and my beautiful, loving daughters, Evelyn and Madelyn, for all the love and patience during the many months I spent writing this book. I would also like to thank my parents, Ravi and Lakshmi Alla, for all the support and encouragement they continue to bestow upon me.

Suman Kalyan Adari

I would like to thank my parents, Venkata and Jyothi Adari, and my loving dog, Pinky, for supporting me throughout the entire process. I would especially like to thank my sister, Niharika Adari, for helping me with edits and proofreading and helping me write the appendix chapter.

Introduction

This book is intended for all audiences ranging from beginners at machine learning, to advanced machine learning engineers, or even to machine learning researchers who wish to learn how to better organize their experiments.

The first two chapters cover the premise of the problem followed by the book, which is that of integrating MLOps principles into an anomaly detector model based on the credit card dataset. The third chapter covers what MLOps actually is, how it works, and why it can be useful.

The fourth chapter goes into detail about how you can implement and utilize MLFlow in your existing projects to reap the benefits of MLOps with just a few lines of code.

The fifth, sixth, and seventh chapters all go over how you can operationalize your model and deploy it on AWS, Microsoft Azure, and Google Cloud, respectively. The seventh chapter goes over how you can host a model on a virtual machine and connect to the server from an external source to make your predictions, so should any MLFlow functionality described in the book become outdated, you can always go for this approach and simply serve models on some cluster on the cloud.

The last chapter, Appendix, goes over how you can utilize Databricks, the creators of MLFlow, to organize your MLFlow experiments and deploy your models.

The goal of the book is to hopefully impart to you, the reader, knowledge of how you can use the power of MLFlow to easily integrate MLOps principles into your existing projects. Furthermore, we hope that you will become more familiar with how you can deploy your models to the cloud, allowing you to make model inferences anywhere on the planet so as long as you are able to connect to the cloud server hosting the model.

At the very least, we hope that more people do begin to adopt MLFlow and integrate it into their workflows, since even as a tool to organize your workspace, it massively improves the management of your machine learning experiments and allows you to keep track of the entire model history of a project.

Researchers may find MLFlow to be useful when conducting experiments, as it allows you to log plots on top of any custom-defined metric of your choosing. Prototyping becomes much easier, as you can now keep track of that one model which worked perfectly as a proof-of-concept and revert back to those same weights at any time while you keep tuning the hyperparameters. Hyperparameter tuning becomes much simpler and more organized, allowing you to run a complex script that searches over several different hyperparameters at once and log all of the results using MLFlow.

With all the benefits that MLFlow and the corresponding MLOps principles offer to machine learning enthusiasts of all professions, there really are no downsides to integrating it into current work environments. With that, we hope you enjoy the rest of the book!

CHAPTER 1

Getting Started: Data Analysis

In this chapter, we will go over the premise of the problem we are attempting to solve with the machine learning solution we want to operationalize. We will also begin data analysis and feature engineering of our data set.

Introduction and Premise

Welcome to *Beginning MLOps with MLFlow*! In this book, we will be taking an example problem, developing a machine learning solution to it, and operationalizing our model on AWS SageMaker, Microsoft Azure, Google Cloud, and Datarobots. The problem we will be looking at is the issue of performing anomaly detection on a credit card data set. In this chapter, we will explore this data set and show the overall structure while explaining a few techniques on analyzing this data. This data set can be found at `www.kaggle.com/mlg-ulb/creditcardfraud`.

If you are already familiar with how to analyze data and build machine learning models, feel free to grab the data set and skip ahead to 3 to jump right into MLOps.

S. Alla and S. K. Adari, *Beginning MLOps with MLFlow*,
https://doi.org/10.1007/978-1-4842-6549-9_1

Otherwise, we will first go over the general process of how machine learning solutions are generally created. The process goes something like this:

1. **Identification of the problem:** First of all, you need to have an idea of what the problem is, what can be done about it, what has been done about it, and why it is a problem worth solving.

   Here's an example of a problem: an invasive snake species harmful to the local environment has infested a region. This species is highly venomous and looks very similar to a harmless species of snake native to this same environment. Furthermore, the invasive species is destructive to the local environment and is outcompeting the local species.

   In response, the local government has issued a statement encouraging citizens to go out and kill the venomous, invasive species on sight, but the problem is that it turns out citizens have been killing the local species as well due to how easy it is to confuse the two species.

   What can be done about this? A possible solution is to use the power of machine learning and build an application to help citizens identify the snake species. What has been done about it? Perhaps someone released an app that does a poor job at distinguishing the two species, which doesn't help remedy the current situation. Perhaps fliers have been given out, but it can be hard to identify every member of a species correctly based on just one picture.

Why is it a problem worth solving? The native species is important to the local environment. Killing the wrong species can end up exacerbating the situation and lead to the invasive species claiming the environment over the native species. And so building a computer vision-based application that can discern between the various snake species (and especially the two species relevant to the problem) could be a great way to help citizens get rid of the right snake species.

2. **Collection of data:** After you've identified the problem, you want to collect the relevant data. In the context of the snake species classification problem, you want to find images of various snake species in your region. The location depends on how big of a scale your project will operate on. Is it going to identify any snake in the world? Just snakes in Florida?

   If you can afford to do so, the more data you collect, the better the potential training outcomes will be. More training examples can introduce increased variety to your model, making it better in the long run. Deep learning models scale in performance with large volumes of data, so keep that in mind as well.

3. **Data analysis:** Once you've collected all the raw data, you want to clean it up, process it, and format it in a way that allows you to analyze the data better.

   For images, this could be something like applying an algorithm to crop out unnecessary parts of the image to focus solely on the snake. Additionally, maybe

you want to center-crop the image to remove all the extra visual information in the data sample. Either way, raw image data is rarely ever in good enough condition to be used directly; it almost always requires processing to get the relevant data you want.

For unstructured data like images, formatting this data in a way good enough to analyze it could be something like creating a directory with all of the respective snake species and the relevant image data. From there, you can look at the count of images for each snake species class that you have and determine if you need to retrieve more samples for a particular species or not.

For structured data, say the credit-card data set, processing the raw data can mean something like getting rid of any entries with null values in them. Formatting them in a way so you can analyze them better can involve dimensionality-reduction techniques such as principal component analysis (PCA). Note: It turns out that most of the data in the credit card data set has actually been processed with PCA in part to preserve the privacy of the users the data has been extracted from.

As for the analysis, you can construct multiple graphs of different features to get an idea of the overall distribution and how the features look plotted against each other. This way, you can see any significant relationships between certain features that you might keep in mind when creating your training data.

There are some tools you can use in order to find out what features have the greatest influence on the label, such as **phi-k correlation**. By allowing you to see the different correlation values between the individual features and the target label, you can gain a deeper understanding of the relationships between the features in this data set. If needed, you can also drop features that aren't very influential from the data. In this step, you really want to get a solid understanding of your data so you can apply a model architecture that is most suitable for it.

4. **Feature engineering and data processing:** Now you can use the knowledge you gained from analyzing the various features and their relationships to each another to potentially construct new features from combinations of several existing ones. For example, the Titanic data set is a great example that you can apply feature engineering to. In this case, you can take information such as class, age, fare, number of siblings, number of parents, and so on to create as many features as you can think up.

   Feature engineering is really about giving your model a deeper context so it can learn the task better. You don't necessarily want to create random features for the sake of it, but something that's potentially relevant like number of female relatives, for example. (Since females were more likely to survive the sinking of the Titanic, could it be possible that if a person had more female relatives, they were less likely to survive as preference was given to their female relatives instead?)

The next step after feature engineering is data processing, which is a step involving all preparations made to process the data to be passed into the model. In the context of the snake species image data, this could involve normalizing all the values to be between 0 and 1 as well as "batching" the data into groups.

This step also usually creates several subsets of your initial data: a **training data set**, a **testing data set**, and a **validation data set**. We will go into more detail on the purpose of each of these data sets later. For now, a **training data set** contains the data you want the model to learn from, the **testing data set** contains data you want to evaluate the model's performance on, and the **validation data set** is used to either select a model or help tune a model's hyperparameters to draw out a better performance.

5. **Build the model:** Now that the data processing is done, this step is all about selecting the proper architecture and building the model. For the snake species image data, a good choice would be to use a convolutional neural network (CNN) because they work very well for any tasks involving images. From there, it is up to you to define the specific architecture of the model with respect to its layer composition.

6. **Training, evaluating, and validating:** When you're training your CNN model, you're usually passing in batches of data until the entire data makes a full pass through the model. From the results of this "forward pass," calculations are made that tell the model how to adjust the weights as they are made going backwards across the network in what's called the "backward

pass." The training process is essentially where the model learns how to perform the task and gets better at it the more examples it sees.

After the training process, either the evaluation step or the validation step can come next. As long as the testing set and validation set come from different distributions (the validation set can be derived from the training set, while the testing set can be derived from the original data), the model is technically seeing new data in the evaluation and validation processes. The model will never learn anything from the evaluation data, so you can test your model anytime.

Model evaluation is where the model's performance metrics such as accuracy, precision, recall, and so on are evaluated on a data set that it has never seen before. We will go into more detail on the evaluation step once it becomes more relevant in the next chapter, Chapter 2.

Depending on the context, the exact purpose of validation can differ, along with the question of whether or not evaluation should be performed first after training. Let's define several sample scenarios where you would use validation:

- **Selecting a model architecture:** Of several model types or architectures, you use k-fold cross-validation, for example, to quickly train and evaluate each of the models on some data partition of the validation set to get an idea of how they are performing. This way, you can get a good idea of which model is performing best, allowing you to pick a model and continue with the rest of the process.

- **Selecting the best model:** Of several trained models, you can use something like k-fold cross-validation to quickly evaluate each model on the validation data to allow you to get an idea of which ones are performing best.
- **Tuning hyperparameters:** Quickly train a model and test it with different hyperparameter setups to get an idea of which configurations work better. You can start with a broad range of hyperparameters. From there, you can use the results to narrow the range of hyperparameters until you get to a configuration where you are satisfied. Models in deep learning, for example, can have many hyperparameters, so using validation to tune those hyperparameters can work well in deep learning settings. Just beware of diminishing returns. After a certain precision with the hyperparameter setting, you're not going to see that big of a performance boost in the model.
- **Indication of high variance:** This validation data is slightly different from the other three examples. In the case of neural networks, this data is derived from a small split of the training data. After one full pass of the training data, the model evaluates on this validation data to calculate metrics such as loss and accuracy.

  If your training accuracy is high and training loss is low, but the validation accuracy is low and the validation loss is high, that's an indication that your model suffers from **high variance**. What

this means is that your model has not learned to generalize what it is "learning" to new data, as the validation data in this case is comprised of data it has never seen before. In other words, your model is **overfitting**. The model just isn't recreating the kind of performance it gets on the training data on new data that it hasn't seen before.

If your model has poor training accuracy and high training loss, then your model suffers from **high bias**, meaning it isn't learning how to perform the task correctly on the training data at all.

This little validation split during the training process can give you an early indication of when overfitting is occurring.

7. **Predicting:** Once the model has been trained, evaluated, and validated, it is then ready to make predictions. In the context of the snake species detector, this step involves passing in visual images of the snake in question to get some prediction back. For example, if the model is supposed to detect the snake, draw a box around it, and label it (in an object detection task), it will do so and display the results in real time in the application.

   If it just classifies the snake in the picture, the user simply sends their photo of a snake to the model (via the application) to get a species classification prediction along with perhaps a probability confidence score.

Hopefully now you have a better idea of what goes on when creating machine learning solutions.

With all that in mind, let's get started on the example, where you will use the credit card data set to build simple anomaly detection models using the data.

Credit Card Data Set

Before you perform any data analysis, you need to first collect your data. Once again, the data set can be found at the following link: `www.kaggle.com/mlg-ulb/creditcardfraud`.

Following the link, you should see something like the following in Figure 1-1.

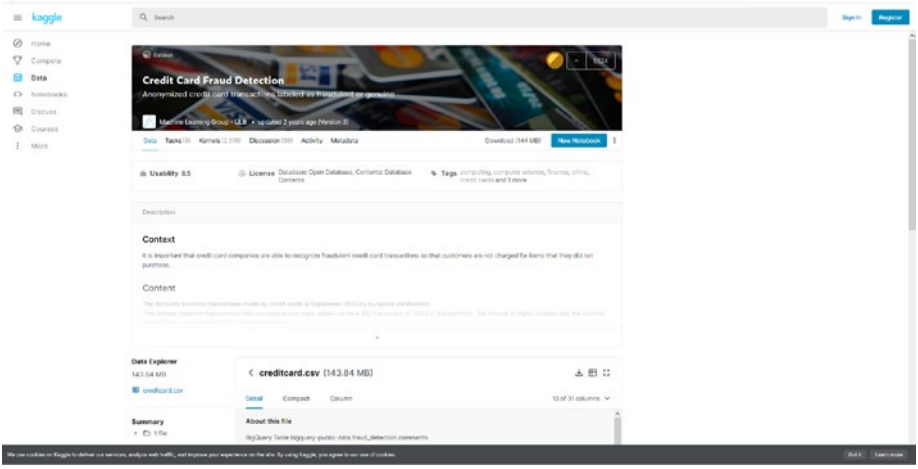


Figure 1-1. *Kaggle website page on the credit card data*

From here, you want to download the data set by clicking the Download (144 MB) button next to New Notebook. It should take you to a sign-in page if you're not already signed in, but you should be able to download the data set after that.

Once the zip file finishes downloading, simply extract it somewhere to reveal the credit card data set. Now let's open up Jupyter and explore this data set. Before you start this step, let's go over the exact packages and their versions:

- Python 3.6.5
- numpy 1.18.3
- pandas 0.24.2
- matplotlib 3.2.1

To check your package versions, you can run a command like

```
pip show package_name
```

Alternatively, you can run the following code to display the version in the notebook itself:

```
import module_name
print(module_name.__version__)
```

In this case, `module_name` is the name of the package you're importing, such as numpy.

Loading the Data Set

Let's begin! First, open a new notebook and import all of the dependencies and set global parameters for this notebook:

```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pylab import rcParams

rcParams['figure.figsize'] = 14, 8
```

Refer to Figure 1-2.

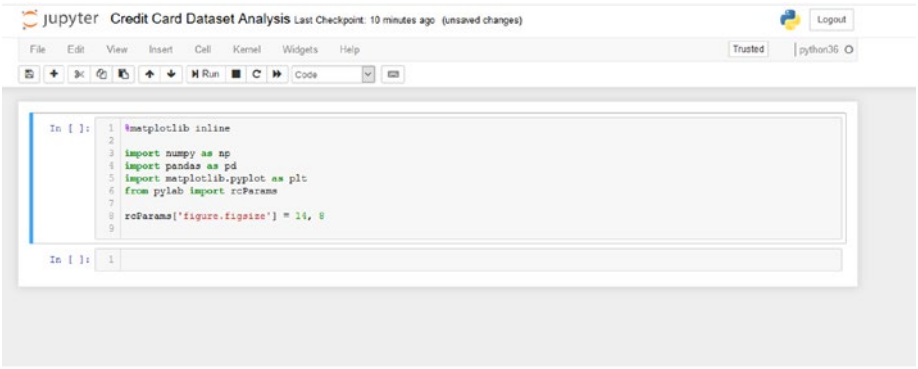


Figure 1-2. *Jupyter notebook cell with some import statements as well as a global parameter definition for the size of all matplotlib plots*

Now that you have imported the necessary libraries, you can load the data set. In this case, the `data` folder exists in the same directory as the notebook file and contains the `creditcard.csv` file. Here is the code:

```
data_path = "data/creditcard.csv"

df = pd.read_csv(data_path)
```

Refer to Figure 1-3.

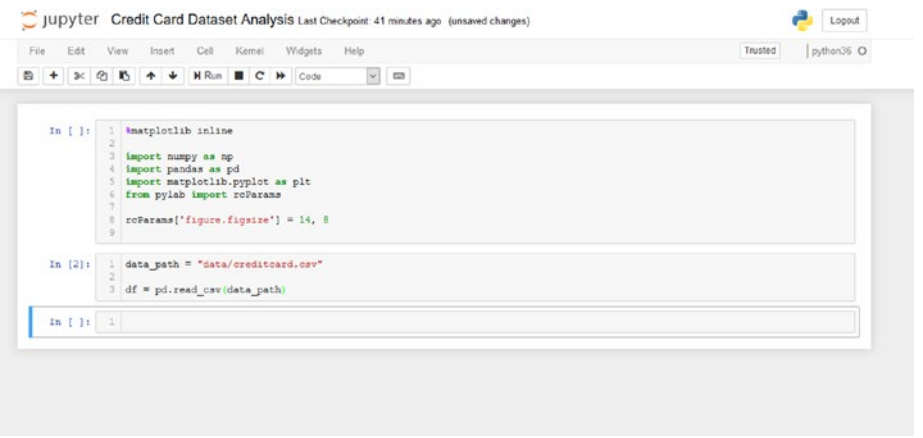


Figure 1-3. *Defining the data path to the credit card data set .csv file, reading its contents, and creating a pandas data frame object*

Now that the data frame has been loaded, let's take a look at its contents:

```
df.head()
```

Refer to Figure 1-4.

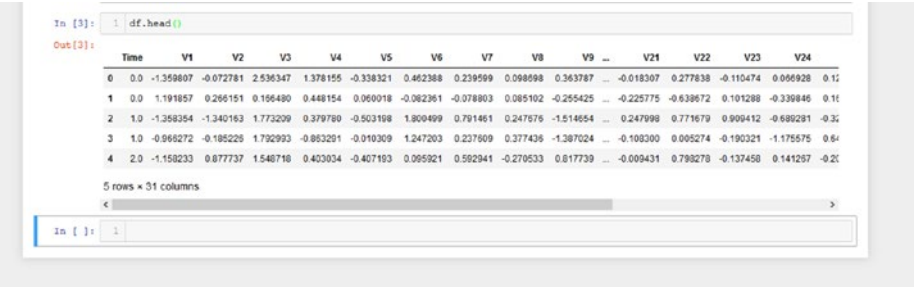


Figure 1-4. *Calling the head() function on the data frame to display the first five rows of the data frame*

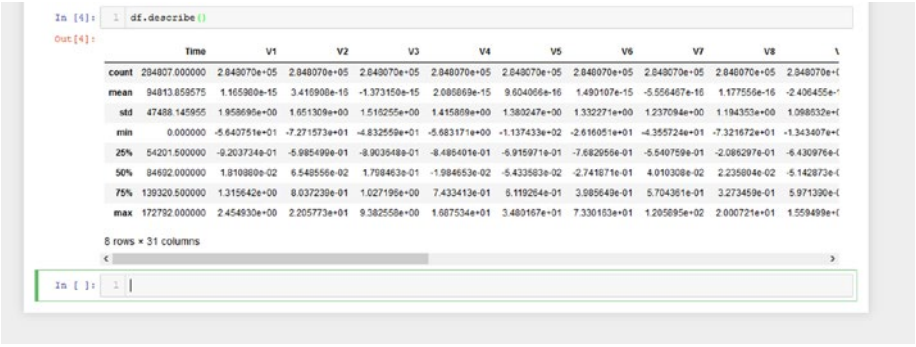
If you are not familiar with the `df.head(n)` function, it essentially prints the first n rows of the data frame. If you did not pass any arguments, like in the figure above, then the function defaults to a value of five, printing the first five rows of the data frame.

Feel free to play around with that function as well as use the scroll bar to explore the rest of the features.

Now, let's look at some basic statistical values relating to the values in this data frame:

```
df.describe()
```

Refer to Figure 1-5.



In [4]:
```
df.describe()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+(
mean	94813.859575	1.165980e-15	3.416908e-16	-1.373150e-15	2.086869e-15	9.604066e-16	1.490107e-15	-5.556467e-16	1.177556e-16	-2.406455e-(
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+(
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+(
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-(
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-(
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-(
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+(

8 rows × 31 columns

In [ ]:
```
```

Figure 1-5. *Calling the describe() function on the data frame to get statistical summaries of the data in each column*

Feel free to scroll right and look at the various statistics for the rest of the columns. As you can see in Figure 1-5, the function generates statistical summaries for data in each of the columns in the data frame.

The main takeaway here is that there are a huge number of data points. In fact, you can check the shape of the data frame by simply calling

```
df.shape
```

Refer to Figure 1-6.

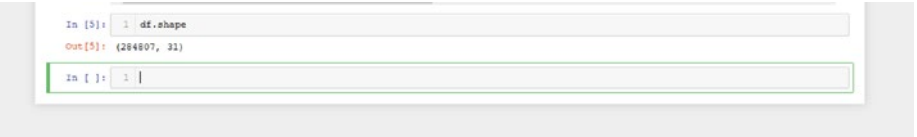


Figure 1-6. *Calling the shape() function on the data frame to get an output in the format (number_of_rows, number_of_columns)*

There are 284,807 rows and 31 columns in this data frame. That's a lot of entries! Not only that, but if you look at Figure 1-5, you'll see that the values can get really large for the column `Time`. In fact, keep scrolling right, and you'll see that values can get very large for the column `Amount` as well. Refer to Figure 1-7.

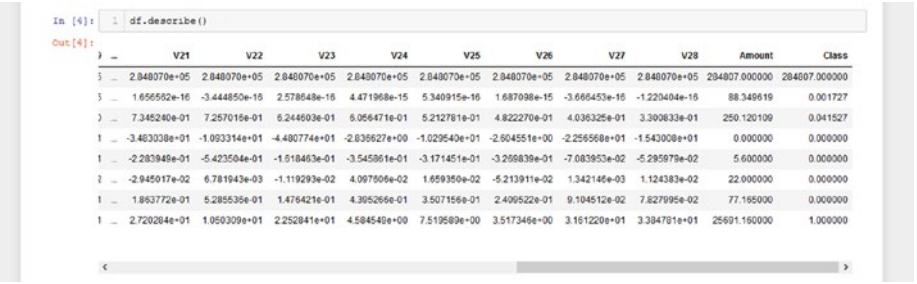


Figure 1-7. *Scrolling right in the output of the describe function reveals that the maximum data value in the column Amount is also very large, just like the maximum data value in the column Time*

As you can see, there are at least two columns with very large values. What this tells you is that later on, when building the various data sets for the model training process, you definitely need to scale down the data. Otherwise, such large data values can potentially mess up the training process.

Normal Data and Fraudulent Data

Since there are only two classes, `normal` and `fraud`, let's split up the data frame by class and continue with the data analysis. In the context of anomaly detection, the `fraud` class is also the `anomaly` class, hence why we chose to name the data frame representing fraudulent transaction data `anomalies` and interchangeably refer to this class as either `fraud` or `anomaly`.

Here is the code:

```
anomalies = df[df.Class == 1]
normal = df[df.Class == 0]
```

After that, run the following in a separate cell:

```
print(f"Anomalies: {anomalies.shape}")
print(f"Normal: {normal.shape}")
```

Refer to Figure 1-8.

Figure 1-8. *Defining data frames for fraudulent/anomalous data and for normal data and printing their shapes*

From here, you can see that the data is overwhelmingly biased towards normal data, and that anomalies only comprise a vast minority of data points in the overall data set. What this tells you is that you will have to craft the training, evaluation, and validation sets more carefully so each of these sets will have a good representation of anomaly data.

In fact, let's look at this disparity in a graphical manner just to see how large the difference is:

```
class_counts = pd.value_counts(df['Class'], sort = True)
class_counts.plot(kind = 'bar', rot=0)
plt.title("Class Distribution")
plt.xticks(range(2), ["Normal", "Anomaly"])
plt.xlabel("Label")
plt.ylabel("Counts")
```

Refer to Figure 1-9.

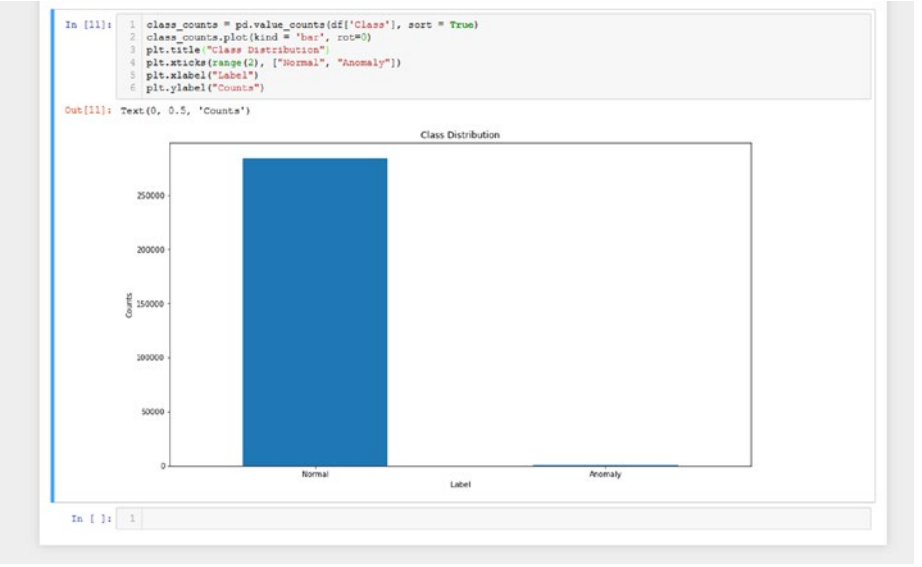


Figure 1-9. *A graph visually demonstrating the difference in counts for normal data and anomalous data*

The graph visually shows the immense difference between the number of data values of the two classes.