Data Science for Supply Chain Forecasting

Nicolas Vandeput
SupChains

Data Science for Supply Chain Forecasting

Nicolas Vandeput

2nd edition – 2021

nicolas.vandeput@supchains.com

Thinking must never submit itself, neither to a dogma, nor to a party, nor to a passion, nor to an interest, nor to a preconceived idea, nor to whatever it may be, if not to facts themselves, because, for it, to submit would be to cease to be.

Henri Poincare

Acknowledgments

Second Edition

We can see in Raphael's fresco *The School of Athens*, philosophers—practitioners and theorists alike—debating and discussing science. I like to use the same approach when working on projects: discussing ideas, insights, and models with other supply chain data scientists worldwide. It is always a pleasure for me.

For the second edition of *Data Science for Supply Chain Forecasting*, I surrounded myself with a varied team of supply chain practitioners who helped me to review chapter after chapter, model after model. I would like to thank each of them for their time, dedication, and support.

I would like to thank my dear friend Gwendoline Dandoy for her work on this book. She helped me to make every single chapter as clear and as simple as possible. She worked tirelessly on this book, as she did on my previous book *Inventory Optimization: Models and Simulations*. Along with her help to review this book, I could always count on her support, iced tea, and kindness. Thank you, Gwendoline.

Thanks to Mike Arbuzov for his help with the advanced machine learning models. It is always a pleasure to discuss with such a data science expert.

I had the chance this year to be surrounded by two inspiring, brilliant entrepreneurs: João Paulo Oliveira, co-founder of BiLD analytic (a data science consultancy company) and Edouard Thieuleux, founder of AbcSupplyChain (do not hesitate to visit his website abc-supplychain.com for supply chain training material and coaching). I would like to thank them both for their help reviewing the book. It is always a pleasure to discuss models and entrepreneurship with them.

I would like to thanks Michael Gilliland (author of *The Business Forecasting Deal*—the book that popularized the forecast value added framework) for his help and numerous points of advice on Part III of the book.

I was also helped by a group of talented supply chain practitioners, both consultants and in-house experts. I would like to thank Léo Ducrot for his help on the third part of the

book. Léo is an expert in supply chain planning and inventory optimization—discussing supply chain best practices with him is a rewarding pleasure. I would also like to thank Karl-Eric Devaux, who helped me to review the first chapters of the book; thanks to his amazing consultancy experience as demand planner. I am blessed by the help and advice that Karl-Eric shared with me over the last years. Thanks, also, to Steven Pauly (who earlier helped me for my inventory book) for his advice and ideas for the third part of the book.

I could also count on the incredible online community of supply chain and data science practitioners. Many thanks to Evangelos Spiliotis (PhD and research fellow at the National Technical University of Athens), who reviewed the statistical models in detail. His strong academic background and wide experience with statistical models was of great help. Thank you to Suraj Vissa (who also kindly helped me with my previous book) for his review of multiple chapters. Vincent Isoz, for his careful review of the neural network chapter. You can read his book about applied mathematics, *Opera Magistris*, here: `www.sciences.ch/Opera_Magistris_v4_en.pdf`. Thank you to Eric Wilson for his review of Part III (do not hesitate to check his demand planning YouTube channel: `www.youtube.com/user/DemandPlanning`). Thanks, also, to Mohammed Hichame Benbitour for his review of Part I.

As for the first edition of this book—and my second book!—I could count on the help of an all-star team of friends: François Grisay and his experience with supply chains, as well as Romain Faurès, Nathan Furnal, and Charles Hoffreumon with their experience with data science.

I would also like to mention a few friends for their support and help. Thank you to my friend Emmeline Everaert for the various illustrations she drew for this second edition. Thanks, also, to my sister Caroline Vandeput, who drew the timelines you can see through the book. Thank you to My-Xuan Huynh, Laura Garcia Marian, and Sébastien Van Campenhoudt for their reviews of the first chapters.

Finally, thank you to my (previous) master's students Alan Hermans, Lynda Dhaeyer, and Vincent Van Loo for their reviews.

Nicolas Vandeput
September 2020
nicolas.vandeput@supchains.com

First Edition

Discussing problems, models, and potential solutions has always been one of my favorite ways to find new ideas—and test them. As with any other big project, when I started to write *Data Science for Supply Chain Forecasting*, I knew discussions with various people would be needed to receive feedback. Thankfully, I have always been able to count on many friends, mentors, and experts to share and exchange these thoughts.

First and foremost, I want to express my thanks to Professor Alassane Ndiaye, who has been a true source of inspiration for me ever since we met in 2011. Not only does Alassane have the ability to maintain the big picture and stay on course in any situation—especially when it comes to supply chain—but he also has a sense of leadership that encourages each and everyone to shine and come face to face with their true potential. Thank you for your trust, your advice, and for inspiring me, Alassane.

Furthermore, I would like to thank Henri-Xavier Benoist and Jon San Andres from Bridgestone for their support, confidence, and the many opportunities they have given me. Together, we have achieved many fruitful endeavors, knowing that many more are to come in the future.

Of course, I also need to mention Lokad's team for their support, vision, and their incredible ability to create edge models. Special thanks to Johannes Vermorel (CEO and founder) for his support and inspiration—he is a real visionary for quantitative supply chain models. I would also like to thank the all-star team, Simon Schalit, Alexandre Magny, and Rafael de Rezende for the incredible inventory model we have created for Bridgestone.

There are few passionate professionals in the field of supply chains who can deal with the business reality and the advanced quantitative models. Professor Bram De Smet is one of those. He has inspired me, as well as many other supply chain professionals around the globe. In February 2018, when we finally got the chance to meet in person, I shared my idea of writing a book about supply chain and data science. He simply said, "Just go for it and enjoy it to the fullest." Thank you, Bram, for believing in me and pushing me to take that first step.

Just like forests are stronger than a single tree by itself, I like to surround myself with supportive and bright friends. I especially would like to thank each and every one of the following amazing people for their feedback and support: Gil Vander Marcken, Charles Hoffremont, Bruno Deremince, and Emmeline Everaert, Romain Faurès, Alexis Nsamzinshuti, François Grisay, Fabio Periera, Nicolas Pary, Flore Dargent, and Gilles Belleflamme. And of course, a special thanks goes to Camille Pichot. They have all helped me to make this book more comprehensive and more complete. I have always appreciated feedback from others to improve my work, and I would never have been able to write this book alone without the help of this fine team of supportive friends.

On another note, I would also like to mention Daniel Stanton for the time he took to share his experience with business book publishing with me.

Last but not least, I would like to thank Jonathan Vardakis truly. Without his dedicated reviews and corrections, this book would simply not have come to its full completion. Throughout this collaboration, I have realized that we are a perfect fit together to write a book. Many thanks to you, Jon.

Nicolas Vandeput
November 2017
nicolas.vandeput@supchains.com

About the Author

Nicolas Vandeput is a supply chain data scientist specialized in demand forecasting and inventory optimization. He founded his consultancy company SupChains in 2016 and co-founded SKU Science—a smart online platform for supply chain management—in 2018. He enjoys discussing new quantitative models and how to apply them to business reality. Passionate about education, Nicolas is both an avid learner and enjoys teaching at universities; he has taught forecasting and inventory optimization to master's students since 2014 in Brussels, Belgium. He published *Data Science for Supply Chain Forecasting* in 2018 and *Inventory Optimization: Models and Simulations* in 2020.

Contents

Foreword – Second Edition

In a recent interview, I was asked what the two most promising areas in the field of forecasting were. My answer was "Data Science" and "Supply Chain" that combined, were going to fundamentally shape the theory and practice of forecasting in the future, providing unique benefits to business firms able to exploit their value. The second edition of *Data Science for Supply Chain Forecasting* is essential reading for practitioners in search of information on the newest developments in these two fields and ways of harnessing their advantages in a pragmatic and useful way.

Nicolas Vandeput, a supply chain data scientist, is an academic practitioner perfectly knowledgeable in the theoretical aspects of the two fields, having authored two successful books, but who is also a consultant, having founded a successful company, and well connected with some of the best known practitioners in the supply chain field as referenced in his acknowledgments. The third part of this book has benefited from advice from another prominent practitioner, Michael Gilliland, author of the *Business Forecasting Deal* (2nd ed.) while Evangelos Spiliotis, my major collaborator in the M4 and M5 competitions, has reviewed in detail the statistical models included in the book.

Nicolas' deep academic knowledge combined with his consulting experience and frequent interactions with experienced experts in their respected fields are the unique ingredients of this well-balanced book covering equally well both the theory and practice of forecasting. This second edition expands on his successful book, published in 2018, with more than 50% new content and a large, second part of over 150 pages, describing machine learning (ML) methods, including gradient boosting ones similar to the lightGBM that won the M5 accuracy competition and was used by the great majority of the 50 top contestants. In addition, there are two new chapters in the third part of the book, covering the critical areas of judgmental forecasting and forecast value added aimed at guiding the effective supply chain implementation process within organizations.

The objective of *Data Science for Supply Chain Forecasting* is to show practitioners how to apply the statistical and ML models described in the book in simple and actionable "do-it-yourself" ways by showing, first, how powerful the ML methods are, and second, how to implement them with minimal outside help, beyond the "do-it-yourself" descriptions provided in the book.

Prof Spyros Makridakis
Founder of the Makridakis Open Forecasting Center (MOFC)
and organizer of the M competitions
Institute For the Future (IFF), University of Nicosia

Foreword – First Edition

Tomorrow's supply chain is expected to provide many improved benefits for all stakeholders, and across much more complex and interconnected networks than the current supply chain.

Today, the practice of supply chain science is striving for excellence: innovative and integrated solutions are based on new ideas, new perspectives and new collaborations, thus enhancing the power offered by data science.

This opens up tremendous opportunities to design new strategies, tactics and operations to achieve greater anticipation, a better final customer experience and an overall enhanced supply chain.

As supply chains generally account for between 60% and 90% of all company costs (excluding financial services), any drive toward excellence will undoubtedly be equally impactful on a company's performance as well as on its final consumer satisfaction.

This book, written by Nicolas Vandeput, is a carefully developed work emphasizing how and where data science can effectively lift the supply chain process higher up the excellence ladder.

This is a gap-bridging book from both the research and the practitioner's perspective, it is a great source of information and value.

Firmly grounded in scientific research principles, this book deploys a comprehensive set of approaches particularly useful in tackling the critical challenges that practitioners and researchers face in today and tomorrow's (supply chain) business environment.

Prof. Dr. Ir. Alassane B. Ndiaye
Professor of Logistics & Transport Systems
Universite Libre de Bruxelles, Belgium

Introduction

Artificial intelligence is the new electricity.

Andrew Ng[1]

In the same way electricity revolutionized the second half of the 19th century, allowing industries to produce more with less, artificial intelligence (AI) will drastically impact the decades to come. While some companies already use this new electricity to cast new light upon their business, others are still using old oil lamps or even candles, using manpower to manually change these candles every hour of the day to keep the business running.

As you will discover in this book, AI and machine learning (ML) are not just a question of coding skills. Using data science to solve a problem will require more a scientific mindset than coding skills. We will discuss many different models and algorithms in the later chapters. But as you will see, you do not need to be an IT wizard to apply these models. There is another more important story behind these: a story of experimentation, observation, and questioning everything—a truly scientific method applied to supply chain. In the field of data science as well as supply chain, simple questions do not come with simple answers. To answer these questions, you need to think like a scientist and use the right tools. In this book, we will discuss how to do both.

Supply Chain Forecasting Within all supply chains lies the question of planning. The better we evaluate the future, the better we can prepare ourselves. The question of future uncertainty, how to reduce it, and how to protect yourself against this unknown has always been crucial for every supply chain. From negotiating contract volumes with suppliers to setting safety stock targets, everything relates to the ultimate question:

[1]Andrew Ng is the co-founder of Coursera, the leading online-classes platform.

What Is Tomorrow Going to Be Like?

Yesterday, big companies provided forecasting software that allowed businesses to use a statistical forecast as the backbone of their S&OP[2] process. These statistical forecast models were proposed sixty years ago by Holt and Winters[3] and haven't change much since: at the core of any statistical forecast tool, you still find exponential smoothing. Software companies sell the idea that they can add a bit of extra intelligence into it, or some less-known statistical model, but in the end, it all goes back to exponential smoothing, which we will discuss in the first part of this book. In the past, one demand planner on her/his own personal computer couldn't compete with these models.

Today, things have changed. Thanks to the increase in computing power, the in-flow of data, better models, and the availability of free tools, one can make a difference. **You** can make a difference. With a few coding skills and an appetite for experimentation, powered by machine learning models, you will be able to bring to any business more value than any off-the-shelf forecasting software can deliver. We will discuss machine learning models in Part II.

We often hear that the recent rise of artificial intelligence (or machine learning) is due to an increasing amount of data,- available, as well as cheaper computing power. This is not entirely true. Two other effects explain the recent interest in machine learning. In previous years, many machine learning models were improved, giving better results. As these models became better and faster, the tools to use them have become more user-friendly. It is much easier today to use powerful machine learning models than it was ten years ago.

Tomorrow, demand planners will have to learn to work hand-in-hand with advanced ML-driven forecast models. Demand planners will be able to add value to those models as they understand the ML shortcomings. We will discuss this in Part III.

How to Read This Book

Data Science for Supply Chain Forecasting is written the way I wish someone explained to me how to forecast supply chain products when I started my career. It is divided into three parts: we will first discuss statistical models, then move to machine learning models, and, finally, discuss how to manage an *efficient* forecasting process.

Old-school Statistics and Machine Learning One could think that these statistical models are already outdated and useless as machine learning models will take over. But this is wrong. These old-school models will allow us to *understand* and *see* the demand

[2]The sales and operations planning (S&OP) process focuses on aligning mid- and long-term demand and supply.

[3]See Section 3.3 for more information about Holt-Winters models.

patterns in our supply chain. Machine learning models, unfortunately, won't provide us any explanation nor understanding of the different patterns. Machine learning is only focused on one thing: getting the right answer. The *how* does not matter. This is why both the statistical models and the machine learning models will be helpful for you.

Concepts and Models The first two parts of this book are divided into many chapters: each of them is either a new model or a new concept. We will start by discussing statistical models in Part I, then machine learning models in Part II. Both parts will start with simple models and end with more powerful (and complex) ones. This will allow you to build your understanding of the field of data science and forecasting step by step. Each new model or concept will allow us to overcome a limitation or to go one step further in terms of forecast accuracy.

On the other hand, not every single existing forecast model is explained here. We will only focus on the models that have proven their value in the world of supply chain forecasting.

Do It Yourself We also make the decision not to use any black-box forecasting function from Python or Excel. The objective of this book is not to teach you how to use software. It is twofold. Its first purpose is to teach you how to experiment with different models on your own datasets. This means that you will have to tweak the models and experiment with different variations. You will only be able to do this if you take the time to implement these models yourself. Its second purpose is to allow you to acquire in-depth knowledge on how the different models work as well as their strengths and limitations. Implementing the different models yourself will allow you to learn by doing as you test them along the way.
At the end of each chapter, you will find a *Do It Yourself* (DIY) section that will show you a step-by-step implementation of the different models. I can only advice you to start testing these models on your own datasets ASAP.

Can I Do This? Is This Book for Me?

Data Science for Supply Chain Forecasting has been written for supply chain practitioners, demand planners, and analysts who are interested to understand the inner workings of the forecasting science.[4] By the end of the book, you will be able to create, fine-tune, and use **your own models** to populate a demand forecast for your supply chain. Demand planners often ask me what the best model is for demand forecasting. I always explain that there is no such thing as a *perfect* forecasting model that could beat any other model for any business. As you will see, tailoring models to your demand dataset will allow you to achieve a better level of accuracy than by using black-box tools. This will be especially appreciable

[4]Even though we will focus on supply chain demand forecasting, the principles and models explained in this book can be applied to any forecasting problem.

for machine learning, where there is definitely no one-size-fits-all model or silver bullet: machine learning models need to be tailor-fit to the demand patterns at hand.

You do not need technical IT skills to start using the models in this book today. You do not need a dedicated server or expensive software licenses—only your own computer. You do not need a PhD in Mathematics: we will only use mathematics when it is directly useful to tweak and understand the models. Often—especially for machine learning—a deep understanding of the mathematical inner workings of a model will not be necessary to optimize it and understand its limitations.

The Data Scientist's Mindset

As the business world discovers data science, many supply chain practitioners still rely on rules of thumb and simple approximations to run their businesses. Most often, the work is done directly in Excel. A paradigm shift will be needed to go from manual approximations done in Excel toward automated powerful models in Python. We need to leave oil lamps behind and move to electricity. This is what we will do—step by step—in this book. Before discussing our supply chain data-scientist tools, let's discuss what our data scientist mindset should be.

Data is gold. If artificial intelligence is the new electricity—allowing us to achieve more in a smarter way—data is the modern gold. Gold, unfortunately, does not grow on trees; it comes from gold ore that needs to be extracted and cleaned. Data is the same: it needs to be mined, extracted, and cleaned. We even have to think about where to mine to get the data. As supply chain data scientists, we are both goldsmiths and miners. Even though this book does not cover the specific topic of data cleaning nor the question of data governance, it will show you how to make the best use out of data.

Data cleaning: it takes time, it is not sexy, it is required.

Start small, and iterate. It is easy to lose yourself in details as you try to model the real business world. To avoid this, we will always start tackling broad supply chain questions with simple models. And then we will iterate on these models, adding complexity layers one by one. It is an illusion to think that one could grasp all the complexity of a supply chain at once in one model. As your understanding of a supply chain grows, so does the complexity of your model.

Experiment! There is no definitive answer nor model to each supply chain question. We are not in a world of one-size-fits-all. A model that worked for one company might not work for you. This book will propose many models and ideas and will give you the tools to play with them and to experiment. Which one to choose in the end is up to you! I can only encourage you to experiment with small variations on them—and then bigger ones—until you find the one that suits your business.

Unfortunately, many people forget that experimenting means trial and error. Which means that you will face the risk of failing. Experimenting with new ideas and models is not a linear task: days or weeks can be invested in dead-ends. On the other hand, a single stroke of genius can drastically improve a model. What is important is to fail fast and start a new cycle rapidly. Don't get discouraged by a couple of trials without improvement.

Automation is the key to fast experimentation. As you will see, we will need to run tens, hundreds, or even thousands of experiments on some datasets to find the best model. In order to do so, only automation can help us out. It is tremendously important to keep our data workflow fully automated to be able to run these experiments without any hurdle. Only automation will allow you to scale your work.

Automation is the key to reliability. As your model grows in complexity and your datasets grow in size, you will need to be able to reliably populate results (and act upon them). Only an automated data workflow coupled to an automated model will give you reliable results over and over again. Manual work will slow you down and create random mistakes, which will result in frustration.

Don't get misled by overfitting and luck. As we will see in Chapter 8, overfitting (i.e., your model will work extremely well on your current dataset, but fail to perform well on new data) is the number one curse for data scientists. Do not get fooled by luck or overfitting. You should always treat astonishing results with suspicion and ask yourself the question: *Can I replicate these results on new (unseen) data?*

Sharing is caring. Science needs openness. You will be able to create better models if you take the time to share their inner workings with your team. Openly sharing results (good and bad) will also create trust among the team. Many people are afraid to share bad results, but it is worth doing so. Sharing bad results will allow you to trigger a debate among your team to build a new and better model. Maybe someone external will bring a brand-new idea that will allow you to improve your model.

Simplicity over complexity. As a model grows bigger, there is always a temptation to add more and more specific rules and exceptions. Do not go down this road. As more special rules add up in a model, the model will lose its ability to perform reliably well on new data. And soon you will lose the understanding of all the different interdependencies. You should always prefer a structural fix to a specific new rule (also known as *quick fix*). As the pile of quick fixes grows bigger, the potential amount of interdependences will exponentially increase and you will not be able to identify why your model works in a specific way.

Communicate your results. Communication skills are important for data scientists. Communicating results in the best way is also part of the data science process: clarity comes from simplicity. When communicating your results, always ask yourself the following questions:

- Who am I communicating to?

- What are they interested in?
- Do I show them everything they are interested in?
- Do I show them only what they are interested in?

In a world of big data, it is easy to drown someone in numbers and graphs. Just keep your communication simple and straight to the point. Remember that our brain easily analyzes and extrapolates graphs and curves. So prefer a simple graph to a table of data when communicating your results.

> *Perfection is achieved not when there is nothing more to add, but when there is nothing left to take away.*
>
> Antoine de Saint-Exupéry

The Data Scientist's Toolkit

We will use two tools to build our models, experiment, and share our results: Excel and Python.

Excel

Excel is the data analyst's Swiss knife. It will allow you to easily perform simple calculations and to plot data. The big advantage of Excel compared to any programming language is that we can **see** the data. It is much easier to debug a model or to test a new one if you see how the data is transformed at each step of the process. Therefore, Excel can be the first go-to in order to experiment with new models or data.

Excel also has many limitations. It won't perform well on big datasets and will hardly allow you to automate difficult tasks.

Python

Python is a programming language initially published in 1991 by Guido van Rossum, a Dutch computer scientist. If Excel is a Swiss knife, Python is a full army of construction machines awaiting instructions from any data scientist. Python will allow you to perform computations on huge datasets in a fast, automated way. Python also comes with many libraries dedicated to data analysis (pandas), scientific computations (NumPy and SciPy), or machine learning (scikit-learn).[5] These will soon be your best friends.

[5]See Pedregosa et al. (2011); Virtanen et al. (2020); Oliphant (2006); Hunter (2007); McKinney (2010).

Why Python? We chose to use Python over other programming languages as it is both user-friendly (it is easy to read and understand) and one of the most used programming languages in the world.

Should You Start Learning Python? Yes, you should.

Excel will be perfect to visualize results and the different data transformation steps you perform, but it won't allow you to scale your models to bigger datasets nor to easily automate any data cleaning. Excel is also unable to run any machine learning algorithm.

Many practitioners are afraid to learn a coding language. Everyone knows a colleague who uses some macros/VBA in Excel—maybe you are this colleague—and the complexity of these macros might be frightening. **Python is much simpler than Excel macros.** It is also **much more powerful**. As you will see for yourself in the following chapters, even the most advanced machine learning models won't require so many lines of code or complex functions. It means that you do not have to be an IT genius to use machine learning on your own computer. You can do it yourself, today. Python will definitively give you an edge compared to anyone using Excel.

Today is a great day to start learning Python. Many resources are available, such as videos, blogs, articles, and books. You can, for example, look for Python courses on the following online platforms:

www.edx.org
www.coursera.org
www.udemy.com
www.datacamp.com

I personally recommend the MIT class *"Introduction to Computer Science and Programming Using Python,"* available on EdX.[6] This will teach you everything you need to know about Python to start using the models presented in this book.

If you want a short introduction to Python and want to learn along the way, I introduce the most useful concepts in Appendix A. This will be enough for you to understand the first code extracts.

Python Code Extracts

Simplicity vs. Efficiency The various Python code extracts throughout the book are made with the objectives of simplicity and clarity. Simple code is much easier to understand, maintain, share, and improve than complex code. This simplification was sometimes done at the expense of efficiency or speed. This means that the codes are not as fast as an

[6]See MITx (2019).

experienced Python user could produce, but the implementations are easy to understand—which is the primary goal here.

Python Libraries We will use throughout the book some of Python's very well-known libraries. As you can see here, we will use the usual import conventions. For the sake of clarity, we won't show the `import` lines over and over again in each code extract.

```python
import numpy as np
import pandas as pd
import scipy.stats as stats
from scipy.stats import norm
import matplotlib.pyplot as plt
```

Other Resources

You can download the Python code shown in this book as well as the Excel templates on `supchains.com/resources-2nd` (password: `XXXXXXXXXX`). There is also a Glossary (and an Index) at the end of the book, where you can find a short description of all the specific terms we will use. Do not hesitate to consult it if you are unsure about a term or an acronym.

Part I

Statistical Forecasting

Chapter 1

Moving Average

The first forecast model that we will develop is the simplest. As supply chain data scientists, we love to start experimenting quickly. First, with a simple model, then with more complex ones. Henceforth, this chapter is more of a pretext to set up our first forecast function in Python and our Excel template—we will use both in all of the following chapters.

1.1 Moving Average Model

The moving average model is based on the idea that **future demand is similar to the recent demand we observed**. With this model, we simply assume that the forecast is the average demand during the last n periods. If you look at monthly demand, this could translate as: *"We predict the demand in June to be the average of March, April, and May."*

If we formalize this idea, we obtain this formula:

$$f_t = \frac{1}{n}\sum_{i=1}^{n} d_{t-i}$$

Where,

f_t is the forecast for period t
n is the number of periods we take the average of
d_t is the demand during period t

Initialization As you will see for further models, we always need to discuss how to initialize the forecast for the first periods. For the moving average method, we won't have a forecast until we have enough historical demand observations. So the first forecast will be done for $t = n + 1$.

Future Forecast Once we are out of the historical period, we simply define any future forecast as the last forecast that was computed based on historical demand. This means that, with this model, the future forecast is flat. This will be one of the major restrictions of this model: its inability to extrapolate any trend.

Notation

In the scientific literature, you will often see the output you want to predict noted as y. This is due to the mathematical convention where we want to estimate y based on x. A prediction (a forecast in our case) would then be noted $\hat{y}$. This hat represents the idea that we do an estimation of y. To make our models and equations as simple to read and understand as possible, we will avoid this usual convention and use something more practical:

Demand will be noted as **d**
Forecast will be noted as **f**

When we want to point to a specific occurrence of the forecast (or the demand) at time t, we will note it f_t (or d_t). Typically:

$\boldsymbol{d_0}$ is the demand at period 0 (e.g., first month, first day, etc.)
$\boldsymbol{f_0}$ is the forecast for the demand of period 0

We will call the demand of each period a **demand observation**. For example, if we measure our demand on monthly buckets, it means that we will have 12 demand observations per year.

1.2 Insights

In Figure 1.1, we have plotted two different moving average forecasts.

As you can see, the moving average forecast where $n = 1$ is a rather specific case: the forecast is the demand with a one-period lag. This is what we call a naïve forecast: “Tomorrow will be just as today.”

Naïve Forecast

A naïve forecast is the simplest forecast model: it always predicts the last available observation.

A naïve forecast is interesting as it will instantly react to any variation in the demand, but on the other hand, it will also be sensitive to **noise** and outliers.

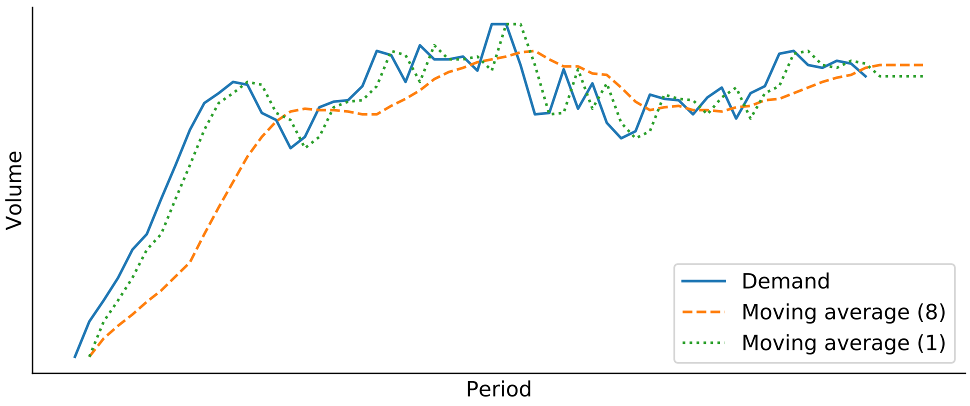


Figure 1.1: Moving averages.

Noise

In statistics, the noise is an unexplained variation in the data. It is often due to the randomness of the different processes at hand.

To decrease this sensitivity, we can go for a moving average based on more previous demand observations ($n > 1$). Unfortunately, the model will also take more time to react to a change in the demand level. In Figure 1.1, you can observe that the moving average with $n = 8$ takes more time to *react* to the changing demand level during the first phase. But during the second phase, the forecast is more stable than the naïve one. **We have to make a trade-off between reactivity and smoothness.** As you will see, we will have to make this trade-off repeatedly in all the exponential smoothing models that we will see later.

Limitations

There are three main limitations at the core of a moving average.

1. **No Trend** The model does not see any trend (and therefore won't project any). We will learn how to include those in Chapters 5 and 7.
2. **No Seasonality** The model will not properly react to seasonality. We will include seasonality in Chapters 9 and 11.
3. **Flat Historical Weighting** A moving average will allocate an *equal* weight to all the historical periods that are taken into account. For example, if you use a moving average with $n = 4$, you allocate a weight (importance) of 25% to the last four periods. But, the latest observation should somehow be more important than the one four periods ago. For example, if you want to forecast June based on the demand

of the latest months, you should give more importance to the demand observed in May compared to any other month.
Although, you might also want to give a small importance to the demand observed five periods ago: May might be the *most* interesting month, but there may be *some* interest to look at last December as well.
We will solve this in Chapter 3 by using an *exponential*—rather than a *flat* —weighting of the historical periods.

1.3 Do It Yourself

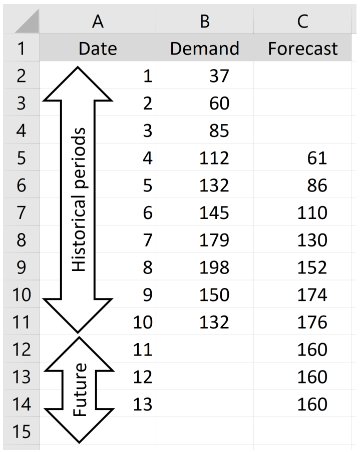
Excel

Let's build an example with a moving average model based on the last three demand occurrences ($n = 3$), as shown in Figure 1.2, by following these steps:

1. We start our data table by creating three columns:
   Date in column `A`
   Demand in column `B`
   Forecast in column `C`
   You can define the first line as Date = 1 (cell `A2=1`) and increase the date by one on each line.
2. For the sake of the example, we will always use the same dummy demand in Excel. You can type these numbers (starting on date 1 until date 10): `37, 60, 85, 112, 132, 145, 179, 198, 150, 132`.
3. We can now define the first forecast on date 4. You can simply use the formula `C5=AVERAGE(B2:B4)` and copy and paste it until the end of the table. You can continue until row 12 (date 11), which will be the last forecast based on historical demand.
4. The future forecasts will all be equivalent to this last point. You can use `C13=C12` and copy and paste this formula until as far as you want to have a forecast. In the example, we go until date 13.
5. You should now have a table that looks like Figure 1.2.

Python

If you are new to Python and you want to get a short introduction, I introduce the most useful concepts in Appendix A. This will be enough to understand the first code extracts and learn along the way.



	A	B	C
1	Date	Demand	Forecast
2	1	37	
3	2	60	
4	3	85	
5	4	112	61
6	5	132	86
7	6	145	110
8	7	179	130
9	8	198	152
10	9	150	174
11	10	132	176
12	11		160
13	12		160
14	13		160
15			

Historical periods
Future

Figure 1.2: Final table for moving average.

Moving Average Function

Throughout the book, we will implement multiple models in separate functions. These functions will be convenient as they will all use the same kind of inputs and return similar outputs. These functions will be the backbone of your statistical forecast toolbox, as by keeping them consistent you will be able to use them in an optimization engine, as shown in Chapter 6.

We will define a function `moving_average(d, extra_periods=1, n=3)` that takes three inputs:

d A time series that contains the historical demand (can be a list or a NumPy array)

`extra_periods` The number of periods we want to forecast in the future

n The number of periods we will average

```python
def moving_average(d, extra_periods=1, n=3):

    # Historical period length
    cols = len(d)
    # Append np.nan into the demand array to cover future periods
    d = np.append(d,[np.nan]*extra_periods)
    # Define the forecast array
    f = np.full(cols+extra_periods,np.nan)

    # Create all the t+1 forecast until end of historical period
    for t in range(n,cols):
        f[t] = np.mean(d[t-n:t])

    # Forecast for all extra periods
    f[t+1:] = np.mean(d[t-n+1:t+1])

    # Return a DataFrame with the demand, forecast & error
    df = pd.DataFrame.from_dict({'Demand':d,'Forecast':f,'Error':d-f})

    return df
```

Python Mastery – NumPy

If you are new to Python, you will notice that we have introduced two special elements in our code:

`np.nan` is a way to represent something that is not a number (nan stands for **n**ot **a** **n**umber).

In our function, we used `np.nan` to store dummy values in our array `f`, until they get replaced by actual values. If we had initialized `f` with actual digits (e.g., ones or zeros), this could have been misleading as we wouldn't know if these ones or zeros were actual forecast values or just the dummy initial ones.

`np.full(shape,value)` this function will return an array of a certain `shape`, filled in with the given `value`.

In our function, we used `np.full()` to create our forecast array `f`.

Our function `moving_average(d,extra_periods,n)` will return a DataFrame. We can save it to use it later, as shown here with a simple demand time series:

```
d = [28,19,18,13,19,16,19,18,13,16,16,11,18,15,13,15,13,11,13,10,12]
df = moving_average(d, extra_periods=4, n=3)
```

Visualization with Pandas

You can easily plot any DataFrame simply by calling the method `.plot()` on it. Typically, if you want to plot the demand and the forecast that we just populated, you can simply type:

```
df[['Demand','Forecast']].plot()
```

You will get a figure similar to Figure 1.3.

You can also customize `.plot()` by specifying some parameters.

`figsize(width,height)` Defines the size of the figure. Dimensions are given in inches.

`title` Displays a title if given.

`ylim=(min,max)` Determines the range of the y axis of our plot.

`style=[]` Defines the style of each of the lines that are plotted. `'-'` will be a continuous line whereas `'--'` will be a discontinous line.

Here's an example:

```
df[['Demand','Forecast']].plot(figsize=(8,3), title='Moving average', ylim=(0,
↪ 30), style=['-','--'])
```

By default, `.plot()` will use the DataFrame index as the x axis. Therefore, if you want to display a legend on the x axis, you simply can name the DataFrame index.

```
df.index.name = 'Period'
```

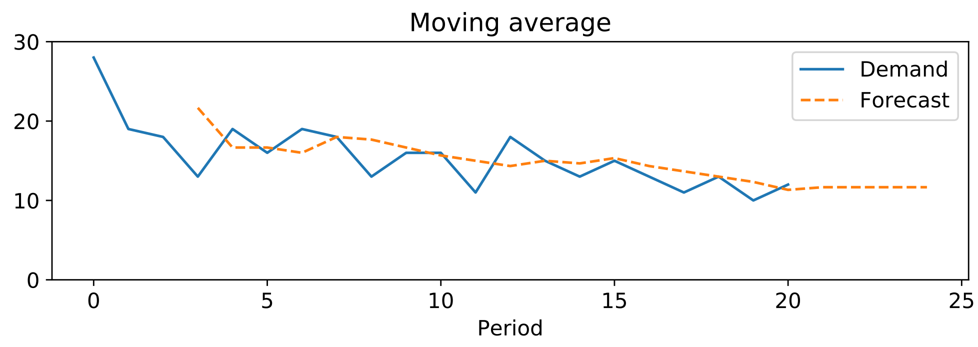


Figure 1.3: `.plot()` output.

As you can see in Figure 1.3, the future forecast (as of period 20) is flat. As discussed, this is due to the fact that this moving average model does not *see* a trend and, therefore, can't project any.

Chapter 2

Forecast KPI

Note to the Reader

This chapter focuses on the *quantitative* aspects of forecast accuracy. For the sake of simplicity, we will look at the error on the very next period (lag 1) and at a single item at a time. In Chapter 27, we will discuss which lags are the most important as well as how to deal with forecast error across a product portfolio.

2.1 Forecast Error

Now that we have created our first forecast model, we need to quantify its accuracy. As you will see, measuring forecast accuracy (or error) is not an easy task, as **there is no one-size-fits-all indicator**. Only experimentation will show you which **Key Performance Indicator** (**KPI**) is best for you. As you will see, each indicator will avoid some pitfalls but will be prone to others.

The first distinction we have to make is the difference between the **accuracy** of a forecast and its **bias**.

Accuracy

The accuracy of your forecast measures how much spread you had between your forecasts and the actual values. The accuracy gives an idea of the magnitude of the errors, but not their overall direction.

Bias

The bias represents the overall direction of the historical average error. It measures if your forecasts were on average too high (i.e., you *overshot* the demand) or too low (i.e., you *undershot* the demand).

Of course, as you can see in Figure 2.1, what we want to have is a forecast that is both accurate and unbiased.

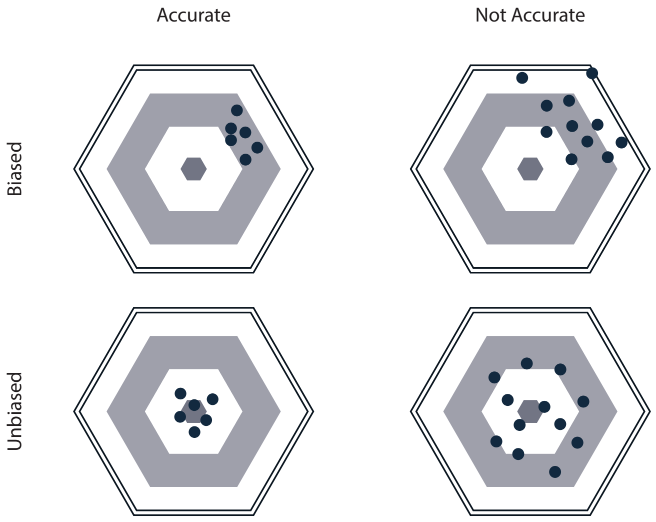


Figure 2.1: Accuracy and bias.

Computing the Forecast Error

Let's start by defining the error during one period (e_t) as the difference between the forecast (f_t) and the demand (d_t).

$$e_t = f_t - d_t$$

Note that with this definition, if the forecast overshoots the demand, the error will be positive; if the forecast undershoots the demand, the error will be negative.

DIY

Excel You can easily compute the error as the forecast minus the demand.

Starting from our example from Section 1.3, you can do this by inputting `=C5-B5` in cell `D5`. This formula can then be dragged onto the range `C5:D11`.

Python You can access the error directly via `df['Error']` as it is included in the DataFrame returned by our function `moving_average(d)`.

2.2 Bias

The (average) bias of a forecast is defined as its average error.

$$\text{bias} = \frac{1}{n} \sum_{n} e_t$$

Where n is the number of historical periods where you have both a forecast and a demand (i.e., periods where an error can be computed).

The bias alone won't be enough to evaluate your forecast accuracy. Because a positive error in one period can offset a negative error in another period, a forecast model can achieve very low bias and not be accurate at the same time. Nevertheless, a highly biased forecast is already an indication that something is wrong in the model.

Scaling the Bias The bias computed as in the formula above will give you an *absolute* value like 43 or -1400. As a demand planner investigating your product forecasts, you should ask yourself the following question: *Is 43 a good bias?* Without information about the product's average demand, you cannot answer this question. Therefore, a more relevant KPI would be the *scaled* bias (or *normalized* bias). We can compute it by dividing the total error by the total demand (which is the same as dividing the average error by the average demand).

$$\text{bias\%} = \frac{\frac{1}{n}\sum e_t}{\frac{1}{n}\sum d_t} = \frac{\sum e_t}{\sum d_t}$$

Attention Point

A common mistake (especially in Excel) is to divide the average error observed in a specific time period by the average demand observed in another (wider) time range. As you will see in the DIY sections, be sure to divide the average error by the average demand during the **corresponding** period.

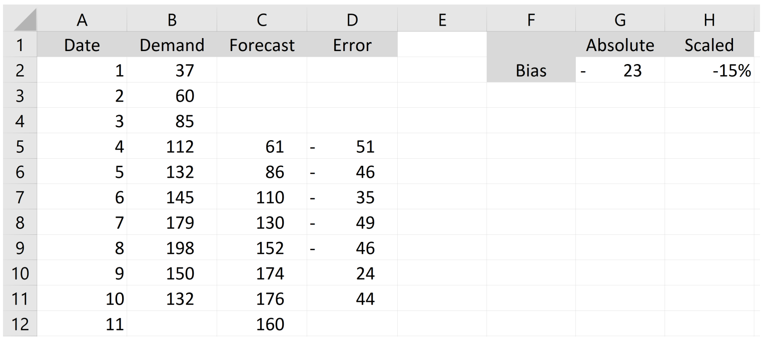
Pro-Tip

It usually brings no insights to compute the bias of *one* item during *one* period. You should either compute it for many products at once (during one period) or compute it for a single item over many periods (best to perform its computation over a full season cycle).

DIY – Bias in Excel

You can compute the average bias by simply averaging the forecast error in Excel.

In our example, as shown in Figure 2.2, you can compute it as `G2=AVERAGE(D5:D11)`. You can also compute the scaled bias by dividing the absolute bias by the average demand with `H2=G2/AVERAGE(B5:B11)`. Note that we use the corresponding cell range in both columns `B` and `D`: we do **not** include the demand observed in periods 1 to 3 in our demand average, as there is no forecast error computed in those periods.



	A	B	C	D	E	F	G	H
1	Date	Demand	Forecast	Error			Absolute	Scaled
2	1	37				Bias	- 23	-15%
3	2	60						
4	3	85						
5	4	112	61	- 51				
6	5	132	86	- 46				
7	6	145	110	- 35				
8	7	179	130	- 49				
9	8	198	152	- 46				
10	9	150	174	24				
11	10	132	176	44				
12	11		160					

Figure 2.2: Bias computation in Excel.

DIY – Bias in Python

In Python, we will create a function `kpi(df)` that will take a forecast DataFrame as an input and print the bias.

```python
def kpi(df):
    dem_ave = df.loc[df['Error'].notnull(),'Demand'].mean()
    bias_abs = df['Error'].mean()
    bias_rel = bias_abs / dem_ave
    print('Bias: {:0.2f}, {:.2%}'.format(bias_abs,bias_rel))
```

Python Mastery – String Formatting

We use here specific formatting instructions in the `print()` function in order to print the absolute and scaled bias as (rounded) absolute value and percentage, respectively. Python provides many string formatting options. See `pyformat.info` for detailed explanations.

When computing the scaled bias, it is important to divide the bias by the average demand during the periods where we could measure forecast error. We define the relevant demand average in line 2, where we select only the rows where an error is defined.

Finally, you should obtain these results:

```
d = [37, 60, 85, 112, 132, 145, 179, 198, 150, 132]
df = moving_average(d, extra_periods=4, n=3)
kpi(df)
>> Bias: 22.95, 15.33%
```

2.3 MAPE

The **Mean Absolute Percentage Error** (or MAPE) is one of the most commonly used KPIs to measure forecast accuracy. MAPE is computed as the average of the **individual** absolute errors divided by the demand (each period is divided separately). To put it simply, it is the average of the percentage absolute errors.

$$\text{MAPE} = \frac{1}{n}\sum_{n}\frac{|e_t|}{d_t}$$

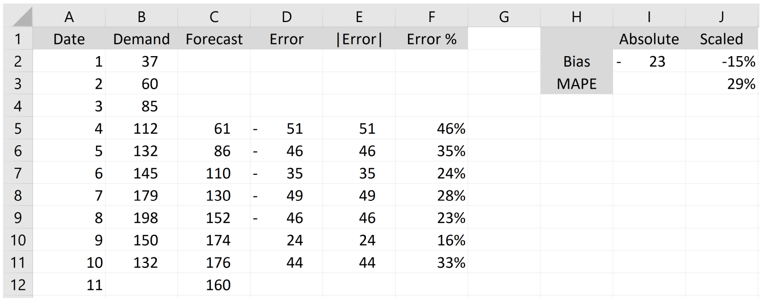
MAPE is a strange forecast KPI. It is quite well-known among business managers, despite being a really poor accuracy indicator. As you can see in the formula, MAPE divides each error individually by the demand, so it is skewed: high errors during low-demand periods will have a major impact on MAPE. You can see this from another point of view: if you choose MAPE as an error KPI, an extremely low forecast (such as 0) can only result in a maximum error of 100%, whereas any too-high forecast will not be capped to a specific percentage error. Due to this, optimizing MAPE will result in a strange forecast that will most likely undershoot the demand. Just avoid it. If MAPE is mentioned in this book, it is not to promote its use, but as a plea *not* to use it.

DIY – MAPE in Excel

Computing the MAPE will require the following modifications in our previous Excel data table:

1. Add two new columns between columns `D` and `E`.
2. Compute the absolute error (`|Error|`) in column `E`. Cell `E5` should be defined as `=ABS(D5)`. You can then drag out this formula until cell `E11`.
3. You can then compute the scaled error in column `F`. Start by typing `F5=E5/B5` and drag out this formula until `F11`.
4. You can now compute the MAPE in cell `J3` as `=AVERAGE(F5:F11)`.

You should obtain a table similar to Figure 2.3



	A	B	C	D	E	F	G	H	I	J
1	Date	Demand	Forecast	Error	|Error|	Error %			Absolute	Scaled
2	1	37						Bias	- 23	-15%
3	2	60						MAPE		29%
4	3	85								
5	4	112	61	- 51	51	46%				
6	5	132	86	- 46	46	35%				
7	6	145	110	- 35	35	24%				
8	7	179	130	- 49	49	28%				
9	8	198	152	- 46	46	23%				
10	9	150	174	24	24	16%				
11	10	132	176	44	44	33%				
12	11		160							

Figure 2.3: MAPE computation in Excel.

Going Further

You can use an array formula to compute the MAPE without the two extra error columns. You can define cell `J3` as:

```
J3 = AVERAGE(ABS(D5:D11)/B5:B11)
```

If you are not familiar with Excel array formulas, these are formulas that can perform operations over multiple cells (hence the term *array*). In order to use one, simply type your formula (for example, the one above), then validate the cell by pressing `CTRL+SHIFT+ENTER` (and not simply `ENTER`). If the formula is properly validated, you should see it being surrounded by `{ }`. Array formulas are powerful tools in Excel, but can be confusing for the users. Use them with care.

DIY – MAPE in Python

Let's update our `kpi(df)` function to print the MAPE as well.

```python
def kpi(df):
    dem_ave = df.loc[df['Error'].notnull(),'Demand'].mean()
    bias_abs = df['Error'].mean()
    bias_rel = bias_abs / dem_ave
    print('Bias: {:0.2f}, {:.2%}'.format(bias_abs,bias_rel))
    MAPE = (df['Error'].abs()/df['Demand']).mean()
    print('MAPE: {:.2%}'.format(MAPE))
```

Note that, unlike for the bias, here we don't need to worry about selecting the proper demand range to compute the MAPE. As we divide `df['Error']` directly by `df['Demand']` **before** computing the mean, pandas by default is removing the rows where the demand is not defined.

You should obtain the following results:

```
kpi(df)
>> Bias: 22.95, 15.33%
>> MAPE: 29.31%
```

2.4 MAE

The **Mean Absolute Error** (MAE) is a very good KPI to measure forecast accuracy. As the name implies, it is the mean of the absolute error.

$$\text{MAE} = \frac{1}{n}\sum_{n} |e_t|$$

As for the bias, the MAE is an absolute number. If you are told that MAE is 10 for a particular item, you cannot know if this is good or bad. If your average demand is 1,000, it is, of course, astonishing, but if the average demand is 1, an MAE of 10 is a very poor accuracy. To solve this, it is common to divide MAE by the average demand to get a scaled percentage:

$$\text{MAE\%} = \frac{\frac{1}{n}\sum |e_t|}{\frac{1}{n}\sum d_t} = \frac{\sum |e_t|}{\sum d_t}$$

Attention Point

Many practitioners use the MAE formula and call it MAPE. This can cause a lot of confusion. When discussing forecast error with someone, I advise you to explicitly specify how you compute the forecast error to be sure to compare apples with apples.

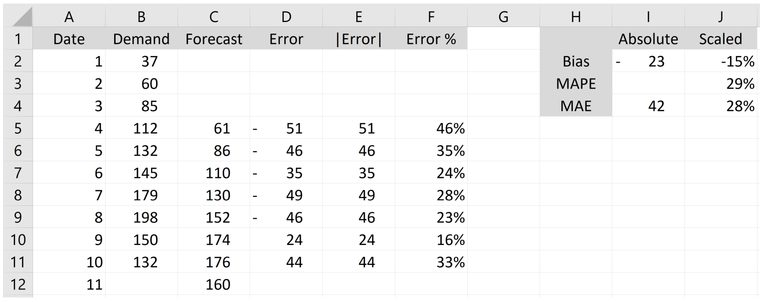
DIY – MAE in Excel

As we already have a column with the absolute error (column `E`), we can easily compute the mean absolute error with these two simple formulas:

MAE (absolute): `I4 = AVERAGE(E5:E11)`

MAE (scaled): `J4 = I4/AVERAGE(B5:B11)`

You should obtain results as shown in Figure 2.4.



	A	B	C	D	E	F	G	H	I	J
1	Date	Demand	Forecast	Error	|Error|	Error %			Absolute	Scaled
2	1	37						Bias	- 23	-15%
3	2	60						MAPE		29%
4	3	85						MAE	42	28%
5	4	112	61	- 51	51	46%				
6	5	132	86	- 46	46	35%				
7	6	145	110	- 35	35	24%				
8	7	179	130	- 49	49	28%				
9	8	198	152	- 46	46	23%				
10	9	150	174	24	24	16%				
11	10	132	176	44	44	33%				
12	11		160							

Figure 2.4: MAE computation in Excel.

DIY – MAE in Python

We can add a few lines to compute the MAE in our `kpi(df)` function.

```python
def kpi(df):
    dem_ave = df.loc[df['Error'].notnull(),'Demand'].mean()
    bias_abs = df['Error'].mean()
    bias_rel = bias_abs / dem_ave
    print('Bias: {:0.2f}, {:.2%}'.format(bias_abs,bias_rel))
    MAPE = (df['Error'].abs()/df['Demand']).mean()
    print('MAPE: {:.2%}'.format(MAPE))
    MAE_abs = df['Error'].abs().mean()
    MAE_rel = MAE_abs / dem_ave
    print('MAE: {:0.2f}, {:.2%}'.format(MAE_abs,MAE_rel))
```

You should now obtain these results:

```
kpi(df)
>> Bias: 22.95, 15.33%
>> MAPE: 29.31%
>> MAE: 42.29, 28.24%
```

2.5 RMSE

The **Root Mean Square Error** (RMSE) is a difficult KPI to interpret, as it is defined as the square root of the average squared forecast error. Nevertheless, it can be very helpful, as we will see later.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{n} e_t^2}$$

Just as for MAE, RMSE is not scaled to the demand, so it needs to be put in percentages to be understandable. We can then define RMSE% as:

$$\text{RMSE\%} = \frac{\sqrt{\frac{1}{n}\sum e_t^2}}{\frac{1}{n}\sum d_t}$$

Actually, many algorithms—especially for machine learning—are based on the **Mean Square Error** (MSE), which is directly related to RMSE.

$$\text{MSE} = \frac{1}{n}\sum_{n} e_t^2$$

Many algorithms use MSE instead of RMSE since MSE is faster to compute and easier to manipulate. But it is not scaled to the original error (as the error is squared), resulting in a KPI that we cannot relate to the original demand scale. Therefore, we won't use it to evaluate our statistical forecast models.

DIY – RMSE in Excel

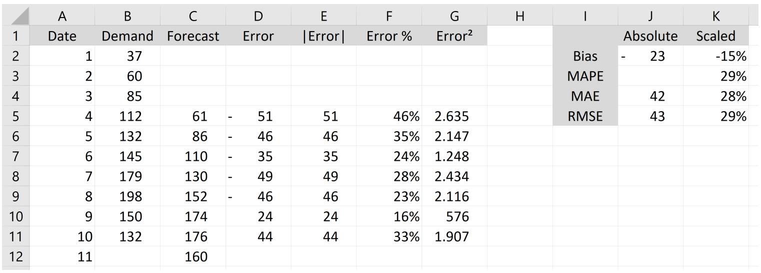
As shown in Figure 2.5, let's add a new column `Error²` in column `G`. You can type `=D5^2` in cell `G5` and then drag out the formula until cell `G11`.

Afterward, you can compute the RMSE in cell `J5` by using the following formula:

```
J5 = SQRT(AVERAGE(G5:G11))
```

The RMSE% can be computed in cell K5 by dividing the RMSE by the average demand:

K5 = J5/AVERAGE(B5:B11)



	A	B	C	D	E	F	G	H	I	J	K
1	Date	Demand	Forecast	Error	|Error|	Error %	Error^2			Absolute	Scaled
2	1	37							Bias	- 23	-15%
3	2	60							MAPE		29%
4	3	85							MAE	42	28%
5	4	112	61	- 51	51	46%	2.635		RMSE	43	29%
6	5	132	86	- 46	46	35%	2.147				
7	6	145	110	- 35	35	24%	1.248				
8	7	179	130	- 49	49	28%	2.434				
9	8	198	152	- 46	46	23%	2.116				
10	9	150	174	24	24	16%	576				
11	10	132	176	44	44	33%	1.907				
12	11		160								

Figure 2.5: RMSE computation in Excel.

DIY – RMSE in Python

As usual, we can simply extend our kpi(df) function to include our new KPI.

```python
def kpi(df):
    dem_ave = df.loc[df['Error'].notnull(),'Demand'].mean()
    bias_abs = df['Error'].mean()
    bias_rel = bias_abs / dem_ave
    print('Bias: {:0.2f}, {:.2%}'.format(bias_abs,bias_rel))
    MAPE = (df['Error'].abs()/df['Demand']).mean()
    print('MAPE: {:.2%}'.format(MAPE))
    MAE_abs = df['Error'].abs().mean()
    MAE_rel = MAE_abs / dem_ave
    print('MAE: {:0.2f}, {:.2%}'.format(MAE_abs,MAE_rel))
    RMSE_abs = np.sqrt((df['Error']**2).mean())
    RMSE_rel = RMSE_abs / dem_ave
    print('RMSE: {:0.2f}, {:.2%}'.format(RMSE_abs,RMSE_rel))
```

You should obtain these results:

```
kpi(df)
>> Bias: 22.95, 15.33%
>> MAPE: 29.31%
>> MAE: 42.29, 28.24%
>> RMSE: 43.20, 28.85%
```

A Question of Error Weighting

Compared to MAE, RMSE does not treat each error the same. It gives more importance to the biggest errors. That means that one big error is enough to get a very bad RMSE.

Let's use an example with a dummy demand time series.

Period	1	2	3	4	5	6	7	8	9	10	11	12
Demand	10	12	14	8	9	5	8	10	12	11	10	15

Let's imagine we want to compare two slightly different forecasts.

Period	1	2	3	4	5	6	7	8	9	10	11	**12**
Demand	10	12	14	8	9	5	8	10	12	11	10	**15**
Forecast #1	12	14	15	10	7	4	5	8	12	14	13	**8**
Error #1	2	2	1	2	-2	-1	-3	-2	0	3	3	**-7**
Forecast #2	12	14	15	10	7	4	5	8	12	14	13	**9**
Error #2	2	2	1	2	-2	-1	-3	-2	0	3	3	**-6**

The only difference in the two datasets is the forecast on the latest demand observation: forecast #1 undershot it by 7 units and forecast #2 undershot it by *only* 6 units. Note that for both forecasts, period 12 is the worst period in terms of accuracy. If we look at the KPI of these two forecasts, this is what we obtain:

KPI	MAE	RMSE
Forecast #1	2.33	2.86
Forecast #2	2.25	2.66

What is interesting here is that by just changing the error of this last period (the one with the *worst* accuracy) by a single unit, we decrease the total RMSE by 6.9% (2.86 to 2.66), but MAE is only reduced by 3.6% (2.33 to 2.25), so the impact on MAE is nearly twice as low. Clearly, RMSE puts much more importance on the largest errors, whereas MAE gives the same importance to each error. You can try this for yourself and reduce the error of one of the *most* accurate periods to observe the impact on MAE and RMSE.

Spoiler: There is nearly no impact on RMSE.[1]

As we will see later, RMSE has some other very interesting properties.

[1] Remember, RMSE is not so much impacted by low forecast error. So reducing the error of the period that has already the lowest forecast error won't significantly impact RMSE.

2.6 Which Forecast KPI to Choose?

We went through the definitions of these KPIs (bias, MAPE, MAE, RMSE), but it is still unclear what difference it can make for our model to use one instead of another. You might think that using RMSE instead of MAE or MAE instead of MAPE doesn't change anything. But nothing is further from the truth.

Let's do a quick example to show this. Imagine a product with a low and rather flat weekly demand that occasionally has a big order (maybe due to promotions, or to clients ordering in batches). Here is the weekly demand observed so far:

	W1	W2	W3	W4	W5
Mon	3	3	4	1	5
Tue	1	4	1	2	2
Wed	5	5	1	1	12
Thu	20	4	3	2	1
Fri	13	16	14	5	20

Now let's imagine we propose three different forecasts for this product. The first one predicts 2 pieces/day, the second one 4 and the last one 6. Let's plot the actual demand and the forecasts in Figure 2.6.

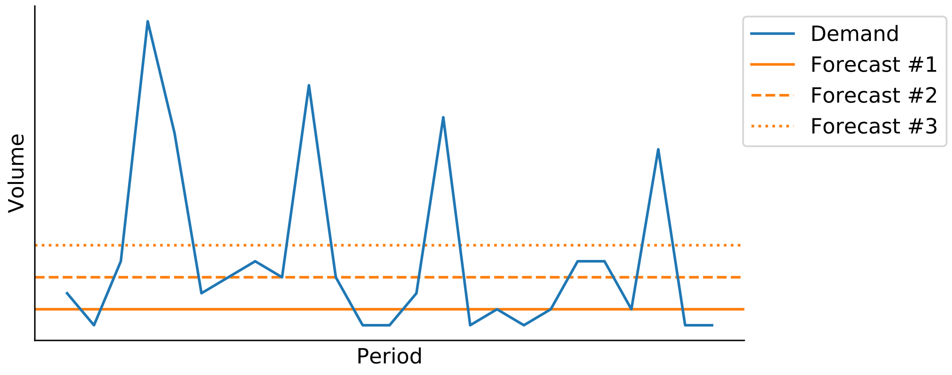


Figure 2.6: Demand and forecasts.

You can see in Table 2.1 how each of these forecasts performed in terms of bias, MAPE, MAE, and RMSE on the historical period. Forecast #1 was the best during the historical periods in terms of MAPE, forecast #2 was the best in terms of MAE, and forecast #3 was the best in terms of RMSE and bias (but the worst on MAE and MAPE).

Table 2.1: KPI comparison.

	Forecast #1	Forecast #2	Forecast #3
Bias	-3.9	-1.9	**0.1**
MAPE	**64%**	109%	180%
MAE	4.4	**4.1**	4.8
RMSE	7.1	6.2	**5.9**

Let's now reveal how these forecasts were made:

Forecast #1 is just a very low amount. It resulted in the best MAPE (but the worst RMSE).

Forecast #2 is the demand *median*.[2] It resulted in the best MAE.

Forecast #3 is the average demand. It resulted in the best RMSE and bias (but the worst MAPE).

Median vs. Average – Mathematical Optimization

Before discussing the different forecast KPIs further, let's take some time to understand why a forecast of the median will get a good MAE while a forecast of the mean will get a good RMSE.

Note to the Reader

The math ahead is not required for you to use the KPIs or the further models in this book. If these equations are unclear to you, this is not an issue—don't get discouraged. Just skip them and jump to the conclusion of the **RMSE** and **MAE** paragraphs.

RMSE

Let's start with RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum e_t^2}$$

Actually, to simplify the algebra, let's use a simplified version, the Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum e_t^2 = \frac{1}{n}\sum (f_t - d_t)^2$$

[2]The median is the value for which half the dataset is higher and half of the dataset is lower.

If you set MSE as a target for your forecast model, it will *minimize* it. You can minimize a mathematical function by setting its derivative to zero. Let's try this.

$$\frac{\partial\,\text{MSE}}{\partial f} = \frac{\partial \frac{1}{n}\sum (f_t - d_t)^2}{\partial f}$$
$$\frac{2}{n}\sum(f_t - d_t) = 0$$
$$\sum f_t = \sum d_t$$

Conclusion To optimize a forecast's (R)MSE, the model will have to aim for the total forecast to be equal to the total demand. That is to say that optimizing (R)MSE aims to produce a prediction that is correct *on average* and, therefore, unbiased.

MAE

Now let's do the same for MAE.

$$\frac{\partial\,\text{MAE}}{\partial f} = \frac{\partial \frac{1}{n}\sum |f_t - d_t|}{\partial f}$$

Or,

$$|f_t - d_t| = \begin{cases} f_t - d_t & d_t < f_t \\ \text{indefinite} & d_t = f_t \\ d_t - f_t & d_t > f_t \end{cases}$$

and

$$\frac{\partial |f_t - d_t|}{\partial f} = \begin{cases} 1 & d_t < f_t \\ \text{indefinite} & d_t = f_t \\ -1 & d_t > f_t \end{cases}$$

Which means that

$$\frac{\partial\,\text{MAE}}{\partial f} = \frac{1}{n}\sum \begin{cases} 1 & d_t < f_t \\ -1 & d_t > f_t \end{cases}$$

Conclusion To optimize MAE (i.e., set its derivative to 0), the forecast needs to be as many times higher than the demand as it is lower than the demand. In other words, we are looking for a value that splits our dataset into two equal parts. This is the exact definition of the *median*.

MAPE

Unfortunately, the derivative of MAPE won't show some elegant and straightforward property. We can simply say that MAPE is promoting a very low forecast as it allocates a high weight to forecast errors when the demand is low.

Conclusion

As we saw in the previous section, we have to understand that a big difference lies in the mathematical roots of RMSE, MAE, and MAPE. The optimization of RMSE will seek to be correct on average. The optimization of MAE will try to overshoot the demand as often as undershoot it, which means targeting the demand median. Finally, the optimization of MAPE will result in a biased forecast that will undershoot the demand. In short, **MAE is aiming at demand median, and RMSE is aiming at demand average.**

MAE or RMSE – Which One to Choose?

Is it best to aim for the median or the average of the demand? Well, the answer is not black and white. As we will discuss in the next pages, each technique has some benefits and some risks. Only experimentation will reveal which technique works best for a specific dataset. You can even choose to use both RMSE and MAE.

Let's take some time to discuss the impact of choosing either RMSE or MAE on forecast bias, outliers sensitivity, and intermittent demand.

Bias

For many products, you will observe that the median demand is not the same as the average demand. The demand will most likely have some peaks here and there that will result in a skewed distribution. These skewed demand distributions are widespread in supply chain, as the peaks can be due to periodic promotions or clients ordering in bulk. This will cause the demand median to be below the average demand, as shown in Figure 2.7.

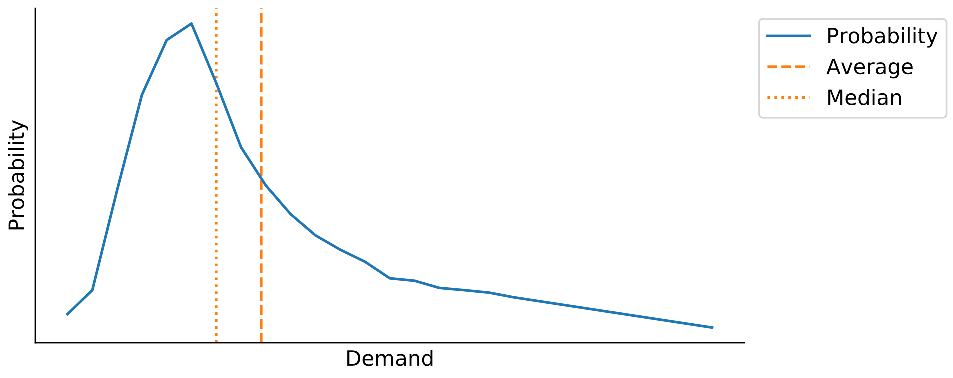


Figure 2.7: Median vs. Average.

This means that a forecast that is **minimizing MAE will result in a bias**, most often resulting in an undershoot of the demand. A forecast that is minimizing RMSE will not result in bias (as it aims for the average). This is definitely MAE's main weakness.

Sensitivity to Outliers

As we discussed, RMSE gives a bigger importance to the highest errors. This comes at a cost: a sensitivity to outliers. Let's imagine an item with a smooth demand pattern as shown in Table 2.2.

Table 2.2: Demand without outliers (median: 8.5, average: 9.5).

Period	1	2	3	4	5	6	7	8	9	10
Demand	16	8	12	9	6	12	5	7	6	14

The median is 8.5 and the average is 9.5. We already observed that if we make a forecast that minimizes MAE, we will forecast the median (8.5) and we would be on average undershooting the demand by 1 unit (bias = -1). You might then prefer to minimize RMSE and to forecast the average (9.5) to avoid this situation. Nevertheless, let's now imagine that we have one new demand observation of 100, as shown in Table 2.3.

Table 2.3: Demand with outliers (median: 8.5, average: 18.1).

Period	1	2	3	4	5	6	7	8	9	10
Demand	16	8	12	9	6	12	5	7	6	100

The median is still 8.5—it hasn't changed!—but the average is now 18.1. In this case, you might not want to forecast the average and might revert back to a forecast of the median.

Generally speaking, the median is more robust to outliers than the average. In a supply chain environment, this is important because we can face many outliers due to demand peaks (marketing, promotions, spot deals) or encoding mistakes. We will discuss outliers further in Chapter 10.

Intermittent Demand

Is robustness to outliers always a good thing? No.

Unfortunately, as we will see, the median's robustness to outliers can result in a very annoying effect for items with intermittent demand.

Let's imagine that we sell a product to a single client. It is a highly profitable product and our unique client seems to make an order one week out of three, but without any recognizable pattern. The client always orders the product in batches of 100. We then have an average weekly demand of 33 pieces and a demand median of... 0.

We have to populate a weekly forecast for this product. Let's imagine we do a first forecast that aims for the average demand (33 pieces). Over the long-term, we will obtain a total squared error of 6 667 (RMSE of 81.6), and a total absolute error of 133.

Week	Demand	Forecast	Error	|Error|	Error^2
1	100	33	67	67	4 445
2	0	33	-33	33	1 111
3	0	33	-33	33	1 111
Total	100	100	0	133	6 667

Now, if we forecast the demand median (0), we obtain a total absolute error of 100 (MAE of 33) and a total squared error of 10.000 (RMSE of 58).

Week	Demand	Forecast	Error	|Error|	Error^2
1	100	0	-100	100	10 000
2	0	0	0	0	0
3	0	0	0	0	0
Total	100	0	-100	100	10 000

As we can see, MAE is a bad KPI to use for intermittent demand. As soon as you have more than half of the periods without demand, the optimal forecast is... 0!

Going Further

A trick to use against intermittent demand items is to aggregate the demand to a higher time horizon. For example, if the demand is intermittent at a weekly level, you could test a monthly forecast or even a quarterly forecast. You can always disaggregate the forecast back into the original time bucket by simply dividing it. This technique can allow you to use MAE as a KPI and smooth demand peaks at the same time.

Conclusion

MAE provides protection against outliers, whereas RMSE provides the assurance to get an unbiased forecast. Which indicator should you use? There is, unfortunately, no definitive answer. As a supply chain data scientist, you should experiment: if using MAE as a KPI results in a high bias, you might want to use RMSE. If the dataset contains many outliers, resulting in a skewed forecast, you might want to use MAE.

Pro-Tip – KPI and Reporting

Note as well that you can choose to report forecast accuracy to management using one or more KPIs (typically MAE and bias), but use another one (RMSE?) to optimize your models. We will further discuss how to manage the forecasting process in Part III.

Part II

Machine Learning

Chapter 12

Machine Learning

Tell us what the future holds, so we may know that you are gods.
Isaiah 41:23

What Is Machine Learning?

Until now, we have been using old-school statistics to predict demand. But with the recent rise of machine learning algorithms, we have new tools at our disposal that can easily achieve outstanding performance in terms of forecast accuracy for a typical supply chain demand dataset. As you will see in the following chapters, these models will be able to learn many relationships that are beyond the ability of traditional exponential smoothing models. For example, we will discuss how to add external information to our model in Chapters 20 and 22.

So far, we have created different algorithms that have used a predefined model to populate a forecast based on historical demand. The issue was that these models couldn't adapt to historical demand. If you use a double exponential smoothing model to predict a seasonal product, it will fail to interpret the seasonal patterns. On the other hand, if you use a triple exponential smoothing model on a non-seasonal demand, it might overfit the noise in the demand and interpret it as a seasonality.

Machine learning is different. Here, the algorithm (i.e., the machine) will learn relationships from a training dataset (i.e., our historical demand) and then apply these relationships on new data. Whereas a traditional statistical model will apply a predefined relationship (model) to forecast the demand, a machine learning algorithm will not assume *a priori* a particular relationship (like seasonality or a linear trend); it will **learn** these patterns directly from the historical demand.

For a machine learning algorithm to learn how to make predictions, we will have to feed it with both the inputs and the desired respective outputs. It will then automatically understand the relationships between these inputs and outputs.

Another important difference between using machine learning and exponential smoothing models to forecast our demand is that machine learning algorithms will **learn patterns from our entire dataset**. Exponential smoothing models will treat each item individually and independently from the others. Because it uses the entire dataset, a machine learning algorithm will apply what works best to each product. One could improve the accuracy of an exponential smoothing model by increasing the length of each time series (i.e., providing more historical periods for each product). Using machine learning, we will be able to increase our model's accuracy by providing more of the products' data to be ingested by the model.

Welcome to the world of machine learning.

12.1 Machine Learning for Demand Forecasting

To make a forecast, the question we will ask the machine learning algorithm is the following: *Based on the last* n *periods of demand, what will the demand be during the next period(s)?*

We will train the model by providing it with the data in a specific layout:

- *n* consecutive periods of demand as input.
- the demand of the (very) next period(s) as output.

Let's see an example (with a quarterly forecast to simplify the table):

Table 12.1: Historical demand formatting for machine learning.

	Inputs				**Output**
Product	**Year 1**				**Year 2**
	Q1	**Q2**	**Q3**	**Q4**	**Q1**
#1	5	15	10	7	6
#2	7	2	3	1	1
#3	18	25	32	47	56
#4	4	1	5	3	2

For our forecasting problem, we will basically show our machine learning algorithm different extracts of our historical demand dataset as inputs and, as a desired output, what the very next demand observation was. In our example in Table 12.1, the algorithm will learn the relationship between the last four quarters of demand, and the demand of the next quarter. The algorithm will *learn* that if we have 5, 15, 10, and 7 as the last four demand observations, the next demand observation will be 6, so that its prediction should be 6.

Next to the data and relationships from product #1, the algorithm will also learn from products #2, #3, and #4. In doing so, the idea is for the model to use *all* the data provided to give us better forecasts.

Most people will react to this idea with two very different thoughts. Either people will think that "it is simply impossible for a computer to look at the demand and make a prediction" or that "as of now, the humans have nothing left to do." Both are wrong.

As we will see later, machine learning can generate very accurate predictions. And as the human controlling the machine, we still have to ask ourselves many questions, such as:

- Which data should we feed to the algorithm for it to understand the proper relationships? We will discuss how to include other data features in Chapters 20, 22, and 23; and how to select the relevant ones in Chapters 18 and 24.
- Which machine learning algorithm should be used? There are many different ones: we will discuss new models in Chapters 13, 15, 17, 19, 21, and 25.
- Which parameters should be used in our model? As you will see, each machine learning algorithm has some parameters that we can tweak to improve its accuracy (see Chapter 14).

As always, there is no definitive, one-size-fits-all answer. Experimentation will help you find what is best for your dataset.

12.2 Data Preparation

The first step of any machine learning algorithm project is to properly clean and format the data. In our case, we need to format the historical demand dataset to obtain one similar to Table 12.1.

Naming Convention During our data cleaning process, we will use the standard data science notation and call the inputs X and the outputs Y. Specifically, the datasets `X_train` and `Y_train` will contain all the historical demand that we will use to **train** our algorithm (`X_train` being the inputs and `Y_train` the outputs). The datasets `X_test` and `Y_test` will be used to **test** our model.

You can see in Table 12.2 an example of a typical historical demand dataset you should have at the beginning of a forecasting project.

We now have to format this dataset to something similar to Table 12.1. Let's say for now that we want to predict the demand of a product during one quarter based on the demand observations of this product during the previous four quarters.[1] We will populate the datasets `X_train` and `Y_train` by going through the different products we have, and, each time, create a data sample with four consecutive quarters as `X_train` and the following

[1]We'll discuss variations of this in Chapters 18 and 24.

Table 12.2: Typical example of historical demand dataset.

Product	Year 1				Year 2				Year 3			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
#1	5	15	10	7	6	13	11	5	4	11	9	4
#2	7	2	3	1	1	0	0	1	3	2	4	5
#3	18	25	32	47	56	70	64	68	72	67	65	58
#4	4	1	5	3	2	5	3	1	4	3	2	5

quarter as `Y_train`. This way, the machine learning algorithm will learn the relationship(s) between one quarter of demand and the previous four.

You can see in Table 12.3 an illustration for the first iterations. Loop #1 uses Y1Q1 to Y1Q4 to predict Y2Q1, Loop #2 is shifted by one quarter: we use Y1Q2 to Y2Q1 to forecast Y2Q2, etc.

Table 12.3: Training and test sets creation.

Loop	Product	Year 1				Year 2				Year 3			
		Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
#1	#1	5	15	10	7	6							
#1	#2	7	2	3	1	1							
#1	#3	18	25	32	47	56							
#1	#4	4	1	5	3	2							
#2	#1		15	10	7	6	13						
#2	#2		2	3	1	1	0						
#2	#3		25	32	47	56	70						
#2	#4		1	5	3	2	5						
#3	#1			10	7	6	13	11					
...	...			...	...	...	...	...					
#8	#1								5	4	11	9	4
#8	#2								1	3	2	4	5
#8	#3								68	72	67	65	58
#8	#4								1	4	3	2	5

Our `X_train` and `Y_train` datasets will look like Table 12.4.

Remember that our algorithm will learn relationships in `X_train` to predict `Y_train`. So we could write that as `X_train` $\rightarrow$ `Y_train`.

In order to validate our model, we need to keep a test set aside from the training set. Remember, the data in the test set won't be shown to the model during its training phase. The test set will be kept aside and used after the training to evaluate the model accuracy against *unseen* data (as a final test).[2]

[2]See Chapters 4 and 8 for a discussion about training and test sets, as well as underfitting and overfitting, respectively.

Table 12.4: Training set.

Loop	Product	X_train				→	Y_train
#1	#1	5	15	10	7	→	6
#1	#2	7	2	3	1	→	1
#1	#3	18	25	32	47	→	56
#1	#4	4	1	5	3	→	2
#2	#1	15	10	7	6	→	13
#2	#2	2	3	1	1	→	0
#2	#3	25	32	47	56	→	70
...	...	...	...	...	...	→	...

In our example, if we keep the last loop as a test set (i.e., using demand from Y2Q4 to Y3Q3 to predict Y3Q4) we would have a test set as shown in Table 12.5.

Table 12.5: Test set.

X_test				Y_test
5	4	11	9	4
1	3	2	4	5
68	72	67	65	58
1	4	3	2	5

That means that our algorithm won't see these relationships during its training phase. It will be tested on the accuracy it achieved on these specific prediction exercises. We will measure its accuracy on this test set and assume its accuracy will be similar when predicting future demand.

Dataset Length

It is important for any machine learning exercise to pay attention to how much data is fed to the algorithm. The more, the better. On the other hand, the more periods we use to make a prediction (we will call this `x_len`), the fewer we will be able to loop through the dataset. Also, if we want to predict more periods at once (`y_len`), it will cost us a part of the dataset, as we need more data (`Y_train` is longer) to perform one loop in our dataset.

Typically, if we have a dataset with *n* periods, we will be able to make $1+\texttt{n}-\texttt{x\_len}-\texttt{y\_len}$ loops through it.

$$\texttt{loops} = 1 + \texttt{n} - \texttt{x\_len} - \texttt{y\_len}$$

Also keep in mind that you will have to keep some of those loops aside as a test set. Optimally, you should have enough loops of test set to cover a full season (so that you are sure that your algorithm captures all demand patterns properly).

$$\begin{aligned}\texttt{train\_loops} &= 1 + \texttt{n} - \texttt{x\_len} - \texttt{y\_len} - \texttt{s\_len}\\ \texttt{test\_loops} &= \texttt{s\_len}\end{aligned}$$

For the training set, a best practice is to keep—at the very least—enough runs to loop through two full years. Overall, for a monthly dataset, you should then have at least $35 + \texttt{x\_len} + \texttt{y\_len}$ periods (as shown below) in order for the algorithm to have two full seasonal cycles to learn any possible relationships and a full season to be tested on.

$$\begin{aligned}\texttt{train\_loops} &= 1 + \texttt{n} - \texttt{x\_len} - \texttt{y\_len} - \texttt{s\_len}\\ 24 &= 1 + \texttt{n} - \texttt{x\_len} - \texttt{y\_len} - 12\\ \texttt{n} &= 23 + \texttt{x\_len} + \texttt{y\_len} + 12\end{aligned}$$

12.3 Do It Yourself – Datasets Creation

Data Collection

The dataset creation and cleaning is a crucial part of any data science project. To illustrate all the models we will create in the next chapters, we will use the historical sales of cars in Norway from January 2007 to January 2017 as an example dataset. You can download this dataset on `supchains.com/download`.[3] You will get a `csv` file called `norway_new_car_sales_by_make.csv`. This dataset, graphed in Figure 12.1, contains the sales of 65 carmakers across 121 months. On average, a bit more than 140,000 new cars are sold in Norway per year. If we assume that the price of a new car is around $30,000 on average in Norway, the market can be roughly estimated to be worth $4B.

Attention Point

This dataset is modest in terms of size, allowing for fast training and manipulation. This is perfect for a learning exercise and trying out new models and ideas. But this dataset is not big enough to showcase all the power of advanced machine learning models—they should show better results on bigger datasets.

[3] The data is compiled by the Opplysningsrådet for Veitrafikken (OFV), a Norwegian organization in the automotive industry. It was initially retrieved by Dmytro Perepølkin and published on Kaggle.

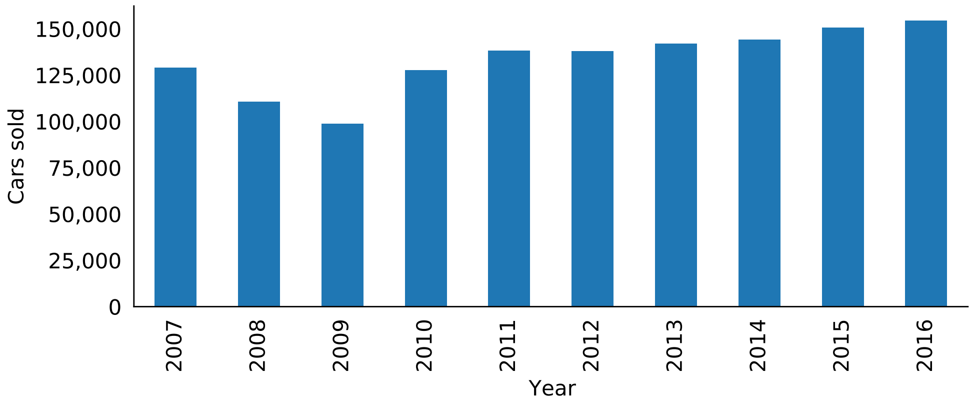


Figure 12.1: Cars sold per year in Norway.

Bring Your Own Dataset

In the next chapters, we will discuss various different models and apply them to this example dataset. But what we are actually interested in is **your own dataset**. Do not waste any time, and start gathering some historical demand data so that you can test the following models on your own dataset as we progress through the different topics. It is recommended that you start with a dataset of at least 5 years of data and more than a hundred different products. The bigger, the better.

We will discuss in the following chapters many models and ideas and apply them to this dataset. The thought processes we will discuss can be applied to any demand dataset; but the results obtained, as well as the parameters and models selected, won't be the same on another dataset.

We will make a function `import_data()` that will extract the data from this `csv` and format it with the dates as columns and the products (here car brands) as lines.

```python
def import_data():
    data = pd.read_csv('norway_new_car_sales_by_make.csv')
    data['Period'] = data['Year'].astype(str) + '-' + data['Month'].astype(str)
    ↪ .str.zfill(2)
    df = pd.pivot_table(data=data,values='Quantity',index='Make',columns='Period',
    ↪ aggfunc='sum',fill_value=0)
    return df
```

We used two special functions:

`.str.zfill(x)` adds leading zeros to a string until it reaches a length of x.[4]

`pd.pivot_table()` creates a pivot table from a DataFrame. The main inputs are:

- **`data`** the DataFrame used
- **`index`** which column(s) from the DataFrame should be used as the rows of the pivot table
- **`columns`** which column(s) from the DataFrame should be used as the columns of the pivot table
- **`values`** what value to show in the pivot
- **`aggfunc`** how to aggregate the values
- **`fill_value`** fill-in value in case of missing data

You can print the results in an Excel file for later reference by using `df.to_excel('demand.xlsx',index=False)`. It is always good practice to visually check what the dataset looks like to be sure that the code worked as intended.

This is what you should obtain if you print `df`.

```
print(df)
>> Period        2007-01  2007-02  2007-03  ...  2016-11  2016-12  2017-01
>> Make                                     ...
>> Alfa Romeo         16        9       21  ...        4        3        6
>> Aston Martin        0        0        1  ...        0        0        0
>> Audi              599      498      682  ...      645      827      565
>> BMW               352      335      365  ...     1663      866     1540
>> Bentley             0        0        0  ...        0        0        0
>> ...               ...      ...      ...  ...      ...      ...      ...
>> Think               2        0        0  ...        0        0        0
>> Toyota           2884     1885     1833  ...     1375     1238     1526
>> Volkswagen       2521     1517     1428  ...     2106     2239     1688
>> Volvo             693      570      656  ...      754     1235     1158
>> Westfield           0        0        0  ...        0        0        0
>>
>> [65 rows x 121 columns]
```

Training and Test Sets Creation

Now that we have our dataset with the proper formatting, we can create our training and test sets. For this purpose, we will create a function `datasets` that takes as inputs:

`df` our historical demand dataset

[4]See `www.w3schools.com/python/ref_string_zfill.asp` for more information and examples.

`x_len` the number of historical periods we will use to make a prediction

`y_len` the number of future periods we want to predict

`test_loops` the number of loops (iterations) we want to leave out as a final test. If you want to populate a future forecast (i.e., you are not running an optimization exercise), you have to set `test_loops` to 0.

The function will return `X_train`, `Y_train`, `X_test`, and `Y_test`.

```
1  def datasets(df, x_len=12, y_len=1, test_loops=12):
2      D = df.values
3      rows, periods = D.shape
4  
5      # Training set creation
6      loops = periods + 1 - x_len - y_len
7      train = []
8      for col in range(loops):
9          train.append(D[:,col:col+x_len+y_len])
10     train = np.vstack(train)
11     X_train, Y_train = np.split(train,[-y_len],axis=1)
12 
13     # Test set creation
14     if test_loops > 0:
15         X_train, X_test = np.split(X_train,[-rows*test_loops],axis=0)
16         Y_train, Y_test = np.split(Y_train,[-rows*test_loops],axis=0)
17     else: # No test set: X_test is used to generate the future forecast
18         X_test = D[:,-x_len:]
19         Y_test = np.full((X_test.shape[0],y_len),np.nan) #Dummy value
20 
21     # Formatting required for scikit-learn
22     if y_len == 1:
23         Y_train = Y_train.ravel()
24         Y_test = Y_test.ravel()
25 
26     return X_train, Y_train, X_test, Y_test
```

Let's discuss a few of the artifices we had to use in our function.

np.split(array, indices, axis) cuts an array at specific indices along an axis. In our function, we use it to cut `train` in two pieces at the column `x_len` in order to create `X_train` and `Y_train` (see row 11). We also use it (along the row axis this time) to cut the test set out of the training set (see rows 14 and 15).

array.ravel() reduces the dimension of a NumPy array to 1D (see rows 23 and 24). `Y_train` and `Y_test` are always created by our function as `2D arrays` (i.e., arrays with rows and columns). If we only want to predict one period at a time, these arrays will then only have one column (and multiple rows). Unfortunately, the functions we will use later (from the scikit-learn library) will need `1D arrays` if we forecast only one period.

We can now easily call our new function `datasets(df)` as well as `import_data()`.

```
df = import_data()
X_train, Y_train, X_test, Y_test = datasets(df, x_len=12, y_len=1, test_loops=12)
```

We obtain the datasets we need to feed our machine learning algorithm (`X_train` and `Y_train`) and the datasets we need to test it (`X_test` and `Y_test`). Note that we set `test_loops` as 12 periods. That means that we will test our algorithm over 12 different loops (i.e., we will predict 12 times the following period `[y_len=1]` based on the last 12 periods `[x_len=12]`).

Forecasting Multiple Periods at Once You can change `y_len` if you want to forecast multiple periods at once. In the following chapters, we will keep `y_len = 1` for the sake of simplicity.

Future Forecast If `test_loops==0`, the function will return the latest demand observations in `X_test` (and `Y_test` set as a dummy value), which will allow you to populate the future forecast, as we will see in Section 12.5.

What About Excel? So far, Excel could provide us an easy way to see the data and our statistical model relationships. But it won't get us any further. Unfortunately, Excel does not provide the power to *easily* format such datasets into the different parts we need (`X_train`, `X_test`, `Y_train`, and `Y_test`). Moreover, with most of our machine learning models, the dataset size will become too large for Excel to handle correctly. Actually, another major blocking point is that Excel does not provide any machine learning algorithm.

12.4 Do It Yourself – Linear Regression

Now that we have created our training and test sets, let's create a forecast benchmark. We want to have an indication of what a simple model could do in order to compare its accuracy against our more complex models. Basically, if a model got an error of *x* percent, we will compare this against the error of our benchmark in order to know if *x* is a good result or not.

We will use a simple linear regression as a benchmark. If you are not familiar with the concept of linear regression, you can just picture it as a straight line that will match historical demand and project it into the future. Many Python libraries propose models to compute linear regressions, but we will choose scikit-learn, as we will later use this library for all our machine learning models. Let's stay consistent from one model to another.

```python
from sklearn.linear_model import LinearRegression
reg = LinearRegression() # Create a linear regression object
reg = reg.fit(X_train,Y_train) # Fit it to the training data
# Create two predictions for the training and test sets
Y_train_pred = reg.predict(X_train)
Y_test_pred = reg.predict(X_test)
```

As you can see, we created a model object called `reg` that we fit to our training data (`X_train`, `Y_train`), thanks to the method `.fit()` (remember, we should fit our model to the training set and not to the test set). We then populated a prediction based on `X_train` and `X_test` via the method `.predict()`.

KPI Function

We can now create a KPI function `kpi_ML()` that will display the accuracy of our model. This function will be similar to the one defined in Chapter 2. Here, we use a DataFrame in order to print the various KPI in a structured way.

```python
def kpi_ML(Y_train, Y_train_pred, Y_test, Y_test_pred, name=''):
    df = pd.DataFrame(columns = ['MAE','RMSE','Bias'],index=['Train','Test'])
    df.index.name = name
   df.loc['Train','MAE'] = 100*np.mean(abs(Y_train - Y_train_pred))/np.mean(Y_train)
    df.loc['Train','RMSE'] = 100*np.sqrt(np.mean((Y_train - Y_train_pred)**2))/np
    ↪   .mean(Y_train)
    df.loc['Train','Bias'] = 100*np.mean((Y_train - Y_train_pred))/np.mean(Y_train)
    df.loc['Test','MAE'] = 100*np.mean(abs(Y_test - Y_test_pred))/np.mean(Y_test)
    df.loc['Test','RMSE'] = 100*np.sqrt(np.mean((Y_test - Y_test_pred)**2))/np
    ↪   .mean(Y_test)
    df.loc['Test','Bias'] = 100*np.mean((Y_test - Y_test_pred))/np.mean(Y_test)
    df = df.astype(float).round(1) #Round number for display
    print(df)
```

This is what we obtain:

```
kpi_ML(Y_train, Y_train_pred, Y_test, Y_test_pred, name='Regression')
>>               MAE  RMSE Bias
>> Regression
>> Train       17.8% 43.9% 0.0%
>> Test        17.8% 43.7% 1.6%
```

You can see that the RMSE is much worse than the MAE. This is usually the case, as demand datasets contain a few exceptional values that drive the RMSE up.[5]

Car sales per month, per brand, and at national level are actually easy to forecast, as those demand time series are stable without much seasonality. This is why the linear benchmark provides such good results here. On top of that, we only predict one month at a time and linear approximation works well in the short term. On a different dataset, with a longer forecast horizon (and more seasonality), linear regressions might not be up to the challenge. Therefore, don't be surprised to face much worse results on your own datasets. We will discuss in the following chapters much more advanced models that will be able to keep up with longer-term forecasting horizons, seasonality, and various other external drivers.

Attention Point

Do not worry if the MAE and RMSE of your dataset is much above the example benchmark presented here. In some projects, I have seen MAE as high as 80 to 100%, and RMSE well above 500%. Again, we use a linear regression benchmark precisely to get an order of magnitude of the complexity of a dataset.

12.5 Do It Yourself – Future Forecast

In Section 12.3, we created a function `datasets()` to populate a training and a test set. In Section 12.4, we used our training and test sets to evaluate a (simple) model. We can now change our hat from data scientist—working with training and test sets to evaluate models—to demand planner—using a model to populate a baseline forecast.

We will create a future forecast—the forecast to be used by **your** supply chain—by using our `datasets()` function and set `test_loops` to `0`. The function will then return `X_test` filled-in with the latest demand observations—thus we will be able to use it to predict *future* demand. Moreover, as we do not keep data aside for the test set, the training dataset (`X_train` and `Y_train`) will include the whole historical demand. This will be helpful, as we will use as much training data as possible to feed the model.

```python
X_train, Y_train, X_test, Y_test = datasets(df, x_len=12, y_len=1, test_loops=0)
reg = LinearRegression()
reg = reg.fit(X_train,Y_train)
forecast = pd.DataFrame(data=reg.predict(X_test), index=df.index)
```

The DataFrame `forecast` now contains the forecast for the future periods.

[5]See Chapter 2 for a thorough discussion about forecast KPI and the impact of outliers and extreme values.

```
print(forecast.head())
>>                          0
>> Make
>> Alfa Romeo       6.187217
>> Aston Martin     1.032483
>> Audi           646.568622
>> BMW           1265.032834
>> Bentley          1.218092
```

Now that we have a proper dataset, a benchmark to beat, and we know how to generate a future forecast, let’s see how far machine learning can get us.

Bibliography

Abu-Rmileh, A. (2019). The multiple faces of 'feature importance' in XGBoost. https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7. Online; accessed 06 July 2020.

Baraniuk, C. (2015). The cyborg chess players that can't be beaten. *BBC*. https://www.bbc.com/future/article/20151201-the-cyborg-chess-players-that-cant-be-beaten. Online; accessed 21 July 2020.

Bourret Sicotte, X. (2018). AdaBoost: Implementation and intuition. https://xavierbourretsicotte.github.io/AdaBoost.html. Online; accessed 26 May 2020.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L. et al. (1998). Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849. Online; accessed 22 May 2020.

Brown, R. (1956). Exponential smoothing for predicting demand. *Management Science*.

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Cunff, A.-L. L. (2019). Confirmation bias: believing what you see, seeing what you believe. https://nesslabs.com/confirmation-bias. Online; accessed 23 July 2020.

Drucker, H. (1997). Improving regressors using boosting techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, page 107–115, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Fildes, R. and Goodwin, P. (2007). Good and bad judgement in forecasting: Lessons from four companies. *Foresight*, 8:5–10.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Gardner, E. S. and Mckenzie, E. (1985). Forecasting trends in time series. *Management Science*, 31(10):1237–1246.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Gilliland, M. (2002). Is forecasting a waste of time? *Supply Chain Management Review*, 6(4):16–23.

Gilliland, M. (2010). *The Business Forecasting Deal: Exposing Myths, Eliminating Bad Practices, Providing Practical Solutions*. John Wiley & Sons, Hoboken, N.J.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. J. Wiley; Chapman & Hall.

Hewamalage, H., Bergmeir, C., and Bandara, K. (2020). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Holt, C. C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Hyndman, R. J. and Athanasopoulos, G. (2018). Forecasting: principles and practice. `https://otexts.com/fpp2/`. Online; accessed 22 May 2020.

Insights, M. T. R. (2019). The AI effect: How artificial intelligence is making health care more human. *MIT Technology Review Insights*. `https://www.technologyreview.com/hub/ai-effect/`. Online; accessed 21 July 2020.

Kashnitsky, Y. (2020). mlcourse.ai – open machine learning course. `https://mlcourse.ai/articles/topic10-boosting/`. Online; accessed 13 August 2020.

Kearns, M. (1988). Thoughts on hypothesis boosting. `https://www.cis.upenn.edu/~mkearns/papers/boostnote.pdf`. Online; accessed 26 May 2020.

Kearns, M. and Valiant, L. G. (1989). Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 433–444, New York, NY, USA. Association for Computing Machinery.

Kingma, D. P. and Ba, J. (2015). Adam: a method for stochastic optimization. *International Conference on Learning Representations*, pages 1–13.

Kissinger, H. A., Schmidt, E., and Huttenlocher, D. (2019). The Metamorphosis. *The Atlantic*. `https://www.theatlantic.com/magazine/archive/2019/08/henry-kissinger-the-metamorphosis-ai/592771/`. Online; accessed 28 May 2020.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

Kurenkov, A. (2015). A 'brief' history of neural nets and deep learning. `http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/`. Online; accessed 16 August 2020.

LeCun, Y. (2019). *Quand la machine apprend: la révolution des neurones artificiels et de l'apprentissage profond*. Odile Jacob, Paris.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

Levenson, R. M., Krupinski, E. A., Navarro, V. M., and Wasserman, E. A. (2015). Pigeons (columba livia) as trainable observers of pathology and radiology breast cancer images. *PLOS ONE*, 10(11):1–21.

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56–61.

Minsky, M. and Papert, S. (1969). Perceptrons: An introduction to computational geometry. *Cambridge tiass., HIT*.

MITx (2019). Introduction to computer science and programming using python.

Morgan, J. N. and Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58(302):415–434.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press. `http://neuralnetworksanddeeplearning.com/chap5.html`. Online; accessed 16 August 2020.

Oliphant, T. E. (2006). *A Guide to NumPy*, volume 1. Trelgol Publishing USA.

Oliva, R. and Watson, N. (2009). Managing functional biases in organizational forecasts: A case study of consensus forecasting in supply chain planning. *Production and Operations Management*, 18(2):138–151.

Oskolkov, N. (2019). How to cluster in high dimensions. `https://towardsdatascience.com/how-to-cluster-in-high-dimensions-4ef693bacc6`. Online; accessed 10 September 2020.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Sanderson, G. (2017). Neural networks. `https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi`. Online; accessed 16 August 2020.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Sculley, D. (2010). Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, pages 1177–1178.

Shead, S. (2019). Alphabet's deepmind losses soared to $570 million in 2018. *Forbes*. `https://www.forbes.com/sites/samshead/2019/08/07/deepmind-losses-soared-to-570-million-in-2018/#4358b1633504`. Online; accessed 18 August 2020.

Silver, N. (2015). *The Signal and the Noise: Why So Many Predictions Fail—but Some Don't*. Penguin Books.

Stetka, B. (2015). Using pigeons to diagnose cancer. *Scientific American*. `https://www.scientificamerican.com/article/using-pigeons-to-diagnose-cancer/`. Online; accessed 26 May 2020.

Tetlock, P. E. and Gardner, D. (2016). *Superforecasting: The Art and Science of Prediction*. Broadway Books.

Times, T. N. Y. (1958). New navy device learns by doing. *The New York Times*. `https://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html`. Online; accessed 16 August 2020.

Tversky, A. and Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, 185(4157):1124–1131.

Vandeput, N. (2020). *Inventory Optimization: Models and Simulation*. De Gruyter.

VanderPlas, J. (2016). In-depth: Kernel density estimation. `https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html`. Online; accessed 09 June 2020.

Vincent, J. (2019). Microsoft invests $1 billion in OpenAI to pursue holy grail of artificial intelligence. *The Verge*. `https://www.theverge.com/2019/7/22/20703578/microsoft-openai-investment-partnership-1-billion-azure-artificial-general-intelligence-agi`. Online; accessed 18 August 2020.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Wallis, K. F. (2014). Revisiting francis galton's forecasting competition. *Statistical Science*, pages 420–424.

Wason, P. C. (1960). On the failure to eliminate hypotheses in a conceptual task. *Quarterly Journal of Experimental Psychology*, 12(3):129–140.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD dissertation, Harvard University*.

Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management Science*, 6(3):324–342.

Glossary

accuracy The accuracy of your forecast measures how much spread you had between your forecasts and the actual values. The accuracy of a forecast gives an idea of the magnitude of the errors but not their overall direction. *See page* 11

alpha A smoothing factor applied to the demand level in the various exponential smoothing models. In theory: $0 < \alpha \leq 1$; in practice: $0 < \alpha \leq 0.6$. *See page* 30

array A data structure defined in NumPy. It is a list or a matrix of numeric values. *See page* 288

bagging Bagging (short word for *Bootstrap Aggregation*) is a method for aggregating multiple sub-models into an *ensemble* model by averaging their predictions with equal weighting. *See page* 153

beta A smoothing factor applied to the trend in the various exponential smoothing models. In theory: $0 < \beta \leq 1$; in practice: $0 < \beta \leq 0.6$. *See page* 45

bias The bias represents the overall direction of the historical average error. It measures if your forecasts were on average too high (i.e., you *overshot* the demand) or too low (i.e., you *undershot* the demand). *See page* 12

Boolean A Boolean is a value that is either `True` or `False`: 1 or 0. *See page* 222

boosting Boosting is a class of *ensemble* algorithms in which models are added *sequentially*, so that later models in the sequence will correct the predictions made by earlier models in the sequence. *See page* 184

bullwhip effect The bullwhip effect is observed in supply chains when small variations in the downstream demand result in massive fluctuations in the upstream supply chain. *See page* 50

classification Classification problems require you to classify data samples in different categories. *See page* 133

data leakage In the case of forecast models, a data leakage describes a situation where a model is given pieces of information about future demand. *See page* 32

DataFrame A DataFrame is a table of data as defined by the pandas library. It is similar to a table in Excel or an SQL database. *See page* 290

demand observation This is the demand for a product during one period. For example, a demand observation could be the demand for a product in January last year. *See page* 4

ensemble An ensemble model is a (meta-)model constituted of many sub-models. *See page* 152

epoch One epoch consists, for the neural network learning algorithm, to run through all the training samples. The number of epochs is the number of times the learning algorithm will run through the entire training dataset. *See page* 262

Euclidean distance The Euclidean distance between two points is the length of a straight line between these two points. *See page* 231

evaluation set An evaluation set is a set of data that is left aside from the training set to be used as a monitoring dataset during the training. A validation set or a holdout set can be used as an evaluation set. *See page* 210

feature A feature is a type of information that a model has at its disposal to make a prediction. *See page* 134

gamma A smoothing factor applied to the seasonality (either additive or multiplicative) in the triple exponential smoothing models. In theory: $0 < \gamma \leq 1$; in practice: $0.05 < \gamma \leq 0.3$. *See page* 75

holdout set Subset of the training set that is kept aside during the training to validate a model against unseen data. The holdout set is made of the last periods of the training set to replicate a test set. *See page* 178

inertia In a K-means model, the inertia is the sum of the distances between each data sample and its associated cluster center. *See page* 231

instance An (object) instance is a technical term for an occurrence of a class. You can see a class as a blueprint and an instance of a specific realization of this blueprint. The class (blueprint) will define what each instance will look like (which variables will constitute it) and what it can do (what methods or functions it will be able to perform). *See page* 137

level The level is the average value around which the demand varies over time. *See page* 29

Mean Absolute Error $MAE = \frac{1}{n}\sum |e_t|$ *See page* 17

Mean Absolute Percentage Error $MAPE = \frac{1}{n}\sum \frac{|e_t|}{d_t}$ *See page* 15

Mean Square Error $MSE = \frac{1}{n}\sum e_t^2$ *See page* 19

naïve forecast The simplest forecast model: it always predicts the last available observation. *See page* 4

noise In statistics, the noise is an unexplained variation in the data. It is often due to the randomness of the different processes at hand. *See page* 5

NumPy One of the most famous Python libraries. It is focused on numeric computation. The basic data structure in NumPy is an array. *See page* 288

pandas Pandas is a Python library specializing in data formatting and manipulation. It allows the use of DataFrames to store data in tables. *See page* 290

phi A damping factor applied to the trend in the exponential smoothing models. This reduces the trend after each period. In theory: $0 < \phi \leq 1$; in practice: $0.7 \leq \phi \leq 1$. *See page* 65

regression Regression problems require an estimate of a numerical output based on various inputs. *See page* 133

Root Mean Square Error $RMSE = \sqrt{\frac{1}{n}\sum e_t^2}$ *See page* 19

S&OP The sales and operations planning (S&OP) process focuses on aligning mid- and long-term demand and supply. *See page* xviii

SKU A stock keeping unit refers to a specific material kept in a specific location. Two different pieces of the same SKU are indistinguishable. *See page* 271

supervised learning A supervised machine learning model is a model that is fed with both inputs and desired outputs. It is up to the algorithm to understand the relationship(s) between these inputs and outputs. *See page* 229

test set A test set is a dataset kept aside to test our model against unseen data after its fitting (training). *See page* 41

training set A training set is a dataset used to fit (train) our model. *See page* 41

trend The trend is the average variation of the time series level between two consecutive periods. *See page* 45

unsupervised learning An unsupervised machine learning model is a model that is only fed with inputs and no specific desired outputs. It is up to the machine learning algorithm to order (categorize) the different data observations. You can simply see it as asking your algorithm to give a label to each data observation. *See page* 229

validation dataset A validation set is a random subset of the training set that is kept aside during the training to validate a model against unseen data. *See page* 143

weak model A weak model is a model that is slightly more accurate than luck. *See page* 184

Index