Nicolas Vandeput

Data Science for Supply Chain Forecasting

Nicolas Vandeput

Data Science for Supply Chain Forecasting

2nd edition

DE GRUYTER

ISBN 978-3-11-067110-0
e-ISBN (PDF) 978-3-11-067112-4
e-ISBN (EPUB) 978-3-11-067120-9

Library of Congress Control Number: 2020951955

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the Internet at http://dnb.dnb.de.

Cover image: Raphael - Stitched together from vatican.va, Public Domain,
https://commons.wikimedia.org/w/index.php?curid=4406048
Typesetting: VTeX UAB, Lithuania
Printing and binding: CPI books GmbH, Leck

www.degruyter.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 in reviewing 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 abcsupplychain.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 thank 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

https://doi.org/10.1515/9783110671124-201

previous book) for his review of multiple chapters. And 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 specializing 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.

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

https://doi.org/10.1515/9783110671124-202

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

https://doi.org/10.1515/9783110671124-203

Contents

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:

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 changed 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

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

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.

https://doi.org/10.1515/9783110671124-204

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 inflow 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 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 advise 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 in understanding 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 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.

---

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.

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

Part I: **Statistical Forecasting**

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.

https://doi.org/10.1515/9783110671124-001

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.

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).

$$e_t = f_t - d_t$$

https://doi.org/10.1515/9783110671124-002

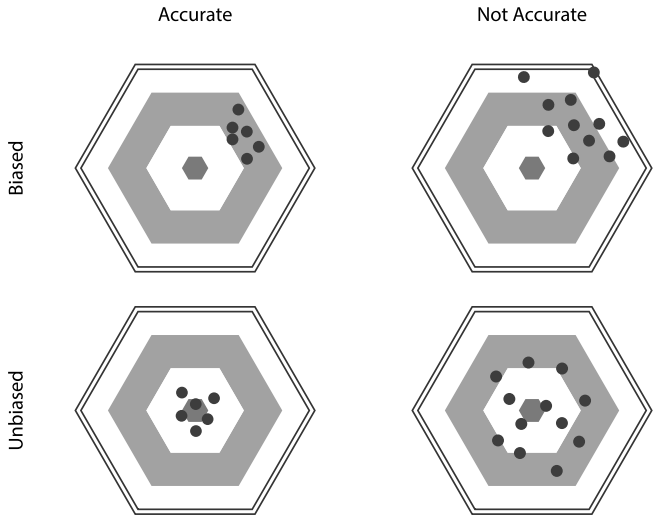


Figure 2.1: Accuracy and bias.

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).

3 Exponential Smoothing

3.1 The Idea Behind Exponential Smoothing

Simple exponential smoothing is one of the simplest ways to forecast a time series; we will use it in later chapters as a building block in many more powerful models. Just as for a moving average, the basic idea of this model is to assume that the future will be more or less the same as the (recent) past. The only pattern that this model will be able to learn from demand history is its **level**.

Level

The level is the average value around which the demand varies over time. As you can observe in Figure 3.1, the level is a smoothed version of the demand.

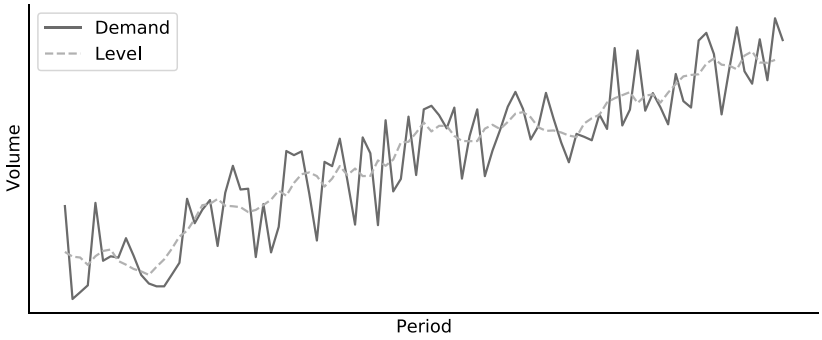


Figure 3.1: Demand level estimation.

The exponential smoothing model will then forecast the future demand as its last estimation of the level. It is important to understand that there is no definitive mathematical definition of the level—instead it is up to our model to **estimate** it.

The simple exponential smoothing model will have some advantages compared to a naïve[1] or a moving average model (see Chapter 1):

- The weight that is put on each observation decreases **exponentially**[2] over time. In other words, in order to determine the forecast, the historical, most recent period has the highest importance; then each subsequent (older) period has less and less importance. This is often better than moving average models, where the same importance (weight) is given to a handful of historical periods.
- Outliers and noise have less impact than with a naïve forecast.

1 Remember, a naïve forecast simply projects the latest available observation in the future.

2 We'll discuss this in more detail in the paragraph "Why Is It Called *Exponential* Smoothing?"

https://doi.org/10.1515/9783110671124-003

3.2 Model

The underlying idea of any exponential smoothing model is that, at each period, the model will **learn** a bit from the **most recent demand observation** and **remember** a bit of **the last forecast** it did. The smoothing parameter (or learning rate) **alpha** (α) will determine how much importance is given to the most recent demand observation (see Figure 3.2). Let's represent this mathematically:

$$f_t = \alpha d_{t-1} + (1 - \alpha) f_{t-1}$$
$$0 < \alpha \leq 1$$

The magic about this formula is that the last forecast made by the model was already including a part of the previous demand observation and a part of the previous forecast. **This means that the previous forecast includes everything the model learned so far based on demand history**.

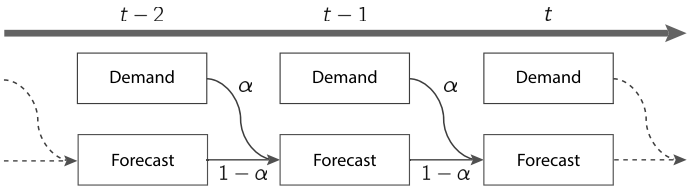


Figure 3.2: Simple exponential smoothing algorithm.

What Is the Intuition Behind This Formula?

$\boldsymbol{\alpha}$ is a ratio (or a percentage) of how much importance the model will allocate to the most recent observation compared to the importance of demand history.

$\boldsymbol{\alpha d_{t-1}}$ represents the learning rate times the previous demand observation. You could say that the model attaches certain importance (alpha) to the last demand occurrence.

$\boldsymbol{(1 - \alpha) f_{t-1}}$ represents how much the model remembers from its previous forecast. Note that this is where the recursive magic happens, as f_{t-1} was itself defined as partially d_{t-2} and f_{t-2}.

There is an important trade-off to be made here between *learning* and *remembering*, between being reactive and being stable. If alpha is high, the model will allocate more importance to the most recent demand observation (i. e., the model will learn fast), and it will be reactive to a change in the demand level. But it will also be sensitive to outliers and noise. On the other hand, if alpha is low, the model won't rapidly notice a change in level, but will also not overreact to noise and outliers.

4 Underfitting

All models are wrong, but some are helpful.

George Box

A model aims to describe reality. As reality can be rather complex, a model will be built on some assumptions (i. e., simplifications), as summarized by statistician George Box. Unfortunately, due to these assumptions or some limitations, some forecast models will not be able to predict or properly explain the reality they are built upon.

We say that a model is **underfitted** if it does not explain reality accurately enough.

To analyze our model's abilities, we will divide our dataset (i. e., the historical demand in the case of a forecast) into two different parts: the training set and the test set.

Training Set

The training set is used to train (*fit*) our model (i. e., optimize its parameters).

Test Set

The test set is the dataset that will assess the accuracy of our model against **unseen** data. This dataset is kept aside from the model during its training phase, so that the model is not aware of this data and can thus be tested against unseen—and right away available—data.

Typically, in the case of statistical forecast models, we use historical demand as the training set to optimize the different parameters (α for the simple exponential smoothing model). To test our forecast, we can keep the latest periods out of the training set to see how our models behave during these periods.

We need to be very careful with our test set. We can never use it to optimize our model. Keep in mind that this dataset is here to show us how our model would perform against new data. If we optimize our model on the test set, we will never know what accuracy we can expect against new demand.

One could say that an underfitted model lacks a good understanding of the training dataset. As the model does not perform properly on the training dataset, it will not perform well on the test set either. In the case of demand forecasting, a model that does not achieve good accuracy on historical demand will not perform properly on future demand either.

We will now look into two possible causes of underfitting and how to solve them.

https://doi.org/10.1515/9783110671124-004

5 Double Exponential Smoothing

When the facts change, I change my mind. What do you do, sir?

John Maynard Keynes

5.1 The Idea Behind Double Exponential Smoothing

We saw with the simple exponential smoothing algorithm how we could create a simple forecast, assuming that the future of the demand would be similar to its (recent) past. A major issue of this simple smoothing is that it can only see a level, and is unable to identify and project a **trend**.

Trend

We define the trend as the average variation of the time series level between two consecutive periods. Remember that the level is the average value around which the demand varies over time.

If you assume that your time series follows a trend, you will most likely not know its magnitude in advance—especially as this magnitude could vary over time. This is fine, because we will create a model that will learn *by itself* the trend over time. Just as for the level, this new model will estimate the trend based on an exponential weight **beta** (β), giving more (or less) importance to the most recent observations.

5.2 Double Exponential Smoothing Model

Remember that for the simple exponential smoothing model, we updated the forecast at each period partially based on

1. the **most recent observation** of the demand.
2. the **previous estimation** of the demand (that is the previous forecast).

Similarly, the general idea behind exponential smoothing models is that each demand component (currently, the level and the trend, later the seasonality, as well) will be updated after each period based on two same pieces of information: the last observation and the previous estimation of this component. Let's apply this principle to estimate the demand level (a_t and trend b_t).

Level Estimation

Let's see how the model will estimate the level:

$$a_t = \alpha d_t + (1 - \alpha)(a_{t-1} + b_{t-1})$$

https://doi.org/10.1515/9783110671124-005

6 Model Optimization

It is difficult to make predictions, especially about the future.

Author unknown

Now that we have seen a couple of forecast models, we can discuss parameter optimization. Let's recap the models we have seen so far and their different parameters,
Moving Average n (Chapter 1)
Simple Exponential Smoothing alpha (Chapter 3)
Double Exponential Smoothing alpha, beta (Chapter 5)

As we have seen in the double exponential smoothing case, a wrong parameter optimization will lead to catastrophic results. To optimize our models, we could manually search for the best parameter values. But remember, that would be against our supply chain data science best practices: we need to automate our experiments in order to scale them. Thanks to our favorite tools—Excel and Python—we will be able to automatically look for the best parameter values. The idea is to set an objective (either RMSE or MAE),[1] automatically run through different parameter values, and then select the one that gave the best results.

6.1 Excel

To optimize the parameters in Excel, we will use Excel Solver. If you have never used Excel Solver before, do not worry. It is rather easy.

Solver Activation

The first step is to activate Solver in Excel. If you have a Windows machine with Excel 2010 or a more recent version, you can activate it via the following steps,

1. Open Excel, go to `File/Options/Add-ins`.
2. Click on the `Manage` drop-down menu, select `Excel Add-ins` and click on the `Go...` button just to the right of it.
3. On the add-ins box, click on the `Solver Add-in` check box and then click the `OK` button to confirm your choice.
4. Let's now confirm that Solver is activated. In the Excel ribbon, go to the `Data` tab, there on the sub-menu `Analyze` (normally on the far right), you should see the `Solver` button.

1 We discussed their strengths and limitations in Chapter 2.

https://doi.org/10.1515/9783110671124-006

7 Double Smoothing with Damped Trend

7.1 The Idea Behind Double Smoothing with Damped Trend

One of the limitations of the double smoothing model is the fact that the trend is assumed to go on forever. In 1985, Gardner and McKenzie proposed in their paper "Forecasting Trends in Time Series" to add a new layer of intelligence to the double exponential model: a **damping factor**, **phi** (ϕ), that will exponentially reduce the trend over time. One could say that this new model **forgets** the trend over time.[1] Or that the model remembers only a fraction (ϕ) of the previous estimated trend.

Practically, the trend (b) will be reduced by a factor ϕ in each period. In theory, ϕ will be between 0 and 1—so that it can be seen as a % (like α and β). Nevertheless, in practice, it is often between 0.7 and 1. At the edge cases if $\phi = 0$, we are back to a simple exponential smoothing forecast; and if $\phi = 1$, the damping is removed and we deal with a double smoothing model.

7.2 Model

We will go back to the double exponential smoothing model and multiply all b_{t-1} occurrences by ϕ. Remember that $\phi \leq 1$, so that the damped trend is a muted version of the double smoothing model. We then have:

$$a_t = \alpha d_t + (1 - \alpha)\,(a_{t-1} + \boldsymbol{\phi} b_{t-1})$$
$$b_t = \beta(a_t - a_{t-1}) + (1 - \beta)\,\boldsymbol{\phi} b_{t-1}$$

The forecast for the next period would then be:

$$f_{t+1} = a_t + b_t \phi$$

Or, to be more general, for a forecast made on period t for period $t + \lambda$:

$$f_{t+\lambda} = a_t + b_t \sum_{i=1}^{\lambda} \phi^i$$

Unfortunately, this generalization is not straightforward to implement in Excel. For example, let's imagine you want to forecast period $t + 3$. You would have:

$$f_{t+3} = a_t + b_t \phi + b_t \phi^2 + b_t \phi^3$$

Note that if $\phi = 1$, then we are back to a normal double smoothing model. Setting $\phi = 1$ basically means that the model won't *forget* the trend over time.

1 See Gardner and Mckenzie (1985).

https://doi.org/10.1515/9783110671124-007

8 Overfitting

With four parameters, I can fit an elephant, and with five, I can make him wiggle his trunk.
John Von Neumann

In Chapter 4, we saw the issue of underfitting a dataset. This happens when a model is not able to learn the patterns present in the training dataset. As we saw, underfitting is most likely due to the model not being smart enough to understand the patterns in the training dataset. This could be solved by using a more complex model.

On the other end of the spectrum, we have the risk for a model to **overfit** a dataset. If a model overfits the data, it means that it has recognized (or learned) patterns from the noise (i. e., randomness) of the training set. As it has learned patterns from the noise, it will reapply these patterns in the future on new data. **This will create an issue as the model will show (very) good results on the training dataset but will fail to make good predictions on the test set**. In a forecasting model, that means that your model will show good accuracy on historical demand but will fail to deliver as good results on future demand.

In other words, overfitting means that you learned patterns that worked **by chance** on the training set. And as these patterns most likely won't occur again on future data, you will make wrong predictions.

Overfitting is the number one enemy of many data scientists for another reason. Data scientists are always looking to make the best models with the highest accuracy. When a model achieves (very) good accuracy, it is always tempting to think that it is simply excellent and call it a day. But a careful analysis will reveal that the model is just overfitting the data. Overfitting can be seen as a mirage: you are tempted to think that there is an oasis in the middle of the desert, but actually, it is just sand reflecting the sky. As we learn more complex models, underfitting will become less of an issue, and overfitting will become the biggest risk. This risk will be especially present with machine learning, as we will see in Part II. Therefore, we will have to use more complex techniques to prevent our models from overfitting our datasets. Our battle against overfitting will reach a peak in Chapters 18 and 24, when we discuss feature optimization.

Let's go over some examples of overfitting and the tools to avoid it.

8.1 Examples

Supply Chain Forecasting

Let's imagine that we have observed the demand of a new product over 20 periods (as shown in Figure 8.1), and now we want to use our optimization algorithm from Chapter 6 to make a forecast.

https://doi.org/10.1515/9783110671124-008

9 Triple Exponential Smoothing

9.1 The Idea Behind Triple Exponential Smoothing

With the first two exponential smoothing models we saw, we learned how to identify the level and the trend of a time series and used these pieces of information to populate our forecast. After that, we added an extra layer of intelligence to the trend by allowing the model to partially forget it over time.

Unfortunately, the simple and double exponential smoothing models do not recognize seasonal patterns and therefore cannot extrapolate any seasonal behavior in the future. Seasonal products—with high and low seasons—are common for many supply chains across the globe, as many different factors can cause seasonality. This limitation is thus a real problem for our model.

In order for our model to learn a seasonal pattern, we will add a third layer of exponential smoothing. The idea is that the model will learn **multiplicative seasonal factors** that will be applied to each period inside a full seasonal cycle. As for the trend (β) and the level (α), the seasonal factors will be learned via an exponential weighting method with a new specific learning rate: **gamma** (γ).

Multiplicative seasonal factors mean, for example, that the model will know that the demand is increased by 20% in January (compared to the yearly average) but reduced by 30% in February.

We will discuss the case of additive seasonality in Chapter 11.

9.2 Model

The main idea is that the forecast is now composed of the level (a) plus the (damped) trend (b) **multiplied by** the seasonal factor (s).

$$\text{Forecast} = (\text{Level} + \text{Trend})\,\text{Season}$$

$$f_{t+1} = (a_t + \phi b_t)\, s_{t+1-p}$$

Pay attention to the fact that, we need to use the seasonal factors that were calculated during the previous season: s_{t-p}, where p (for periodicity) is the season length.[1] The different seasonal factors (s) can be seen as **percentages** to be applied to the level in order to obtain the forecast. For example, for a monthly forecast, the statement, "We sell 20% more in January" would be translated as $s_{\text{january}} = 120\%$.

1 Typically, the periodicity will be 12 for yearly cycles.

https://doi.org/10.1515/9783110671124-009

10 Outliers

> *I shall not today attempt further to define this kind of material ... and perhaps I could never succeed in intelligibly doing so. But I know it when I see it.*
>
> Potter Stewart

In 1964, Potter Stewart was a United States Supreme Court Justice. He was discussing not outliers, but whether the movie *The Lovers* was obscene or not.

As you work on forecasts with the different models we saw—and the following models we will see later—you will notice that your dataset has outliers. And even though *I know it when I see it* might be the only practical definition, these outliers pose a real threat to supply chains. These high (or low) points will result in overreactions in your forecast or your safety stocks, ultimately resulting in (at best) manual corrections or (at worst) dead stocks, losses, and a nasty bullwhip effect. Actually, when you look at blogs, books, articles, or software on forecasting, the question of outlier detection is often eluded. This is a pity. **Outlier detection is serious business**.

These outliers pop out all the time in modern supply chains. They are mostly due to two main reasons:

Mistakes and Errors These are obvious outliers. If you spot such kind of errors or encoding mistakes, it calls for process improvement to prevent these from happening again.

Exceptional Demand Even though some demand observations are real, it does not mean they are not *exceptional* and shouldn't be cleaned or smoothed. This kind of exceptional sales is actually not so uncommon in supply chains. Think about promotions, marketing, strange customer behaviors, or destocking. Typically, you might not want to take into account for your forecast the exceptional –80% sales you did last year to get rid of an old, nearly obsolete inventory.

If you can spot outliers and smooth them out, you will make a better forecast. I have seen numerous examples where the forecast error was reduced by a couple of percentages, thanks to outlier cleaning. Flagging outliers manually is a time-intensive, error-prone, and un rewarding process; few demand planners will take the time necessary to review those. Therefore, the bigger the dataset, the more important it is to *automate* this detection and cleaning. As data scientists, we automate tasks to scale our processes. Let's see how we can do this for outliers detection.

In the following pages, we will discuss three and a half ideas to spot these outliers and put them back to a reasonable level.

10.1 Idea #1 – Winsorization

As we said, an outlier is an exceptionally high or low value. Based on this simple definition, a first idea to detect outliers would be to simply cut down the top x highest

https://doi.org/10.1515/9783110671124-010

11 Triple Additive Exponential Smoothing

11.1 The Idea Behind Triple Additive Exponential Smoothing

So far, we have discussed four different exponential smoothing models:
- Simple exponential smoothing
- Double exponential smoothing with (additive) trend
- Double exponential smoothing with (additive) damped trend
- Triple exponential smoothing with (additive) damped trend and multiplicative seasonality

The last model we saw is potent but still has some limitations due to its seasonality's multiplicative aspect. The issue of multiplicative seasonality is how the model reacts when you have periods with very low volumes. Periods with a demand of 2 and 10 might have an absolute difference of 8 units. Still, their relative difference is 500%, so the seasonality (which is expressed in relative terms) could drastically change. We will then replace this multiplicative seasonality with an additive one.

With multiplicative seasonality, we could interpret the seasonal factors as a percentage increase (or decrease) of the demand during each period. One could say "We sell 20% more in January but 30% less in February." Now, the seasonal factors will be absolute amounts to be added to the demand level. One could say "We sell 150 units more than usual in January but 200 less in February."

11.2 Model

The model idea is that the forecast is now composed of the level plus the (damped) trend, **plus** an additive seasonal factor s.

$$\text{Forecast} = \text{Level} + \text{Trend} + \text{Season}$$
$$f_{t+1} = a_t \quad + \phi b_t \quad + s_{t+1-p}$$

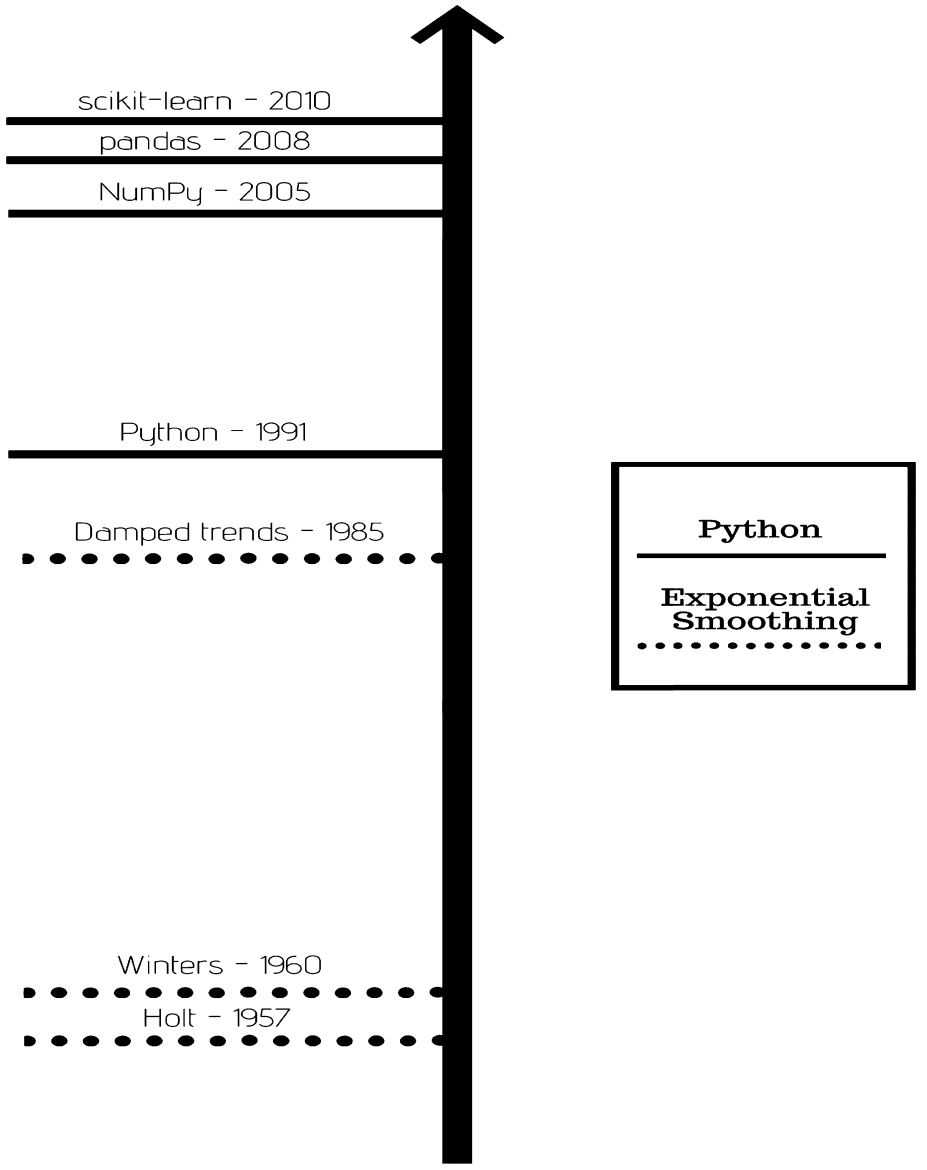
Pay attention—like for the multiplicative model, we use the seasonal factors from the previous cycle: s_{t-p} (where p denotes the season length—p for periodicity). The seasonal factors can now be read as an amount to add (or subtract) from the period level to obtain the forecast. For example, if $s_{\text{january}} = 20$, it means you will sell 20 pieces more in January than in an average month.

Component Updates

We calculate the different components as such:

$$a_t = \alpha\,(d_t - s_{t-p}) + (1-\alpha)\,(a_{t-1} + \phi b_{t-1})$$

https://doi.org/10.1515/9783110671124-011


scikit-learn – 2010
pandas – 2008
NumPy – 2005
Python – 1991
Damped trends – 1985
Winters – 1960
Holt – 1957
Python
Exponential Smoothing

Part II: **Machine Learning**

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.

https://doi.org/10.1515/9783110671124-012

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.

Product	Inputs Year 1				Output 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.

13 Tree

It's a dangerous business, Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to.

J. R. R. Tolkien, *The Lord of the Rings*

As a first machine learning algorithm, we will use a **decision tree**. Decision trees are a class of machine learning algorithms that will create a map (a tree, actually) of questions to make a prediction. We call these trees **regression trees** if we want them to predict a number, or **classification trees** if we want them to predict a category or a label.

Regression

Regression problems require an estimate of a numerical output based on various inputs. For example, forecasting is a regression problem.

Classification

Classification problems require you to classify data samples in different categories. For example, identifying pictures as cat or dog is a classification problem.

In order to make a prediction, the tree will start at its foundation with a yes/no question, and based on the answer, it will continue asking new yes/no questions until it gets to a final prediction. Somehow you might see these trees as a big game of "Guess Who?" (the famous '80s game): the model will ask multiple consecutive questions until it gets to a good answer.[1]

In a decision tree, each question is called a **node**. For example, in Figure 13.1, "Does the person have a big nose?" is a node. Each possible final answer is called a **leaf**. In Figure 13.1, each leaf contains only one single person. But that is not mandatory. You could imagine that multiple people have a big mouth and a big nose. In such case, the leaf would contain multiple values.

The different pieces of information that a tree has at its disposal to split a node are called the **features**.

Feature

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

1 If you do not think an algorithm can easily make a prediction based on a few yes/no questions, I advise you to check out Akinator—a genius who will find any character you can think of within a few questions (en.akinator.com).

https://doi.org/10.1515/9783110671124-013

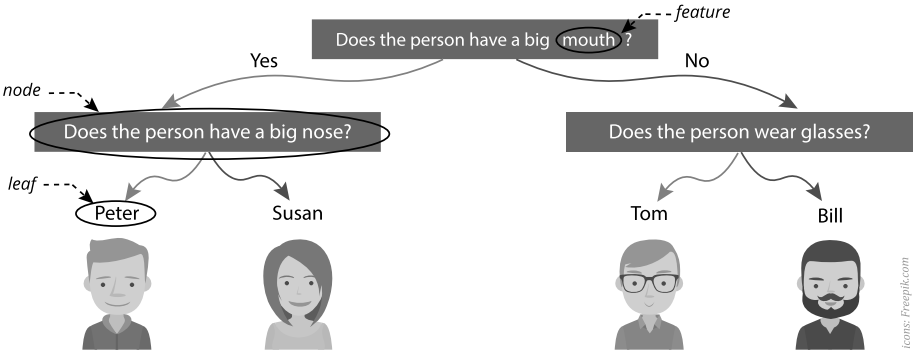


Figure 13.1: Decision tree for *Guess Who?*

For example, the tree we had in Figure 13.1 for the game "Guess Who?" could split a node on the three features: mouth, nose, and glasses.

13.1 How Does It Work?

To illustrate how our tree will grow, let's go back to Table 13.1, our quarterly dummy dataset.

Table 13.1: Training dataset.

X_train				Y_train
5	15	10	7	6
15	10	7	6	13
10	7	6	13	11
7	6	13	11	5
6	13	11	5	4
13	11	5	4	11
11	5	4	11	9
7	2	3	1	1

Based on this training dataset, a **smart** question to ask yourself in order to make a prediction is: *Is the first demand observation >7?*

As shown in Table 13.2, this is a useful question, as you know that the answer (Yes/No) will provide an interesting indication of the behavior of the demand for the next quarter. If the answer is *yes*, the demand we try to predict is likely to be rather high (≥ 8), and if the answer is *no*, then the demand we try to predict is likely to be low (≤ 7).

14 Parameter Optimization

When we created our regression tree in Chapter 13, we chose some parameters:

```
tree = DecisionTreeRegressor(max_depth=5, min_samples_split=15,
    min_samples_leaf=5)
```

But are we sure these are the best? Maybe if we set `max_depth` to 7, we could improve our model accuracy? It is unfortunately impossible to know *a priori* what the best set of parameters is. But that's fine—we are supply chain data scientists, we love to run experiments.

14.1 Simple Experiments

As a first technique, we will run through different values for the maximum depth, test each of them, and select the best one. As you can see in Figure 14.1, increasing the maximum depth of our tree (i. e., allowing it to ask more consecutive questions) continuously improves the accuracy over the *training* set. But we reach a plateau for the *test* set accuracy as of around six consecutive questions. Actually, the accuracy on the test set is even a bit worse if the maximum depth is 14 rather than 6. It is important to understand that if we allow our model to ask eight consecutive questions or more, we overfit it to the training set, and it won't perform well on the test set. So should we choose `max_depth=6` and call it a day? No.

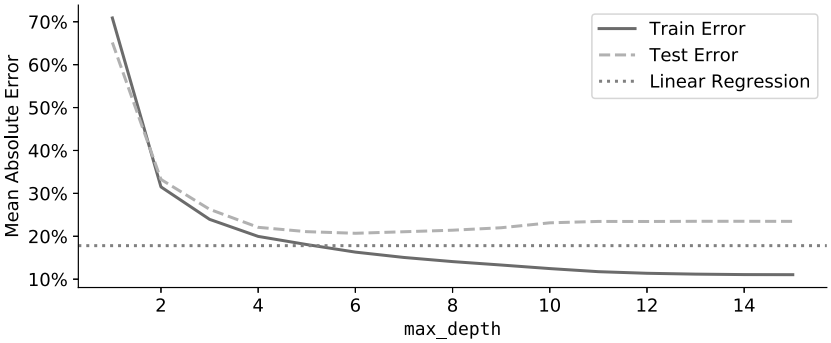


Figure 14.1: Experimentation for maximum depth optimization.

If we did this, we would actually be optimizing this parameter directly against the test set. We would then face the risk of overfitting the model to the test set and not being able to replicate similar results against new data.

https://doi.org/10.1515/9783110671124-014

15 Forest

15.1 The Wisdom of the Crowd and Ensemble Models

In social choice theory[1] there is a concept called **the wisdom of the crowd**. This idea explains that the average opinion of a group of people is going to be more precise (on average) than the opinion of a single member of the group. Let's explore a simple example: if you want to make a forecast for the demand of a product next month, it is better to ask the opinion of many different team members (salespeople, marketing, CEO, supply chain planners, financial analysts) and take the average of the different forecasts, rather than to trust only one team member blindly.

A Brief History of the Wisdom of the Crowd

In 1906, Francis Galton, an English scientist, went visiting a livestock fair. He witnessed a contest: the villagers were invited to guess the weight of an ox. Eight hundred people took part in this contest, all noting down their guess on tickets. Galton noted that "the hope of a prize and the joy of competition prompted each competitor to do his best. The competitors included butchers and farmers, some of whom were highly expert in judging the weight of cattle." After the event, he was able to perform a statistical analysis of the various guesses. To his surprise, averaging all the guesses resulted in a virtually perfect weight estimation.[a] Moreover, this average guess was beating the actual winner of the contest, and the guesses made by experts.

a See Wallis (2014) for more information.

In their (excellent) book *Superforecasting: The Art and Science of Prediction*, Philip E. Tetlock and Dan Gardner explain how one can harness the power of the wisdom of the crowd among a team to predict anything from stock-price value to presidential elections.[2] For wisdom to emerge out of a crowd, you need three main elements. First, each individual needs independent sources of information. Second, they must make independent decisions (not being influenced by the others). And third, there must be a way to gather and weigh those individual predictions into a final one (usually taking the average or median of the various predictions).

1 The social choice theory is the science of combining different individual choices into a final global decision. This scientific field emerged in the late 18th century with the voting paradox. It states that a group's collective preference can be cyclic (i. e., A is preferred to B, B is preferred to C, and C is preferred to A) even if none of the preferences of each individual are cyclic.

2 See Tetlock and Gardner (2016).

https://doi.org/10.1515/9783110671124-015

16 Feature Importance

In some ways, AI is comparable to the classical oracle of Delphi, which left to human beings the interpretation of its cryptic messages about human destiny.

Henry A. Kissinger, Eric Schmidt, Daniel Huttenlocher[1]

Statistical models are interesting, as they can show us the interactions between the different input variables and the output. **Such models are easy to understand and to interpret**. In the first part of the book, we worked with many different exponential smoothing models: we could analyze them by simply looking at their level, trend, or seasonality over time. Thanks to them, we can easily answer questions such as *Do we have a seasonality?* or *Is there a trend?* And if the forecast for a specific period is strange, we can look at how the sub-components (level, trend, seasonality) are behaving to understand where the error comes from.

This is unfortunately not the case with machine learning. These models are very difficult to interpret. A forest will never give you an estimation of the level, trend, or seasonality of a product. You won't even know if the model *sees* a seasonality or a trend. Nevertheless, we have one tool at our disposal to understand how our machine learning algorithm thinks: the feature importance.

As you remember, we created a machine learning algorithm that looked at the last 12 months of historical car sales in Norway to predict the sales in the following month. In order to do that, we trained our model by showing it many different sequences of 13 months so that it could learn from these sequences how to predict the 13th month based on the 12 previous ones.

Before we continue with new models and further optimization, let's discuss the importance of these inputs. We do not know yet which of these 12 historical months is the most important to predict the sales of the 13th.

We would like to answer questions such as:

- Is M-1 more important than M-12?
- Is M-5 of any help?

When we grow a tree or a forest, we can get an idea of each feature's importance (in our specific case, each feature is one of the 12 previous periods). There are different definitions and ways to compute each feature's importance; we will focus on the one used by the scikit-learn library. Basically, the importance of a feature is the reduction it brings to the objective that the algorithm tries to minimize. In our case, we want to bring the MAE or the RMSE down. Therefore, the **feature importance** is measured as the forecast accuracy brought by each feature. The feature importance is then normalized (i. e., each feature importance is scaled so that the sum of all features importance is 1).

1 See Kissinger et al. (2019).

https://doi.org/10.1515/9783110671124-016

17 Extremely Randomized Trees

The random forest idea was that we could obtain a better forecast accuracy by taking the average prediction of many *different* trees. To create those slightly different trees from the same initial training dataset, we used two tricks. The first was to limit the number of features the algorithm could choose from each node split. The second trick was to create different random subsets of the initial training dataset (thanks to bootstrapping and only keeping a subset of the initial samples).

In 2006, Belgian researchers Pierre Geurts, Damien Ernst, and Louis Wehenkel introduced a third idea to further increase the differences between each tree.[1] At each node, the algorithm will now choose a split point **randomly** for each feature and then select the best split among these. It means that, for our Norwegian car sales dataset, this new method will draw at each node one random split point for each of the 12 previous months (our features) and, among these 12 potential splits, choose the best one to split the node. The fact that an Extremely Randomized Trees (or ETR) draws split points randomly seems counter intuitive. Still, this will increase further the *difference* between each tree in the ETR, resulting in better overall accuracy. Remember, an ensemble model (such as the ETR or the forest) is more accurate, as its sub-models are different.

17.1 Do It Yourself

We will once again use the scikit-learn library, so our code will be very similar to the one we used for the random forest.

```
from sklearn.ensemble import ExtraTreesRegressor
ETR = ExtraTreesRegressor(n_jobs=-1, n_estimators=200, ⌟
 ↪  min_samples_split=15, min_samples_leaf=4, max_samples=0.95, ⌟
 ↪  max_features=4, max_depth=8, bootstrap=True)
ETR.fit(X_train,Y_train)
```

The input parameters we use (`n_jobs`, `n_estimators`, `max_depth`, `max_features`, `min_samples_leaf`, `min_samples_split`) are the same as for the random forest.

Let's print the results.

```
Y_train_pred = ETR.predict(X_train)
Y_test_pred = ETR.predict(X_test)
kpi_ML(Y_train, Y_train_pred, Y_test, Y_test_pred, name='ETR')
```

1 See Geurts et al. (2006).

https://doi.org/10.1515/9783110671124-017

18 Feature Optimization #1

The more you know about the past, the better prepared you are for the future.

Theodore Roosevelt

So far, we have created three different models (a regression tree, a random forest, and a set of extremely randomized trees), and we have optimized each of them via a random search automatically running through some possible parameter sets and testing them via k-fold cross-validation.

We took the time to choose a model and optimize its parameters. There is one thing that we haven't optimized (yet): the features that we use.

The different models that we used so far have an interesting feature: once fitted to a training dataset, they can show each input's (i. e., feature) importance. Remember that, in our case, the input features are the different historical periods that we use to populate our forecast. For our car sales in Norway, we used the last 12 months of sales to predict the next month: these are our features. We previously looked at the feature importance of our random forest and saw that M-1 was the most critical month, leaving half of the other months useless. As you can see in Figure 18.1, if we do the same exercise with the ETR model, we obtain a much flatter curve.

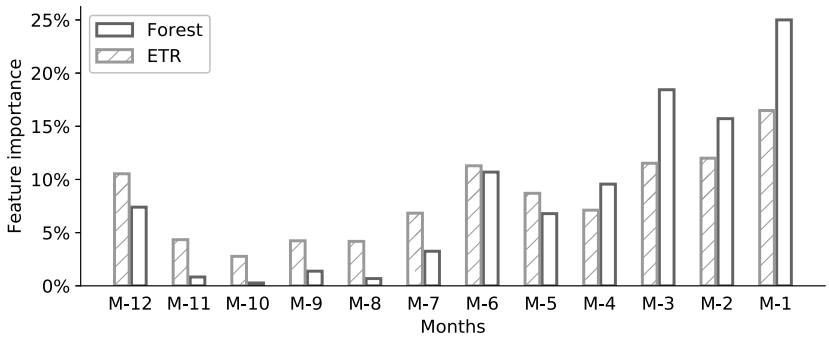


Figure 18.1: Feature importance.

How can we explain this? As you might remember, the difference between the extra trees model and the random forest lies in the fact that the split point chosen at each node is not the optimal one, but can only be selected from a set of random points for each selected feature. So even though M-1 should be the most useful feature, in many cases, the random split point proposed for this feature is not the best across the different possibilities the algorithm can choose from.

Based on this graph, we can then ask ourselves an important question: *what if we used 13 months instead of 12 to make a forecast?* To answer this question, we will have to do a lot of experiments (on our training set).

https://doi.org/10.1515/9783110671124-018

19 Adaptive Boosting

Note to the Reader

We introduce the Adaptive Boosting model in this chapter more as an essential historical step in the path of machine learning rather than as a plea to use it. We will see in Chapter 21 a more powerful model—and one that is simpler to use, which can be seen as the evolution of Adaptive Boosting.

Can a set of weak learners create a single strong learner?

Michael Kearns, Leslie Valiant

19.1 A Second Ensemble: Boosting

In Chapters 15 and 17, two models—the forest and the extremely random trees—that both created a very good prediction by averaging many slightly different, good models. These models were called *ensemble bagging* models: *ensemble* as they use an ensemble of different models to make a prediction; *bagging* as they use the average of many sub-models to make a prediction.

In the late 1980s, Michael Kearns and Leslie Valiant asked the following question, *"Can a set of weak learners create a single strong learner?"*[1]

Weak Model

A weak model (or weak learner) is a model that is slightly more accurate than random guessing—typically, a simple, shallow tree.

If we reframe this question, we could say: *Can we obtain a good forecasting model by using only (very) simple trees?* In 1990, Robert E. Schapire answered this positively. In 1997, Yoav Freund and Robert E. Schapire published an algorithm together that could create a strong learner based on weak ones.[2] To do so, they used another *ensemble* technique called **boosting**.

1 See Kearns (1988), Kearns and Valiant (1989). In his work, Kearns noted that "the resolution of this question is of theoretical interest and possibly of practical importance." As you will see, boosting algorithms further pushed the frontiers of machine learning power.

2 See Schapire (1990), Freund and Schapire (1997) for the original papers. For a detailed review and discussion between multiple academic authors, see Breiman et al. (1998) (available here: projecteuclid.org/download/pdf_1/euclid.aos/1024691079). Those initial papers discussed Adaptive Boosting for classification problems; the first implementation of AdaBoost for regression problems was published by Drucker (1997), which is the algorithm used in scikit-learn.

https://doi.org/10.1515/9783110671124-019

20 Demand Drivers and Leading Indicators

Felix, qui potuit rerum cognoscere causas – Fortunate, who was able to know the causes of things.
Virgil (70–19 BC)

Until now, we have made our forecasts solely based on historical demand. We have discussed in Chapter 14 how to optimize our model, and we have discussed in Chapter 18 how to optimize the number of historical periods we should take into account to make a prediction. We haven't discussed the following yet: which other factors could we be looking at to predict future demand?

For many businesses, historical demand is not the only—or main—factor that drives future sales. Other internal and external factors drive the demand as well. You might sell more or less depending on the weather, the GDP growth, unemployment rate, loan rates, and so on. These **external factors** (external, as a company does not control them) are often called **leading indicators**.

The demand can also be driven by company decisions: price changes, promotions, marketing budget, or another product's sales. As these factors result from business decisions, we will call them **internal factors**.

As you can see in Figure 20.1, we will group both internal and external factors into the term **exogenous factors** (as these factors are *exogenous* to the historical demand), or, more simply, **demand drivers**.

Figure 20.1: Demand drivers classification.

20.1 Linear Regressions?

As we discussed in the previous chapters, one of the limitations of the different exponential smoothing models was their inability to deal with exogenous information. Typically, in most forecasting training and classes, you will be shown linear regressions such as: "Our sales increase by 10% during weekends," or "Demand increases by 10% for every degree above 25C°." You will be shown data where this works quite well, as in Figure 20.2.

https://doi.org/10.1515/9783110671124-020

21 Extreme Gradient Boosting

21.1 From Gradient Boosting to Extreme Gradient Boosting

In 2001, Jerome H. Friedman proposed a new concept to boost trees: **Gradient Boosting**.[1] The general concept of Gradient Boosting and Adaptive Boosting is essentially the same: they are both ensemble models boosting (stacking) trees on top of each other based on the model mistakes. The main difference is that in Gradient Boosting, each new weak learner is stacked directly on the model's current errors rather than on a weighted version of the initial training set.

Extreme Gradient Boosting

As is common, this first algorithm was refined by the original author over time. Later, Chen and Guestrin (from the University of Washington) proposed a new gradient boosting algorithm—called *Extreme Gradient Boosting* or *XGBoost*—and formalized it in 2016.[2] The data science community has since widely used this implementation, with excellent results. This is simply one of the most powerful machine learning algorithms currently available. In this chapter, we will focus on XGBoost rather than the *regular* Gradient Boosting. The differences between Gradient Boosting and Extreme Gradient Boosting lies in some mathematical workings of each model and their respective implementations.[3] As users, XGBoost will bring us three improvements compared to AdaBoost and regular Gradient Boosting:

- XGBoost is faster.
- XGBoost is (generally) better
- XGBoost allows for more parameters to be optimized.

21.2 Do It Yourself

Installation

Scikit-learn proposes an implementation of the original Gradient Boosting algorithm proposed by J. Friedman.[4] We will prefer the library developed around Chen's algorithm to benefit from his efficient and powerful implementation.[5]

1 See Friedman (2001) for the original paper, and Kashnitsky (2020) (mlcourse.ai/articles/topic10-boosting) for a detailed explanation and comparison against AdaBoost.

2 See Chen and Guestrin (2016).

3 The main mathematical difference between XGBoost and GBoost lies in the fact that in XGBoost uses a *regularized* objective function that penalizes both the number of leaves in a tree and extreme weights given to individual leaves. Even though the exact mathematics involved are beyond the scope of this book—see Chen and Guestrin (2016) for the detailed model—we will use those regularization parameters to improve our model.

4 See an example of implementation here, scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html

5 See the official website xgboost.readthedocs.io

https://doi.org/10.1515/9783110671124-021

22 Categorical Features

As we saw in Chapter 20, we can improve our forecast by enriching our dataset—that is by adding external information to our historical demand. We saw that it might not be straightforward (nor meaningful) for all businesses to use such external macroeconomic elements. On the other hand, most supply chains serve different markets (often through different channels) and have different product families. What if a machine learning model could benefit from these extra pieces of information: *Am I selling this to market A? Is this product part of family B?*

In our car sales dataset, we could imagine that instead of only having sales in Norway, we could have the sales in different markets across Europe. You could then feed the algorithm with the sales per country and indicate the market of each data sample (e. g., Sweden, Finland). We could also imagine segmenting into four categories: low cost, normal, premium, and luxury brands—or simply allowing the model to see the brand it is forecasting.

Unfortunately, most of the current machine learning libraries (including scikit-learn) cannot directly deal with categorical inputs. This means that you won't be able to fit your model based on an `X_train` dataset, as shown in Table 22.1.

Table 22.1: `X_train` with categorical data.

`X_train`					`Y_train`
Segment	**Demand**				**Demand**
Premium	5	15	10	7	6
Normal	2	3	1	1	1
Low cost	18	25	32	47	56
Luxury	4	1	5	3	2

The machine learning models that we discussed can only be trained based on numerical inputs. That means that we will have to transform our categorical inputs into numbers.

22.1 Integer Encoding

A first way to transform categories into numbers is simply to allocate a value to each category, as shown in Table 22.2.

This is a rather simple and straightforward way to transform a categorical input into a numerical one. It is meaningful if you have an **ordinal** relationship between all the categories. An ordinal relationship means that the different categories have a **nat-**

https://doi.org/10.1515/9783110671124-022

23 Clustering

As discussed in Chapter 22, it can be helpful for both data scientists and machine learning models to classify the various products they have. Unfortunately, you do not always receive a pre-classified dataset. Could a machine learning model help us classify it? Yes, of course. In order to do so, we will have to use **unsupervised** machine learning models.

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. It is called supervised, as you show the model what the desired output is.

All the machine learning models we have seen so far are called supervised models (you cannot ask your algorithm to make a forecast if you never show it what a good forecast looks like).

Unsupervised Learning

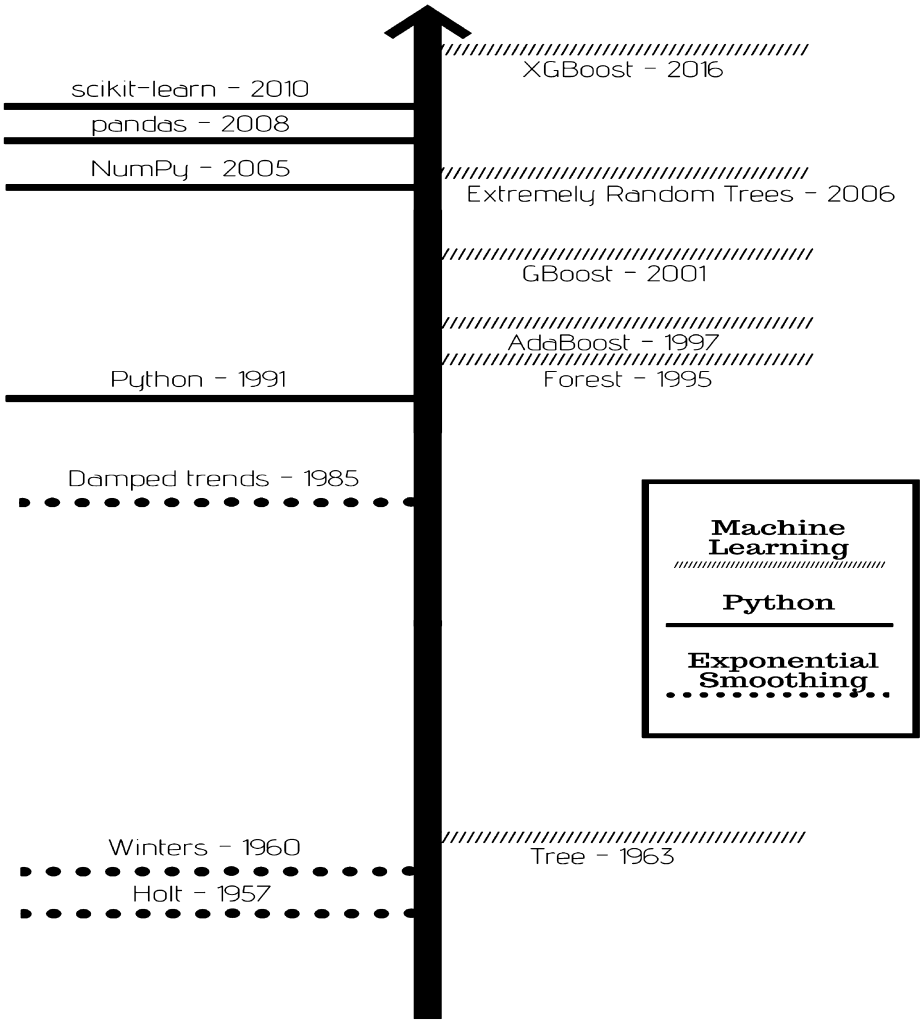
An **unsupervised** machine learning model is a model that is only fed with inputs and no specific desired outputs. It is up to the machine learning algorithm to make some order of (categorize) the different data observations. You can simply see it as asking your algorithm to assign a label to each data observation. This is called unsupervised learning, as you do not label samples in any pre-given way, henceforth not requiring any specific output.

Unsupervised learning, using given meaningful features, can cluster anything from products to clients or from social behaviors to neighborhoods in a city. Usually, a product catalog is segmented based on product families (or brands). But products within the same family might not share the same demand patterns (seasonality, trends). For example, meat for barbecues will sell in a similar way as outside furniture, but not regular meat. Student notebooks will be purchased with the same seasonality as business notebooks, although they look alike.

23.1 K-means Clustering

A very famous unsupervised machine learning model is called **K-means**. This algorithm will classify each data observation into k different clusters (see Figure 23.1). Let's look in detail at how this works.

https://doi.org/10.1515/9783110671124-023


XGBoost – 2016
scikit-learn – 2010
pandas – 2008
NumPy – 2005
Extremely Random Trees – 2006
GBoost – 2001
AdaBoost – 1997
Python – 1991
Forest – 1995
Damped trends – 1985
Machine Learning
Python
Exponential Smoothing
Winters – 1960
Tree – 1963
Holt – 1957

25 Neural Networks

Note to the Reader

Neural networks (or *deep learning*, which usually refers to "deep" neural networks) is a vast subject. This chapter is only an introduction to the topic. It will be enough, though, to get you started in no time using some of the most recent best practices.

If you are interested in learning more about neural networks, do not hesitate to apply for Andrew Ng's deep learning specialization on Coursera (www.coursera.org/specializations/deep-learning).

The 2010s saw the deep learning tsunami.

In 2012, the revolution started at the ImageNet Large Scale Visual Recognition Challenge, a data science competition to classify pictures. A team led by Alex Krizhevsky (from the University of Toronto) achieved an unprecedented error rate of 15.3%, whereas the second best model got only 26.2%. Krizhevsky used a specific type of neural networks: convolutional neural networks. Moreover, he trained his network using graphical processing units (GPU) instead of the traditional CPUs.[1] In 2012, no other participants were using neural networks in this challenge. The next year, in the 2013 edition, all participants used similar neural networks.

In 2016, AlphaGo—an AI developed by DeepMind—beat the Go world champion. Computers had ruled chess-playing since 1997, after Deep Blue beat Kasparov in a famous match.[2] But beating humans at playing Go is a much more complex challenge, with an estimated 10^{170} possible combinations against 10^{120} for chess. Many experts were surprised by this early victory that wasn't expected to happen until a decade later. DeepMind was acquired by Google in 2014 for more than $500 million.

In 2017, DeepMind released AlphaGo Zero, which beat the late 2016-AlphaGo 100 games to 0. AlphaGo Zero only required three days of training to achieve this, learning to play Go only by competing against itself. In 2017, DeepMind reported paying $243 million to its 700 employees. This amount increased twofold in 2018.[3]

In 2020, OpenAI released the third iteration of an AI specialized in producing texts: GPT-3. This AI can write poetry, tell stories, solve equations, code websites, and even write articles about itself. Earlier, in 1950, Alan Turing (1912–1946, famous En-

1 See Krizhevsky et al. (2012).

2 Deep Blue suffered a bug during the first game. Kasparov interpreted it as a stroke of genius from the computer. He lost confidence resulting in him losing the match, despite using a good technique in the first match: he played an unusual opening to confuse the computer. See Silver (2015) for the whole story.

3 See Shead (2019).

https://doi.org/10.1515/9783110671124-025

glish computer scientist) proposed the "Turing test" to assess if a machine could imitate human language well enough to be confused for a human. GPT-3 is now writing journal articles that are often barely recognizable from human ones. OpenAI—funded by Elon Musk—was initially a non-profit organization. It became a *capped* profit organization in 2019, capping the returns on investment at a hundred times the initial amount. Shortly after, they secured a $1 billion investment from Microsoft.[4]

25.1 How Neural Networks Work

An (artificial) neural network is a **network** of **neurons** using specific **activation functions** trained by an **optimization algorithm**. In this section and the following section, we will define and explain each of these terms.

Neuron

As shown in Figure 25.1, an (artificial) neuron is a (mathematical) unit receiving information from various **weighted inputs** (the weights are noted w_i and the inputs x_i) and a **bias** (b). The neuron will sum those weighted inputs and its bias (we note the sum z with $z = b + \sum w_i x_i$), and then apply an **activation function** ($f(z)$) to the result. This activation function determines the neuron's behavior: how it reacts to inputs. In simple words, a neuron transforms a sum of inputs by using an activation function.

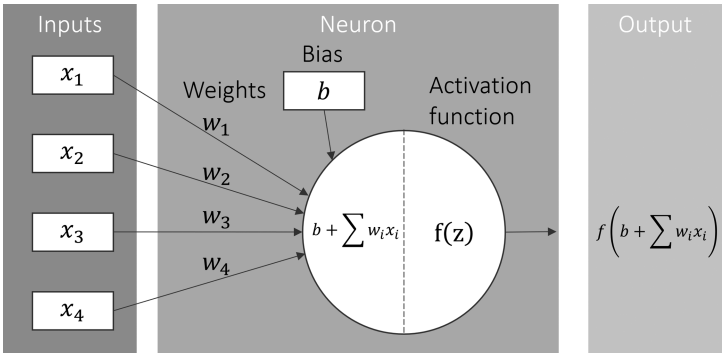
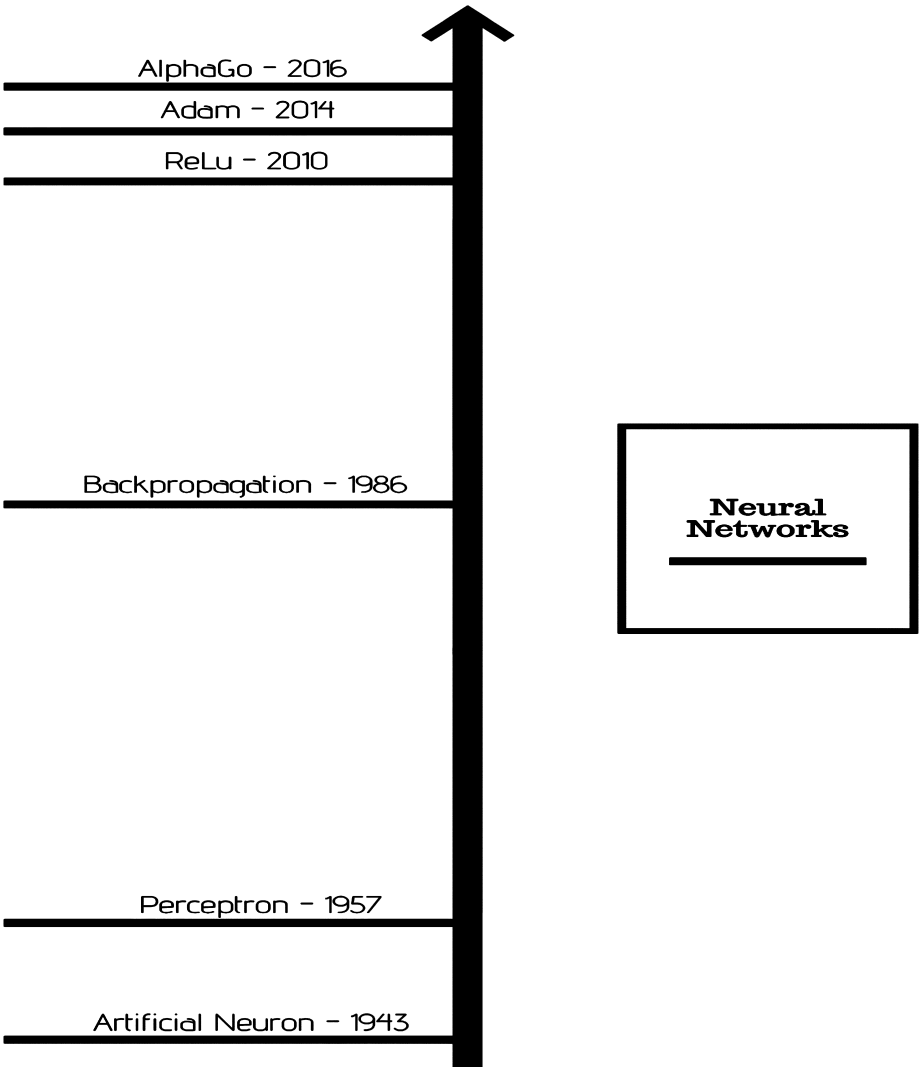


Figure 25.1: Artificial neuron with four inputs.

4 See Vincent (2019).


AlphaGo - 2016
Adam - 2014
ReLu - 2010
Backpropagation - 1986
Perceptron - 1957
Artificial Neuron - 1943
Neural Networks

Part III: **Data-Driven Forecasting Process Management**

26 Judgmental Forecasts

> *The desire to be right and the desire to have been right are two desires, and the sooner we separate them, the better off we are. The desire to be right is the thirst for truth. On all counts, both practical and theoretical, there is nothing but good to be said for it. The desire to have been right, on the other hand, is the pride that goes before a fall. It stands in the way of our seeing we were wrong, and thus blocks the progress of our knowledge.*
>
> Willard Van Orman Quine

In Parts I and II, we discussed how to make a statistical or ML-driven forecast engine. This engine will populate a forecast baseline for you. Is forecast baseline the end of the forecasting process in a supply chain? No, it can still be enriched by humans by using various sources of insights and information. A human-created forecast is called a **judgmental forecast**, as it relies on human judgment. Using judgmental forecasts comes with a lot of risks, as we will discuss in this chapter. Nevertheless, if done properly, it will add value to your baseline forecast (as we will discuss in Chapter 27).

Judgmental forecasts are also appropriate to forecast products when you lack historical data to use a model, or when significant changes are ongoing (due to changing client behavior, new competition, changing legislation, the COVID-19 pandemic, etc.).

Before discussing judgmental forecasts further, keep in mind that a demand forecast should be the best *unbiased* estimate of a supply chain's future demand. This is nothing like a *budget*, a *sales target* to incentivize sale representatives, or a *production plan*.

26.1 Judgmental Forecasts and Their Blind Spots

As shown in Figure 26.1, Oliva and Watson (2009) proposed a framework to classify sources of bias in judgmental forecasts into two categories: intentional and unintentional sources. Let's review those in detail.

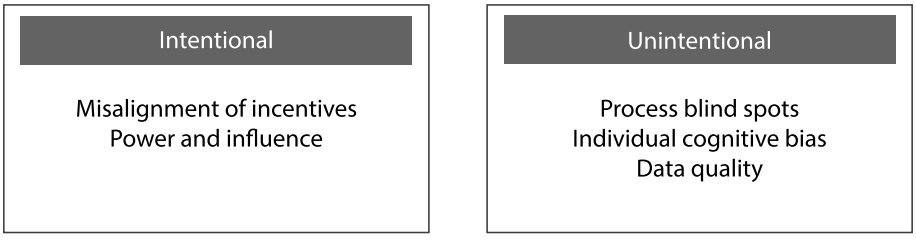


Figure 26.1: Sources of bias in judgmental forecasts.

https://doi.org/10.1515/9783110671124-026

27 Forecast Value Added

In this chapter, we will identify how to improve a forecast baseline using the best practices of judgmental forecast (as seen in Chapter 26). We will first discuss, in Section 27.1, how to use smart KPIs to manage a portfolio of products and focus on products that matter the most. Then, in Section 27.2, we will see how to track the value added by each team in the forecasting process.

27.1 Portfolio KPI

When we discussed forecast KPIs in Chapter 2, we considered each item separately: we wanted our forecast engine to find the best model for each product, independently of the others. On the other end, demand planners do not have the time to inspect every single product (on every single location) one by one. As a demand planner, you need to scale up from analyzing in detail one single item to working on a portfolio with thousands of SKUs. To do so, you will have to prioritize your work by focusing on products that matter the most. In short: your time is constrained, so you should prioritize it. The forecast engine's time isn't, so you can let it do its best on each item.

Smart Weighting

As a demand planner, you have the opportunity to review *some* of your product forecasts during each forecast exercise. But on which items should you spend your time? The first idea would be to look at the products with the most significant forecast error. Let's imagine an example where you are responsible for nails and hammers. As shown in Table 27.1, the absolute forecast error is more significant on the nails (500 pieces) than on the hammers (50 pieces). Should you, therefore, focus your time and efforts on nails?

Table 27.1: Forecast KPI.

Product	Forecast	Demand	Error	|Error|
Hammer	150	100	50	50
Nail	1000	1500	−500	500
Total	1150	1600	−450	550
			−28.1%	34.4%

Obviously, not every SKU is created equal: some bring more profits, some are costly, some use constrainted resources (e. g., space), some are of strategic importance...

https://doi.org/10.1515/9783110671124-027

Now It's Your Turn!

Your task is not to foresee the future, but to enable it.

Antoine de Saint-Exupéry

The real purpose of this book was not *just* to explain different models but, more importantly, to **give you the appetite to use them**. This book is a toolbox that gives you the tools and models to create your **own forecast models**. Hopefully, it has ignited ideas for you to create *unique* models and has taught you the methodology to test them.

You have learned in Part I how to create a robust statistical baseline forecast by using the different exponential smoothing models. Moreover, we have discussed different ideas (especially the different initialization methods) to tweak them to any dataset. We also have created a robust model to detect—and correct—outliers in an automated fashion. We have discussed various forecast KPIs, ensuring that you will be able to assess your models properly.

In Part II, you have discovered how machine learning could create advanced forecast models that could learn relationships across a whole dataset. Machine learning will allow you to classify your products and uncover potential *complex* relationships with any demand driver.

Finally, in Part III, we discussed the best practices required for judgmental forecasting and how to use the forecast value added framework to create—and sustain—an efficient forecasting process.

Now, it is your turn to work on creating your own models. The most important message is: **You can do it**. Start your journey by collecting historical demand data. Once you have gathered enough data (aim for five years), build your first model using simple machine learning models (such as random forests). You can then refine your model by either including new features (pricing, promotions, historical stock-outs, etc.) or using more advanced ML techniques. Remember that to succeed, you will have to test many ideas; avoid overfitting, and avoid the temptation to add too much complexity at once. Openly discuss your ideas, models, and results with others.

You will achieve astonishing results.

The future of demand planning is human-machine teams working hand-in-hand. You have learned here the keys to creating your own machine-learning driven models and how to best interact with them to create even more value.

This is the future of supply chain forecasting.

https://doi.org/10.1515/9783110671124-028

A Python

If this is your first time using Python, let's take some time to quickly introduce and discuss the different libraries and data types we are going to use. Of course, the goal of this book is not to give you full training in Python; if you wish to have an in-depth introduction (or if you are not yet convinced to use Python), please refer to the recommended courses in the Introduction.

How to Install Python

There are multiple ways to install Python on your computer. An easy way to do this is to install the Anaconda distribution on www.anaconda.com/download. Anaconda is a well-known platform used by data scientists all over the world. It works on Windows, Mac, and Linux. Anaconda will take care of installing all the Python libraries you need to run the different models that we are going to discuss. It will also install Spyder and Jupyter Notebook, two Python-code editors that you can use to type and run your scripts. Feel free to check both and use your favorite.

Lists

The most basic object we will use in Python is a list. In Python, a list is simply an ordered sequence of any number of objects (e. g., strings, numbers, other lists, more complex objects, etc.). You can create a list by encoding these objects between `[]`. Typically, we can define our first time series `ts` as:

```
ts = [1,2,3,4,5,6]
```

These lists are very efficient at storing and manipulating objects, but are not meant for number computation. For example, if we want to add two different time series, we can't simply ask `ts + ts2`, as this is what we would get:

```
ts = [1,2,3,4,5,6]
ts2 = [10,20,30,40,50,60]
ts + ts2
Out: [1, 2, 3, 4, 5, 6, 10, 20, 30, 40, 50, 60]
```

Python is returning a new, *longer* list. That's not exactly what we wanted.

https://doi.org/10.1515/9783110671124-029

NumPy Arrays

This is where the famous **NumPy**[1] library comes to help. Since its initial release in 2006, NumPy has offered us a new data type: a NumPy array. This is similar to a list, as it contains a sequence of different numeric values, but differs in the way that we can easily call any mathematical function up on them. You can create one directly from a list like this:

```
import numpy as np
ts = np.array([1,2,3,4,5,6])
```

As you will see, NumPy is most often imported as `np`.

We can now simply add our `array ts` to any other array.

```
ts2 = np.array([10,20,30,40,50,60])
ts + ts2
Out: array([11, 22, 33, 44, 55, 66])
```

Note that the result is another NumPy array (and not a simple list).

NumPy most often works very well directly with regular lists, because we can use most of the NumPy functions directly on them. Here is an example:

```
alist = [1,2,3]
np.mean(alist)
Out: 2.0
```

You can always look for help on the NumPy official website.[2] As you will see yourself, most of your Google searches about NumPy functions will actually end up directly in their documentation.

Slicing Arrays

To select a particular value in a list (or an array), you simply have to indicate between `[]` the index of its location inside the list (array). The catch—as with many coding languages—is that the index starts at 0 and not at 1; so the first element in your list will have the index 0, the second element will have the index 1, and so on.

1 NumPy is short for **Num**eric **Py**thon.

2 docs.scipy.org/doc/numpy/reference/

```
alist = ['cat','dog','mouse']
alist[1]
Out: 'dog'
anarray = np.array([1,2,3])
anarray[0]
Out: 1
```

If you want to select multiple items at once, you can simply indicate a range of index with this format: `[start:end]`. Note the following:

- If you do not give a `start` value, Python will assume it is 0.
- If you do not give an `end`, it will assume it is the `end` of the list.

Note that the result will **include** the `start` element but will **exclude** the `end` element.

```
alist = ['cat','dog','mouse']
alist[1:]
Out: ['dog','mouse']
anarray = np.array([1,2,3])
anarray[:1]
Out: np.array([1])
```

If you give a negative value as the end, Python will start by counting backward from the last element of your list/array (–1 being the last element of your array).

```
alist = ['cat','dog','mouse']
alist[-1]
Out: ['mouse']
alist[:-1]
Out: ['cat','dog']
```

You can slice a multi-dimensional array by separating each dimension with a comma.

```
anarray = np.array([[1,2],[3,4]])
anarray[0,0]
Out: 1
anarray[:,-1]
Out: array([2, 4])
```

Pandas DataFrames

Pandas is one of the most used libraries in Python (it was created by Wes McKinney in 2008). The name comes from **pan**el **da**ta, because it helps to order data into tables. Think Excel-meets-databases in Python. This library introduces a new data type: a **DataFrame**. If you're a database person, just think of a DataFrame as an SQL table. If you're an Excel person, just imagine a DataFrame as an Excel table. Actually, a DataFrame is a sort of data table in which each column would be a NumPy array with a specific name. That will come in pretty handy, because we can select each column of our DataFrame by its name.

There are many ways to create a DataFrame. Let's create our first one by using a list of our two time series.

```
import pandas as pd
pd.DataFrame([ts,ts2])
```

```
Out:
    0   1   2   3   4   5
0   1   2   3   4   5   6
1  10  20  30  40  50  60
```

The convention is to import pandas as `pd` and to call our main DataFrame `df`. The output we get is a DataFrame where we have 6 columns (named `'0'`,`'1'`,`'2'`,`'3'`,`'4'` and `'5'`) and 2 rows (actually, they also have a name—or index—`'0'` and `'1'`).

We can easily edit the column names:

```
df = pd.DataFrame([ts,ts2])
df.columns = ['Day1','Day2','Day3','Day4','Day5','Day6']
print(df)
```

```
Out:
   Day1  Day2  Day3  Day4  Day5  Day6
0     1     2     3     4     5     6
1    10    20    30    40    50    60
```

Pandas comes with very simple and helpful official documentation.[3] When in doubt, do not hesitate to look into it. As with NumPy, most of your Google searches will end up there.

3 pandas.pydata.org/pandas-docs/stable/

Creating a DataFrame from a Dictionary

Another way to create a DataFrame is to construct it based on a dictionary of lists or arrays. A **dictionary** is a collection of elements that links (unique) keys to values. You can create one by including between `{}` a `key` and a `value` (both can be any Python object).

```
dic = {'Small product':ts,'Big product':ts2}
dic
Out:
{'Small product': array([1, 2, 3, 4, 5, 6]),
 'Big product': array([10, 20, 30, 40, 50, 60])}
```

Here, the `key 'Small product'` will give you the value `ts`, whereas the `key 'Big product'` will give you ts2.

```
dic['Small product']
Out: array([1, 2, 3, 4, 5, 6])
dic['Small product'] + dic['Big product']
Out: array([11, 22, 33, 44, 55, 66])
```

We can now create a DataFrame directly from this dictionary.

```
df = pd.DataFrame.from_dict(dic)
Out:
   Small product  Big product
0              1           10
1              2           20
2              3           30
3              4           40
4              5           50
5              6           60
```

We now have a DataFrame where each product has its own column and where each row is a separate period.

Slicing DataFrames

There are many different techniques to slice a DataFrame to get the element or the part you want. This might be confusing for beginners, but you'll soon understand that each of these has its uses and advantages. Do not worry if you get confused or overwhelmed: you won't need to apply all of these right now.

- You can select a specific column by passing the name of this column directly to the DataFrame—either with `df['myColumn']`, or even more directly with `df.myColumn`.
- You can select a row based on its index value by simply typing `df[myIndexValue]`.
- If you want to select an element based on both its row and column, you can call the method `.loc` on the DataFrame and give it the index value and the column name you want. You can, for example, type `df.loc[myIndexValue,'myColumn']`.
- You can also use the same slicing method as for lists and arrays based on the position of the element you want to select. You then need to call the method `.iloc` to the DataFrame. Typically, to select the first element (top left corner), you can type `df.iloc[0,0]`.

As a recap, here are all the techniques you can use to select a column or a row:

```
df['myColumn']
df.myColumn
df[myIndexValue]
df.loc[myIndexValue,'myColumn']
df.iloc[0,0]
```

Exporting DataFrames

A DataFrame can easily be exported as either an Excel file or a CSV file.

```
df.to_excel('FileName.xlsx',index=False)
df.to_csv('FileName.csv',index=False)
```

The parameter `index` will indicate if you want to print the DataFrame index in the output file.

Other Libraries

We will also use other very well-known Python libraries. We used the usual import conventions for these libraries throughout the book. For the sake of clarity, we did not show the `import` lines over and over in each code extract.

SciPy is a library used for all kinds of scientific computation, optimization, as well as statistical computation (SciPy stands for **Sci**entific **Py**thon). The documentation is available on docs.scipy.org/doc/scipy/reference but it is unfortunately not always as clear as we would wish it to be. We mainly focus on the statistical tools and only import them as `stats`.

```
import scipy.stats as stats
```

To make our code shorter, we will import some functions directly from `scipy.stats` in our examples. We can import these as such:

```
from scipy.stats import normal
```

Matplotlib is the library used for plotting graphs. Unfortunately, matplotlib is not the most user-friendly library, and its documentation is only rarely helpful. If we want to make simple graphs, we will prefer using the `.plot()` method from pandas.

```
import matplotlib.pyplot as plt
```

Seaborn is another plotting library built on top of matplotlib. It is actually more user-friendly and provides some refreshing visualization compared to matplotlib. You can check the official website seaborn.pydata.org, as it provides clear and inspiring examples.

```
import seaborn as sns
```

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. *Comptes Rendus 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, pages 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.

https://doi.org/10.1515/9783110671124-030

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, pages 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* 10

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* 28

array A data structure defined in NumPy. It is a list or a matrix of numeric values. *See page* 266

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* 139

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* 41

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* 10

Boolean A Boolean is a value that is either `True` or `False`: 1 or 0. *See page* 203

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* 168

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* 45

classification Classification problems require you to classify data samples in different categories. *See page* 122

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* 30

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* 268

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* 139

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* 242

Euclidean distance The Euclidean distance between two points is the length of a straight line between these two points. *See page* 210

https://doi.org/10.1515/9783110671124-031

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* 192

feature A feature is a type of information that a model has at its disposal to make a prediction. *See page* 122

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* 70

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* 162

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* 211

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* 126

level The level is the average value around which the demand varies over time. *See page* 27

Mean Absolute Error $\text{MAE} = \frac{1}{n}\sum|e_t|$ *See page* 16

Mean Absolute Percentage Error $\text{MAPE} = \frac{1}{n}\sum\frac{|e_t|}{d_t}$ *See page* 14

Mean Square Error $\text{MSE} = \frac{1}{n}\sum e_t^2$ *See page* 17

naïve forecast The simplest forecast model: it always predicts the last available observation. *See page* 5

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* 266

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* 268

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* 60

regression Regression problems require an estimate of a numerical output based on various inputs. *See page* 122

Root Mean Square Error $\text{RMSE} = \sqrt{\frac{1}{n}\sum e_t^2}$ *See page* 17

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

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* 251

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* 209

test set A test set is a dataset kept aside to test our model against unseen data after its fitting (training). *See page* 37

training set A training set is a dataset used to fit (train) our model. *See page* 37

trend The trend is the average variation of the time series level between two consecutive periods. *See page* 41

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* 209

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* 131

weak model A weak model is a model that is slightly more accurate than luck. *See page* 167

Index