

EXPERT INSIGHT

# Machine Learning and Generative AI for Marketing

**Take your data-driven marketing strategies  
to the next level using Python**



**Yoon Hyup Hwang  
Nicholas C. Burtch**

**<packt>**

# **Machine Learning and Generative AI for Marketing**

**Take your data-driven marketing  
strategies to the next level using  
Python**

Yoon Hyup Hwang

Nicholas C. Burtch



# **Machine Learning and Generative AI for Marketing**

Copyright © 2024 Packt Publishing

*All rights reserved.* No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

**Senior Publishing Product Manager:** Tushar Gupta

**Acquisition Editor – Peer Reviews:** Swaroop Singh

**Project Editor:** Amisha Vathare

**Content Development Editor:** Shruti Menon

**Copy Editor:** Safis Editing

**Technical Editor:** Karan Sonawane

**Proofreader:** Safis Editing

**Indexer:** Manju Arasan

**Presentation Designer:** Ganesh Bhadwalkar

**Developer Relations Marketing Executive:** Meghal Patel

First published: August 2024

Production reference: 1230824

Published by Packt Publishing Ltd.

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB, UK.

ISBN: 978-1-83588-940-4

[www.packtpub.com](http://www.packtpub.com)

# Contributors

## About the authors

**Yoon Hyup Hwang** is a data science and engineering leader who has authored multiple books on applied ML and data science. He has over a decade of experience and expertise in delivering extremely high ROI data products and solutions that result in multi-million-dollar annual recurring revenue and savings across various industries that include finance, insurance, ads and marketing, manufacturing, and supply chain. He holds an MSE degree in Computer and Information Technology from the University of Pennsylvania and a BA in Economics from the University of Chicago.



*I would like to start by thanking my beautiful wife, Sunyoung. By sacrificing our weekends and family times so that I could work on writing this book over the past several months, she played a critical role in getting this book done and published.*

*To my family, I would like to thank you for your consistent love, support, and belief in me. I would not be where I am now without you.*

*I would also like to thank everyone on my publishing team at Packt who helped me uphold the highest quality standard.*

*Lastly, I would like to thank everyone reading this book. I hope you enjoy it and find it helpful!*

---

**Nicholas C. Burtch**, PhD, is a recognized data science researcher and thought leader with over ten years of experience in leading complex, data-driven projects. He has an extensive track record of deploying end-to-end ML solutions for understanding large-scale structured and unstructured data in industries ranging from finance to scientific research. Nick has published dozens of peer-reviewed research articles that have received thousands of citations and is a US patent holder. He received his PhD and MS in Chemical Engineering from the Georgia Institute of Technology and holds a BS in Chemical Engineering from the University of Michigan.



*I would like to express my deepest gratitude to the people closest to me who supported me throughout the creation of this book. Your encouragement and understanding have been invaluable. A special thanks to my loved ones for their patience and understanding throughout this journey.*

*I am also thankful to the Packt Publishing team for their guidance and commitment to excellence.*

*Finally, to the readers, I hope this book provides you with valuable insights that you can leverage in your own professional journey.*

---

# About the reviewers

**Dr. Na’im R. Tyson** is a seasoned data scientist with extensive experience in **natural language processing (NLP)** and **machine learning (ML)**. He holds a PhD in Linguistics from Ohio State University and an MS in Computational Linguistics from Georgetown University. His expertise spans developing ML models, implementing data ingestion pipelines, and refining NLP applications. Currently an Adjunct Assistant Professor at New York University, Dr. Tyson is passionate about education and holds several patents in language learning systems and keyword extraction methods, focusing on practical applications that drive meaningful results.

**Oluwole Fagbohun** currently serves as the Vice President of Engineering and Environmental Data at ChangeBlock, where he leads initiatives to combat climate change through innovative ML solutions. He holds a master’s degree in data science, has published several peer-reviewed papers, and is a published author. He is a frequent speaker at multiple conferences and universities and has mentored participants at various hackathons. As the founder of Readrly, an EdTech company, Oluwole leverages AI techniques to enhance children’s reading skills by providing captivating stories from around the globe.

Oluwole is passionate about playing chess and cherishes spending quality time with his family. Originally hailing from Nigeria, he now resides in London with his wife and three children.



*I would like to extend my heartfelt gratitude to my wife and children for their unwavering support, now and*

*always. Special thanks to my parents for their enduring love and encouragement.*

---

## Beta reader

**Konstantin Pikal** is a PhD Candidate at LUISS University in Rome, specializing in marketing technologies. With over a decade of experience as a marketing leader, he has developed an expertise in data-driven marketing, data analytics, and content marketing. His research focuses on bridging the gap between marketing research and practical applications, aiming to bring academic insights into real-world marketing strategies and to classrooms around the world.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)

# Contents

1. [Preface](#)
  1. [Who this book is for](#)
  2. [What this book covers](#)
  3. [To get the most out of this book](#)
  4. [Get in touch](#)
2. [The Evolution of Marketing in the AI Era and Preparing Your Toolkit](#)
  1. [The evolution of marketing with AI/ML](#)
    1. [The pre-AI era of marketing](#)
    2. [The advent of digital marketing](#)
    3. [The integration of AI/ML in marketing](#)
  2. [Core data science techniques in marketing](#)
    1. [Predictive analytics](#)
    2. [Understanding customer data](#)
    3. [A/B testing and experimentation](#)
    4. [Segmentation and targeting](#)
    5. [Customer lifetime value modeling](#)
    6. [Integrating AI for enhanced insights](#)
  3. [Setting up the Python environment for AI/ML projects](#)
    1. [Installing the Anaconda distribution](#)
    2. [Installing essential Python libraries for AI/ML](#)
    3. [Integrating your environment with JupyterLab](#)
    4. [Verifying your setup](#)
    5. [Navigating the Jupyter notebook](#)
  4. [Training your first ML model](#)
    1. [Step 1: Importing the necessary libraries](#)
    2. [Step 2: Loading the data](#)
    3. [Step 3: Exploratory data analysis](#)
    4. [Step 4: Preparing the data for ML](#)
    5. [Step 5: Training a model](#)
    6. [Step 6: Evaluating the model](#)
  5. [Summary](#)
3. [Decoding Marketing Performance with KPIs](#)
  1. [Understanding marketing KPIs](#)
    1. [Awareness stage](#)

2. [Engagement stage](#)
3. [Conversion stage](#)
4. [Retention stage](#)
5. [Loyalty stage](#)
2. [Computing and visualizing KPIs from data with Python](#)
  1. [Conversion rate](#)
    1. [Overall conversion rate](#)
    2. [Demographics and conversion rate](#)
    3. [Sales channel and conversion rate](#)
  2. [CLV](#)
    1. [Overall CLV](#)
    2. [Geolocation and CLV](#)
    3. [Product and CLV](#)
  3. [CPA](#)
    1. [Overall CPA](#)
    2. [Sales channel and CPA](#)
    3. [Promotions and CPA](#)
  4. [ROI](#)
  5. [Overall ROI](#)
    1. [ROI per sales channel](#)
    2. [ROI per promotion](#)
  6. [Correlations across KPIs](#)
3. [Tracking marketing performance with KPIs](#)
  1. [Choosing the right visualization for KPIs](#)
  2. [Ongoing KPI tracking](#)
4. [Summary](#)
4. [Unveiling the Dynamics of Marketing Success](#)
  1. [Why people engage in regression analysis](#)
    1. [Target variable](#)
    2. [Continuous variables](#)
    3. [Categorical variables](#)
      1. [Factorize](#)
      2. [Categorical](#)
      3. [Dummy variables](#)
      4. [Regression analysis](#)
    4. [Interaction variables](#)
  2. [Why people convert with decision tree interpretation](#)

1. [Target variable](#)
2. [Continuous versus categorical variable](#)
3. [Decision tree analysis](#)
3. [Why people churn with causal inference](#)
  1. [Causal effect estimation](#)
    1. [Causal model](#)
    2. [Estimation](#)
    3. [Refutation](#)
  2. [Graphical causal model](#)
    1. [Causal influence](#)
    2. [Anomaly attribution](#)
  4. [Summary](#)
5. [Harnessing Seasonality and Trends for Strategic Planning](#)
  1. [Time series analysis basics](#)
    1. [Basic time series trends](#)
    2. [Moving averages](#)
    3. [Autocorrelation](#)
    4. [Product trends](#)
  2. [Trend and seasonality decomposition](#)
    1. [Additive time series decomposition](#)
    2. [Multiplicative time series decomposition](#)
  3. [Time series forecasting models](#)
    1. [ARIMA](#)
      1. [Training the ARIMA model](#)
      2. [ARIMA model diagnostics](#)
      3. [Forecasting with the ARIMA model](#)
    2. [Prophet time-series modeling](#)
      1. [Training a Prophet model](#)
      2. [Forecasting with a Prophet model](#)
    3. [Other time-series models](#)
    4. [Summary](#)
6. [Enhancing Customer Insight with Sentiment Analysis](#)
  1. [Introduction to sentiment analysis in marketing](#)
    1. [The significance of sentiment analysis](#)
    2. [Advancements in AI and sentiment analysis](#)
  2. [Practical example: Twitter Airline Sentiment dataset](#)
  3. [Preparing data for sentiment analysis](#)

1. [Traditional NLP techniques for data preparation](#)
  1. [Cleaning text data](#)
  2. [Tokenization and stop word removal](#)
  3. [Lemmatization](#)
2. [Class imbalance](#)
  1. [Evaluating class balance](#)
  2. [Addressing class imbalance](#)
  3. [GenAI for data augmentation](#)
4. [Performing sentiment analysis](#)
  1. [Building your own ML model](#)
    1. [Feature engineering](#)
    2. [Model training](#)
    3. [Model evaluation](#)
    4. [Classification report](#)
  2. [Using pre-trained LLMs](#)
    1. [Implementing pre-trained models](#)
    2. [Evaluating model performance](#)
5. [Translating sentiment into actionable insights](#)
  1. [Creating your own dataset](#)
    1. [Collecting twitter data](#)
    2. [Collecting data from other platforms](#)
    3. [Performing NER on a dataset for a fictional retailer](#)
  2. [Understanding topics and themes](#)
    1. [Using word clouds](#)
    2. [Discovering latent topics with LDA](#)
    3. [Temporal trends: tracking the brand narrative](#)
    4. [Mapping sentiments using geospatial analysis](#)
6. [Summary](#)
7. [Leveraging Predictive Analytics and A/B Testing for Customer Engagement](#)
  1. [Predicting customer conversion with tree-based algorithms](#)
    1. [Tree-based machine learning algorithms](#)
    2. [Building random forest models](#)
      1. [Target and feature variables](#)
      2. [Training a random forest model](#)
      3. [Predicting and evaluating random forest model](#)
    3. [Gradient boosted decision tree \(GBDT\) modeling](#)

1. [Training GBDT model](#)
  2. [Predicting and Evaluating GBDT Model](#)
2. [Predicting customer conversion with deep learning algorithms](#)
  1. [Train and test sets for deep learning models](#)
  2. [Wide neural network modeling](#)
    1. [Training the wide neural network model](#)
    2. [Predicting and evaluating wide neural network model](#)
  3. [Deep neural network modeling](#)
    1. [Training deep neural network model](#)
    2. [Predicting and evaluating the deep neural network model](#)
  3. [Conducting A/B testing for optimal model choice](#)
    1. [Simulating A/B Testing](#)
    2. [Two-tailed T-test](#)
  4. [Summary](#)
8. [Personalized Product Recommendations](#)
  1. [Product analytics with market basket analysis](#)
    1. [Apriori algorithm – finding frequent itemsets](#)
    2. [Association rules](#)
  2. [Collaborative filtering](#)
    1. [User-based collaborative filtering](#)
      1. [Recommending by the most similar customer](#)
      2. [Recommending by the top products bought by similar customers](#)
    2. [Item-based collaborative filtering](#)
      1. [Recommending by the most similar items](#)
      2. [Recommending by the purchase history](#)
  3. [Other frequently used recommendation methods](#)
  4. [Summary](#)
9. [Segmenting Customers with Machine Learning](#)
  1. [One-time versus repeat customers](#)
    1. [The need to retain customers](#)
    2. [Analyzing the impact of retaining customers](#)
  2. [Customer segmentation with purchase behaviors](#)
    1. [K-means clustering](#)
      1. [Without log transformation](#)
      2. [With log transformation](#)

2. [Silhouette score](#)
  3. [Customer segmentation with product interests](#)
  4. [Summary](#)
10. [Creating Compelling Content with Zero-Shot Learning](#)
  1. [Fundamentals of generative AI](#)
    1. [A probabilistic approach](#)
    2. [Foundational models](#)
      1. [Generative adversarial networks](#)
      2. [Variational autoencoders](#)
      3. [Long short-term memory networks](#)
      4. [Transformers](#)
    3. [When GenAI is the right fit](#)
  2. [Introduction to pre-trained models and ZSL](#)
    1. [Contextual embeddings](#)
    2. [Semantic proximity](#)
    3. [Pre-trained models](#)
      1. [Model weights](#)
      2. [Model architecture](#)
      3. [Preprocessing steps](#)
    4. [Zero-shot learning](#)
      1. [Mechanics of learning and prediction](#)
      2. [Output parameters](#)
    3. [ZSL for marketing copy](#)
      1. [Preparing for ZSL in Python](#)
      2. [Creating an effective prompt](#)
        1. [Example 1: Product descriptions](#)
        2. [Example 2: Blog post titles](#)
        3. [Example 3: Social media captions](#)
      3. [Impact of parameter tuning](#)
      4. [Summary](#)
  11. [Enhancing Brand Presence with Few-Shot Learning and Transfer Learning](#)
    1. [Navigating FSL](#)
      1. [Understanding FSL through meta-learning](#)
      2. [Implementing model-agnostic meta-learning in marketing](#)
      3. [Overcoming challenges in FSL](#)
    2. [Navigating transfer learning](#)

1. [The mechanics of transfer learning](#)
2. [Transfer learning using Keras](#)
3. [Transfer learning using API services](#)
4. [Overcoming challenges in transfer learning](#)
3. [Applying FSL to improve brand consistency](#)
  1. [Benchmarking with ZSL and FSL](#)
  2. [Developing an email marketing campaign](#)
    1. [Step 1: Initial email creation](#)
    2. [Step 2: Collect and analyze initial metrics and responses](#)
    3. [Step 3: Iterative refinement](#)
    4. [Step 4: Continued feedback integration](#)
  4. [Summary](#)
12. [Micro-Targeting with Retrieval-Augmented Generation](#)
  1. [Introduction to RAG for precision marketing](#)
    1. [How RAG works](#)
    2. [Mathematical model of RAG retrieval](#)
    3. [The importance of data in RAG](#)
      1. [Data freshness](#)
      2. [Data specificity](#)
    4. [Understanding the retrieval index](#)
      1. [Indexing strategies](#)
      2. [Data curation and updating](#)
    5. [RAG implementation challenges](#)
    6. [Applications of RAG in marketing](#)
  2. [Building a knowledge retrieval system for marketing with LangChain](#)
    1. [Introduction to LangChain](#)
    2. [Understanding the external dataset](#)
    3. [Designing the retrieval model with LangChain](#)
      1. [Install and connect to Elasticsearch](#)
      2. [Indexing data in Elasticsearch](#)
      3. [Data ingestion into Elasticsearch](#)
      4. [Integrating with LangChain using GPT](#)
  3. [Implementing RAG for micro-targeting based on customer data](#)
    1. [Determining the campaign strategy](#)
      1. [Message timing](#)

- 2. [Choosing the brand](#)
  - 2. [Using LangChain for micro-targeting](#)
    - 1. [Case study 1: Targeted product discounts](#)
    - 2. [Case study 2: Product upselling](#)
    - 3. [Case study 3: Real-time content customization for bpw.style](#)
  - 4. [Summary](#)
- 13. [The Future Landscape of AI and ML in Marketing](#)
  - 1. [Consolidating key AI and ML concepts](#)
  - 2. [Next-generation AI technologies in marketing](#)
    - 1. [From RAG to ReAct](#)
      - 1. [How ReAct works](#)
      - 2. [Applications in marketing](#)
    - 2. [Model architecture advances](#)
      - 1. [Diffusion models](#)
      - 2. [Neural radiance fields](#)
      - 3. [Capsule networks](#)
    - 3. [Multi-modal GenAI](#)
      - 1. [Technical foundations](#)
      - 2. [Multi-modal applications](#)
  - 3. [AR and VR as emerging digital platforms](#)
    - 1. [The role of ML in AR for marketing](#)
      - 1. [Personalized AR experiences](#)
      - 2. [Geolocation-based AR campaigns](#)
      - 3. [Seamless product integration](#)
    - 2. [The role of ML in VR for marketing](#)
      - 1. [Immersive storytelling and brand experiences](#)
      - 2. [Behavioral data insights](#)
      - 3. [Virtual storefronts and events](#)
  - 4. [Summary](#)
- 14. [Ethics and Governance in AI-Enabled Marketing](#)
  - 1. [Ethical considerations in AI/ML for marketing](#)
    - 1. [Model transparency and explainability](#)
    - 2. [Model explainability tools](#)
    - 3. [Bias mitigation](#)
      - 1. [Addressing training data bias](#)
      - 2. [Algorithmic fairness techniques](#)

3. [Diversity in model evaluation data](#)
4. [Grounding for LLMs](#)
5. [Chain-of-thought reasoning](#)
4. [Balancing privacy with personalization](#)
  1. [Federated learning](#)
  2. [Differential privacy and anonymization](#)
2. [Governance and regulatory compliance](#)
  1. [Intellectual property protection](#)
  2. [Data and model licensing and attribution](#)
    1. [Internal data ownership and security](#)
    2. [Data management and collection practices](#)
  3. [Ethical governance frameworks](#)
    1. [Internal AI ethics committees](#)
    2. [AI policy development and reporting](#)
    3. [Continuous training and awareness](#)
  4. [Regulatory compliance](#)
    1. [General Data Protection Regulation \(GDPR\)](#)
    2. [California Consumer Privacy Act \(CCPA\)](#)
    3. [Global regulations and standards](#)
    4. [Industry-specific guidelines](#)
  3. [Summary](#)
15. [Other Books You May Enjoy](#)
16. [Index](#)

# Preface

In the dynamic world of marketing, the integration of **artificial intelligence (AI)** and **machine learning (ML)** is no longer just an advantage—it's a necessity. AI/ML integration in marketing strategies represents a paradigm shift towards more data-driven, efficient, and impactful marketing practices. With the emergence of data science, businesses can now gain an in-depth understanding of the mechanics of their marketing successes and failures, understanding customer behaviors and interactions with unprecedented clarity. Moreover, the rise of **generative AI (GenAI)** is creating new opportunities for the facile creation of highly personalized, engaging content that resonates with the target audience on a whole new level. This book provides a comprehensive toolkit for harnessing the power of ML and GenAI to craft marketing strategies that not only predict customer behaviors but also captivate and convert, leading to improved cost per acquisition, boosted conversion rates, and increased net sales.

Starting with the basics of Python for data analysis and progressing to more sophisticated ML and GenAI models, this book is your comprehensive guide to understanding and applying AI to enhance marketing strategies. We will use hands-on examples to show you how you can harness the capabilities of AI to unlock deep insights into customer behaviors, craft personalized marketing messages, and drive significant business growth. We'll also explore the ethical implications of AI, which will help you ensure that your marketing strategies are not only effective but also responsible and compliant with current standards.

By the conclusion of this book, you will have gained a robust understanding of various data science, ML, and GenAI techniques tailored specifically for marketing. Armed with this knowledge, you will be equipped to design, launch, and manage marketing campaigns that are not just successful but also cutting-edge, ensuring your business remains at the forefront of the digital marketing revolution.

# Who this book is for

This book targets a diverse group of professionals at the intersection of technology and marketing, including:

- Marketing professionals at any level seeking to leverage AI/ML for data-driven decision-making and enhance their customer engagement strategies
- Data scientists and analysts in the marketing domain looking to apply advanced AI/ML techniques to solve real-world marketing challenges
- ML engineers and software developers aiming to build or integrate AI-driven tools and applications for marketing purposes
- Business leaders and entrepreneurs who must understand the impact of AI on marketing to drive innovation and maintain their competitive advantage in today's landscape

Each reader is presumed to have a foundational proficiency in Python programming and a basic to intermediate grasp of ML principles and data science methodologies. They are likely already in or aspire to be in roles where the application of AI/ML directly influences marketing outcomes and business strategies.

# What this book covers

*Chapter 1, The Evolution of Marketing in the AI Era and Preparing Your Toolkit*, explores the evolution of marketing in the AI era, core AI/ML techniques shaping marketing's future, and setting up a Python environment for marketing projects.

*Chapter 2, Decoding Marketing Performance with KPIs*, explains that every data-driven or AI/ML-driven marketing strategy starts with optimization and enhancement goals tied to **key performance metrics (KPIs)**. Understanding and analyzing these KPIs is critical for evaluating the effectiveness of marketing campaigns and strategies. This chapter discusses how to identify and compute crucial marketing KPIs, leveraging them for explanatory data analysis, and using these insights to drive marketing decisions.

*Chapter 3, Unveiling the Dynamics of Marketing Success*, investigates the underlying dynamics contributing to marketing success. By utilizing data science and AI/ML techniques, you can gain deep insights into the factors driving successful marketing campaigns. The topics covered include analyzing customer interactions, identifying successful marketing patterns, and optimizing strategies based on data-driven insights.

*Chapter 4, Harnessing Seasonality and Trends for Strategic Planning*, discusses seasonality and trends, which play a vital role in shaping marketing strategies. This chapter focuses on leveraging AI/ML to identify and capitalize on predictable fluctuations in consumer behavior and market dynamics. By understanding these patterns, you can strategically plan and

execute campaigns that align with consumer interests and market conditions.

*Chapter 5, Enhancing Customer Insight with Sentiment Analysis*, discusses customer sentiment, which is a valuable indicator of brand perception and customer satisfaction. This chapter explores how sentiment analysis, powered by AI/ML, can enhance customer insights. You will learn how to analyze customer feedback, reviews, and social media interactions to tailor your strategies to improve customer experience and engagement.

*Chapter 6, Leveraging Predictive Analytics and A/B Testing for Customer Engagement*, covers predictive analytics and A/B testing, which are essential tools for enhancing customer engagement. This chapter explains how to use AI/ML models to predict customer behavior and engagement levels. It also covers the design and execution of effective A/B tests to optimize marketing decisions and improve user experiences.

*Chapter 7, Personalized Product Recommendations*, explores personalized product recommendations, which significantly enhance the likelihood of purchase by aligning offerings with customer preferences and behaviors. This chapter covers various techniques for generating personalized recommendations, including market basket analysis, collaborative filtering, and other recommendation methods.

*Chapter 8, Segmenting Customers with Machine Learning*, discusses customer segmentation, which is a critical aspect of targeted marketing. This chapter discusses using ML to segment customers based on various factors, such as purchase behavior, demographics, and interests. It covers techniques like K-means clustering and leveraging **large language models (LLMs)** for deeper customer segmentation insights.

*Chapter 9, Creating Compelling Content with Zero-Shot Learning,* introduces **zero-shot learning (ZSL)**, a technique that allows AI models to generate relevant marketing content without prior direct examples. ZSL enhances creativity and speed, allowing you to produce content for new categories and contexts using learned patterns and knowledge extrapolation. You will find practical examples and best practices for integrating ZSL into marketing strategies, showcasing how to create dynamic content for product descriptions, blog posts, and social media.

*Chapter 10, Enhancing Brand Presence with Few-Shot Learning and Transfer Learning,* covers **few-shot learning (FSL)**, which is a method that's used to adapt AI models to new tasks using only a small number of labeled examples.

This chapter covers the principles of FSL, including meta-learning, which enable models to generalize effectively to new scenarios. The practical examples demonstrate how you can apply FSL to marketing campaigns, emphasizing its importance in capturing nuanced aspects of brand ethos and refining marketing strategies based on customer feedback.

*Chapter 11, Micro-Targeting with Retrieval-Augmented Generation,* this chapter explores **retrieval-augmented generation (RAG)** as a tool for precision marketing, combining GenAI with real-time information retrieval. This hybrid approach allows the creation of highly personalized content by accessing and incorporating current data during the generation process. This chapter details RAG's operational framework and its application in micro-targeting, showcasing how it can improve consumer engagement and conversion by providing contextually appropriate and accurate content.

*Chapter 12, The Future Landscape of AI and ML in Marketing,* consolidates key AI and ML concepts from previous chapters and discusses their future

applications in marketing. It covers emerging technologies such as multi-modal GenAI, advanced model architectures, and the integration of AI with augmented and virtual reality. These advancements promise to create more dynamic, responsive, and personalized marketing strategies. This chapter provides insights into how these technologies will shape the future of marketing, enabling more immersive consumer experiences and innovative marketing practices.

*Chapter 13, Ethics and Governance in AI-Enabled Marketing*, addresses the ethical considerations and governance challenges associated with AI technologies in marketing. It explores data privacy, algorithmic bias, and the need for transparency, emphasizing their impact on consumer trust and brand integrity. The chapter also covers major regulatory frameworks such as GDPR and CCPA, providing strategies for responsible AI deployment, model transparency, and compliance. Finally, it discusses practical guidelines for mitigating bias, ensuring data privacy, and establishing robust governance structures to promote ethical AI use in marketing.

## To get the most out of this book

To maximize the benefits of this book, it is recommended that you have basic familiarity with Python. However, the book is structured to accommodate a wide range of expertise levels, from beginners to advanced practitioners. Here are a few tips to help you get the most out of this book:

- 1. Engage with the code examples:** Practical examples are provided throughout the book to illustrate key concepts. Actively engaging with

these examples by running the code and experimenting with variations will deepen your understanding and enhance your skills.

2. **Apply the knowledge:** Try to apply the techniques and strategies discussed in the book to your own marketing projects. This practical application will help reinforce the concepts and demonstrate their real-world relevance.
3. **Stay updated:** The fields of AI and ML are rapidly evolving. Staying up to date with the latest advancements, tools, and best practices through continuous learning and professional development will ensure that you remain at the forefront of this exciting domain.

The line graphs in this book may look slightly different from those generated by the provided code. This is because the graphs have been optimized to ensure clarity and ease of understanding; feel free to play around with visualization to achieve the type of outcome you need!

## Download the example code files

The code bundle for the book is hosted on GitHub at <https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing>. We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

## Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here:

<https://packt.link/gbp/9781835889404>.

## Conventions used

There are a number of text conventions used throughout this book.

**CodeInText**: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. For example: “Mount the downloaded `webStorm-10*.dmg` disk image file as another disk in your system.”

**Bold**: Indicates a new term, an important word, or words that you see on the screen. For instance, words in menus or dialog boxes appear in the text like this. For example: “Select **System info** from the **Administration** panel.”



Important notes or suggestions appear like this.



Tips and tricks appear like this.

## Get in touch

Feedback from our readers is always welcome.

**General feedback:** Email `feedback@packtpub.com` and mention the book’s title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

**Errata:** Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you reported this to us. Please visit

<http://www.packtpub.com/submit-errata>, click **Submit Errata**, and fill in the form.

**Piracy:** If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the material.

**If you are interested in becoming an author:** If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit <http://authors.packtpub.com>.

# Share your thoughts

Once you've read *Machine Learning and Generative AI for Marketing*, we'd love to hear your thoughts! Please [click here to go straight to the Amazon review page](#) for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

# Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily.

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below:



<https://packt.link/free-ebook/9781835889404>

2. Submit your proof of purchase.
3. That's it! We'll send your free PDF and other benefits to your email directly.

[OceanofPDF.com](http://OceanofPDF.com)

# 1

## The Evolution of Marketing in the AI Era and Preparing Your Toolkit

In the age of information, the marketing landscape has been fundamentally transformed by the emergence of data science and **artificial intelligence/machine learning (AI/ML)** technologies. This transformation has reshaped the tools and methodologies marketers employ and redefined how brands engage with their customers. It has resulted in a shift in marketing strategies from targeted advertising to more personalized customer experiences, as well as the usage of more data-driven, efficient, and impactful marketing practices. This book aims to give you the knowledge and skills that you will need to navigate and leverage this transformation. We will provide you with a comprehensive toolkit that will allow you to master data-driven marketing strategies, enhance customer engagement, and stay competitive in the evolving digital landscape.

In this chapter, we begin our journey through the evolution of marketing in the AI era, exploring the core data science and AI/ML techniques that are shaping the future of marketing. We will take a look at the history of marketing technologies, the rise of digital marketing, and the key role that AI/ML plays in revolutionizing the field.

This chapter will also guide you through setting up your Python environment, allowing you to implement AI/ML in your marketing projects, and will introduce you to the essential tools and libraries that will be used throughout this book. We will then walk through an end-to-end example demonstrating some basic best practices when developing an ML model. This foundational knowledge will give you insight into how you can navigate the complex yet exciting field of AI/ML-driven marketing, opening up new opportunities for innovation and growth in your marketing strategies.

In particular, we will cover the following topics in this chapter:

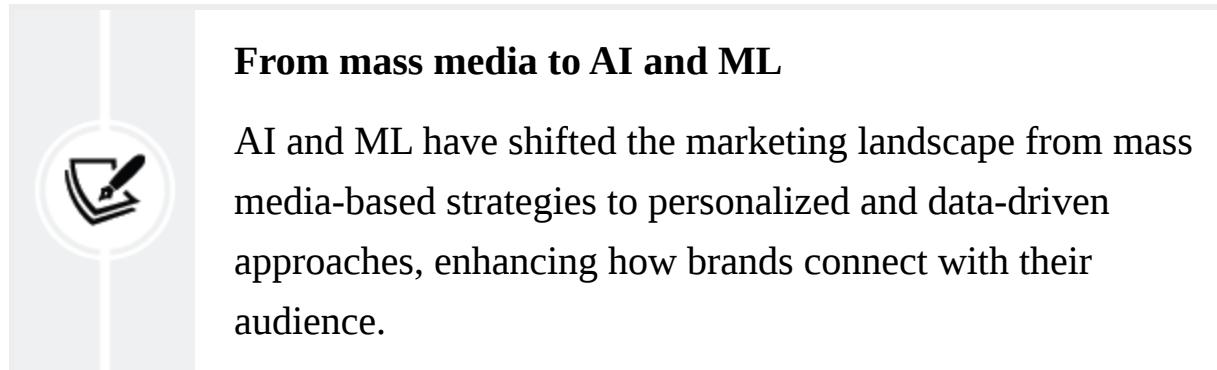
- The evolution of marketing with AI/ML
- Core data science techniques in marketing
- Setting up the Python environment for AI/ML projects

This foundational chapter sets the stage for the advanced discussions that follow, ensuring that you have the requisite knowledge and tools to fully engage with the AI/ML methodologies that will be explored in subsequent chapters.

## **The evolution of marketing with AI/ML**

The marketing landscape has undergone a radical transformation in the last few decades, significantly influenced by the emergence and integration of AI and ML technologies. This evolution of the domain has redefined the tools and strategies used and reshaped the way brands connect with their audiences. The transition from traditional marketing methods to data-driven

and AI-enhanced approaches marks a major shift in the marketing domain, with more personalized, efficient, and engaging marketing practices.



The following timeline shows key milestones in marketing, from traditional methods through to the digital era and, finally, the integration of AI/ML technologies:

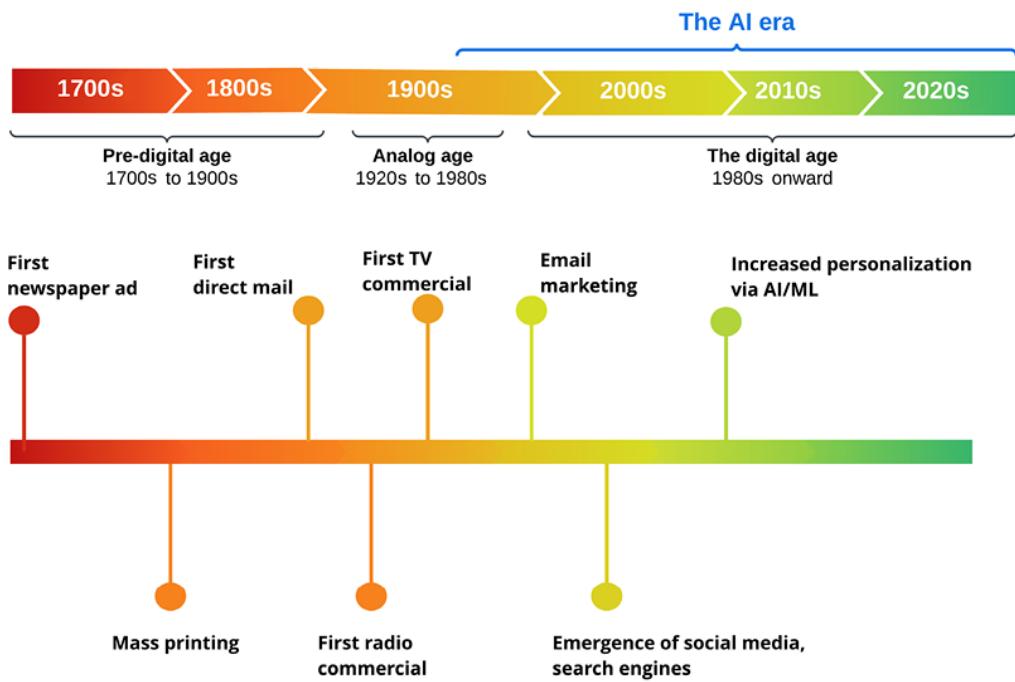


Figure 1.1: A brief history of marketing

Let's look at the characteristics of these time periods in the following sections.

# The pre-AI era of marketing

To appreciate the impact of AI and ML on marketing, it's important to look back at the pre-AI era. Back then, marketing strategies were largely dictated by broad demographic insights, with not much personalization and a heavy reliance on mass media. Marketers were forced to cast wide nets to catch potential customers, and the primary channels for customer engagement were print, television, and radio advertisements.

Traditional marketing also included techniques like direct mailing and coupons. This is evident from Claude Hopkins' scientific advertising principles proposed in the early 20<sup>th</sup> century, which served a purpose similar to today's targeted digital ads. However, while effective to a certain extent, these strategies lacked the precision and personalization that modern consumers have come to expect today. To illustrate the marketing landscape before the digital era, let's look at the following examples of traditional marketing materials, including flyers and billboards, highlighting the dependence on physical media for advertising:



*Figure 1.2: Examples of traditional marketing materials such as flyers, billboards, and print advertisements*

## The advent of digital marketing

In the 1990s, the digital revolution brought about the first major shift, with the introduction of tools and platforms that allowed marketers to target audiences more directly and measure the impact of their campaigns more accurately. The emergence of email marketing, social media, and search engines opened new avenues for customer interaction based on data about customer preferences, behaviors, and feedback.



*Figure 1.3: Digital marketing revolutionized advertising through targeted campaigns using social media, search engines, and email marketing*

By the early 2000s, however, the sheer volume and complexity of this data soon outstripped the human capacity to analyze and utilize it effectively. This complexity was due to the vast amounts of data generated from various digital channels, including diverse formats, the high velocity of data creation, and the need for real-time processing. This data landscape highlighted the limitations of older digital techniques, such as basic email

marketing and static web ads, which could not be easily processed for actionable insights. This set the stage for the integration of AI and ML in marketing.



### The impact of digital marketing

The digital era brought about targeted advertising and measurable campaigns, thanks to emerging tools like search engines, social media, and email marketing, which paved the way for AI and ML applications in marketing.

## The integration of AI/ML in marketing

The AI era can be traced back to the mid-20<sup>th</sup> century when AI first became a buzzword, but the actual onset of technology use happened later. In the marketing domain, AI and ML technologies have been used to leverage big data for insights and personalization since the early 2010s. The emergence of these technologies has marked a transformative era in marketing and enables unprecedented levels of precision and efficiency.

Marketers can now move beyond basic demographic targeting to create highly personalized experiences and predict customer needs and behaviors with remarkable accuracy. As we discuss this most critical era, we will explore how these technologies are revolutionizing every aspect of marketing, from customer segmentation to real-time analytics and personalized content creation.

The following are some of the key aspects of integrating AI/ML in marketing:

- **Predictive analytics and customer insights:** In **Part 2** of this book, we explore how AI and ML empower marketers with predictive analytics to forecast future customer behaviors based on historical data. This predictive capability enables proactive marketing strategies, from anticipating customer needs and preferences to identifying potential churn risks. Additionally, we will discuss explanatory techniques such as clustering, which help in understanding customer segments and behaviors without necessarily predicting future outcomes. Clustering can reveal natural groupings within customer data. These clusters help in identifying distinct market segments and inform more targeted marketing strategies. Marketers can leverage these insights to achieve the following:
  - **Make informed decisions:** Predictive analytics helps identify which marketing strategies are likely to succeed. For example, a retailer can forecast which products are likely to be popular during the holiday season, enabling them to optimize inventory and marketing campaigns accordingly.
  - **Optimize marketing efforts:** Predictive analytics can help marketers allocate resources more efficiently. For instance, a company can use these insights to determine the best times to launch promotions or the most effective channels for reaching their target audience, thus maximizing ROI.
  - **Foster stronger customer relationships:** Marketers can create more personalized experiences by anticipating customer needs and preferences. For example, a streaming service can recommend shows and movies based on a user's past viewing habits, which increases user satisfaction and loyalty.

- **Personalization at scale:** One of the most significant contributions of AI/ML to marketing is the ability to personalize marketing efforts at scale. In **Part 3** of this book, we introduce how we can segment audiences with incredible precision, which we can use to tailor messages, offers, and content to match the interests and needs of each customer. This level of personalization results in more effective marketing strategies that enhance the customer experience and increase both engagement and conversion rates.

For example, an online bookstore can recommend different books to different users based on their past purchases and browsing history. A customer who frequently buys mystery novels might receive personalized recommendations for new releases in that genre, while another customer who prefers self-help books might see curated lists of the latest self-help titles.

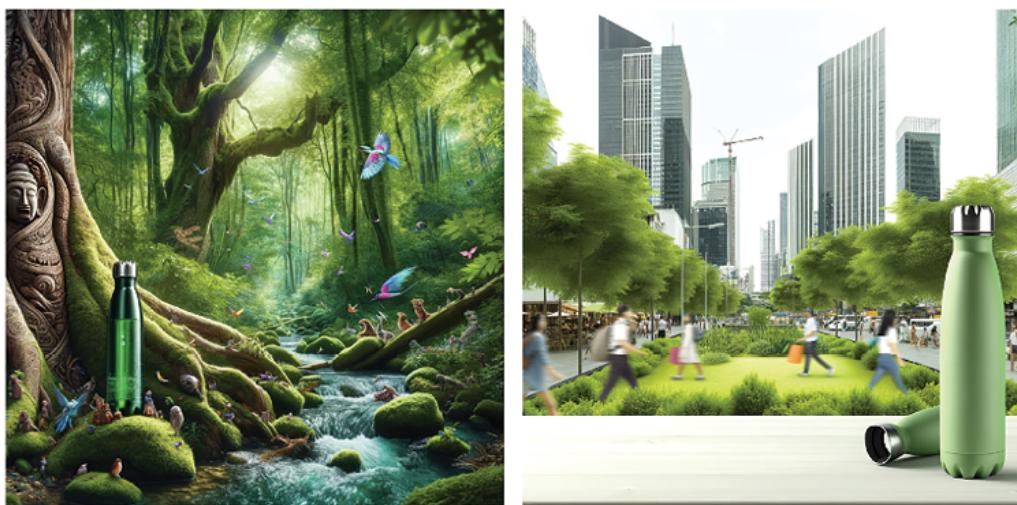


### Why does personalization matter?

Personalization at scale is a significant achievement of AI/ML in marketing, allowing for the customization of messages, offers, and content to individual customer preferences, thus driving engagement and conversion rates.

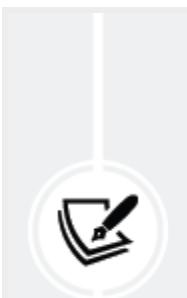
- **The role of genAI in content creation and analytics:** Looking ahead, **Part 4** of this book explores the role of **generative AI (GenAI)** in marketing and how it is likely to further revolutionize content creation, customer engagement, and analytics. GenAI models are capable of generating text, images, and even video content, and offer new possibilities for dynamic and personalized marketing materials. Take,

for example, an online retailer creating digital ads for a water bottle, targeted for distinct personas using GenAI. For the environmentally conscious consumer interested in sustainable living, an image set in nature with a relevant product placement can be presented, as shown on the left in the following figure. Conversely, for the urban enthusiast, the image on the right places the same product within a bustling city scene.



*Figure 1.4: GenAI tailors ads for distinct personas, using nature for eco-conscious consumers and cityscapes for urban enthusiasts*

Additionally, the advanced analytics powered by AI and ML continue to refine customer segmentation, campaign optimization, and ROI measurement, ensuring that marketing strategies are both effective and efficient.



### **Importance of GenAI**

GenAI models are at the forefront of the next evolution in marketing, enabling the creation of personalized text,

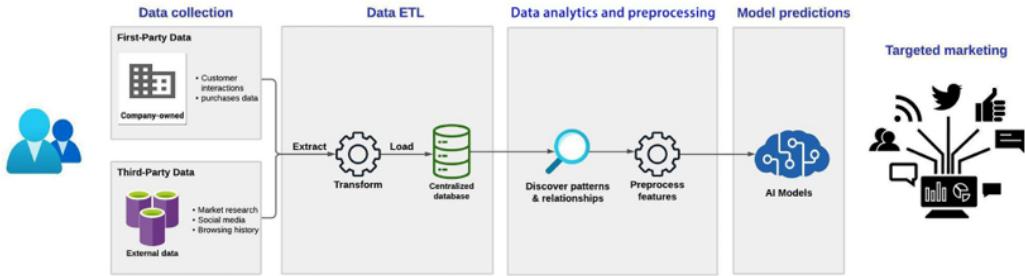
images, and video content, and offering advanced analytics for deeper customer insights.

As we stand on the brink of this new era, the integration of AI and ML in marketing strategies is no longer a luxury but a necessity for brands aiming to remain competitive and resonate with their audiences. Our journey through the evolution of marketing with AI/ML highlights not just a technological revolution but a fundamental shift in how brands connect with their customers, offering more personalized, engaging, and meaningful interactions.

## Core data science techniques in marketing

Applying data science techniques to the dynamic realm of marketing is crucial for understanding complex customer data, optimizing marketing strategies, and enhancing customer experiences. We explore the core data science methodologies shaping modern marketing efforts extensively throughout this book, with *Chapter 2* focusing on decoding marketing performance with **key performance indicators (KPIs)**. This will provide you with a strong foundation for understanding how these techniques drive insights.

When a deep understanding of how customer data and predictive analytics are combined, the result can be a highly effective customer strategy. The following diagram illustrates the general end-to-end process that is involved, starting with data collection and concluding with a targeted marketing campaign:



*Figure 1.5: General process in data-driven marketing, from data collection to targeted campaigns using data science techniques*

In the following subsections, we will go deeper into specific data science techniques such as predictive analytics, customer segmentation, and A/B testing, along with insights and practical examples to illustrate their application in modern marketing.

## Predictive analytics

Predictive analytics stands as a cornerstone of data science in marketing. In *Chapter 3*, we discuss how we can employ statistical models and ML algorithms to forecast future customer behaviors based on historical data. The applications of this range from predicting which customers are most likely to make a purchase to identifying those at risk of churn. By leveraging predictive analytics, marketers can make informed decisions about where to allocate resources, how to design their campaigns, and when to engage with customers. The ability to harness seasonality and trends, a focus of *Chapter 4*, also plays an important role in tailoring marketing strategies to capitalize on predictable fluctuations in consumer behavior and market dynamics.

**Predictive analytics forecasts customer behaviors**



Predictive models improve the accuracy of forecasts by analyzing historical data and identifying patterns that can reflect future behavior.

Examples of different predictive models and their applications in marketing include:

- **Email opening likelihood:** Forecasts the probability of customers opening marketing emails to optimize campaign engagement. For example, predicting that customers are more likely to open emails sent on weekdays.
- **Purchase/conversion likelihood:** Predicts which customers are more likely to purchase, aiding in targeted marketing for increased conversions. For instance, identifying customers who have shown interest in similar products.
- **Customer churn prediction:** Identifies customers at risk of churning, enabling targeted retention strategies to minimize turnover. For example, detecting users who haven't interacted with the service in a certain period.
- **Website login frequency modeling:** Analyzes login patterns to inform engagement strategies and tailor re-engagement campaigns. For example, recognizing that users who log in daily are more likely to engage with new features.
- **Contact frequency modeling:** Determines optimal outreach frequency to maintain engagement without overwhelming customers. For example, finding that weekly updates keep customers engaged without causing fatigue.

# Understanding customer data

Understanding customer data is crucial for any data-driven marketing strategy. This includes different types of information, such as the following:

- **Demographic details:** Understanding the age, gender, location, and other demographic factors helps marketers tailor their messages to specific audiences. For example, promoting winter clothing to customers in colder regions on a seasonal basis.
- **Purchasing history:** Analyzing past purchases allows marketers to recommend similar or complementary products, increasing the likelihood of future sales. For instance, suggesting accessories to customers who recently bought a smartphone.
- **Online behavior patterns:** Tracking website navigation and interaction data helps marketers optimize website layout and content to enhance user experience and engagement. For example, identifying frequent drop-off points in the purchase process and simplifying the checkout procedure to reduce cart abandonment rates.
- **Social media interactions:** Monitoring social media activities and sentiments enables marketers to engage with customers in real time and address their needs or concerns. For instance, leveraging customer feedback on social platforms to improve products and services and creating campaigns that resonate with the audience based on trending discussions.

The challenge for marketers is not just in collecting this data but in analyzing and interpreting it to glean actionable insights. We will discuss this further in *Chapter 5*, where we explore how sentiment analysis can enhance customer insights, offering a better understanding of customer

emotions and opinions that can inform targeted marketing efforts. Later, *Chapter 7* introduces personalized product recommendations, demonstrating the importance of analyzing and interpreting customer data to gain actionable insights.



### From data to insights

Data mining and analysis can transform raw and otherwise overwhelming data into a goldmine of actionable customer insights.

## A/B testing and experimentation

A/B testing is an essential data science technique in the marketer's toolkit. It involves comparing two versions of a marketing asset, such as a web page, email, or ad, to determine which one performs better on a given metric, such as click-through rate or conversion. One version (A) acts as the control, while the other version (B) is a variant of A, incorporating changes meant to improve performance. As covered in *Chapter 6*, A/B testing is rooted in statistical hypothesis testing, providing a data-driven approach to optimizing marketing materials and strategies based on actual customer behavior rather than intuition. The process is succinctly depicted in the following flowchart, which outlines the steps from A/B test creation through to the final selection of the more effective version:

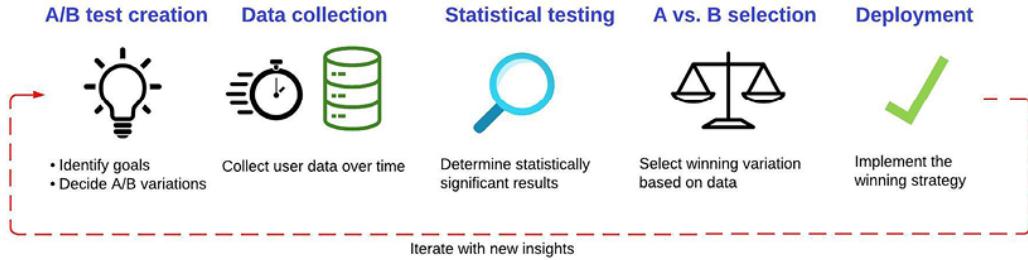


Figure 1.6: Summary of the A/B testing process

## Segmentation and targeting

One of the fundamental applications of data science in marketing is **segmentation**. This process involves dividing a broad customer base into smaller subgroups based on shared characteristics or behaviors. Techniques such as cluster analysis enable marketers to identify distinct segments within their audience, allowing more targeted and personalized marketing efforts. By understanding the specific needs and preferences of each segment, marketers can tailor their messages, offers, and content to resonate more deeply with their target audience. The methodologies and benefits of segmentation are explored further in *Chapter 8*, in which we emphasize how data science enables marketers to identify distinct segments within their audience.



**Segmentation for tailored content**

Segmentation is crucial for ensuring that the intended message is being conveyed to the ideal target group.

## Customer lifetime value modeling

**Customer lifetime value (CLV)** provides insights into how much revenue a customer is expected to generate over their relationship with the brand. CLV modeling is a key aspect of maximizing marketing effort profitability, as we will discuss in *Chapter 2*. Data science techniques allow marketers to model and calculate the CLV for individual customers or segments, providing insights that are invaluable in prioritizing marketing efforts, optimizing customer acquisition costs, and fostering long-term relationships with high-value customers.



**Value of CLV in the long term**

CLV modeling takes a long-term view of customer relationships, allowing better outcomes around customer acquisition and retention.

## Integrating AI for enhanced insights

The integration of AI with data science techniques marks a significant leap forward in marketing analytics. *Chapters 9 to 11* focus on creating compelling content with zero-shot learning, enhancing brand presence with few-shot learning, and micro-targeting with retrieval-augmented generation, illustrating how AI algorithms analyze complex datasets at scale to identify patterns and gain new insights. *Chapter 12* further explores the future landscape of AI/ML in marketing, highlighting upcoming trends and the potential for new AI innovations to shape marketing strategies.

While implementing these powerful AI/ML capabilities, we must also navigate the ethical considerations and governance frameworks to ensure

that we are following responsible marketing practices, which will be the focus of *Chapter 13*.



### Key GenAI concepts

The following GenAI concepts will be covered in detail in *Part 4* of the book:

- **Zero-shot learning:** Predicting new classes without prior training examples
- **Few-shot learning and transfer learning:** Making predictions with few training examples
- **Retrieval-augmented generation (RAG):** Combining data retrieval with response generation for more accurate outputs

## Setting up the Python environment for AI/ML projects

For marketers who are relatively new to the world of AI/ML, setting up a robust Python environment is the first technical step to unlocking the potential of data science in marketing strategies. Python, renowned for its simplicity and powerful libraries, serves as the backbone for most AI/ML projects in today's technology companies. This section will guide you through setting up a Python environment tailored for AI/ML in marketing projects, ensuring that you have the necessary tools and libraries at your disposal. Python's appeal lies in its vast ecosystem of libraries designed for

data analysis, ML, **natural language processing (NLP)**, and more. For AI/ML projects, using an Anaconda distribution is highly recommended due to its simplicity in managing packages and environments. Let's look at how you can get started.

## Installing the Anaconda distribution

Visit the Anaconda website at

<https://www.anaconda.com/download> and download the installer for Python 3. Once the Anaconda distribution has been downloaded, you can follow the installation instructions and create a new environment. This can be done by opening a terminal window and typing:

```
conda create --name ai_marketing python=3.8
conda activate ai_marketing
```

After running these commands, you should have a new Python environment named `ai_marketing` with Python 3.8 installed and activated. This environment will help you manage dependencies for your AI/ML projects effectively, isolating them from other projects on your system.

## Installing essential Python libraries for AI/ML

With your environment set up, the next step is to install the key Python libraries that will power your AI/ML marketing projects. The following are some of the essential Python libraries for AI/ML projects, grouped by their

general functionality. You can install these libraries using Anaconda via the following terminal commands:

- NumPy and pandas for data manipulation:

```
conda install numpy pandas
```

- Matplotlib and Seaborn for data visualization:

```
conda install matplotlib seaborn
```

- scikit-learn for ML:

```
conda install scikit-learn
```

- TensorFlow and Keras for deep learning:

```
conda install tensorflow keras
```

- NLTK and spaCy for NLP:

```
conda install nltk spacy
```

- Hugging Face's Transformers for advanced NLP and generative AI:

```
pip install transformers
```

## Integrating your environment with JupyterLab

JupyterLab offers an interactive coding environment ideal for data exploration, visualization, and presenting step-by-step AI/ML analyses. This can be installed via the following terminal command:

```
conda install jupyterlab
```

To launch JupyterLab, where you can create and manage your notebooks, type the following in the terminal:

```
jupyter lab
```

You should now see a screen similar to what is shown below in a newly launched browser window:

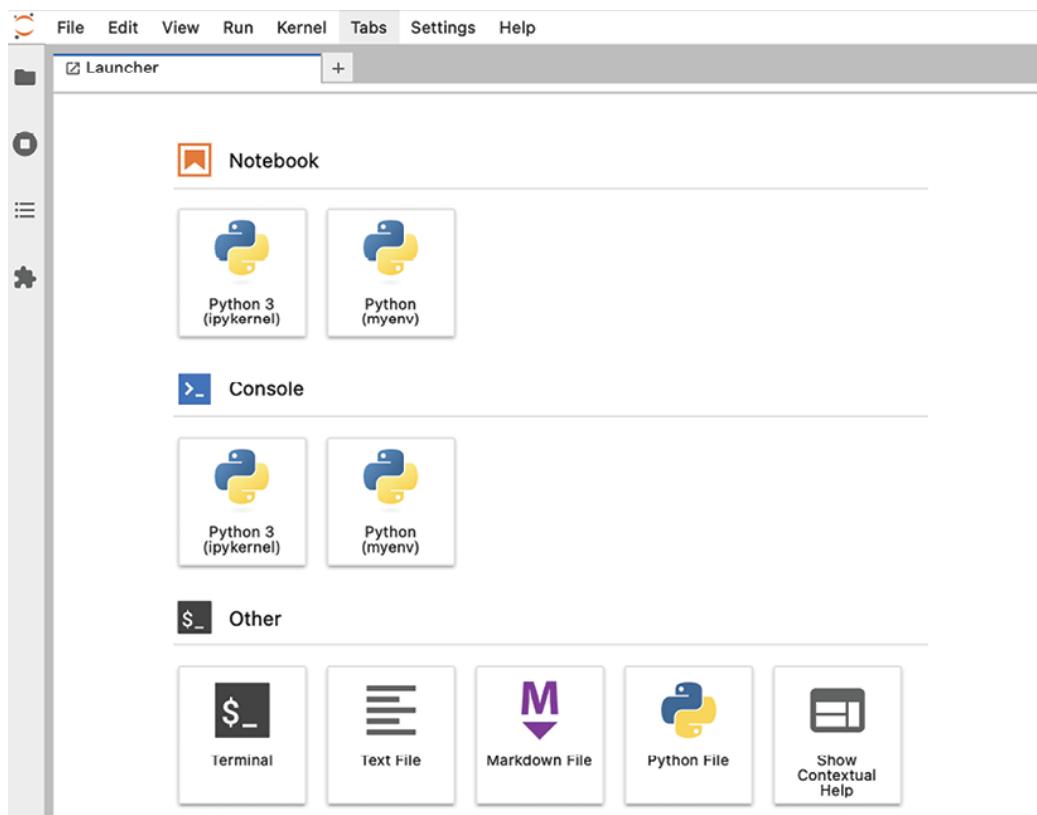


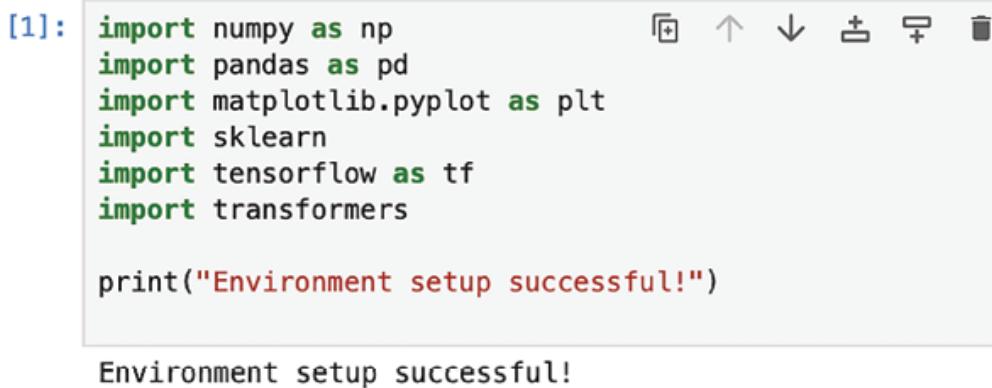
Figure 1.7: Image of the JupyterLab UI after launch

# Verifying your setup

To ensure that everything is set up correctly, run a simple Python script in a Jupyter notebook to verify the installation of key libraries. Click on the icon below the **Notebook** heading of *Figure 1.7* and type the following lines of code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import tensorflow as tf
import transformers
print("Environment setup successful!")
```

Pressing *Shift + Enter* will execute the code in the cell:



The screenshot shows a Jupyter Notebook cell with the following content:

```
[1]: import numpy as np
      import pandas as pd
      import matplotlib.pyplot as plt
      import sklearn
      import tensorflow as tf
      import transformers

      print("Environment setup successful!")
```

Below the code, the output is displayed in red text:

Environment setup successful!

*Figure 1.8: Environment setup successful message*

# Navigating the Jupyter notebook

- With your Jupyter notebook, you can now write, run, and document your Python code in a step-by-step manner. You can find extensive

documentation and tutorials on the [jupyter.org](https://jupyter.org) website. Here are some basics to get you started:

- **Creating new cells:** You can add new cells to your notebook by clicking the + button in the toolbar at the top. This will insert a new cell directly below your current selection.
- **Running cells:** To execute the code within a cell, select the cell and then press *Shift + Enter* or click the *Run* button in the toolbar. This will run the code and display any output below the cell.

You can also use *Ctrl + Enter* to run the cell without moving to the next one.

- **Changing cell types:** Jupyter Notebook supports different types of cells, including code cells for Python code and Markdown cells for text.

You can change the cell type using the dropdown menu in the toolbar. Select *Code* for Python code or *Markdown* for narrative text, equations, or HTML.

- **Using Markdown for documentation:** Markdown cells allow you to add formatted text, bullet points, numbered lists, hyperlinks, images, and more to make your notebook more readable and informative. Basic Markdown syntax includes:

- `#` for headings (e.g., `# Heading 1`, `## Heading 2`)
- `-` or `*` for bullet lists
- `1.`, `2.`, etc., for numbered lists
- ``code`` for inline code
- `[Link text](url)` for hyperlinks
- `[Alt text](Image URL)` for images

- **Saving and sharing notebooks:** Save your notebook regularly by clicking the **Save** icon in the toolbar or using *Ctrl + S* (or *Cmd + S* on a Mac).

You can share your notebook by downloading it in various formats (including `.ipynb`, `HTML`, `PDF`, etc.) from the **File -> Download as** menu.

- **Shortcuts for efficiency:** Jupyter supports several keyboard shortcuts for common operations. Press *Esc* to enter command mode, where you can:
  - Use *A* to insert a new cell above the current cell.
  - Use *B* to insert a new cell below.
  - Use *M* to change the current cell to a Markdown cell.
  - Use *Y* to change it back to a code cell.
  - Use *D, D* (press *D* twice) to delete the current cell.

## Training your first ML model

As we start applying AI/ML in marketing, it's crucial to understand the fundamental steps involved in any data science project. The Iris dataset is a classic classification example that is widely used in ML due to its simplicity and informative features. This will give you a hands-on introduction to the end-to-end process of performing AI/ML data analysis within the Jupyter Notebook environment via the following steps:

- Step 1: Importing the necessary libraries
- Step 2: Loading the data
- Step 3: Exploratory data analysis (EDA)
- Step 4: Preprocessing the data

- Step 5: Model training
- Step 6: Evaluating the model

## Step 1: Importing the necessary libraries

Before getting into the steps, let's start by importing all the required libraries using the following code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import precision_score, recall_score, f1_s
```

## Step 2: Loading the data

The Iris dataset contains 150 records of iris flowers, including measurements of their sepals and petals, along with the species of the flower. Scikit-learn provides easy access to this dataset and loads it into a pandas DataFrame:

```
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['species'] = iris.target_names[iris.target]
iris_df.head()
```

Once the code is entered into your Jupyter notebook and run, it should appear as follows:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

*Figure 1.9: View of the first 5 rows of the Iris dataset*

## Step 3: Exploratory data analysis

After loading the dataset, it's time to dive deeper into the underlying data characteristics:

1. We can first understand the structure of our data via the following command:

```
print(iris_df.info())
```

This gives us the following output:

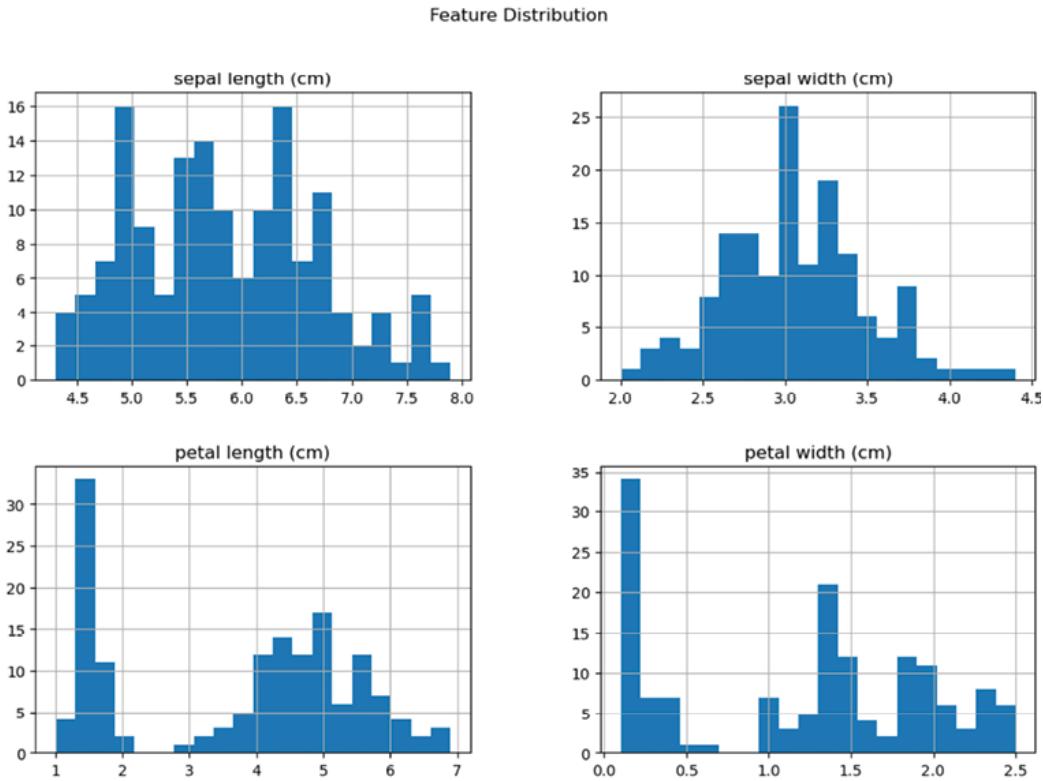
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   sepal length (cm)    150 non-null   float64
 1   sepal width (cm)    150 non-null   float64
 2   petal length (cm)   150 non-null   float64
 3   petal width (cm)   150 non-null   float64
 4   species            150 non-null   object  
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

*Figure 1.10: View of the data structure of the Iris dataset*

The above result gives us insights into the data structure, including column names and data types, non-null counts (missing values can significantly impact the performance of ML models), and memory usage (useful for managing computing resources, especially when working with large datasets).

2. Next, we can visualize the distribution of features to get insights into the nature of the data we're dealing with. Histograms are graphical representations that summarize the distribution of numerical data by dividing it into intervals or “bins” and displaying how many data points fall into each bin:

```
iris_df.hist(figsize=(12, 8), bins=20)
plt.subtitle('Feature Distribution')
plt.show()
```



*Figure 1.11: Histograms of the distribution of features in the Iris dataset*

The histograms for the Iris dataset's features give us valuable insights into their feature characteristics, including their distribution shape (whether they are bell-shaped or skewed), outliers and anomalies (which can significantly affect model performance), and feature separability (if one feature consistently falls into a species bin that doesn't overlap much with the other species, it might be a good predictor for that species).

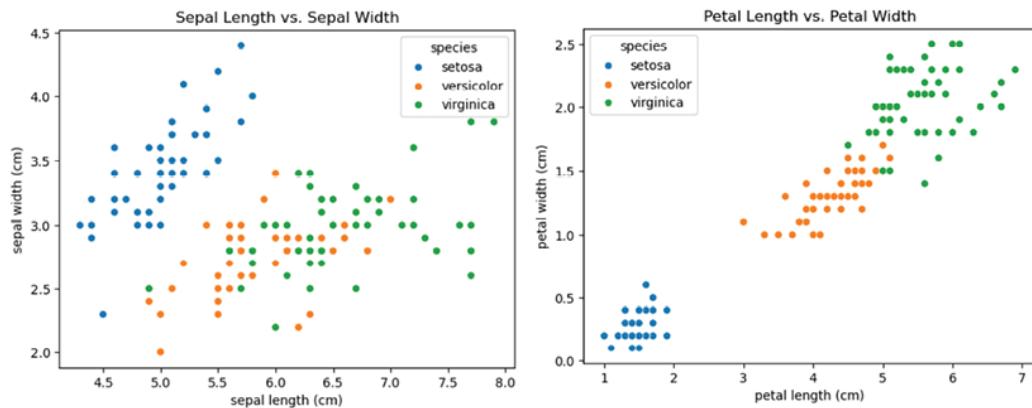
3. Lastly, we can utilize scatter plots to visualize pairwise relationships between features and employ pair plots to gain insight into how each feature interacts with others across the different species. We can generate scatter plots for pairs of features via the following:

```

sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)')
plt.title('Sepal Length vs. Sepal Width')
plt.show()
sns.scatterplot(x='petal length (cm)', y='petal width (cm)')
plt.title('Petal Length vs. Petal Width')
plt.show()

```

This gives us the following output:



*Figure 1.12: Scatter plots of pairwise relationships between features in different species in the Iris dataset*

As shown by the above scatter plots, petal length and petal width may demonstrate clear clustering by species, suggesting they are strong predictors for species classification.

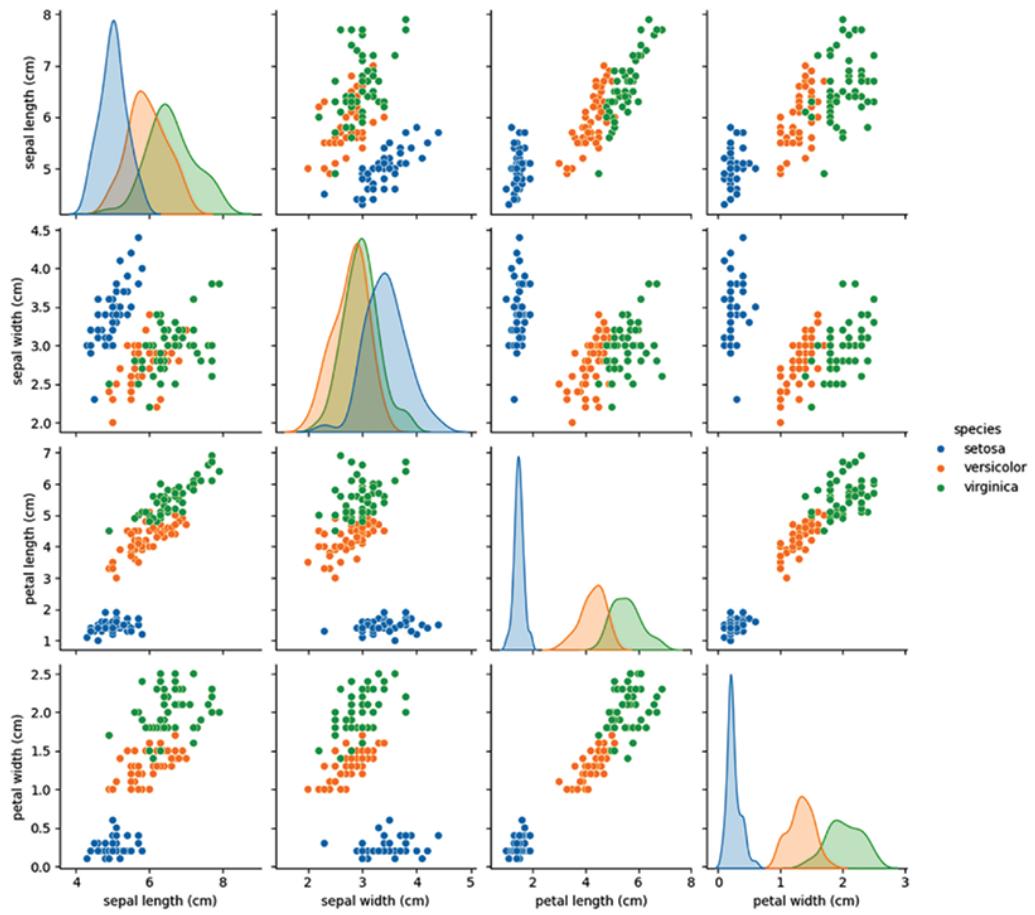
Pair plots offer a more comprehensive view by showing scatter plots for every pair of features in the dataset. Additionally, histograms along the diagonal provide distributions for each feature, segmented by species:

```

sns.pairplot(iris_df, hue='species')
plt.show()

```

This yields the following output:



*Figure 1.13: Pair plots for the Iris dataset, showing scatter plots for each pair of features and histograms along the diagonal*

The above pair plots allow us to quickly identify which features have linear relationships or clear separation between species across multiple dimensions. For instance, the combination of petal length and petal width shows a distinct separation between species, indicating that these features are particularly useful for classification tasks. The histograms along the diagonal help in understanding the distribution of each feature within each species, providing insights into how these distributions can be leveraged for predictive modeling. For instance, if a feature shows a tight, well-defined distribution within a species, it suggests that the feature is a reliable

predictor for that species. Conversely, a feature with a wider spread within a species may be less reliable as a predictor.



### Importance of visual EDA

Visual EDA is a powerful first step in the modeling process. By identifying patterns, clusters, and outliers, we can make informed decisions about feature selection, preprocessing, and the choice of ML models.

## Step 4: Preparing the data for ML

The next crucial step is to prepare our data for ML. This process typically involves selecting features for the model, splitting the data into training and testing sets, and sometimes transforming the features to better suit the algorithms you plan to use.

In supervised learning tasks like the one given here, we distinguish between features (independent variables) and the target (dependent variable). In the `Iris` dataset:

- *Features* include the measurements: sepal length, sepal width, petal length, and petal width.
- *Target* is the species of the iris plant.

For our example, we will follow these steps:

1. We use all four measurements as features to predict the species of the iris plant, making this a multi-class classification problem:

```
X = iris_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]  
y = iris_df['species']
```

2. To assess the performance of an ML model, we split our dataset into a training set and a testing set. The training set is used to train the model, and the testing set is used to evaluate its performance on unseen data. A common split ratio is 80% for training and 20% for testing. We can easily split our data using the `train_test_split` function from scikit-learn:

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

### Importance of data splitting



Splitting the data into training and testing sets is a fundamental practice in ML. It helps in evaluating the model's performance accurately and ensures that it can generalize well to new, unseen data. By training and testing on different sets, we mitigate the risk of overfitting, where the model performs well on the training data but poorly on new data.

3. Some ML algorithms are sensitive to the scale of the data. For example, algorithms that compute distances between data points (like K-nearest neighbors) can be affected by features that are on different scales. Feature scaling can be applied via the following code:

```
scaler = StandardScaler()  
X_train_scaled = scaler.fit_transform(x_train)
```

```
X_test_scaled = scaler.transform(X_test)
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_
X_train_scaled_df.head()
```

This gives us the following output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-1.473937	1.203658	-1.562535	-1.312603
1	-0.133071	2.992376	-1.276006	-1.045633
2	1.085898	0.085709	0.385858	0.289218
3	-1.230143	0.756479	-1.218701	-1.312603
4	-1.717731	0.309299	-1.390618	-1.312603

*Figure 1.14: Scaled features for the first 5 rows of the Iris dataset*

Now, each feature's values are centered around 0 with a unit variance. This step is essential for certain ML algorithms that are sensitive to the scale of the data and ensures that each feature contributes proportionately to the final model.

## Step 5: Training a model

With our data loaded, explored, and prepared, we're now ready to move on to one of the most exciting parts of an ML project: training a model. For this example, we will use a decision tree classifier, a versatile ML algorithm that works well for classification tasks and is easy to understand and interpret as it mimics human decision-making. The decision tree will help us predict the species of iris plants based on the features we prepared earlier.

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.

Each node in the tree represents a feature in the instance being classified, and each branch represents a value that the node can assume. We can train our decision tree classifier using scikit-learn:

```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
```

Once our model is trained, we can use it to make predictions. We will predict the species of iris plants using the features from our testing set:

```
y_pred = dt_classifier.predict(X_test)
print("First few predictions:", y_pred[:5])
```

The following is the output:

```
First few predictions: ['versicolor' 'setosa' 'virginica' 'versi
```



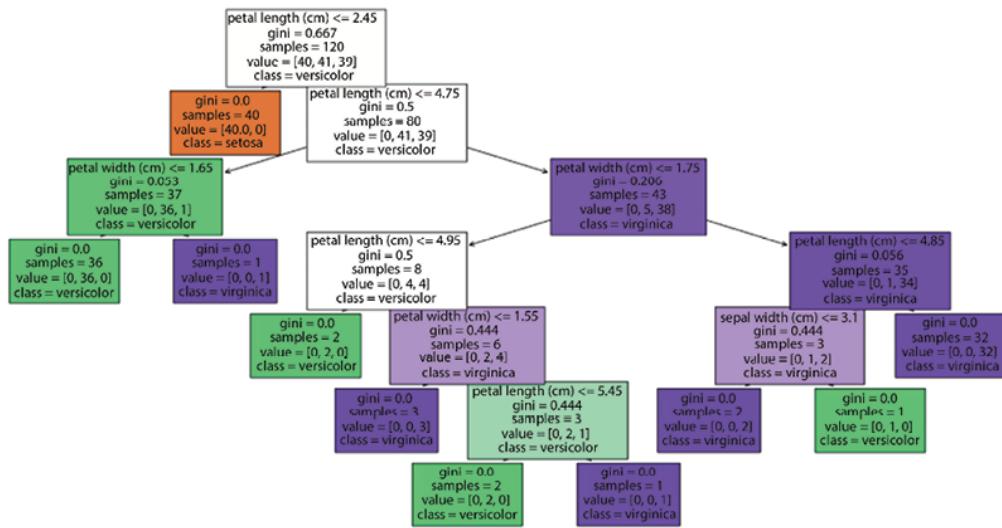
### Why use a decision tree?

Decision trees are a popular choice for classification tasks because they don't require much data preparation, are easy to interpret and visualize, and can handle both numerical and categorical data. For beginners in ML, decision trees offer a clear and intuitive way to understand the basics of model training and prediction.

Visualizing the decision tree can provide insight into how the model makes its decisions:

```
plt.figure(figsize=(20,10))
plot_tree(dt_classifier, filled=True, feature_names=iris.feature
```

This gives us the following output:



*Figure 1.15: Visualization of a decision tree classifier built on the Iris dataset*

The above visualization shows the splits that the tree makes on the features, the criteria for these splits, and the eventual leaf nodes where the final predictions are made based on the majority class from the training samples that fall into that leaf. For those new to ML, seeing this process can clarify how a seemingly simple algorithm can effectively classify instances. Visualizing your model can also highlight areas where the model might be overfitting by creating overly complex decision paths.

## Step 6: Evaluating the model

The last step is to use the model predictions on our test set and evaluate its performance. This step is crucial as it helps us understand how well our model can generalize to unseen data.

Using the trained decision tree classifier, we can now calculate precision, recall, and the F1-score. Let's look at what exactly these are:

- Precision measures the accuracy of the positive predictions. It is the ratio of true positive predictions to the total positive predictions (including both true positives and false positives). High precision indicates that the model is reliable in its positive predictions.
- Recall (or sensitivity) measures the ability of the model to capture all actual positives. It is the ratio of true positive predictions to the total actual positives (including both true positives and false negatives). High recall means that the model is good at capturing positive instances without missing many.
- The F1-score is the harmonic mean of precision and recall, providing a single metric to assess the balance between them. An F1-score reaches its best value at 1 (perfect precision and recall) and worst at 0.

We'll use the built-in function in scikit-learn to calculate these metrics. These predictions can then be compared to the actual species to evaluate some performance metrics of our model:

```
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

This gives us the following output:

```
Precision: 1.00
Recall: 1.00
F1-Score: 1.00
```

Achieving a score of 1.0 in precision, recall, and F1-score is exceptional and indicates perfect model performance on the test set; however, in real-world scenarios, especially with complex and noisy data, such perfect scores are rare and should be approached with caution, as they may not always reflect the model's ability to generalize to unseen data. The Iris dataset is relatively small and well-structured, with clear distinctions between the classes. Thus, this simplicity makes it easier to achieve high-performance metrics compared to real-world datasets, for which model training and evaluation are typically more complex. As we will discuss in future chapters, further output diagnostics such as the confusion matrix can also be used to gain insight into model strengths and weaknesses.

### Understanding model performance



Model accuracy is a vital metric in assessing the effectiveness of an ML model. An accuracy score close to 1.0 indicates a high level of correct predictions. However, it's also important to consider other metrics like precision, recall, and the confusion matrix for a more comprehensive evaluation, especially in datasets with imbalanced classes.

Congratulations! By completing these steps—training a model, making predictions, and evaluating its performance—you've covered the essential workflow of an ML project. The skills and concepts you've practiced here are directly applicable to the exercises you will perform in the following chapters towards creating effective, data-driven marketing campaigns.

## Summary

In this chapter, we started our foundational journey by setting up a Python environment tailored for AI/ML projects, focusing on those in marketing. We also provided a timeline of marketing through the years and gave you some background about where the field currently stands. Using the Iris dataset as a practical example, we walked you through the fundamental steps of loading data, performing EDA, preparing the data for ML, and finally, training and visualizing a model. We also laid the groundwork for understanding how these steps translate into marketing analytics. This exercise demonstrated the versatility of Python and its rich ecosystem of libraries, highlighting their role in data manipulation, ML, NLP, and data visualization.

The example, while not directly related to marketing, teaches you essential skills that are directly applicable to marketing challenges you may face in the real world, such as customer segmentation, predictive analytics, and campaign optimization. Gaining familiarity with these processes gives you a solid foundation for tackling more complex and specialized marketing data analyses. The iterative and exploratory nature of data science work, which offers flexible techniques for testing hypotheses, visualizing data, and sharing insights. This is what makes it so useful for effective analysis. As we move forward, the tools, techniques, and principles introduced in this chapter will serve as building blocks for the more advanced AI/ML applications we will explore. The journey into AI/ML-powered marketing is filled with opportunities to leverage data for strategic advantage, enhance customer engagement, and drive business growth. With the Python environment set up and a preliminary ML project under your belt, you're now ready to dive deeper into the transformative potential of AI and ML in marketing.

In *Chapter 2*, we will discuss the core concepts of decoding marketing performance using KPIs, providing you with the essential tools to measure and optimize your marketing strategies effectively.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](https://OceanofPDF.com)

# 2

## Decoding Marketing Performance with KPIs

Every data-driven or AI/ML-driven marketing strategy starts with a set of optimization and enhancement goals that have key performance metrics tied to them. These key performance metrics or measures are often termed the **key performance indicators (KPIs)** that help you track the performance of your marketing campaigns and define the success criteria of your marketing efforts. Some of the frequently used KPIs in digital and data-driven marketing that we will define and explore in this chapter are **conversion rate, cost per acquisition (CPA), customer lifetime value (CLV), and return on investment (ROI)**.

In this chapter, we will discuss a frequently used set of marketing KPIs, what they tell you about your marketing campaigns, and how they can be used to further improve and optimize your marketing strategies and business outcomes. We will use an insurance product marketing dataset as an example to highlight how Python and its data analysis and visualization libraries, such as `pandas`, `numpy`, `matplotlib`, and `plotly`, can be used for analyzing, visualizing, and generating insights from marketing KPIs.

In particular, we will cover the following topics in this chapter:

- Understanding marketing KPIs

- Computing and visualizing KPIs from data with Python
- Tracking marketing performance with KPIs

**Source code and data:**

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/tree/main/ch.2>



**Data source:**

<https://www.kaggle.com/datasets/pankajjs06/ibm-watson-marketing-customer-value-data>

# Understanding marketing KPIs

As a marketer or marketing data scientist, KPIs are the keys to measuring the successes and failures of any marketing efforts. The proper usage and definition of KPIs ahead of embarking on any marketing campaign is crucial for the effective measurement of marketing successes and failures. However, improper usage of KPIs can result in misinformation being gathered and often wasting significant financial and human resources that have a direct negative impact on the business.

For example, if your marketing goal is to raise brand awareness of a targeted group (i.e., Generation Alpha) but you are not properly breaking down the sources of marketing content impressions (i.e., text vs. image vs. video, or YouTube vs. Instagram vs. TikTok, or influencer referrals vs. paid

ads), it may tell a completely different story than what is actually happening. We will review some of the frequently used KPIs in this section and move on to the implementation and real-world use case examples in the following sections.



### What is a KPI?

A KPI is a key performance indicator that measures and quantifies the progress toward a certain goal or objective. KPIs help organizations identify strengths and weaknesses and make data-driven decisions in every aspect of business, such as marketing, finance, sales, and operation.

In each stage of the marketing funnel, there are KPIs associated with it. The following diagram shows a summary of a typical marketing funnel along with some of the frequently used KPIs for each stage.

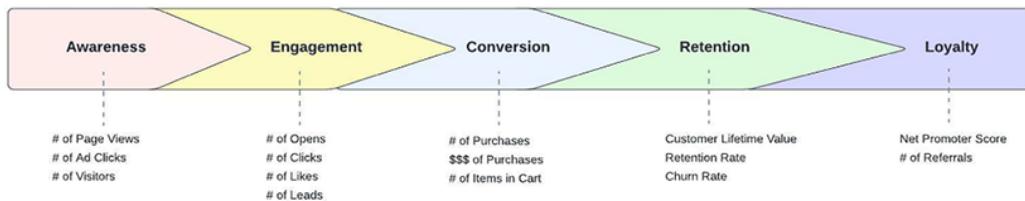


Figure 2.1: Different stages of a typical marketing funnel

Now let's look at each of these stages in some detail.

## Awareness stage

During the *Awareness* stage, the goal is to make customers aware of your organization and products. Thus, brand building and exposure are the ultimate marketing goals. Some of the typical KPIs to measure for the

success of the marketing initiatives during this stage are *Number of Page Views*, *Number of Ad Clicks*, *Number of Visitors*, and *SEO Rankings*. The key here is to optimize marketing spending for maximum brand impressions and exposure.

## Engagement stage

During the *Engagement* stage, the goal is to start interactions and build relationships with prospective customers. You may want potential customers to engage with your organization and products by opening emails, reading newsletters, clicking on links, and signing up for events. Some of the key metrics or KPIs to measure the success of your marketing initiatives during this stage are *Number of Opens*, *Number of Clicks*, *Number of Likes*, and *Number of Qualified Leads*.

In particular, tracking the number of leads, as well as the quality of leads, is directly tied to potential future revenue. The key measure or metric to keep in mind here is the **cost per lead (CPL)**, given by:

$$CPL = \frac{\text{Total Cost of Marketing}}{\text{Number of Leads}}$$

Keeping track of the rise and fall of CPL is important for successful marketing campaigns during this stage, as rising CPL indicates that the efficiency of your marketing strategy is decreasing.

## Conversion stage

The *Conversion* stage is where we harvest all the hard work we have done. The definition of conversion can be different for distinct types of customers.

For e-commerce businesses, conversion may mean leads making item purchases. For media businesses, conversion may mean leads signing up or paying subscriptions for newsletters. Regardless, this is when leads become customers. Some of the key metrics to follow in this stage are *Number of Purchases*, *Amount of Purchases*, *Number of Subscriptions*, or *Number of Items in Cart*.

During this stage, tracking the conversion rate is important to measure the successes and failures of marketing initiatives. The conversion rate is simply:

$$\text{Conversion Rate} = \frac{\text{Number of Conversions}}{\text{Number of Leads}}$$

Tracking the rise and fall of the conversion rate is important, as a falling conversion rate may suggest inefficiency in marketing strategies at this stage, product feature deficiencies, or market shifts in interests. On top of tracking this KPI, delving deeper into the drivers behind the rise and fall of conversion rates is even more important. We will look further into this in the following section on some of the approaches that can be taken to gain insights into conversion rates, and we will discuss more in detail, unveiling the drivers and factors behind the rise and fall of conversion rates, in the following chapter.

## Retention stage

During the *Retention* stage, you focus on turning customers into repeat customers, providing continuous value to the customers, and making them stick to your services or products. Retaining customers and turning them into consistent repeat customers are directly tied to generating steady

revenue streams. Some of the key metrics to track are **customer retention rate (CRR)**, **customer churn rate (CCR)**, **customer lifetime value (CLV)**, and **cost per acquisition (CPA)**.

CRR and CCR can be calculated using the following equations:

$$CRR = \frac{\# \text{ of Customers at the End of Period} - \# \text{ of New Customers during the Period}}{\# \text{ of Customers at the Start of Period}}$$

$$CCR = \frac{\# \text{ of Customers lost during the Period}}{\# \text{ of Customers at the Start of Period}}$$

Rising CRR suggests that customers see value in using your services and products continuously; on the other hand, an increase in CCR suggests the opposite. There can be multiple reasons why customers churn, and delving deeper into data to understand the potential causes is the key to building successful marketing strategies during this stage. We will discuss how to look at data from different angles to have a comprehensive understanding of certain customer behaviors in the following section, and even deeper analysis techniques using machine learning will be discussed in the following chapter.

In the *Retention* stage, understanding CLV and CPA is critical for a sustainable business. Even if CLV is high, where each customer brings in significant revenue, if CPA exceeds CLV, then it will be a net negative business. Even though it is not trivial to calculate CLV, the formulas for CPA and CLV are as follows:

$$CPA = \frac{\text{Total Marketing Spend}}{\# \text{ of Customers Acquired}}$$

$$CLV = \text{Customer Value} * \text{Average Customer Lifespan}$$

As you may notice, *Customer Value* in the CLV formula may depend on many factors, such as the type of business and the definition of the value. We will delve deeper into estimating CLV from an AI/ML point of view in *Chapter 8*, but the key takeaway from this section is that CPA negates CLV, and proper measurement of both KPIs and optimization are critical paths to successful marketing strategies during the *Retention* stage.

## Loyalty stage

This stage is where your customers become advocates of your services or products and bring in more new leads and customers via referrals, word of mouth, or reviews. It is critical to show your appreciation toward these loyal customers, help them spread the word about your services and products, and continue to build strong relationships and connections with your loyal customers.

All key measures discussed previously are relevant and important to be continuously monitored for this stage. For example, the number of page visits and page views that were some of the *Awareness* stage KPIs are still applicable to this stage, so that you can continuously monitor and gauge the loyal customer group's interest level. The number of clicks and likes, which were the KPIs during the *Engagement* stage, for example, are still relevant in the *Loyalty* stage so that you can track the continuous engagement of this group of customers. The number and amount of purchases, as well as the retention and churn rates that were some of the KPIs during the *Conversion* and *Retention* stages, are still important to monitor, as these loyal customers are the ones that bring the most consistent stream of revenues for businesses.

Aside from the aforementioned KPIs, **net promoter score (NPS)** and *number of referrals* and reviews can be some of the other important KPIs to monitor during the *Loyalty* stage.

In order to get the NPS, a survey needs to be conducted to gather customer feedback and, based on the survey results, NPS is calculated as in the following equation:

$$NPS = \% \text{ of Promoters} - \% \text{ of Detractors}$$

For example, if 60% of the survey respondents were promoters and 10% of the respondents were detractors, then your NPS would be 50. NPS ranges from -100 to 100, and the higher the score, the higher the satisfaction that the customers have from your services and products. It would be wise to periodically conduct such satisfaction surveys and monitor NPS, as not addressing decreasing NPS is likely to result in increasing customer churn rate and lower conversion rates.

We have covered some of the key stages in the customer journey and marketing funnel and how you can form KPIs for each stage to track the health of your marketing funnel, as well as how to measure the successes and failures of your marketing strategies for each stage. Not only would you need to monitor KPIs consistently but you would also need to delve deeper into data to identify which parts of your funnel show weaknesses and how various groups of customers behave differently to different marketing strategies. In the following section, we will discuss how to take these learnings into practice with an insurance marketing dataset as an example using Python and start generating reusable and trackable KPIs, and derive actionable items from these KPIs.

# Computing and visualizing KPIs from data with Python

With the foundation of various marketing KPIs that we have discussed so far, let us dive into a potential real-world use case of some of these KPIs, how to compute and visualize them, and how to interpret them for further data-driven decision-making. We will be using an insurance product marketing dataset for our example.

From a financial point of view, the success of marketing is to bring in more revenue with less spending. The most straightforward approach in tracking and measuring this is to look at conversion rate, CLV, and CPA. The conversion rate tells us the percentage of leads converted into paying customers, CLV tells us the estimated revenue being generated per acquired customer, and CPA tells us how much it costs to acquire a paying customer. By combining these three KPIs, we can understand the ROI and contribution to the net income.

Using the above-mentioned example dataset, let's discuss how these KPIs can be measured with Python, help generate deeper insights, and, ultimately, affect the decision-making process for better and more efficient marketing strategies.

## Conversion rate

The first KPI we are going to look at within this data is the *conversion rate*. Remember from the previous section that the conversion rate is the percentage of the leads converted into purchasing a product. In the context of this data, those who converted are those who responded Yes in the `Response` column.

# Overall conversion rate

To calculate the overall conversion rate, we use the `pandas` library:

1. Assuming that the name of the data file is `data.csv`, you can use the following code to import and read the data into a `DataFrame`:

```
import pandas as pd  
df = pd.read_csv("./data.csv")
```

2. Next, we need to convert the text responses in the `Response` column into a binary 0 or 1 value so that we can easily compute the conversion rates. The code to do this is as follows:

```
df["conversion"] = df["Response"].apply(lambda x: 1 if x ==
```

3. In order to compute the overall conversion rate with this data, you simply need to take the average of the values in the newly created `conversion` column as in the following code:

```
df["conversion"].mean()
```

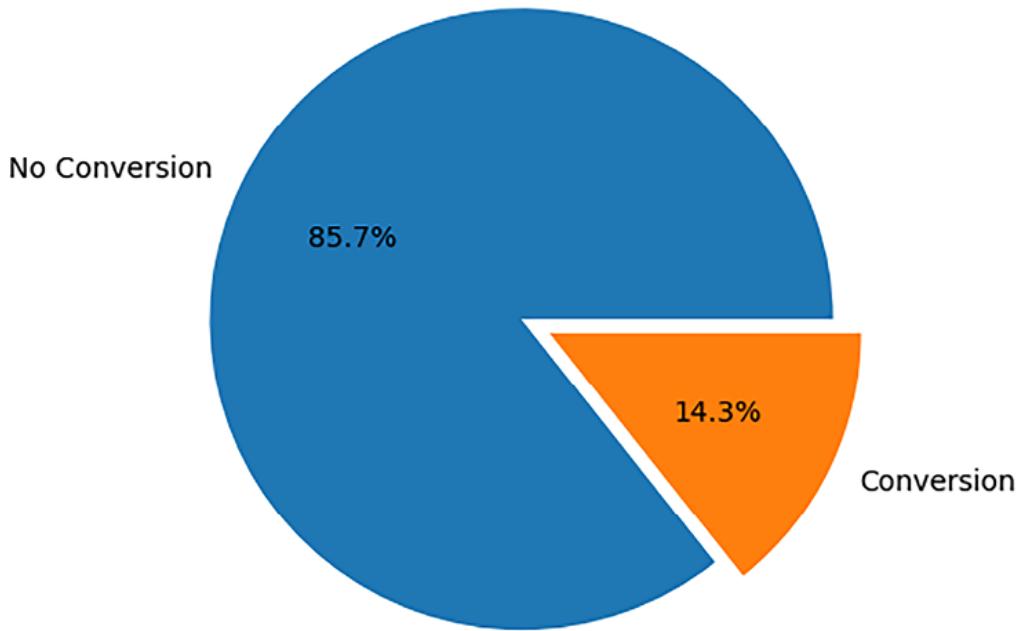
4. One way to visualize and easily understand the overall conversion rate is using a pie chart, which is a great way to show the composition or breakdown of different segments within data. The code to generate such a pie chart is as follows:

```
ax = df.groupby("conversion")["Customer"].count().reset_index()  
figsize=(5, 5),  
y="Customer",  
autopct='%1.1f%%',
```

```
        legend=False,
        labels=["No Conversion", "Conversion"],
        explode=[0.05, 0.05]
    )
ax.set_ylabel(None)
```

Here, we first group by the column, conversion, and count the number of customers in each segment. This results in the number of converted customers vs. non-converted customers.

5. Then, we plot a pie chart for these two groups. The result looks as follows:



*Figure 2.2: Pie chart of Conversion versus No Conversion*

As is clearly noticeable, the majority of the leads did not convert. Only 14.3% of the qualified leads converted into a policy renewal or responded Yes to the renewal offer. This is typically the case when it comes to marketing. Typically, conversion rates are very low, and even low single-digit percentage conversion rates are not abnormal in digital marketing.

The chart itself is insightful in a way that it helps business stakeholders to understand what percentage of the targeted leads converted, but this is not enough. In order to have a full grasp of what the data tells us about the conversion rate, we need to look at various angles.

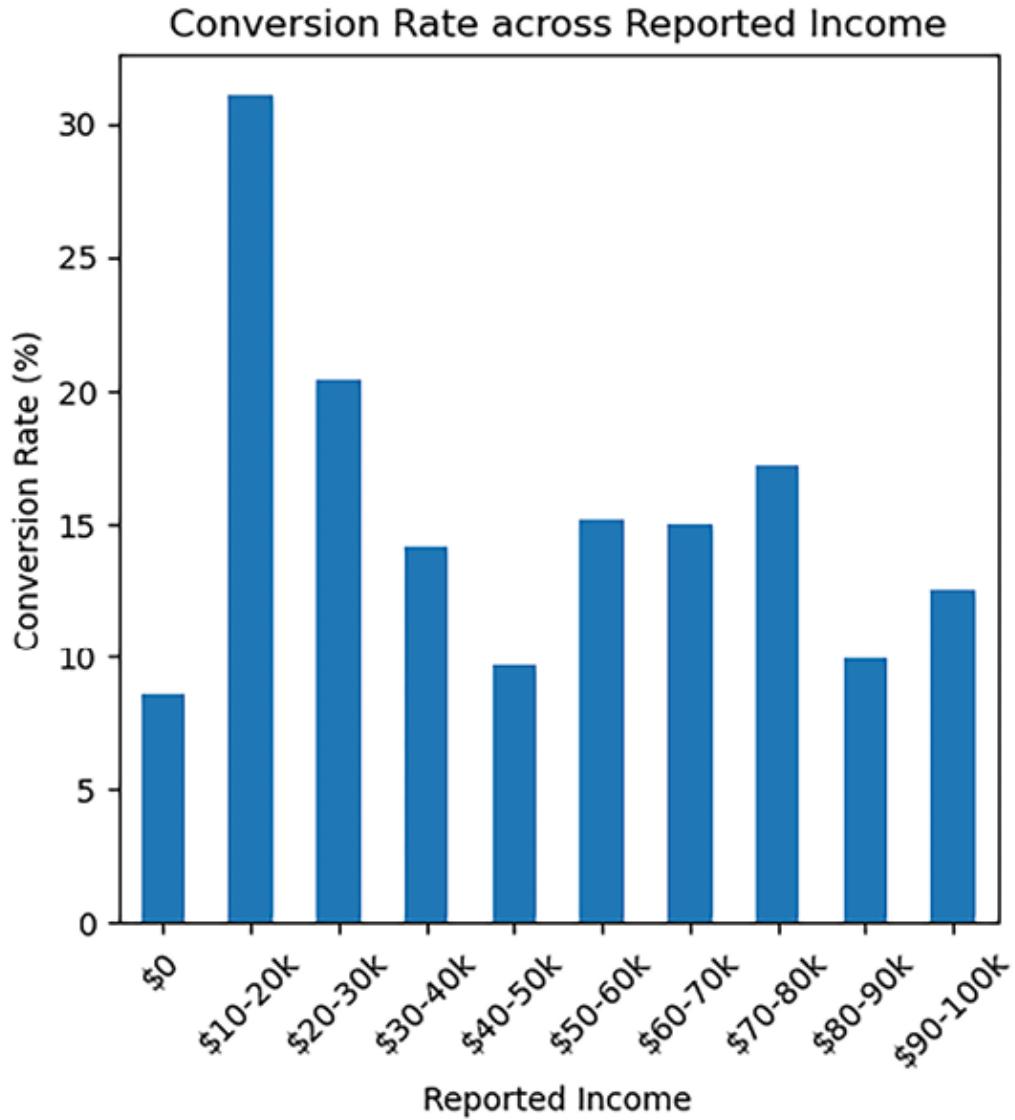
## Demographics and conversion rate

One way to dive deeper to generate more insights into the conversion rate is to look at the demographics of customers and how it may affect the conversion rates. For example, in our data, there is a column called `income`. We can break down the conversion rates across different income levels to see what kind of story the data may tell us about the relationship between income and conversions.

In our example data, the reported income ranges from \$0 to \$100,000. For easier understanding and better visualization, we are going to bucket this into 10 income levels and examine the relationship between income and conversions, as shown in the following code:

```
df["income_category"] = df["Income"].apply(  
    lambda x: 0 if x == 0 else x//10000  
)  
income_conversion = df.groupby("income_category")["conversion"]  
ax = (  
    income_conversion  
).plot.bar(  
    figsize=(5, 5),  
    rot=45,  
)  
ax.set_xticklabels(["$0" if x == 0 else f"${x}0-{x+1}0k" for x  
ax.set_ylabel("Conversion Rate (%)")  
ax.set_xlabel("Reported Income")  
ax.set_title("Conversion Rate across Reported Income")
```

The output of this code looks as follows:



*Figure 2.3: Bar plot of conversion rates across different income bands*

As you can see from this bar plot, people with an income between \$10k and \$20k have the highest conversion rate. Another interesting point to note is this chart shows a bimodal distribution, where one is at \$10–20k and another is at \$70–80k. Lastly, customers with a reported income of \$0 seem to show the lowest conversion rate.



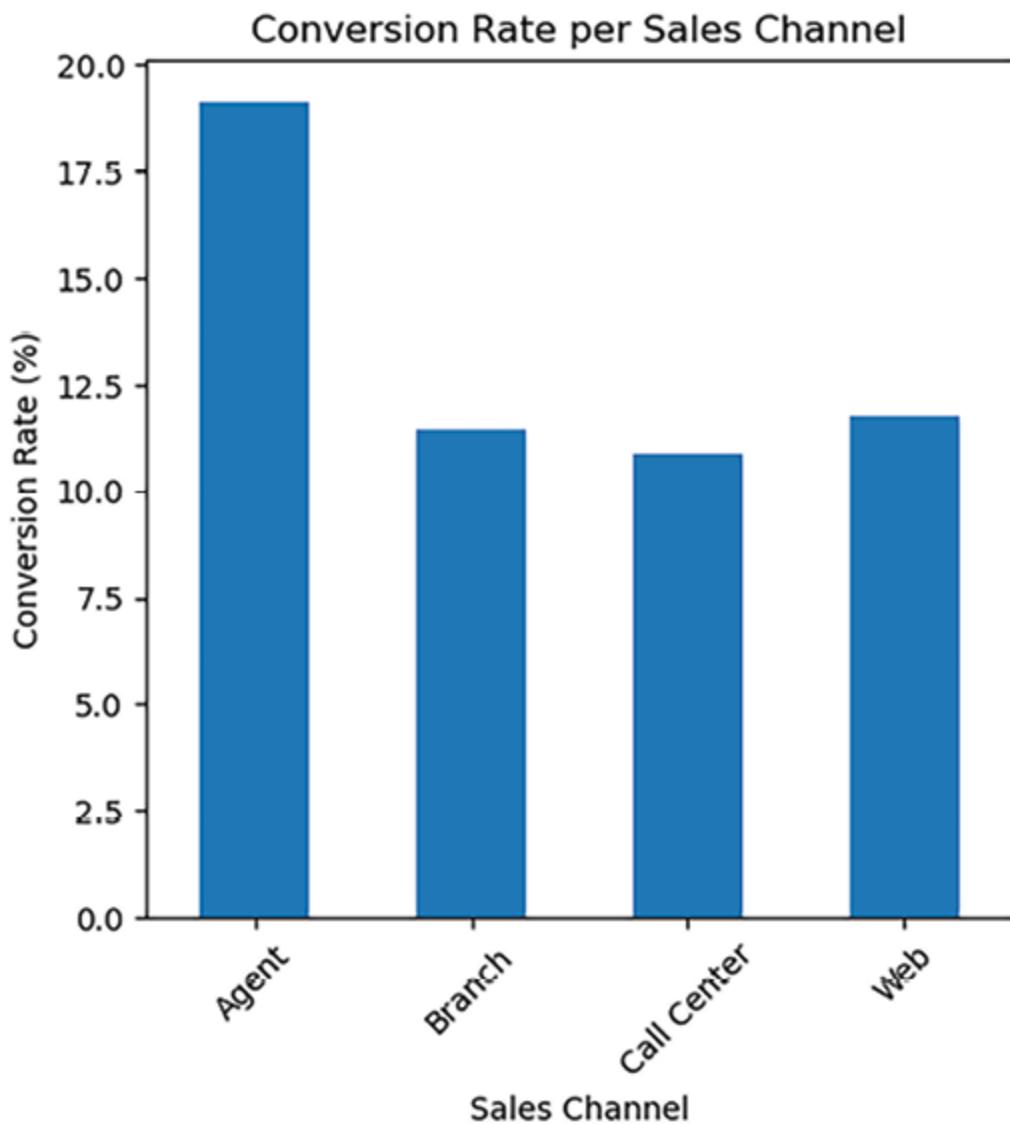
Try breaking down the conversion rates with other demographic variables, such as gender, education, and marital status!

## Sales channel and conversion rate

How customers are contacted can also be a critical factor affecting conversions. We can break down the conversion rates based on `Sales channel`, as in the following code, to understand the relationship between sales channels and conversion rates:

```
ax = (
    df.groupby("Sales Channel")["conversion"].mean() * 100
).plot.bar(
    figsize=(5, 5),
    rot=45
)
ax.set_ylabel("Conversion Rate (%)")
ax.set_title("Conversion Rate per Sales Channel")
```

The resulting bar chart looks as follows:



*Figure 2.4: Bar plot of conversion rates across different sales channels*

As this chart shows, the chance of conversion is the highest when a customer is reached out to by an `Agent`. The other three channels, `Branch`, `call center`, and `web`, have similar chances when it comes to conversion rates. This may be related to the personal connections that agents have with customers and how personal interactions affect a customer's tendency to convert.

Not so surprisingly, the composition of conversions across different sales channels, it looks like the following:

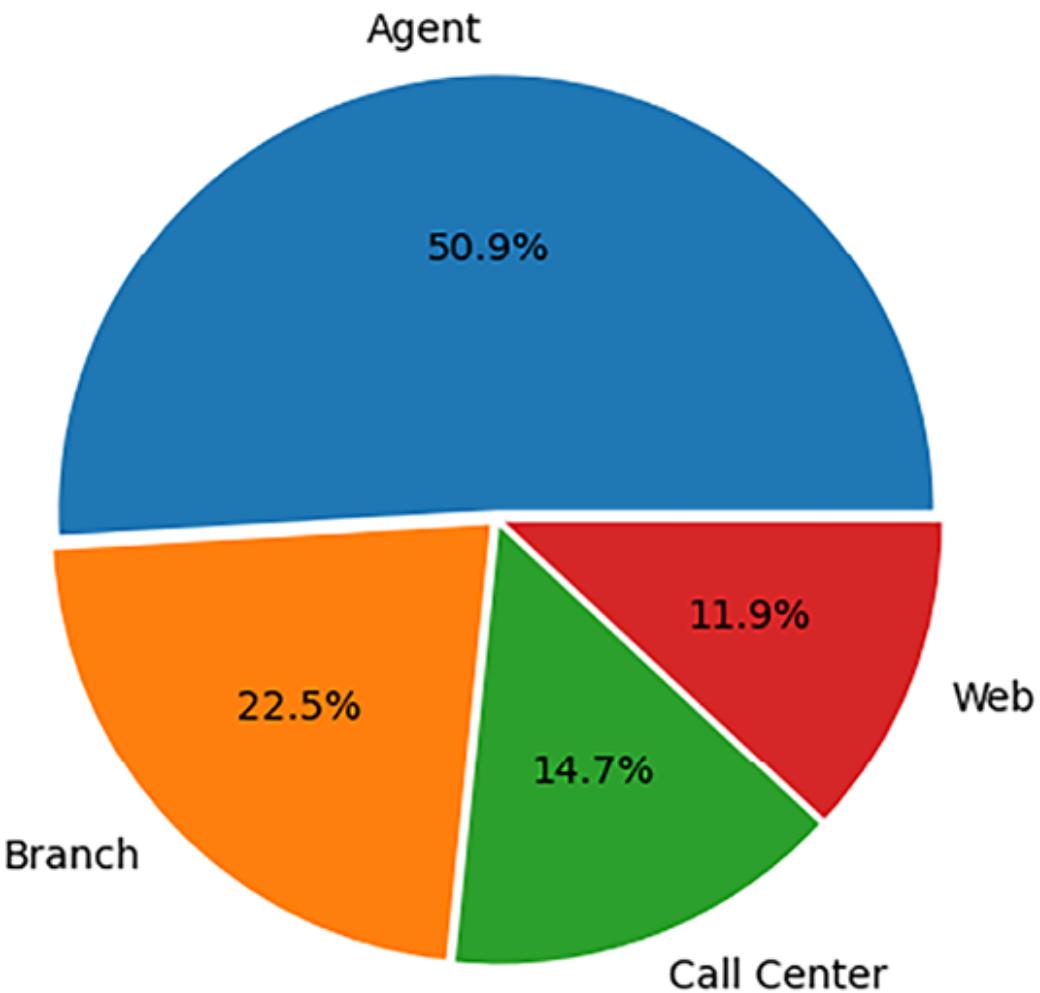


Figure 2.5: Pie chart of conversions across different sales channels

The code to generate this pie chart looks as follows:

```
sales_channel_count = df.groupby("Sales Channel")["conversion"]
ax = sales_channel_count.reset_index().plot.pie(
    figsize=(5, 5),
    y="conversion",
    autopct='%1.1f%%',
    legend=False,
    labels=sales_channel_count.index,
```

```
    explode=[0.02]*df[\"Sales Channel\"].nunique()
)
ax.set_ylabel(None)
```

This pie chart suggests that, out of the conversions, a little above 50% are through Agent interactions, Branch comes next, and then call center and Web have similar impacts on the number of conversion contributions.

In summary, looking at conversion rates from different angles tells us different stories about them. It informs us that the conversions are strong for low (\$10–20k) and high (\$70–80k) income populations, but weak in mid-income (\$40–50k) and \$0 reported income populations. It also gives us insights into which marketing channels worked the best for this renewal marketing campaign, showing that Agent reachouts worked the best among the four channels. Based on these KPI analyses, we are better informed and armed for what aspects to improve and what aspects are known to work well.



Try diving deeper into the conversion rates not only with other variables, such as renew offer type and vehicle class, but also with multiple variables, such as sales channel and state or education and policy and vehicle size.

Bring the hidden insights to the surface!

## CLV

As mentioned previously, CLV is one of the key measures to calculate ROI, as it tells us the potential revenue each customer brings. In this section, we will utilize the pre-computed CLV that was given within the dataset.

However, we will discuss in depth how to build and forecast CLV for each customer in *Chapter 8*. For the sake of focusing on the KPIs and deriving actionable items and insights from KPIs, we will use the given CLV in this chapter.

## Overall CLV

Using the `pandas` library, there are multiple ways to understand the distribution of CLV. One way is to use the `describe` function of a DataFrame as in the following:

```
df["Customer Lifetime Value"].describe()
```

This code will show the distribution as in the following output:

```
count      9134.000000
mean      8004.940475
std       6870.967608
min      1898.007675
25%      3994.251794
50%      5780.182197
75%      8962.167041
max     83325.381190
Name: Customer Lifetime Value, dtype: float64
```

Figure 2.6: Distribution of CLV

As you can see, the `describe` function gives us high-level distribution information about the `Customer Lifetime Value` column, where the average CLV is \$8,004.94 and the median is \$5,780.18. Another way to look at the distribution is with a box plot, using the `plot.box` function of a DataFrame, as shown in the following code:

```
df["Customer Lifetime Value"].plot.box()
```

This gives us the following output:

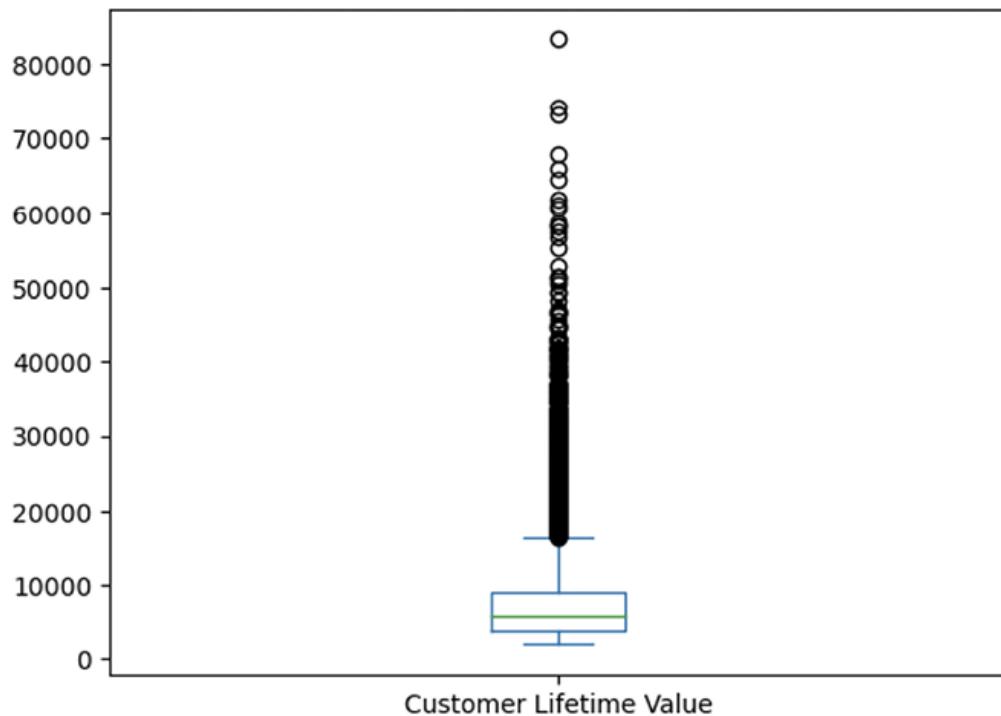


Figure 2.7: Box plot of CLV

As shown from the box plot above, there is a very long tail or group of outliers. This coincides with the output of the `describe` function, where the mean was much higher than the median, which suggests there is a long tail. This skewness in the data is also visible from the histogram as follows:

```
ax=df["Customer Lifetime Value"].hist(bins=25)  
plt.show()
```

We then see the following output:

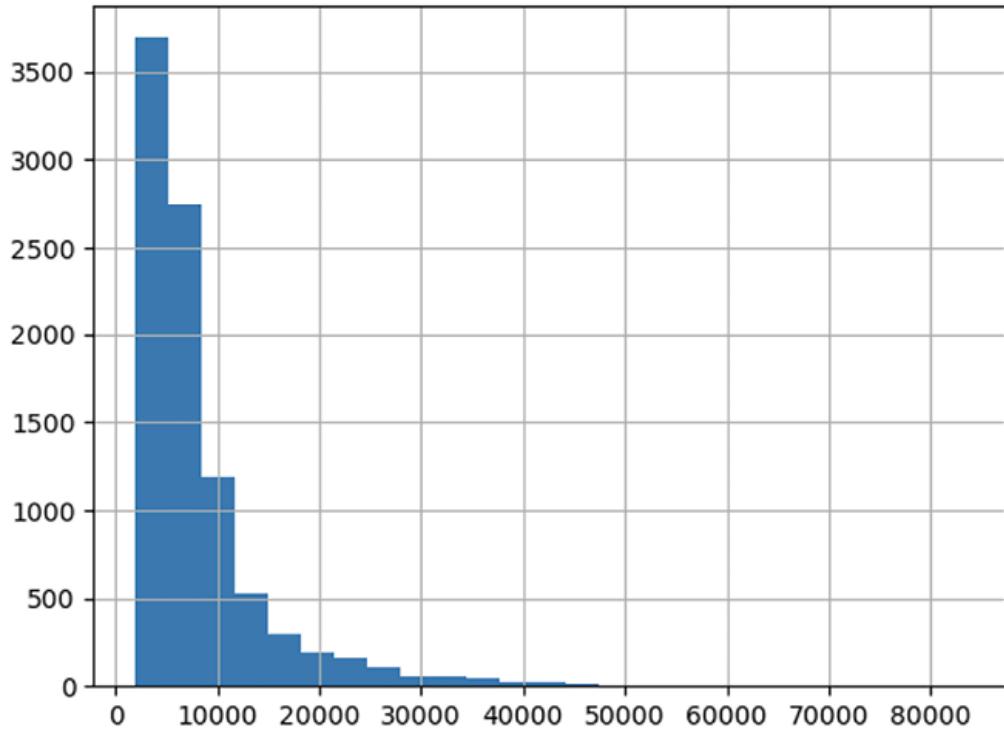


Figure 2.8: Histogram of CLV

As can be seen from the histogram chart, there is a long tail to the right, which corresponds with the outliers shown in the previous box plot and the output of the descriptive statistics.

From a marketing and business standpoint, this signifies that most customers will have a lifetime value below \$10,000 and there will be a subset of high-profile customers with lifetime values above \$10,000. As these groups will behave and react very differently to different marketing strategies, it will be wise to consider personalizing the marketing messages and strategies to different groups of customers. We will dive deeper into personalized marketing in *Chapter 7*; however, it is critical to have a good understanding that the CLV and other KPIs shown in this section show potential different groups that need to be targeted with different marketing messages.

# Geolocation and CLV

Not only should we look at the overall distribution of CLV when analyzing the lifetime values of customers but we should also look at it from various angles. One angle can be geolocation and how geolocation affects the CLV. In the example insurance marketing dataset, there is a column called `state`. We can separately look at CLVs based on different states, using the following code:

```
ax = df.groupby("State")["Customer Lifetime Value"].mean().plot  
    figsize=(5, 5),  
    rot=45  
)  
ax.bar_label(ax.containers[0], fmt='%.1f')  
ax.set_ylabel("Customer Lifetime Value ($)")  
ax.set_title("CLV per State")
```

The resulting bar chart looks like the following:

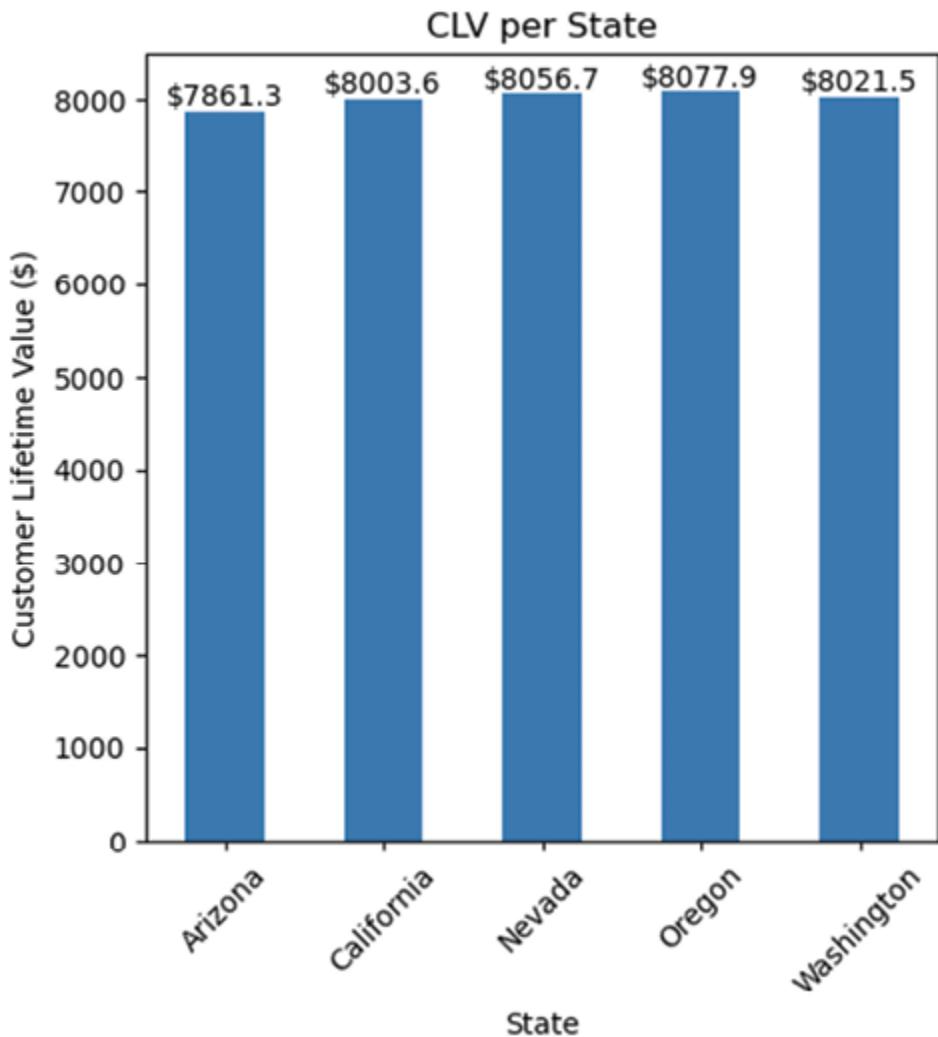


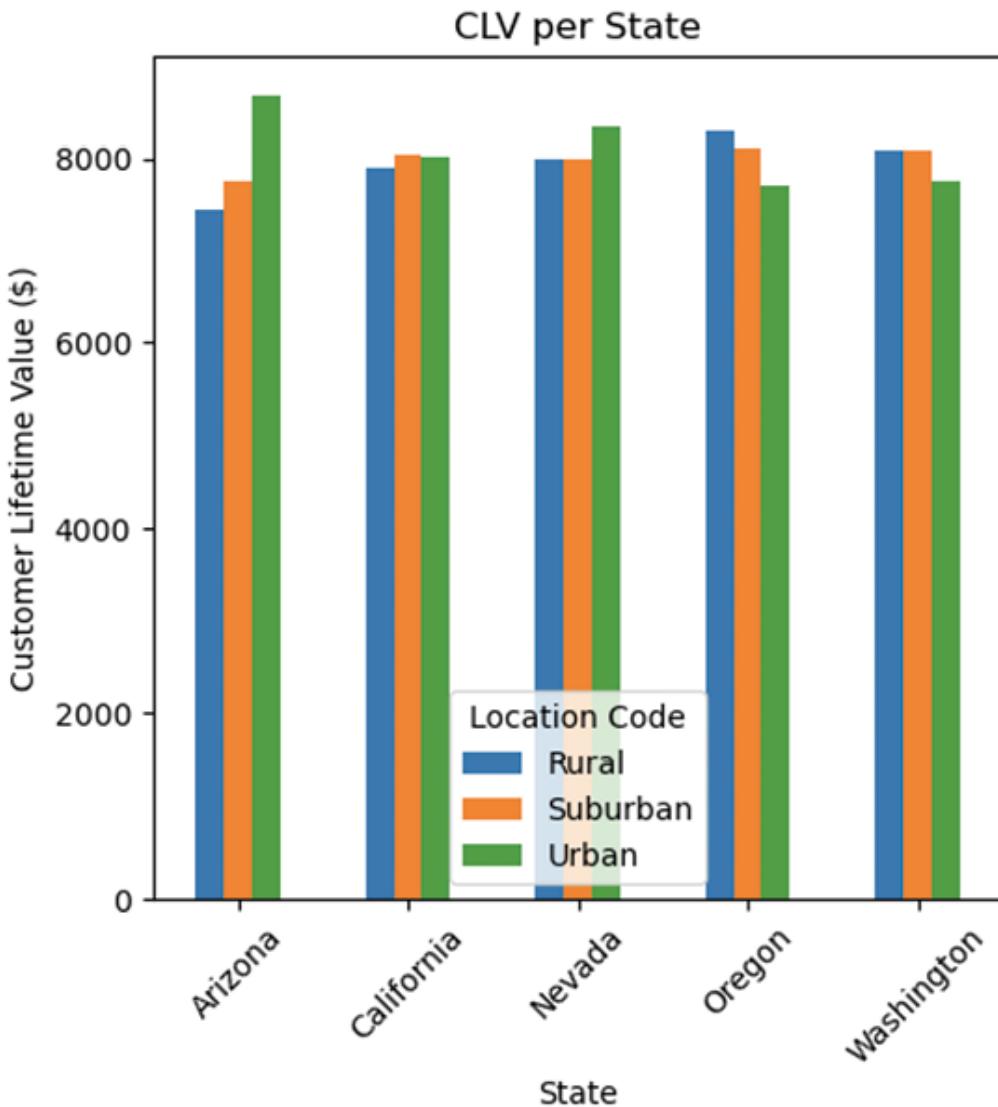
Figure 2.9: Bar plot of CLVs across different states

We have used the `groupby` function to group the data by the `state` column and took the average of `Customer Lifetime Value`. Using the `plot.bar` function, we can visualize the CLVs across different states. Although the average CLVs are similar across different states, `Oregon` has the highest average CLV and `Arizona` has the lowest.

As areas within the same state may also differ, we can go a level deeper into analysis by breaking it down further by the types of areas with the following code:

```
ax = df.groupby([
    "State", "Location Code"
])[[
    "Customer Lifetime Value"
].mean().unstack().plot.bar(
    figsize=(5, 5),
    rot=45
)
ax.set_ylabel("Customer Lifetime Value ($)")
ax.set_title("CLV per State")
```

This gives us the following output:



*Figure 2.10: Bar plot of CLVs across different states and locations*

Compared to the previous chart, we have now grouped by not only `State` but also by `Location Code`. This brings additional insights into how different types of area further affect the CLVs. Previously, we have seen that *Arizona* has the lowest overall CLV compared to other states; however, as this bar plot shows, the urban area of *Arizona* has the highest CLV and the decrease in the CLV was mostly driven by the `Rural` and `Suburban` areas within *Arizona*. Generally speaking, CLVs in urban areas are higher

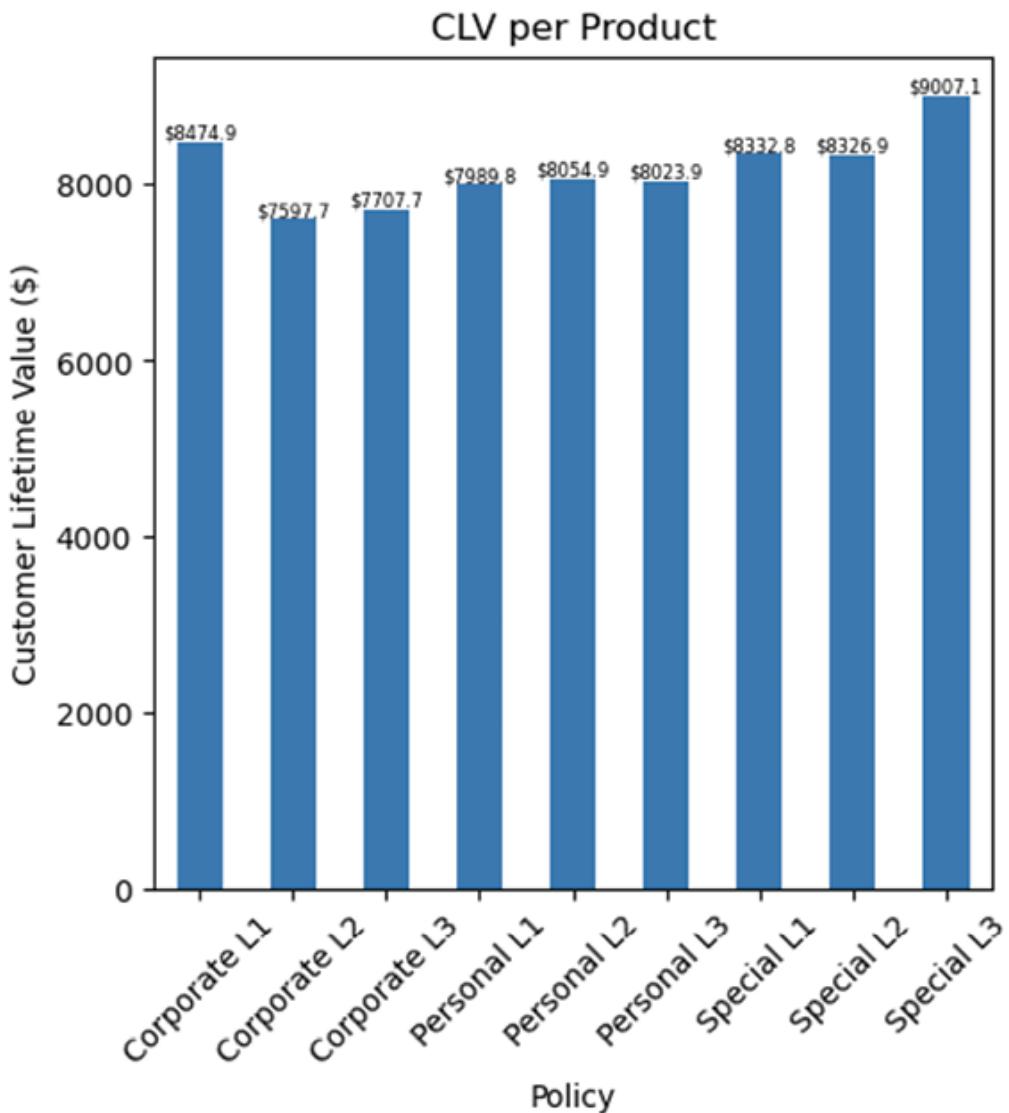
than in rural and suburban areas, except for *Oregon* and *Washington*. Rural areas within the `Oregon` and `Washington` states have higher CLVs than those of urban areas. As can be noted from this exercise, analyzing KPIs from different angles exposes different insights.

## Product and CLV

CLVs often differ by the products that customers purchase. We can examine this relationship in our dataset by looking at the `Policy` column, which is an insurance product that customers have purchased:

```
ax = df.groupby("Policy")["Customer Lifetime Value"].mean().plot  
    figsize=(5, 5),  
    rot=45  
)  
ax.bar_label(ax.containers[0], fmt='%.1f', fontsize=6)  
ax.set_ylabel("Customer Lifetime Value ($)")  
ax.set_title("CLV per Product")
```

This gives us the following bar plot:



*Figure 2.11: Bar plot of CLV per product*

By grouping by `Policy` and averaging CLV, we get the distribution as shown in the previous chart. It is noticeable that customers with `Special L3` policies or products have the highest CLV and those with `Corporate L2` have the lowest CLV. As the types of products that customers purchase are inherently related to the needs and appetites of the customers, they should be taken into consideration when targeting different customer bases. Having

personalized marketing strategies using predictive ML modeling can be a good solution to target different customer bases more efficiently.

## CPA

As mentioned previously, CPA is another critical piece to measure marketing effectiveness. Costs spent for acquiring new customers can vary by business. Some of the common marketing costs for new customer acquisition are for advertisements via different channels, such as emails, mail, TV, social media, promotional merchandise, telemarketing and customer service calls, and other tools, such as website promotions, chatbots, and customer relationship management systems.

As we have seen when analyzing conversion rate, in our example with the insurance marketing dataset, four channels were used to market insurance renewal offers: `Agent`, `Branch`, `Call Center`, and `Web`. Unfortunately, the data in this example does not have the costs spent for each sales channel; thus, we are going to make some assumptions.

Let us assume it costs about \$200 for each `Agent` sale, \$85 for each `Branch` sale, \$30 for each `Call Center` sale, and \$2 for each `Web` sale. In the real-world, you would track the marketing spending for each marketing channel you use. For example, you would need to know how much it costs per minute for each sales agent call, how much it costs for each online ad click, and how much it costs for each email or direct mail sent. With this assumption, we are going to assign *Estimated Acquisition Cost* for each customer as in the following:

```
import numpy as np
cost_distr = {
    "Agent": {"avg": 200, "std": 40},
```

```
        "Branch": {"avg": 85, "std": 17},  
        "Call Center": {"avg": 30, "std": 6},  
        "Web": {"avg": 2, "std": 0.5}  
    }  
    df["est_acquisition_cost"] = df["Sales Channel"].apply(  
        lambda x: np.random.normal(cost_distr[x]["avg"], cost_distr  
    )
```

As you can see from this code, we create a `cost_distr` dictionary with our assumptions for the costs of each marketing channel. Then, we use the `numpy` library's `random.normal` function to generate random numbers for each record from our assumptions and insert them into a column, `est_acquisition_cost`, which will be used in the following subsections:

```
df["est_acquisition_cost"].describe()
```

This is one way to create synthetic data from an assumed distribution. The resulting distribution should look something like the following:

```
count      9134.000000  
mean       106.349839  
std        83.081989  
min        0.458185  
25%        30.941641  
50%        87.969015  
75%        184.168216  
max        340.036821  
Name: est_acquisition_cost, dtype: float64
```

Figure 2.12: Distribution of the generated estimated acquisition cost

Given that these numbers are randomly generated from normal distributions, the exact numbers will vary each time this code is run. However, the overall distribution will look similar to this output.

## Overall CPA

As we have discussed in the previous section, CPA is calculated as the total marketing cost divided by the total number of conversions, which can be done in Python, as in the following code:

```
campaign_cost = df["est_acquisition_cost"].sum() / df["conversi
```

We simply sum across all the values in the `est_acquisition_cost` column, which will give us the total marketing spend, and divide that sum by the number of conversions, which is done by summing across all values in the `conversion` column:

```
print(f"${campaign_cost:.1f}")
```

```
$739.8
```

The value of `campaign_cost` should be similar to `$739.8`.

At the time of writing, the value of `campaign_cost` was `$739.8`; however, given the nature of randomness when we created synthetic data with our assumptions, this value may differ each time you run but should be in the ballpark.

## Sales channel and CPA

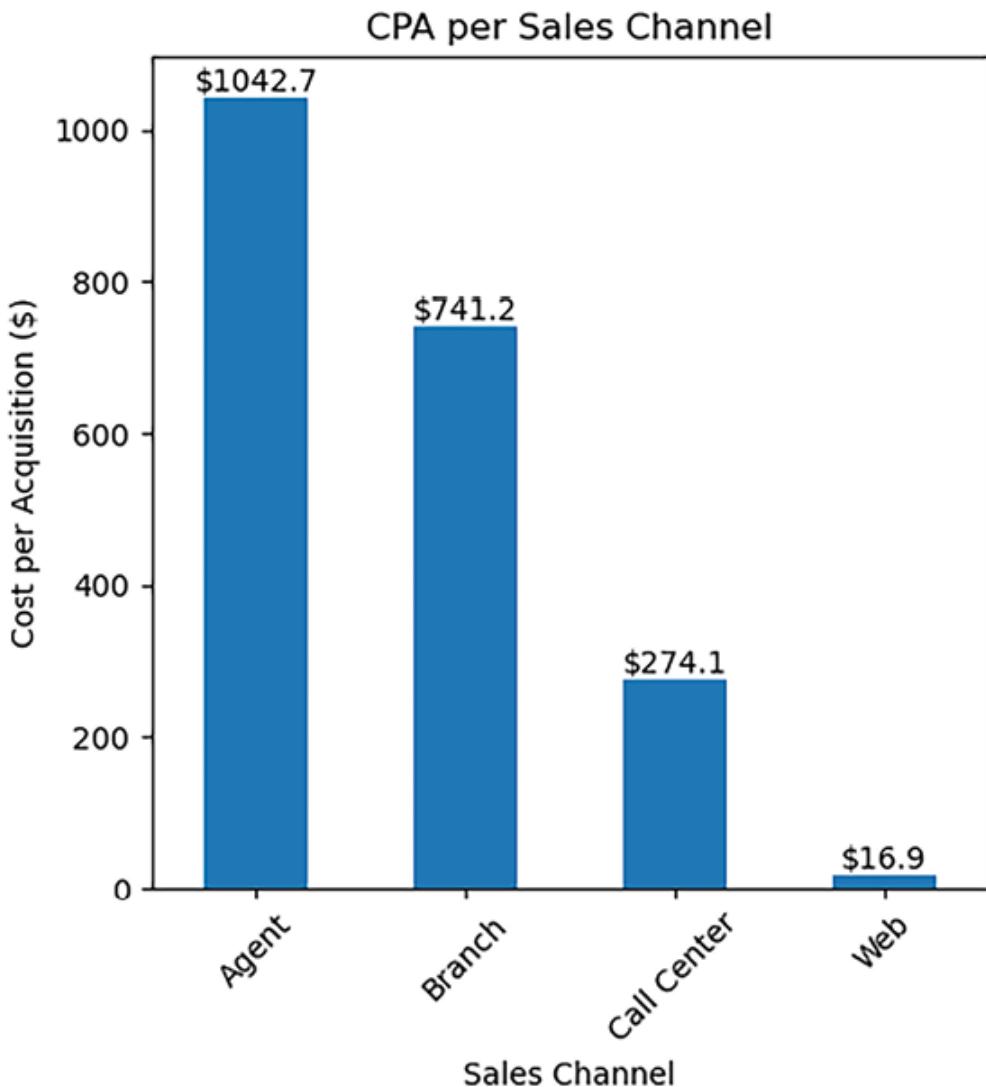
Due to the fact that different sales and marketing channels have different costs, CPA differs by channel. We can also observe this in our data. In order to calculate CPAs per channel, we simply need to calculate the total marketing cost for each channel and divide them by the number of conversions for each channel, as in the following code example:

```
channel_cpa = (
    df.groupby("Sales Channel")["est_acquisition_cost"].sum() /
    df.groupby("Sales Channel")["conversion"].sum()
)
```

In this code, we first apply the `groupby` function to the `Sales Channel` column and sum over the `est_acquisition_cost` column. Then, we apply the `groupby` function to the `Sales channel` column again, but this time, sum over the `conversion` column. When we divide the former `pandas Series` by the latter `pandas Series`, it divides the sum of `est_acquisition_cost` by the sum of `conversion` for each `Sales Channel` column:

```
ax = (
    channel_cpa
).plot.bar(
    figsize=(5, 5),
    rot=45
)
ax.bar_label(ax.containers[0], fmt='%.1f')
ax.set_ylabel("Cost per Acquisition ($)")
ax.set_title("CPA per Sales Channel")
```

The bar chart with the newly created variable `channel_cpa` looks like the following:



*Figure 2.13: Bar plot of CPAs across different sales channels*

As expected, the CPA is the highest with the `Agent` channel and the lowest with the `Web` channel. Despite the fact that the conversion rate is the highest from the `Agent` channel, as we saw in the previous section where we discussed conversion rates across different sales channels, the **CPA** is still the highest. This is a good example to show that managing the cost of marketing spending is as important as having high conversion rates.

# Promotions and CPA

Similar to how CPAs differ by sales channels, CPAs may also differ by the types of marketing promotions. We can observe this from our dataset as well. In order to calculate the CPAs for each promotion, you can use the following code:

```
promo_cpa = (
    df.groupby("Renew Offer Type")["est_acquisition_cost"].sum()
    /
    df.groupby("Renew Offer Type")["conversion"].sum()
)
```

Here, we group by the `Renew Offer Type` column and sum across the `est_acquisition_cost` and `conversion` columns, then divide the former by the latter. The resulting bar plot looks as follows:

```
ax = (
    promo_cpa
).plot.bar(
    figsize=(5, 5),
    rot=45
)
ax.bar_label(ax.containers[0], fmt='%.1f')
ax.set_ylabel("Cost per Acquisition ($)")
ax.set_title("CPA per Promotion")
```

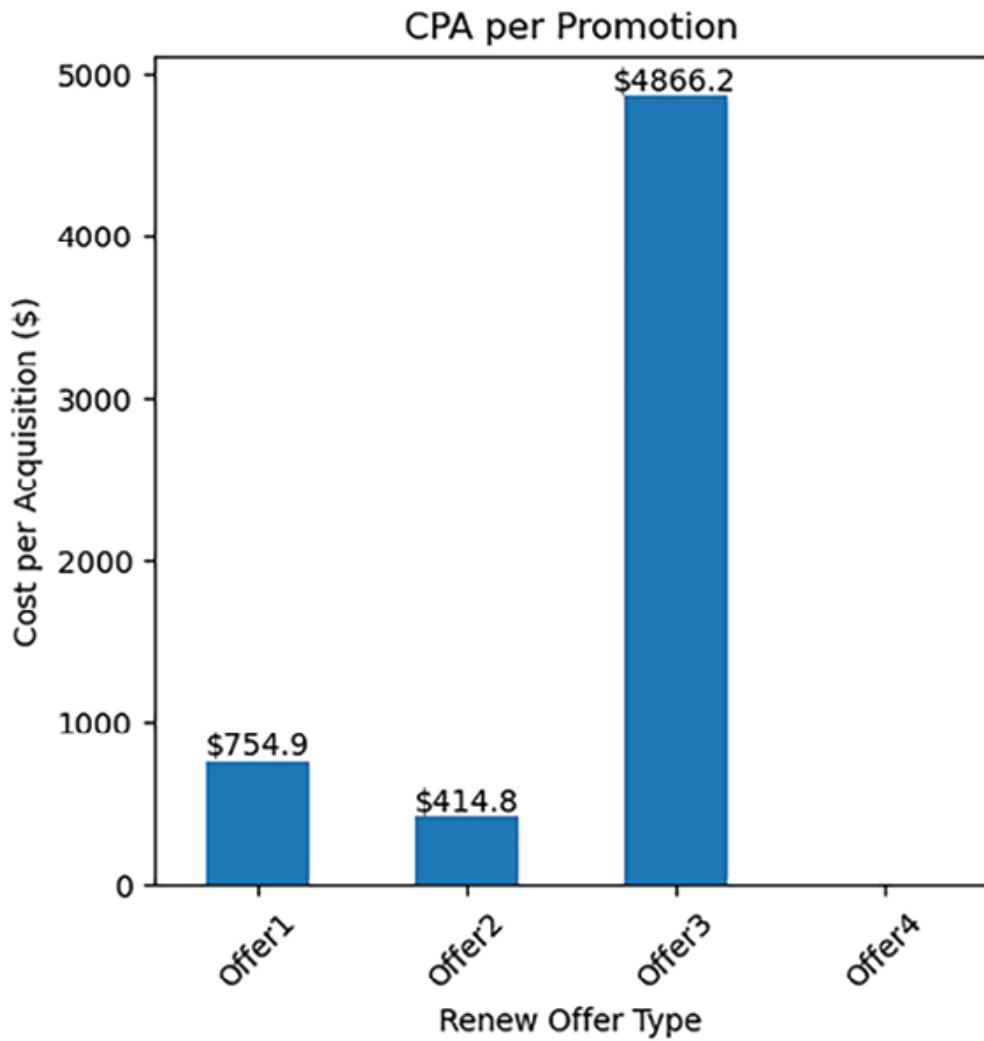


Figure 2.14: Bar plot of CPAs across different promotions

There are two things that stand out from this chart. One is that the CPA for `offer3` is the highest and the other is that the CPA for `offer4` is not shown. The reason why the CPA for `offer4` is not shown in this chart is that there were no conversions for `offer4`.

This can be found when you dive deeper into the conversion rates for each promotion, as in the following:

```
df.groupby("Renew Offer Type")["conversion"].mean()
```

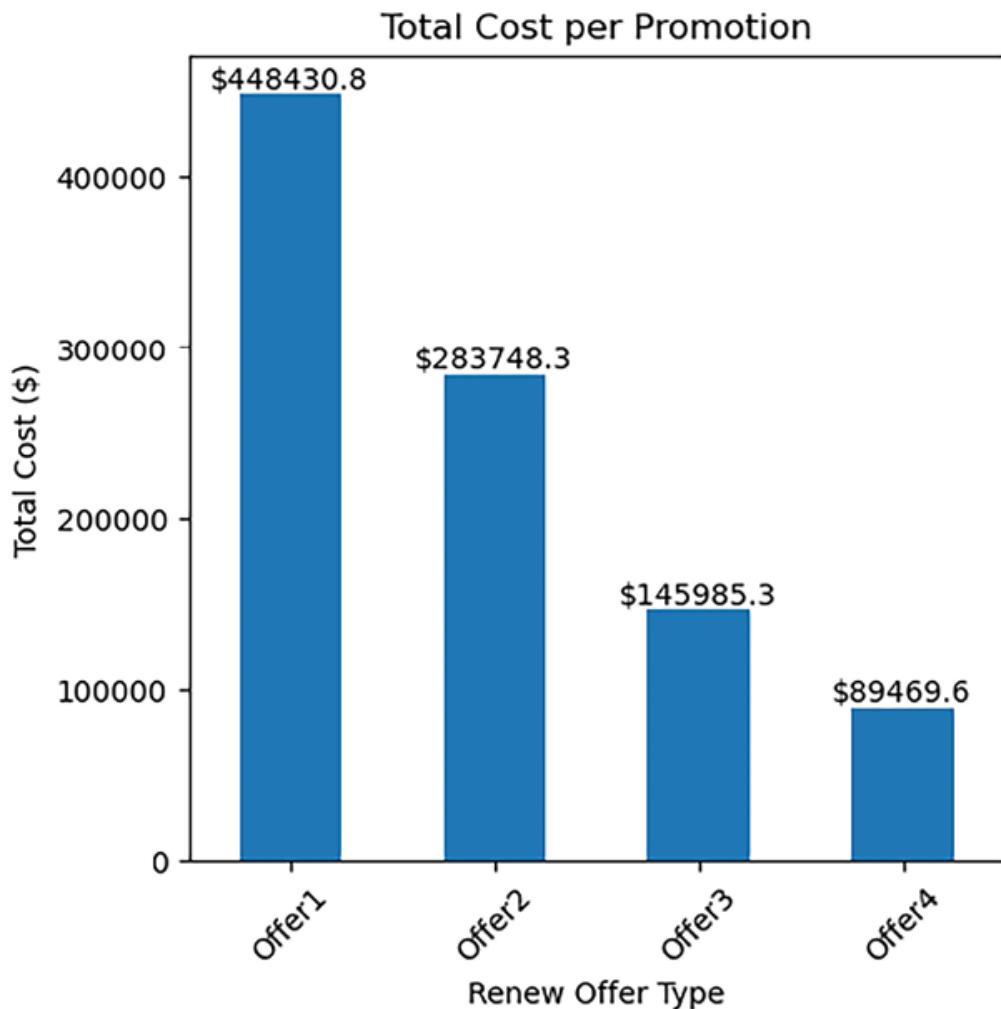
```
Renew Offer Type
Offer1    0.158316
Offer2    0.233766
Offer3    0.020950
Offer4    0.000000
Name: conversion, dtype: float64
```

*Figure 2.15: Conversion rates across different renewal offer types*

Although this answers why the CPA for `offer4` was not shown in the previous bar plot, as there was no conversion, this still does not fully answer the reason why the CPA for `offer3` was so high as there still are some conversions. What is more interesting is that the total marketing spend for `offer3` was the second lowest among the four promotions, as shown in the chart generated using the following code:

```
ax = df.groupby("Renew Offer Type")["est_acquisition_cost"].sum
      figsize=(5,5),
      rot=45
)
ax.bar_label(ax.containers[0], fmt='%.1f')
ax.set_ylabel("Total Cost ($)")
ax.set_title("Total Cost per Promotion")
```

Here is the bar plot:



*Figure 2.16: Bar plot of total costs across different promotions*

This is a good example of how a low conversion rate offsets the low marketing spend, and vice versa. `offer1` has the highest marketing spend and close to 3 times more spending than `offer3`, so the high conversion rate of `offer1` offsets the cost of marketing, which results in a much lower CPA than `offer3`. On the other hand, `offer3` has the second lowest marketing spend but has the highest CPA, because the low conversion rate of `offer3` offsets the low marketing spend. This signifies the necessity of analyzing KPIs in various aspects and understanding the key factors in the rise and fall of certain KPIs.

# ROI

We have come a long way to finally be able to compute the ROI of our marketing initiative. Those conversion rates, CLVs, and CPAs we have analyzed so far will be combined to get the ROI and decide if the marketing money was well spent.

## Overall ROI

In a nutshell, **ROI** is total returns or earnings divided by total spending. One way to approach it within our example is to sum across `Customer Lifetime Value` for those customers that converted and divide it by the sum across the `est_acquisition_cost` column, as in the following code:

```
campaign_cost = df["est_acquisition_cost"].sum()
converted_clv = (df["Customer Lifetime Value"] * df["conversion
print(f"Total CLV: ${converted_clv:, .02f}")
print(f"Total Cost: ${campaign_cost:, .02f}")
print(f"Overall ROI: {(converted_clv - campaign_cost) / campaign_co
```

The output is as follows:

```
Total CLV: $10,274,171.74
Total Cost: $972,130.58
Overall ROI: 9.6x
```

As this output shows, the total CLV added from the converted customers is about \$10.3M and the total marketing spend is about \$968K within our example. This gives us an ROI of 9.6x or 960%, which suggests that the value addition from this marketing initiative outweighs the spending by multiples of 9.6. If this were a real-world marketing campaign with such

performances, this would have been a stellar marketing initiative. Typically, an ROI of 5:1 is a solid outcome, and anything higher is considered great. An ROI of 2:1 or below is considered disappointing.

Another effective way of visualizing this KPI is to visually show value inflows vs. outflows. One way of showing this is using a waterfall chart, as shown in the following:

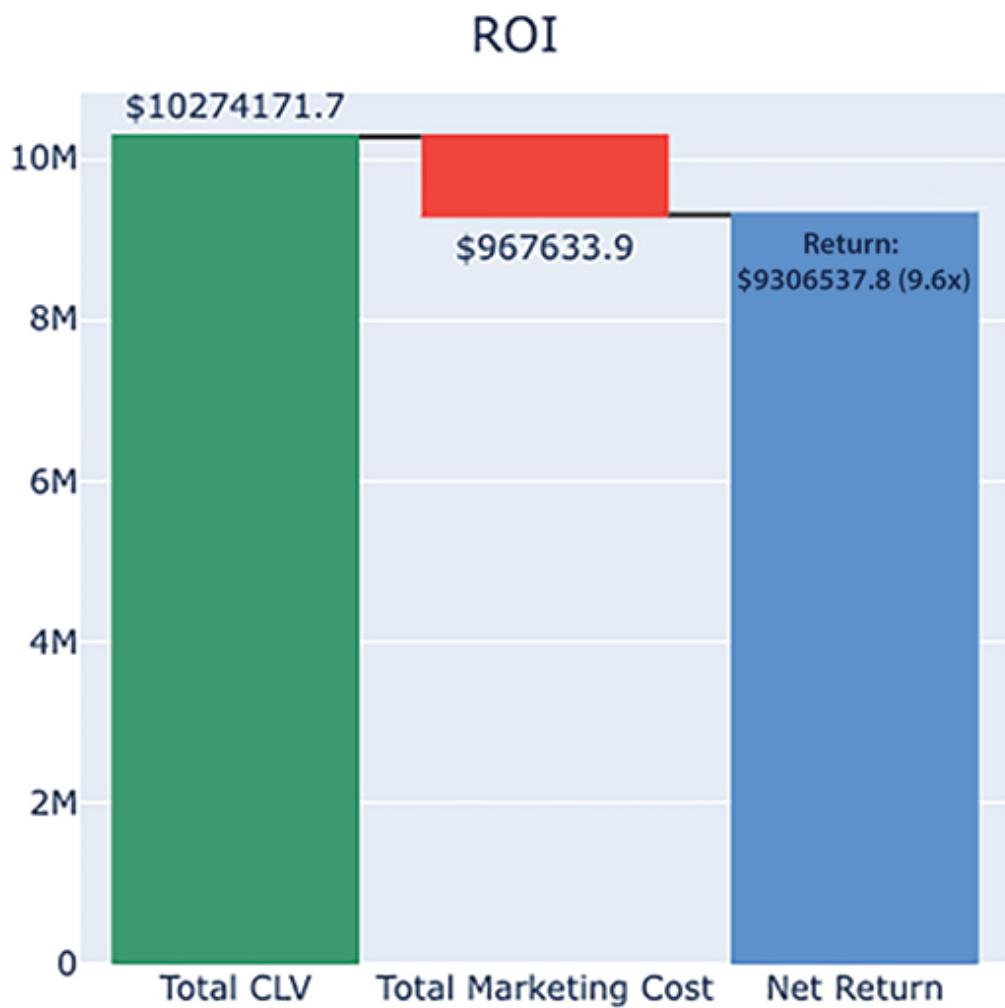


Figure 2.17: Waterfall chart of Net Return

As you can imagine, you can break this down by factors contributing to value additions and subtractions. For example, you can visualize value

additions by each sales channel and how it sums up to the total CLV added, and value subtractions by each sales channel and how it sums up to the total marketing cost. Waterfall charts are a great way to visualize value additions and subtractions and show the net return in one view. The following code can be used to visualize the waterfall chart using the `plotly` library:

```
import plotly.graph_objects as go
fig = go.Figure(
    go.Waterfall(
        name = "Waterfall",
        orientation = "v",
        measure = ["relative", "relative", "total"],
        x = ["Total CLV", "Total Marketing Cost", "Net Return"],
        textposition = "outside",
        text = [
            f"${{converted_clv:.01f}}",
            f"${{campaign_cost:.01f}}",
            f"Return: ${{converted_clv-campaign_cost:.1f}} ({{con
        ],
        y = [converted_clv, -campaign_cost, 0],
        connector = {"line":{"color":"rgb(63, 63, 63)"}},
    )
)
fig.update_layout(
    height=500,
    width=500,
    title = "ROI",
    title_x = 0.5,
    showlegend = True
)
fig.show()
```

As you can see from this code, you can use the `Waterfall` object within the `plotly` library to create a waterfall chart. The key parameters here are:

- `measure`: This is an array of the types of values. Use `relative` to show the relative values, such as additions and subtractions, and use `total` to calculate the sum value.
- `x`: This is an array of x coordinates, where it is an array of labels of each addition and subtraction.
- `y`: This is an array of y coordinates, where it is an array of addition and subtraction values.



Waterfall charts can be an intuitive and efficient way of showing value ins and outs. Try breaking it down to further show factors contributing to value additions and subtractions!

## ROI per sales channel

As briefly mentioned in the previous subsection, analyzing which factor contributes to ROI in which way is important to help understand which marketing strategies or channels work the most efficiently and which do not. The first angle we are going to look at for ROI is *Sales Channel*. The first thing we are going to do is calculate the net return and compute the ROI per sales channel, as in the following:

```
df["return"] = (
    df["Customer Lifetime Value"] * df["conversion"]
) - df["est_acquisition_cost"]
channel_roi = (
    df.groupby("Sales Channel")["return"].sum()
/
    df.groupby("Sales Channel")["est_acquisition_cost"].sum()
)
```

This code creates a new variable, named `return`, which is the CLV minus acquisition cost, and for those customers who have not converted, it is simply a negative acquisition cost, as there is no value addition. Then, the ROI per channel is calculated by grouping it by the `Sales Channel` column and dividing the sum of `return` by the sum of `est_acquisition_cost`. In order to visualize the ROI per sales channel, the following code can be used:

```
ax = (
    channel_roi
).plot.bar(
    figsize=(5, 5),
    rot=45
)
ax.bar_label(ax.containers[0], fmt='%.1fx')
ax.set_ylabel("Return on Investment (multiple)")
ax.set_title("ROI per Sales Channel")
```

The resulting chart looks as follows:

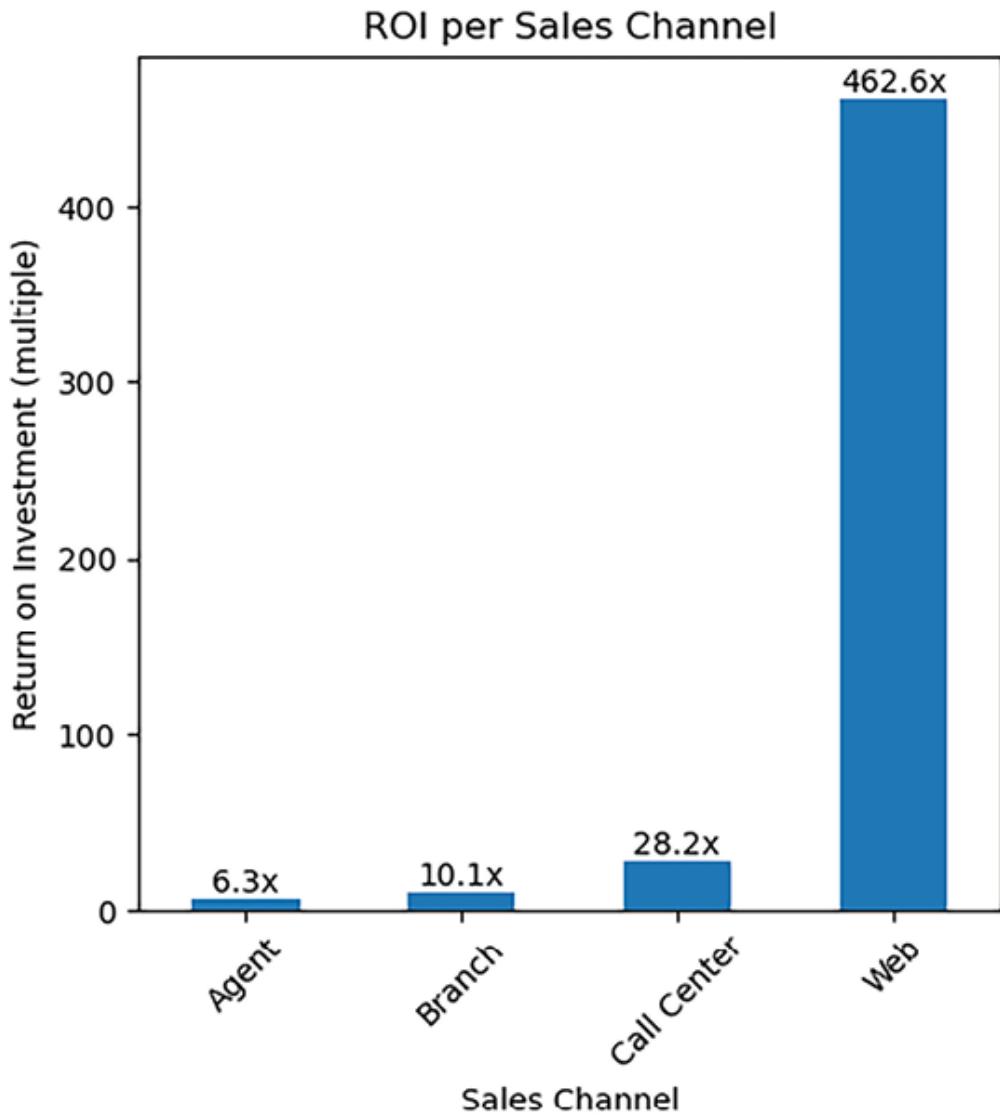


Figure 2.18: Bar plot of ROI across different sales channels

This bar plot suggests the ROI from the `Web` channel is the highest, followed by the `Call Center` channel, and the `Agent` channel is the lowest. As you can imagine, ROI is negatively correlated with CPA and positively correlated with CLV.

The low CPA of the `Web` channel would have been the highest contributing factor for this extremely high ROI, given that the conversion rate for this sales channel was similar to other channels. On the other hand, the high

CPA of the `Agent` channel would have been the most critical factor for a relatively low ROI, as its conversion rate was the highest among the four channels. As you can see, ROI and its breakdowns are insightful KPIs to gain knowledge on how best to manage marketing costs and generate returns.

## ROI per promotion

One last factor we will look at to understand the dynamics of ROIs across different angles is the promotion types. We will look at how `Renew Offer Type` affects the ROI, as shown in the following code:

```
promo_roi = (
    df.groupby("Renew Offer Type")["return"].sum() /
    df.groupby("Renew Offer Type")["est_acquisition_cost"].sum()
)
```

As shown in the code, we group by the `Renew Offer Type` column and divide the sum of `return` by the sum of `est_acquisition_cost`. This can be visualized in a bar chart using the following code:

```
ax = (
    promo_roi
).plot.bar(
    figsize=(5, 5),
    rot=45,
    color=(promo_roi > 0).map({True: 'cornflowerblue', False: 'red'})
)
ax.bar_label(ax.containers[0], fmt='%.1fx')
ax.set_ylabel("Return on Investment (multiple)")
ax.set_title("ROI per Promotion")
```

We get a bar chart like this:

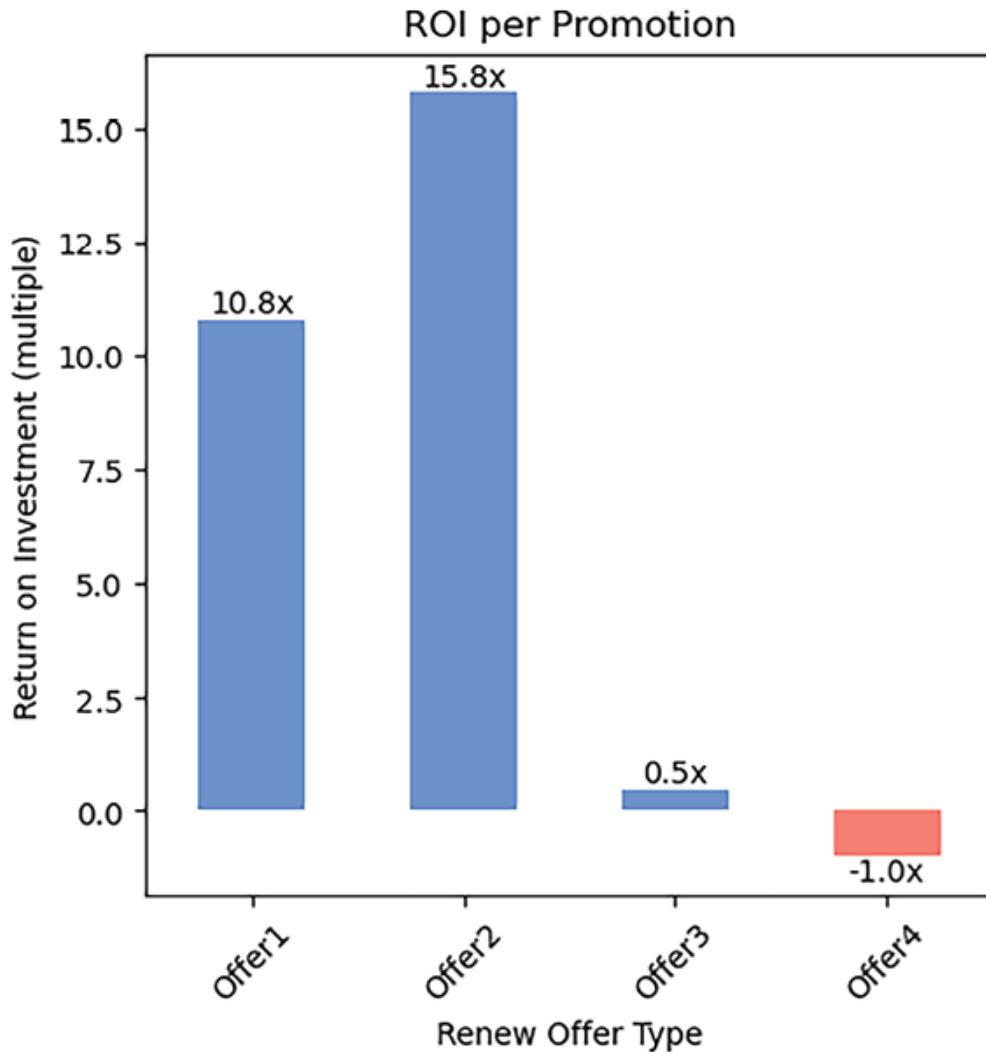


Figure 2.19: Bar chart of ROI among different offers

This bar chart shows that `offer2` has the highest ROI, followed by `offer1`, and `offer4` is the lowest. As previously discovered, there were no conversions for `offer4`; thus, all the marketing spending was lost, resulting in a -100% return or -1x ROI.

As expected from the previous discussion around CPA, `offer3` has the lowest ROI after `offer4`. This is likely due to the high CPA that `offer3`

had, which negatively affected ROI. On the other hand, CPA for `offer2` was the lowest, which helped it to have the highest ROI among the promotion offers.

## Correlations across KPIs

As we have seen so far, there are multiple factors contributing to ROI, which, in turn, necessitates conducting analyses from various points of view. We have examined the basics and the interpretations of KPIs that ultimately touch from the *Engagement* stage to the *Conversion* stage up until this section.

Not only can we examine KPIs with a set of defined factors but we can also leverage a more data-driven and statistical approach to uncover the relationships between various metrics. One approach may be to analyze the correlations among KPIs, as well as between KPIs and other factors. A *heatmap* is a clever way to visualize these relationships and provide a more intuitive understanding of hidden relationships within the data. This analytical technique reveals how different KPIs are interconnected with each other as well as how individual factors are associated with the KPIs, offering strategic insights for optimizing marketing campaigns. The following code can be used to build a heatmap using our example data:

```
import seaborn as sns
import matplotlib.pyplot as plt
correlation_matrix = df[['
    'Income', 'Monthly Premium Auto',
    'Months Since Last Claim', 'Months Since Policy Inception',
    'Number of Open Complaints', 'Number of Policies', 'Total C
    'conversion', 'Customer Lifetime Value', 'est_acquisition_c
]].corr()
plt.figure(figsize=(10, 8))
```

```

sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix Heatmap of Marketing KPIs')
plt.show()

```

This code generates the following heatmap:

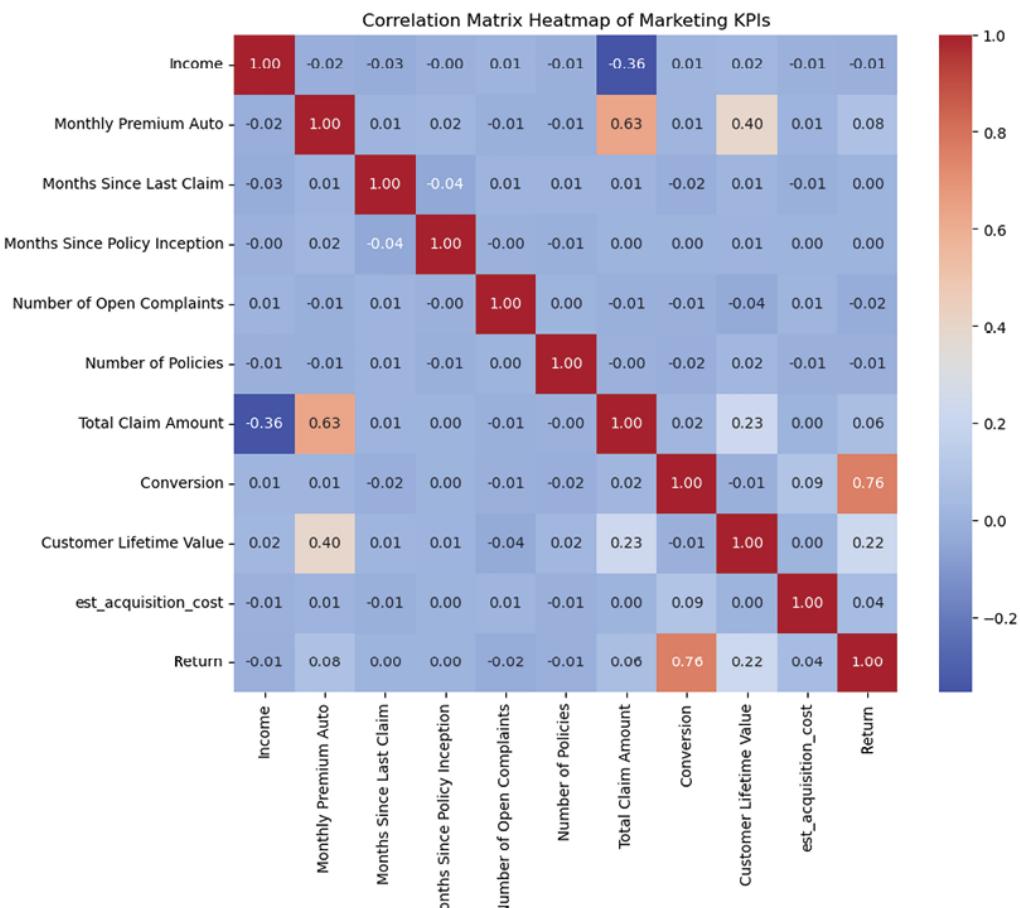


Figure 2.20: Heatmap of relationships among variables

Notably, the correlation between *Monthly Premium Auto* and *Total Claim Amount* stands out with a coefficient of **0.632**, indicating a strong positive relationship. This suggests that as the monthly premium for auto insurance increases, so does the total claim amount. Such insights are invaluable for

insurance companies looking to balance premium rates with claim costs to maximize profitability.

Additionally, the *Income* and *Total Claim Amount* show a significant negative correlation of **-0.355**, highlighting that higher-income customers tend to have lower total claim amounts. This could imply that higher-income segments might represent a lower risk profile, potentially guiding targeted marketing strategies or premium adjustments.



Be creative when analyzing and building KPIs and focus on building a good data story!

## Tracking marketing performance with KPIs

Computing and building KPIs is one key aspect of decoding marketing performances, but staying on top of the progressions and changes in KPIs is another equally important aspect of having a good grasp of your marketing performance. There are numerous ways to visualize KPIs and countless ways of tracking and sharing the KPIs with key stakeholders. In this section, we are going to share some of the common approaches to visualizations and ongoing KPI tracking.

## Choosing the right visualization for KPIs

When presenting KPI data, the choice of visualization plays a crucial role in conveying insights effectively. Here are some guidelines to help choose the

most appropriate type of visualization:

- **Bar charts:** Ideal for comparing quantities across distinct categories, such as sales performance across different regions or products.
- **Pie charts:** For illustrating a composition or proportion of a whole, like market share distribution.
- **Scatter plots:** Use these to explore relationships between two variables, such as ad spend versus sales revenue.
- **Line charts:** Suitable for showing trends over time, such as website traffic growth or sales trends.
- **Area charts:** Typically used for comparing one or more series over time, or numeric values, such as website traffic across different channels.
- **Heat maps:** Effective for showcasing correlations or patterns within large datasets, such as the relationship between various marketing KPIs.
- **Funnel charts:** Perfect for visualizing stages in a process, particularly useful for conversion rates across a marketing or sales funnel.
- **Waterfall charts:** Often used for showing additions and subtractions that sum up to the total, such as cash flows.

## Ongoing KPI tracking

Now that we have experimented with how KPIs can be computed in Python given a dataset, you may wonder what is next. You may also question whether you must run your Python scripts each time you'd like to report KPI metrics. The approach generally taken and recommended for ongoing KPI tracking is utilizing accessible dashboards.



## What is a dashboard?

A dashboard is a way of showing diverse types of visuals of data, so that information can be shared and digested easily by stakeholders. It includes KPIs and other business metrics that need to be monitored. The scope of dashboards is not limited to tracking just marketing KPIs and it can be used in various lines of business, such as finance, operations, technology, customer service, and so forth.

In the business setting, dashboards are typically connected to data warehouses, so that the data being visualized is ingested directly from the data warehouse. This enables real-time or close-to-real-time updates to the dashboards. If the data in the data warehouse is updated, the visuals on the dashboards are automatically updated as well. Often, the data is complex, messy, or not easy to digest, so data ingestion pipelines are built to transform data into a more digestible structure. A typical flow of data to dashboards for tracking marketing KPIs looks as follows:

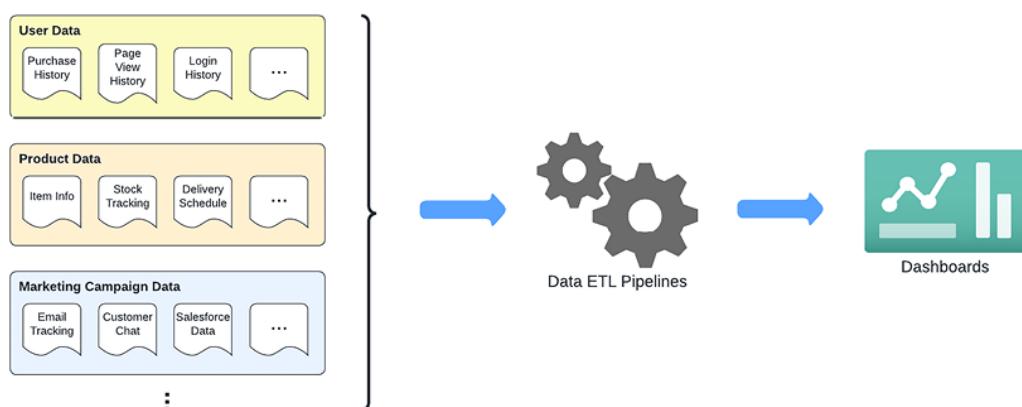


Figure 2.21: Typical flow of data to dashboards

A detailed discussion of data ETL pipeline development is out of scope for this book, but it is a process that *extracts* raw data from various sources, *transforms* it into a format that is easily usable, and *loads* it into data storage. The three key steps, extract, transform, and load, make up the acronym ETL. The outcome of an ETL pipeline for marketing data is easily digestible data for marketing KPI calculations, which are then summarized and visualized on the marketing dashboards. Some of the commonly used dashboard software products in the industry are Tableau, Power BI, and Looker.



Good KPIs and dashboards should provoke meaningful action items and assist decision-making! Consider building goal-oriented KPIs and dashboards that are specific to each marketing goal and initiative.

## Summary

In this chapter, we have laid a concrete foundation for building data and AI/ML-driven marketing models and strategies. We have discussed commonly used marketing KPIs that not only help the business measure the marketing performance but also provoke action items for improvements based on the strengths and weaknesses discovered through multilevel analyses. Using an insurance product marketing dataset as an example, we have seen how these KPIs can be measured and analyzed with Python for further improvements and optimizations in future marketing efforts.

We have also discussed how various dashboarding tools, such as Tableau, Power BI, and Looker, can be used for reusable real-time KPI tracking. As

everyone emphasizes, AI/ML starts with data and deep exploratory analysis to decide what to train AI/ML models with and what to optimize for. The items and KPIs covered in this chapter will come in handy when designing and developing advanced AI/ML models for marketing in the future.

In the next chapter, we are going to learn how we can utilize data science and machine learning algorithms to uncover the drivers behind marketing successes and failures. More specifically, we will discuss how regression analysis and models can be used to dissect the factors affecting marketing engagement, how tree-based models can be used to identify drivers behind conversions, and how causal inference can help understand the reasoning behind customer churn.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)

# 3

## Unveiling the Dynamics of Marketing Success

**Key performance indicators (KPIs)** are great at capturing the current status or results of marketing initiatives. However, they lack the ability to show what drives such results and how different factors may have affected the end results of marketing campaigns. We briefly looked at how you can further analyze relationships between various variables and KPIs with correlation analysis in the last chapter. In this chapter, we are going to delve deeper into how we can utilize **data science (DS)** and **machine learning (ML)** techniques and tools to unveil the dynamics of marketing success.

This chapter touches on three critical phases of the customer journey: engagement, conversion, and churn. These three types of customer behaviors are the key components and marketing goals to optimize for, and marketers often question what factors actually affect certain outcomes, such as why people engage more or less, why people end up converting more or less, and why people churn or don't churn. The three most common approaches that are taken to unveil the factors that affect marketing outcomes, on top of the correlation analysis we have done in the last chapter, are (1) regression analysis, (2) decision tree interpretation, and (3) causal inference. In this chapter, we will discuss how we can perform those

ML techniques in Python to have a better and more in-depth interpretation of the drivers behind marketing successes and failures.

In this chapter, we will cover the following topics:

- Why people engage with regression analysis
- Why people convert with decision tree interpretation
- Why people churn with causal inference

## Why people engage in regression analysis

Usually, increasing the engagement rate is the very first step in marketing. Potential customers need to engage with the company or products first before they can convert into paying customers and then, eventually, repeat or loyal customers. There will be numerous factors that affect how potential customers engage with or click on your marketing campaigns. Customers with certain socio-economic backgrounds may be more attracted to your services and/or products than others. People from certain regions may engage more frequently in your marketing campaigns than others. Some who have been exposed to your previous sales may have a more favorable appetite for your services or products that results in higher engagement rates than others.

Understanding how potential customers interact with different components is the key to achieving higher engagement rates and the start of targeted or personalized marketing. **Regression analysis**, a well-known and widely used technique across various domains and often considered a fundamental concept in ML, is an intuitive and effective approach that we can use to understand the interactions among various components that may affect

higher or lower engagement among your target customers. The two most commonly used types of regression analysis are **linear regression** and **logistic regression**.

A linear regression assumes a linear relationship between a target variable and other factors. The linear relationship is defined as:

$$Y = a + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n$$

where  $Y$  is the target variable, which is the outcome that you observe, each  $x_i$  is the factor that may affect the outcome, each  $b_i$  is the coefficient and degree of impact each factor has on the target variable, and  $a$  is the intercept. Linear regression is used when the target variable is a continuous variable, such as customer lifetime value, sales volume, and customer tenure.

Logistic regression, on the other hand, is used when the target variable is binary, such as `Pass` vs. `Fail`, `True` vs. `False`, and 1 vs. 0. When linear regression estimates the value of an outcome, logistic regression estimates the odds of success, and the relationship is defined as:

$$\log\left(\frac{P(y = 1)}{1 - P(y = 1)}\right) = a + b_1 * x_1 + b_2 * x_2 + b_3 * x_3 + \dots$$

Where  $\log$  denotes the natural logarithm and  $P(y = 1)$  is the probability of the outcome being 1

As logistic regression estimates the probability of an outcome, it is a great tool to use when understanding the impacts of various factors on a binary outcome, such as whether a customer will respond to email marketing or whether a customer will click on an email message. In this section, we will

use an auto insurance marketing dataset to discuss how logistic regression can be used to unveil the drivers behind customer engagement.



#### Source code and data:

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/blob/main/ch.3/Why%20People%20Engage.ipynb>

Data source:

<https://www.kaggle.com/datasets/pankajjs/h06/ibm-watson-marketing-customer-value-data>

## Target variable

The first thing we need to do is define the target or outcome we want to analyze. As we are interested in understanding how various factors affect customer engagement, we will have the `Response` variable as our target variable, which tells us whether a customer responded to a marketing call or not. The following code can be used for encoding the target variable with `0` for when a customer did not respond and `1` for when a customer did respond:

```
import pandas as pd
df = pd.read_csv("./engage-data.csv")
df["Engaged"] = df["Response"].apply(lambda x: 1 if x == "Yes"
```

This code first loads the data into a variable, `df`, by using the `pandas` library's `read_csv` function, and we create a new variable, `Engaged`, by encoding `Yes` as `1` and `No` as `0` from the `Response` column.

## Continuous variables

There are largely two types of variants of factors that may affect the outcome or target variable:

- **Continuous variables:** These are the variables that have real numbers as the values and can include fractions or decimals, such as monetary amounts, sales numbers, and pageview counts.
- **Categorical variables:** These are discrete variables that have a distinct set of options or categories as values, such as education levels, genders, and customer account types.

We will first look at the relationships between some of the continuous variables in our auto insurance marketing dataset and the target variable, `Engaged`:

```
df.describe()
```

	Customer_Lifetime_Value	Income	Monthly_Premium_Auto	Months_Since_Last_Claim	Months_Since_Policy_Inception	Number_of_Open_Complaints	Number_of_Policies	Total_Claim_Amount
count	9134.000000	9134.000000	9134.000000	9134.000000	9134.000000	9134.000000	9134.000000	9134.000000
mean	8004.940475	37657.580000	93.219291	15.097000	48.646594	0.384388	2.961710	434.688794
std	6870.967608	30379.904734	34.407967	10.073257	27.905991	0.916384	2.390182	290.500092
min	1898.007675	0.000000	61.000000	0.000000	0.000000	0.000000	1.000000	0.099007
25%	3994.251794	0.000000	68.000000	6.000000	24.000000	0.000000	1.000000	272.558244
50%	5780.182197	33889.500000	83.000000	14.000000	48.000000	0.000000	2.000000	383.945434
75%	8962.167041	62320.000000	103.000000	23.000000	71.000000	0.000000	4.000000	547.514839
max	83325.381190	99981.000000	298.000000	35.000000	99.000000	5.000000	9.000000	2893.239678

Figure 3.1: Distributions of continuous variables in the dataset

As this screenshot shows, when we run the `df.describe()` function, it shows the overall statistics of the continuous variables. In our auto insurance example dataset, there are eight continuous variables:

```
Customer_Lifetime_Value, Income, Monthly_Premium_Auto,  
Months_Since_Last_Claim, Months_Since_Policy_Inception,  
Number_of_Open_Complaints, Number_of_Policies, and  
Total_Claim_Amount.
```

In order for us to analyze the relationships between these variables and the outcome variable, `Engaged`, we are going to use `Logit` within the `statsmodels` package. The code to fit a logistic regression model is as follows:

```
import statsmodels.api as sm  
continuous_vars = [  
    x for x in df.dtypes[(df.dtypes == float) | (df.dtypes == int)]  
]  
logit = sm.Logit(  
    df['Engaged'],  
    df[continuous_vars])  
.fit()
```

Here, we first import the required module, `statsmodels.api`, and define the continuous variable, `continuous_vars`. Then, we can use the `Logit` object in the `statsmodels` package to train a logistic regression model. We can view the trained logistic regression model with the following code:

```
logit.summary()
```

The output of this code looks as in the following:

Logit Regression Results						
Dep. Variable:	Engaged	No. Observations:	9134 <th data-cs="3" data-kind="parent"></th> <th data-kind="ghost"></th> <th data-kind="ghost"></th>			
Model:	Logit	Df Residuals:	9126 <th data-cs="3" data-kind="parent"></th> <th data-kind="ghost"></th> <th data-kind="ghost"></th>			
Method:	MLE	Df Model:	7			
Date:	Wed, 20 Mar 2024	Pseudo R-squ.:	-0.02546			
Time:	07:27:34	Log-Likelihood:	-3847.1			
converged:	True	LL-Null:	-3751.6			
Covariance Type:	nonrobust	LLR p-value:	1.000			
	coef	std err	z	P> z	[0.025	0.975]
Customer_Lifetime_Value	-6.741e-06	5.04e-06	-1.337	0.181	-1.66e-05	3.14e-06
Income	-2.857e-06	1.03e-06	-2.766	0.006	-4.88e-06	-8.33e-07
Monthly_Premium_Auto	-0.0084	0.001	-6.889	0.000	-0.011	-0.006
Months_Since_Last_Claim	-0.0202	0.003	-7.238	0.000	-0.026	-0.015
Months_Since_Policy_Inception	-0.0060	0.001	-6.148	0.000	-0.008	-0.004
Number_of_Open_Complaints	-0.0829	0.034	-2.424	0.015	-0.150	-0.016
Number_of_Policies	-0.0810	0.013	-6.356	0.000	-0.106	-0.056
Total_Claim_Amount	0.0001	0.000	0.711	0.477	-0.000	0.000

Figure 3.2: Summary of logistic regression results

Let's analyze this logistic regression output. The two most important things to look at are, *coef* and *P>|z|*. The leftmost column, *coef*, in the bottom table is the coefficient for each variable. For example, the coefficient for the variable `Monthly_Premium_Auto` is `-0.0084` and the coefficient for the variable `Total_Claim_Amount` is `0.0001`. This means that customers paying higher monthly premiums are less likely to respond to marketing campaigns, which aligns with the heuristics that high-value customers often are the harder ones to get and encourage engagement. On the other hand, customers with higher claim amounts are more likely to engage, as they may be the ones utilizing insurance policies the most.

The *P>|z|* column suggests the statistical significance each variable has on the outcome. The values lower than `0.05` are typically considered to have a statistically significant impact on the outcome, as p-values lower than `0.05`

suggest there is strong evidence against the null hypothesis that there is no impact of this variable on the target variable. However, values larger than `0.05` do not necessarily suggest it does not have any significance on the outcome. You may also notice the coefficients for `Customer_Lifetime_Value` and `Income` are very small.

This does not suggest they have low impacts on the outcome. As you may have guessed, these two variables have monetary values and the scales for these two variables are in thousands, which results in small coefficient values, but with significant impact on the outcome.

## Categorical variables

We have seen how continuous variables interact with the outcome, `Engaged`, in a positive or negative way. You may wonder, then, what about categorical or discrete variables? We have a handful of categorical variables within our auto insurance example dataset; however, we will use the `Education` and `Gender` variables as examples to discuss how to handle categorical variables when performing regression analysis.

## Factorize

The first approach we can take is to utilize the `factorize` functionality within the `pandas` library. Using `Education` as an example, you can run the following code to encode a textual discrete variable into a numerical variable:

```
labels, levels = df['Education'].factorize()
```

The results of this factorization look like the following:

```
labels, levels = df['Education'].factorize()  
labels
```

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

```
levels
```

```
Index(['Bachelor', 'College', 'Master', 'High School or Below',
```

As you can see from this code snippet, we apply the `factorize` function for the `Education` variable and get the two variables, `labels` and `levels`. The newly created `labels` variable contains numerical values for each record and the `levels` variable contains information about what each numerical value of the `labels` variable means. In this example, `Bachelor` is encoded as `0`, `College` as `1`, `Master` as `2`, `High School or Below` as `3`, and `Doctor` as `4`.

## Categorical

The second approach we can take to encode categorical variables is to use the `Categorical` function within the `pandas` library. The following code shows an example of how to use the `Categorical` function:

```
categories = pd.Categorical(  
    df['Education'],  
    categories=['High School or Below', 'Bachelor', 'College',  
    categories.categories
```

```
Index(['High School or Below', 'Bachelor', 'College', 'Master',
```

```
categories.codes
```

```
array([1, 1, 1, ..., 1, 2, 2], dtype=int8)
```

The `categories` argument in the `categorical` function lets you define the order of the categories. Here, we defined the `Education` category to order from `High School or Below` to `Doctor`. You can access the categories by calling the `categories` attribute and you can get the encodings for each record by calling the `codes` attribute. For example, in our case, `High School or Below` is encoded with `0`, `Bachelor` with `1`, and `Doctor` with `4`.

## Dummy variables

One other approach that can be taken to encode categorical variables is to create dummy variables. Dummy variables are one-hot encoded variables for each category.

For instance, in our example, the following dummy variables can be created for the `Education` variable:

```
pd.get_dummies(df['Education']).head(10)
```

	Bachelor	College	Doctor	High School or Below	Master
0	True	False	False		False
1	True	False	False		False
2	True	False	False		False
3	True	False	False		False
4	True	False	False		False
5	True	False	False		False
6	False	True	False		False
7	False	False	False		False
8	True	False	False		False
9	False	True	False		False

Figure 3.3: Dummy variables created for the Education variable

As you can see from this example, the `pandas` library's `get_dummies` function was used to create dummy variables for the `Education` variable. This creates five new variables: `Bachelor`, `College`, `Doctor`, `High School or Below`, and `Master`. Each of these newly created variables is encoded `0/1` (or `False/True`), where if a given record belongs to a given category, it is encoded as `1` or `True`.

### How to choose which method to use for categorical variable encoding

**One-hot encoding/dummy variables:** Use these when there is no natural ordering in the variable. Both methods create binary variables for each category. Note that creating one-hot encoded and dummy variables for all categories can



result in large dimensional data and make data sparse. Thus, you may want to subselect important categories for one-hot encoding/dummy variables.

**Factorize:** Use this as another quick and efficient method when you have nominal categorical variables that do not have any inherent order.

**Categorical:** When there is a natural ordering in the categorical variable, such as education levels, it is best to use `categorical()` with a specified order. This ensures that the encoding respects the hierarchy of the categories, leading to accurate model predictions and evaluations such as AUC-ROC in analyses where order matters.

## Regression analysis

Now that we have explored different approaches that can be taken to encode categorical variables, we can run a regression analysis on how some of the categorical variables affect the outcome. Using the newly created categorical variables from the previous section, `GenderFactorized` and `EducationFactorized`, we can fit a logistic regression model with the following code to model the relationships and impacts of the `Gender` and `Education` levels on the level of marketing engagement:

```
logit = sm.Logit(  
    df['Engaged'],  
    df[[  
        'GenderFactorized',  
        'EducationFactorized'  
    ]]  
).fit()
```

Running `logit.summary()` will result in the following summary of the fitted logistic regression model:

Logit Regression Results						
<b>Dep. Variable:</b>	Engaged		<b>No. Observations:</b>	9134		
<b>Model:</b>	Logit		<b>Df Residuals:</b>	9132		
<b>Method:</b>	MLE		<b>Df Model:</b>	1		
<b>Date:</b>	Mon, 25 Mar 2024		<b>Pseudo R-squ.:</b>	-0.2005		
<b>Time:</b>	13:32:10		<b>Log-Likelihood:</b>	-4503.7		
<b>converged:</b>	True		<b>LL-Null:</b>	-3751.6		
<b>Covariance Type:</b>	nonrobust		<b>LLR p-value:</b>	1.000		
	coef	std err	z	P> z	[0.025	0.975]
<b>GenderFactorized</b>	-1.1266	0.047	-24.116	0.000	-1.218	-1.035
<b>EducationFactorized</b>	-0.6256	0.021	-29.900	0.000	-0.667	-0.585

*Figure 3.4: Summary of logistic regression results*

As you can see from this logistic regression model output, both variables, `GenderFactorized` and `EducationFactorized`, have negative coefficients or impact on the outcome, `Engaged`. As `Gender` is encoded as `0` for female and `1` for male, this suggests that males are less likely to respond than females.

Similarly, education levels are encoded from 0 to 4, and having a negative coefficient suggests that the higher the education level is, the less likely it is for a customer to respond.

Along with the results we have seen from regression analysis with continuous variables in the previous section, these are great insights that show directional relationships among various factors against the engagement outcome.

# Interaction variables

Regression models are great at identifying and estimating linear relationships between the variables and the target variable. However, variables are likely to have some relationships among themselves. For example, income may have a high correlation with education levels. One approach that can be taken to address such intrinsic relationships within the variables in regression analysis is to introduce multiplicative terms, as shown in the following formula:

$$y = a + b_1 * x_1 + b_2 * x_2 + b_3 * x_1 x_2$$

This way, the regression model considers the interactions between variables that have relationships within themselves. The same `Logit` within the `statsmodels` library can be used for running regression analysis with interaction terms in Python. The following code will fit a logistic regression model with interaction terms:

```
logit = sm.Logit.from_formula(  
    data=df,  
    formula="Engaged ~ Income + EducationFactorized + Income:Education")  
.fit()  
logit.summary()
```

As shown in the code, we are fitting a logistic regression model with two variables, `Income` and `Education`, and one interaction term, `Income x Education`. The following is the summary of the fitted logistic regression model:

Logit Regression Results						
Dep. Variable:	Engaged	No. Observations:	9134 <th data-cs="3" data-kind="parent"></th> <th data-kind="ghost"></th> <th data-kind="ghost"></th>			
Model:	Logit	Df Residuals:	9130 <th data-cs="3" data-kind="parent"></th> <th data-kind="ghost"></th> <th data-kind="ghost"></th>			
Method:	MLE	Df Model:	3			
Date:	Mon, 25 Mar 2024	Pseudo R-squ.:	0.003177			
Time:	13:32:11	Log-Likelihood:	-3739.7			
converged:	True	LL-Null:	-3751.6			
Covariance Type:	nonrobust	LLR p-value:	2.701e-05			
	coef	std err	z	P> z	[0.025	0.975]
Intercept	-2.1009	0.077	-27.403	0.000	-2.251	-1.951
Income	5.065e-06	1.54e-06	3.300	0.001	2.06e-06	8.07e-06
EducationFactorized	0.2158	0.045	4.753	0.000	0.127	0.305
Income:EducationFactorized	-3.229e-06	9.22e-07	-3.502	0.000	-5.04e-06	-1.42e-06

Figure 3.5: Summary of logistic regression results

Looking at the first two components, `Income` and `EducationFactorized`, this output suggests that `Income` and `Education` have positive impacts on engagement likelihood. However, the interaction term, `Income:EducationFactorized`, negates the individual positive effects of `Income` and `Education`, as it has a negative coefficient. In our case of marketing, this means that when you compare customers within the same income level, the higher the education level is, the lower their engagement rate will be. Similarly, this also suggests that for customers with the same education level, the higher their income is, the lower their engagement rate will be. As you can see from this example, by introducing the interaction terms, we can analyze the relationships among the variables more deeply and gain insights that may not be unveiled when analyzed individually.

As we have seen in this section, regression analysis is a powerful tool to unveil the dynamics of the drivers behind the successes and failures of marketing efforts. In our example, we have seen how various variables have

negative and positive impacts on customer engagement rates. As shown, regression analysis is great at identifying linear relationships between various factors and the outcome and can be used to understand the interactions within the factors.

However, it is generally considered that regression models lack the understanding of intervariable relationships. Part of this limitation is because we have to introduce interaction terms for all possible intervariable relationships. As we saw when we created `Income` and `Education` interaction terms previously, we will have to build similar interaction terms to address other possible interactions among the variables. Furthermore, if you want to introduce or consider interactions among more than two variables, it starts to get complicated.

Decision trees, unlike linear regression analysis, have a strength in addressing complex interactions among variables. We will examine the use of decision trees in the following section, which captures the interactions among various factors and how these interactions ultimately affect the final outcome very well.

## **Why people convert with decision tree interpretation**

Once potential customers start engaging with your marketing campaigns, it is time to start converting them into paying customers. Similar to how various backgrounds may affect the tendency to engage, there are numerous factors affecting who converts more often than others. Socio-economic factors, of course, will always play a vital role in affecting the conversion rates. Age, region, and gender may also be important factors in different

conversion results. Months or seasons of the year can also cause conversion rates to vary, depending on the services or products you provide.

Not only single factors but also combinations of various factors may have significant influences on who may convert. A person who is employed and owns a house is more likely to convert for refinance loans than a person who owns a house but is retired. A 25-year-old student is less likely to look for a credit card than a 25-year-old entrepreneur. We have briefly discussed how regression analysis can help identify how factors interacting with each other affect customer behaviors by introducing interaction or multiplicative factors. Another great approach to uncovering the hidden insights around why certain people convert more than others is to use **decision trees** and their power to understand the interactions and relationships among various factors.

Decision trees, as the name suggests, learn from data by growing trees. From a root node, a tree branches out by splitting the data into subcategories and eventually reaching the leaf nodes, from which you can understand what factors are affected in which ways to reach the corresponding leaf nodes. A sample of a decision tree may look like the following:

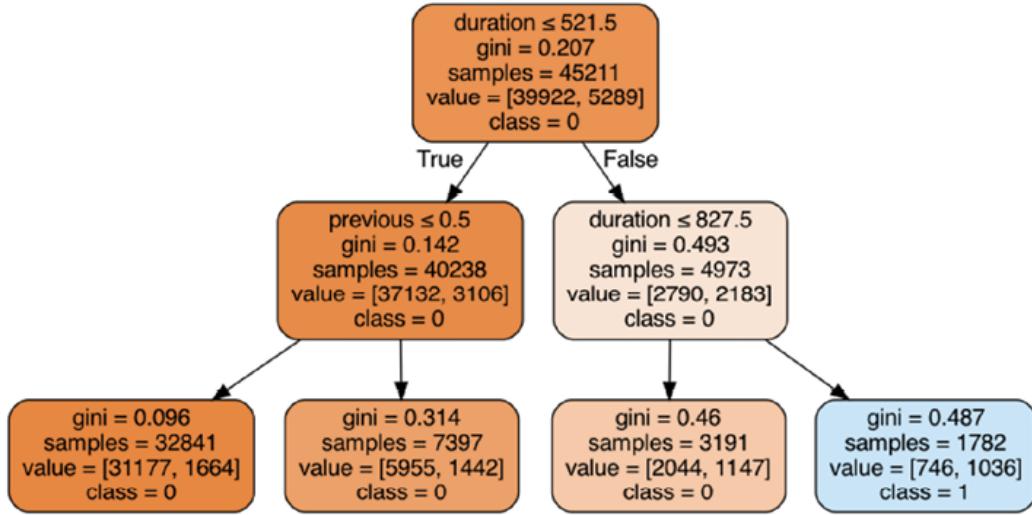


Figure 3.6: Sample of a decision tree

We will have a detailed discussion on how to interpret the decision tree results with an example dataset, so we will save further explanation and discussion for later. Let's talk first about how a decision tree branches out from each node.

There are mainly two methods that are commonly used for splitting the data or branching out into child nodes:

- **Gini Impurity:** Gini impurity measures how impure a partition is. The formula for the Gini impurity measure is as follows:

$$Gini = 1 - \sum_{i=1}^c p_i^2$$

In this formula,  $c$  stands for the class labels, and  $p_i$  stands for the probability of a record with the class label  $i$  being chosen. When all records in each node of a tree are pure with all records being in a single target class, the Gini impurity measure reaches  $0$ .

- **Entropy Information Gain:** Entropy measures how much information it gains from splitting the data with the criteria being tested. The formula for *Entropy* is as follows:

$$\text{Entropy} = - \sum_{i=1}^c p_i \log(p_i)$$

As for the *Gini* measure,  $c$  stands for the class labels, and  $p_i$  stands for the probability of a record with the class label  $i$  being chosen. The split that gives the biggest change in the entropy measure will be chosen to branch out to child nodes, as it suggests that results in the highest information gain.

With this fundamental knowledge about decision trees, we will dive into applying one to understand what drives successful marketing campaigns for better conversion rates.

#### **Source code and data:**

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/blob/main/ch.3/Why%20People%20Convert.ipynb>

#### **Data source:**

<https://archive.ics.uci.edu/dataset/222/bank+marketing>

# Target variable

We will be using a bank marketing dataset for this exercise, where the marketing campaign was to convert customers to subscribe to term deposits. First things first, let's load the data into a DataFrame and encode our target variable. Take a look at the following code:

```
df = pd.read_csv("./convert-data.csv", sep=";")  
df["conversion"] = df["y"].apply(lambda x: 1 if x == "yes" else 0)
```

Here, we are loading the data into a variable, `df`, and encoding the `y` column into `1` for `yes` and `0` for `no`. When you run `df["conversion"].mean()`, you should see about 12% of the customers converted and subscribed to term deposits.

# Continuous versus categorical variable

For our analysis, we will be using the following variables, which have continuous values: `age`, `balance`, `duration`, `campaign`, and `previous`. You can use the following code to see the distribution for these variables:

```
continuous_vars = [  
    "age", "balance", "duration", "campaign", "previous"  
]  
df[continuous_vars].describe()
```

The distribution looks as follows:

	age	balance	duration	campaign	previous
<b>count</b>	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
<b>mean</b>	40.936210	1362.272058	258.163080	2.763841	0.580323
<b>std</b>	10.618762	3044.765829	257.527812	3.098021	2.303441
<b>min</b>	18.000000	-8019.000000	0.000000	1.000000	0.000000
<b>25%</b>	33.000000	72.000000	103.000000	1.000000	0.000000
<b>50%</b>	39.000000	448.000000	180.000000	2.000000	0.000000
<b>75%</b>	48.000000	1428.000000	319.000000	3.000000	0.000000
<b>max</b>	95.000000	102127.000000	4918.000000	63.000000	275.000000

*Figure 3.7: Distribution of continuous variables*

Similar to the previous exercise, there are some categorical variables in this dataset as well. The first discrete text variable that we will convert into numerical values is `month`. If you look at the values of the variable `month`, you will notice that the values are 3-letter month abbreviations. As months have an inherent order, we will convert the values into corresponding numerical values using the following code:

```
months = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug'
df['month'] = df['month'].apply(
    lambda x: months.index(x)+1
)
```

Now, the variable `month` will have values from `1` to `12` for months respectively from `January` to `December`.

The variable `job` is another categorical variable. Unlike the `month` variable, the `job` variable does not have an inherent order within itself. Thus, we will create dummy variables for each of the `job` categories, which will be one-hot encoded for the corresponding job categories. Take a look at the following code:

```
jobs_encoded_df = pd.get_dummies(df['job'])
jobs_encoded_df.columns = ['job_%s' % x for x in jobs_encoded_d
```

The output of this code will look like the following:

```
jobs_encoded_df.head()
```

	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	job_student
0	False	False	False	False	True	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	True	False	False	False	False	False	False
3	False	True	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False

Figure 3.8: Encoded dummy variables

Similar to the previous exercise, we have used the `get_dummies` function to create dummy variables for each of the job categories in the dataset. You can use the following code to add these newly created dummy variables back to the DataFrame:

```
df = pd.concat([df, jobs_encoded_df], axis=1)
```

With this code run, the DataFrame, `df`, will now have newly created dummy variables of the `job` column added to it.

Lastly, we are going to encode `marital` and `housing` variables as well using the following code:

```
marital_encoded_df = pd.get_dummies(df['marital'])
marital_encoded_df.columns = ['marital_%s' % x for x in marital
df = pd.concat([df, marital_encoded_df], axis=1)
df['housing'] = df['housing'].apply(lambda x: 1 if x == 'yes' e
```

As you can see from this code, we have created dummy variables for each of the marital categories, which are `divorced`, `married`, and `single`. Then, we concatenated these newly created dummy variables back to the DataFrame, `df`. We have also converted the textual values of the `housing` column into `1` for yes and `0` for no.



As categorical variable encoding is somewhat covered in depth in the previous exercise, we quickly went through this step for this exercise. Take your time going through this categorical variable encoding step and try different approaches that we discussed during the previous exercise as well!

## Decision tree analysis

We have compiled the variables of our interest, which are `age`, `balance`, `duration`, `campaign`, `previous`, `housing`, `month`, `job`, and `marital status`. We are going to start looking into how these factors have affected the conversion rate outcome of this marketing campaign. First, we are going to define our features and the target or response variable, as in the following code:

```
features = (
    continuous_vars
    + ["housing", "month"]
    + list(jobs_encoded_df.columns)
    + list(marital_encoded_df.columns)
)
response_var = 'conversion'
```

As noted in this code, we are using the `conversion` column as our outcome or target variable and the continuous variables and encoded categorical variables that we have created previously. The resulting list of features should look as follows:

```
features
```

```
['age',
 'balance',
 'duration',
 'campaign',
 'previous',
 'housing',
 'month',
 'job_admin.',
 'job_blue-collar',
 'job_entrepreneur',
 'job_housemaid',
 'job_management',
 'job_retired',
 'job_self-employed',
 'job_services',
 'job_student',
 'job_technician',
 'job_unemployed',
 'job_unknown',
 'marital_divorced',
 'marital_married',
 'marital_single']
```

*Figure 3.9: List of features*

With the feature set and target variable defined, we are ready to build a decision tree. Take a look at the following code first:

```
from sklearn import tree
dt_model = tree.DecisionTreeClassifier(
    max_depth=3
)
dt_model.fit(df[features], df[response_var])
```

We are using the `tree` module within the `sklearn` or `scikit-learn` library and using the `DecisionTreeClassifier` class to train a decision tree model. You will notice one argument that we have defined, `max_depth`. This argument controls how much the tree grows. The deeper the tree grows, the more accurately the tree learns the data; however, it will likely overfit to the dataset and not be generalizable, meaning it will perform worse for the data that has not been observed. It is best to limit how much a tree grows and balance between how well and accurately a tree learns the data and how well this knowledge can be generalized. In this exercise, we will set `max_depth` to `3`. Finally, you can use the `fit` function to let the tree learn from the data.



Try finding a golden spot for how deep a decision tree should grow by splitting the dataset into training and testing sets and choosing the depth that minimizes the difference between the training and testing sets.

The most intuitive method to use the knowledge that the decision tree gained from the data is to visualize how it branched out from each node and how the interactions between the variables lead to the leaf nodes. To do this, we will use the `graphviz` package, which you can install with the following command:

```
conda install python-graphviz
```

You can use the following code to draw the trained decision tree:

```
import graphviz
dot_data = tree.export_graphviz(
    dt_model,
    out_file=None,
    feature_names=features,
    class_names=['0', '1'],
    filled=True,
    rounded=True,
    special_characters=True
)
graph = graphviz.Source(dot_data, format="png")
graph.render("conversion-dt-depth-3")
```

As you can see from the code, we are giving the function the trained model, `dt_model`, and the features, `features`, that were used for training the model. Then, you can output the graph into an image file with the `render` function of the graph. This code will generate an image file named `conversion-dt-depth-3.png` and it will look like the following:

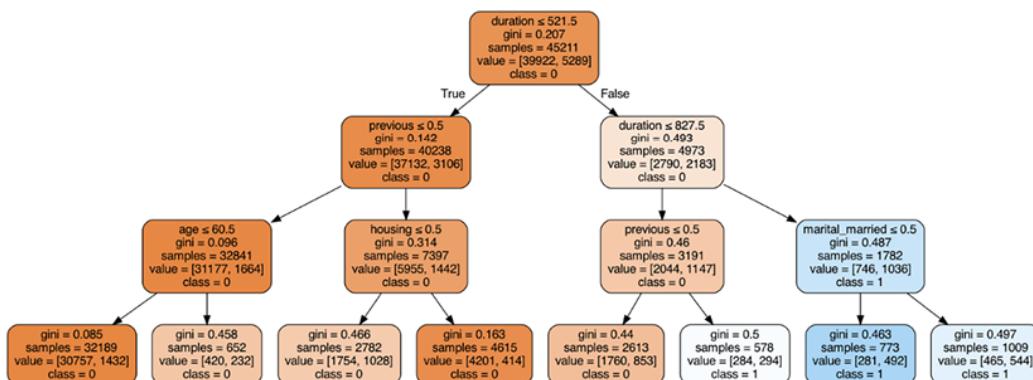
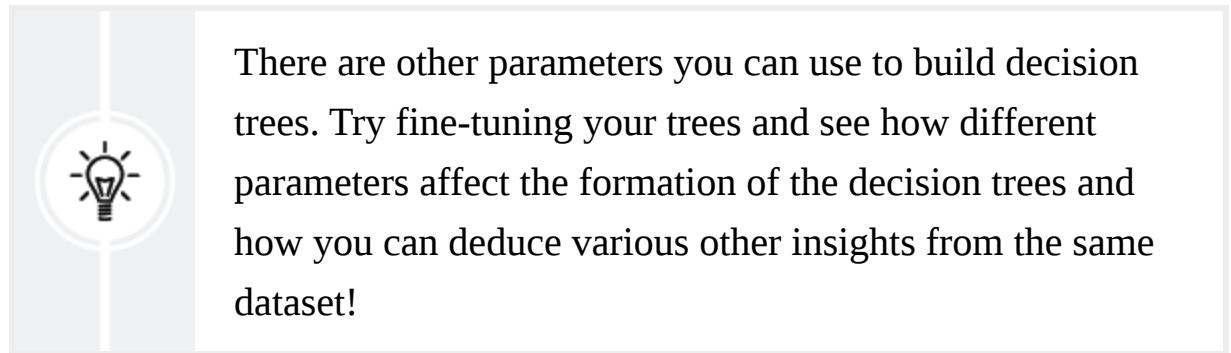


Figure 3.10: Decision tree diagram

As you can see from this decision tree graph, the depth of the tree is 3, which is what we defined it to be when we were training the decision tree. Let's take a closer look at this graph.

The root node is being split based on the variable `duration`. As the arrows suggest, if the duration is less than or equal to `521.5`, then it moves to the left child node; otherwise, it moves to the right child node. If we traverse to the rightmost leaf node by following the path with a duration greater than `827.5` and `marital_married` greater than `0.5`, the majority of the customers in this leaf node have converted. In other words, those customers who have had a contact duration that is longer than `827.5` seconds and who are married have converted more.

On the other hand, if we traverse to the leftmost leaf node, it suggests that those customers who have had a contact duration that is shorter than or equal to `521.5` seconds (root node), 0 previous contacts (left child node at depth 1), and their age is less than or equal to `60.5` (leftmost child node at depth 2) are more likely not to convert than the others.



There are other parameters you can use to build decision trees. Try fine-tuning your trees and see how different parameters affect the formation of the decision trees and how you can deduce various other insights from the same dataset!

As you can see from this example, by traversing through the tree and its branches and nodes, we can see the impacts of different factors of marketing on the customer behavior of conversion. Compared to the regression analysis that we did in the previous section, this decision tree has

the advantage of better understanding more complex interactions and inter-relationships among different factors and how they affect the outcome variable. One thing to note is that both regression analysis and decision tree analysis results show correlations between the factors and the outcome rather than the causal relationships of what causes certain outcomes. Examining the causality of certain outcomes is not an easy task; however, in the following section, we will discuss how we can estimate the causal relationships between the features and the outcome from the data.

## Why people churn with causal inference

A high churn rate is especially a problem in marketing as it negates all the good marketing income that was generated from previous marketing campaigns. It is critical to have an in-depth understanding of why people churn and what factors need to be optimized to reduce the customer churn rate. As we have seen in previous sections, regression analysis and decision tree analysis are great at identifying linear relationships between the potential factors and the outcome and the inter-relationships between various factors and the outcome variable. However, as noted before, these identified relationships or correlations do not necessarily mean causations.

Identifying the causes of certain outcomes (for example, causes of customer churn) is often a difficult and complex task to achieve. This is where **causal analysis** comes in. If regression analysis was used to identify the relationships among the variables and decision tree analysis was used to identify the *interactions* among the variables, causal analysis is used to identify the *causes* and *effects* of certain outcomes.

This is typically done through various experiments with control groups and treatment or experimental groups. The control group, as the name suggests, is the group to which no treatment is given. In a medical experiment setting, this control group is the one that will get a placebo. The treatment group, on the other hand, is the group that actually gets the treatment. Similarly, in a medical experiment setting, this treatment group is the one that gets the actual medical treatments, such as new medicine pills being developed.

This paradigm of the necessity of setting up experiments prior to running a marketing campaign restricts the after-the-fact analysis. However, with the data, we can still run a causal analysis to estimate the effects of different factors on the outcome. We will be utilizing a Python package, `dowhy`, to run causal analysis based on the marketing campaign data. We will be using bank churn data as an example to discuss how a causal analysis can be done to unveil the causes and effects of certain variables on the marketing outcome.

#### **Source code and data:**

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/blob/main/ch.3/Why%20People%20Churn.ipynb>

#### **Data source:**

<https://www.kaggle.com/datasets/shrutimechlearn/churn-modelling>

Installation of the `dowhy` package:



```
pip install dowhy
```

(Note: you may require additional installations for the `dowhy` package. Please refer to the official documentation at <https://pypi.org/project/dowhy/>.)

## Causal effect estimation

The first step to estimating the causal effects of certain variables on the outcome is defining the problem with some assumptions. Let's take the bank churn dataset as an example. First, when you load the data into a DataFrame, you will see variables as in the following:

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   RowNumber        10000 non-null    int64  
 1   CustomerId       10000 non-null    int64  
 2   Surname          10000 non-null    object  
 3   CreditScore      10000 non-null    int64  
 4   Geography         10000 non-null    object  
 5   Gender            10000 non-null    object  
 6   Age               10000 non-null    int64  
 7   Tenure            10000 non-null    int64  
 8   Balance           10000 non-null    float64 
 9   NumOfProducts     10000 non-null    int64  
 10  HasCrCard        10000 non-null    int64  
 11  IsActiveMember   10000 non-null    int64  
 12  EstimatedSalary  10000 non-null    float64 
 13  Exited           10000 non-null    int64  
dtypes: float64(2), int64(9), object(3)

```

*Figure 3.11: Snapshot of the DataFrame*

Here, the `Exited` column is the target outcome we are interested in analyzing the causes for. As you may notice, there are several other columns that may be interesting to see causal effects on customer churn. Some may be interested in seeing whether a high or low credit score affects customer churn. Some others may be interested in seeing whether a long or short tenure affects customer churn. You may also be curious whether having multiple products affects customer churn.

For our exercise in this chapter, we will define our problem or a causal effect of our interest as follows:



*“Does having multiple products affect customer churn?”*

---

With this problem definition, we can build some assumptions that eventually lead to customer churn:

- Having multiple products may affect the churn rate, as it is likely that the people with multiple products will find it more difficult or cumbersome to switch to different banks.
- `CreditScore` may affect whether a customer can have multiple products or not, as the credit score is used to decide whether a person is qualified to have certain bank products or not, such as credit cards or term loans.
- `Age` can also affect whether a customer has multiple products or not, as the older you are, the more likely you will have needed more bank products.
- Similarly, `Tenure`, `Balance`, and `Salary` can affect whether a customer may have multiple products or not. These factors may also affect customer churn.

With these assumptions and the problem statement, let's analyze the causal effect of a `MultipleProduct` variable on customer churn, which, using a do-calculus notation, can be written as:

$$E[Customer\ Churn\mid do(Multiple\ Products)]$$

This, in turn, means the expected change in the likelihood of customer churn due to a change in the variable `MultipleProduct`. As with our assumptions, there can be a set of other variables that affect the outcome

and we may be interested in estimating the causal effect of these covariates on the outcome. In our example, these covariates are `Tenure`, `Balance`, and `Salary`, and the estimated causal effect of these covariates on customer churn can be written as:

$$E[Customer Churn | do(\\text{Multiple Products}), (Tenure, Balance, Salary)]$$

## Causal model

With the pre-defined problem statement and our assumptions, let's start building a causal model.

First, we will need to build a variable that we will use as the treatment. Take a look at the following code:

```
df["MultipleProduct"] = df["NumOfProducts"].apply(lambda x: 1 if
```

In our dataset, there is a column called `NumOfProducts`. We are simply encoding this into 0s and 1s, where customers with more than one product will be encoded as 1s and the rest as 0s.

Now, we will use the `CausalModel` class within the `dowhy` package to define the causal model based on our assumptions, as in the following code:

```
from dowhy import CausalModel
model = CausalModel(
    data=df,
    treatment="MultipleProduct",
    outcome="Exited",
    common_causes=["Balance", "EstimatedSalary", "Tenure"],
    instruments=["Age", "CreditScore"]
)
```

As you can see from this code, we defined the newly created variable, `MultipleProduct`, as the treatment for which we are interested in learning the causal effect on the outcome, `Exited`. We defined `Balance`, `EstimatedSalary`, and `Tenure` as common causes that may have causal effects on the customer churn and on whether a customer may have multiple products or not. Lastly, we defined `Age` and `CreditScore` as instruments as they may affect a customer's ability to have multiple products.

You can visualize this causal model using the following code:

```
model.view_model()
```

The resulting causal graph looks as follows:

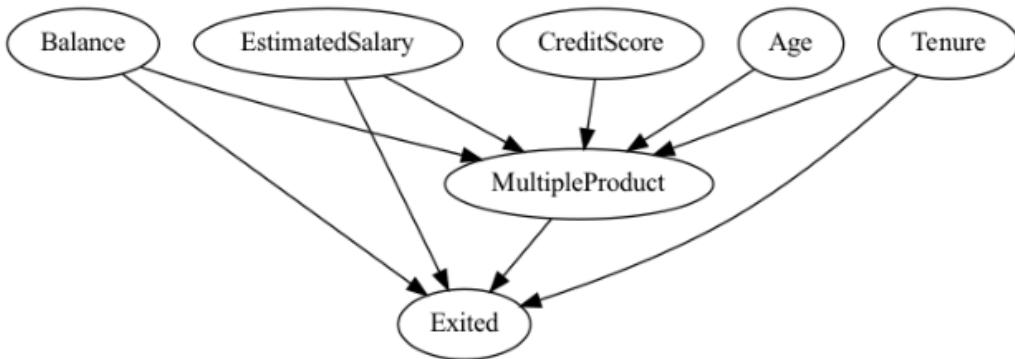


Figure 3.12: Visualization of the causal graph

Note that the actual output may differ each time this code is run.

Here, the instrument variables, `CreditScore` and `Age`, affect the treatment variable, `MultipleProduct`, which affects the outcome variable, `Exited`, and the covariates or common causes, `Balance`, `EstimatedSalary`, and `Tenure`, affect the treatment variable as well as the outcome variable.

Then, we can have the model identify the causal effects, as with the following code:

```
estimands = model.identify_effect()  
print(estimands)
```

This code will show the following expressions for the causal effect under our assumptions:

```
Estimand type: EstimandType.NONPARAMETRIC_ATE  
  
### Estimand : 1  
Estimand name: backdoor  
Estimand expression:  
d  
_____(E[Exited|Tenure,Balance,EstimatedSalary])  
d[MultipleProduct]  
Estimand assumption 1, Unconfoundedness: If U-{\MultipleProduct} and U-Exited then P(Exited|MultipleProduct,Tenure,  
Balance,EstimatedSalary,U) = P(Exited|MultipleProduct,Tenure,Balance,EstimatedSalary)  
  
### Estimand : 2  
Estimand name: iv  
Estimand expression:  
E[  
d  
_____(Exited).  
[d[CreditScore Age]  
d[CreditScore Age]]  
([MultipleProduct])  
-1  
]  
Estimand assumption 1, As-if-random: If U->Exited then ~(U ->{CreditScore,Age})  
Estimand assumption 2, Exclusion: If we remove {CreditScore,Age}->{MultipleProduct}, then ~({CreditScore,Age}->Exite  
d)  
  
### Estimand : 3  
Estimand name: frontdoor  
No such variable(s) found!
```

Figure 3.13: Code expressions for causal effect

This should look familiar as this is the same as the causal effect formulas we discussed previously.

## Estimation

With the causal model defined, we can now estimate the effects of the treatment variable, `MultipleProduct`. The `estimate_effect` function makes the estimation of these effects straightforward, as shown in the following code:

```

estimate = model.estimate_effect(
    estimands,
    method_name = "backdoor.propensity_score_weighting"
)
print(estimate)

```

This should produce the following results:

```

*** Causal Estimate ***

## Identified estimand
Estimand type: EstimandType.NONPARAMETRIC_ATE

### Estimand : 1
Estimand name: backdoor
Estimand expression:
d
-----(E[Exited|Tenure,Balance,EstimatedSalary])
d[MultipleProduct]
Estimand assumption 1, Unconfoundedness: If U→{MultipleProduct} and U→Exited then P(Exited|
Balance,EstimatedSalary,U) = P(Exited|MultipleProduct,Tenure,Balance,EstimatedSalary)

## Realized estimand
b: Exited-MultipleProduct+Tenure+Balance+EstimatedSalary
Target units: ate

## Estimate
Mean value: -0.13904650583124545

```

*Figure 3.14: Causal estimate results*

Based on this result, we estimate the mean effect of the treatment variable, `MultipleProduct`, on the outcome, `Exited`, to be `-0.1390`. This means that having multiple products causes a decrease in the probability of customer churn of around 14%. This confirms our assumption that having multiple products will create stickiness toward the bank and reduce the chance of customer churn.

There are various methods you can use for estimation methods. To name a few:

- Linear regression
- Distance matching



- Propensity score stratification
- Propensity score matching
- Propensity score weighting

We used propensity score weighting for our example, but we recommend trying different estimation methods and analyzing how they affect the causal effect estimations.

## Refutation

Once we have estimated the causal effect of the treatment on the outcome, we need to validate it. This assumption validation step is called **refutation**. We are going to run robustness checks of our assumptions with a few approaches.

The first approach is to introduce a random common cause or covariate variable. You can run this check as follows:

```
refute_results = model.refute_estimate(  
    estimands,  
    estimate,  
    "random_common_cause"  
)  
print(refute_results)
```

The output looks like the following:

```
Refute: Add a random common cause  
Estimated effect:-0.13904650583124545  
New effect:-0.13904650583124542
```

We are using the `refute_estimate` function within our causal model, `model`, with the argument "`random_common_cause`". This refutation method introduces a new, randomly generated variable to test if the causal estimate significantly changes, helping assess the model's robustness against omitted variable bias. As you can see from the output, the original estimated effect and the new effect after introducing a random common cause are the same with about -0.1390 as the estimated causal effect value. This confirms the fact that our assumption was correct.

The second approach to refute our assumption is to randomly subsample the data and test on this random subset. You can run this check as follows:

```
refute_results = model.refute_estimate(  
    estimands,  
    estimate,  
    "data_subset_refuter"  
)  
print(refute_results)
```

The output looks like the following:

```
Refute: Use a subset of data  
Estimated effect:-0.13904650583124545  
New effect:-0.13842175180225635
```

Similar to the previous case, we used the same `refute_estimate` function but with the "`data_subset_refuter`" argument this time. The output suggests that the original estimated effect and the new estimated effect with a random subset of the data are very close. This confirms our assumption was correct about the causal relationship between the `MultipleProduct`

variable and the `Exited` outcome variable and is a generalizable result across the dataset.



Similar to the multiple estimation method options, there are multiple ways you can run refutations. We have experimented with random common cause addition and random subset replacement in this chapter. However, there are various other ways you can validate your assumptions. To name a few:

- **Replace treatment with placebo:** Replace the treatment variable with one known not to have a causal effect on the outcome.
- **Replace with dummy outcome:** Instead of replacing the treatment variable, replace the outcome variable with one unrelated to the treatment.
- **Refute with bootstrapped random sample:** Generate multiple bootstrapped samples from the data and estimate the causal effect on each sample.

We recommend trying different refutation methods as well! The examples in the official documentation may be a set of useful resources and can be found here:

[https://www.pywhy.org/dowhy/v0.11.1/example\\_notebooks/nb\\_index.html](https://www.pywhy.org/dowhy/v0.11.1/example_notebooks/nb_index.html).

## Graphical causal model

While the previous causal effect estimation was to analyze and identify the effects of treatment variables, the **graphical causal model** enables the root cause analysis and the analysis of causal effects and linkages of various factors. This will be a great tool to have when you want to understand the causal effects of underlying variables and how they eventually lead to certain outcomes.

With the same assumptions as previously, we will learn how to use the `dowhy` graphical causal model to better understand causal linkages among the variables.

## Causal influence

One of the key differences using the graphical causal model is that we need to build a directed graph, where we define a start node to the next node. Let's discuss this further with an example by running the following code:

```
import networkx as nx
causal_graph = nx.DiGraph([
    ('CreditScore', 'MultipleProduct'),
    ('Age', 'MultipleProduct'),
    ('MultipleProduct', 'Exited'),
    ('Balance', 'MultipleProduct'),
    ('EstimatedSalary', 'MultipleProduct'),
    ('Tenure', 'MultipleProduct'),
    ('Balance', 'Exited'),
    ('EstimatedSalary', 'Exited'),
    ('Tenure', 'Exited'),
])
```

To install the `networkx` package:



```
pip install networkx
```

As you can see, we are using the `networkx` package's `DiGraph` class to create a directed graph. You can visualize this directed graph with the following code:

```
nx.draw_networkx(causal_graph, arrows=True)
```

The output will look like the following:

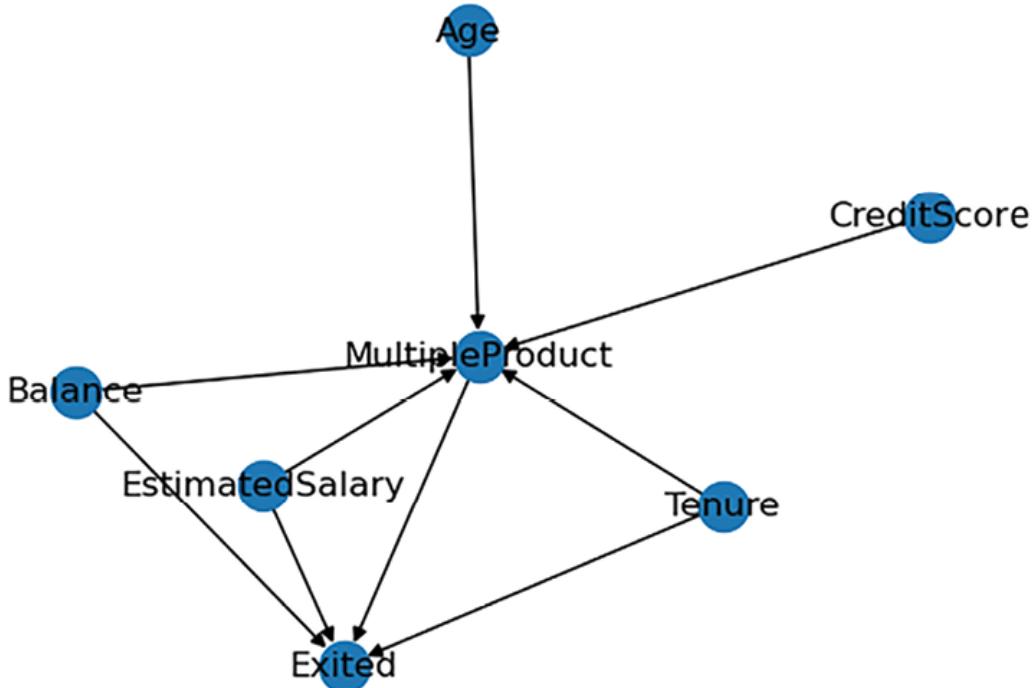


Figure 3.15: Directed graph

As you may notice, this is a similar graph to one we saw in the previous section, where there are directed edges that go from `Age` and `CreditScore` to `MultipleProduct`. Similarly, there are also directed edges that go from `Balance`, `EstimatedSalary`, and `Tenure` to `MultipleProduct` and `Exited`.

Finally, there is a directed edge that goes from `MultipleProduct` to `Exited`. In summary, this is a visualization of our assumptions about how different variables may have causal effects on one another.

With this directed graph, we can now fit a graphical causal model as in the following code:

```
from dowhy import gcm
scm = gcm.StructuralCausalModel(causal_graph)
gcm.auto.assign_causal_mechanisms(scm, df)
gcm.fit(scm, df)
```

Here, we are using the `gcm` module in the `dowhy` package and defining the causal model with the `StructuralCausalModel` class within the `gcm` module. Then, we assign our dataset to each node in the graph with the `assign_causal_mechanisms` function and, finally, fit the graphical causal model with the `fit` function and the structural causal model and dataset as the input to the `fit` function.

Now that our graphical causal model has learned the causal effects of various variables on the outcome, we can quantify these learned relationships using the `arrow_strength` function, as in the following code:

```
arrow_strengths = gcm.arrow_strength(scm, target_node='Exited')
```

You can visualize this by using the following code:

```
total = sum(arrow_strengths.values())
arrow_strengths_perc = {key: val/total*100 for key, val in arrow_strengths.items()}
gcm.util.plot(
    causal_graph,
    causal_strengths=arrow_strengths_perc,
```

```
    figure_size=[8, 5]
)
```

As you may notice from this code, we are normalizing the arrow strengths into percentages of the total in the first two lines of this code. The resulting graph should look like the following:

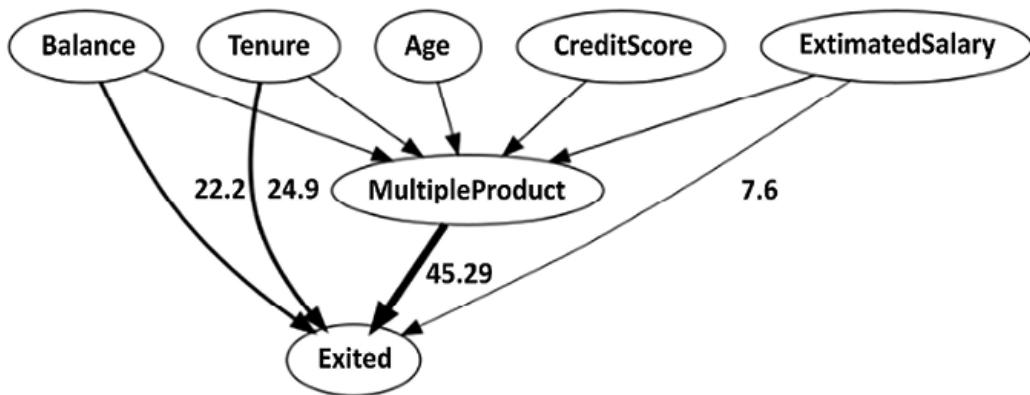


Figure 3.16: Graph with arrow strengths

As this graph suggests, there is the strongest causal relationship between `MultipleProduct` and `Exited`, with about 45% contribution, and `Tenure` follows as the second most impactful variable for customer churn with about 25% contribution. This directed graph with estimated causal impacts on the outcome provides great insight for understanding the quantified causal effects of different variables on the outcome.

While the arrow strengths show us the impacts of direct linkages to the outcome, they do not show us indirect relationships between the instrument variables, `Age` and `CreditScore`, and the outcome variable, `Exited`. One approach we can take to compute and visualize the impacts of all the causal factors on the outcome variable is to quantify the intrinsic causal

contributions that attribute the variance of the outcome variable to all the upstream nodes in the directed graph.

We can use the `intrinsic_causal_influence` function to quantify the intrinsic causal contributions, as in the following code:

```
influence = gcm.intrinsic_causal_influence(  
    scm,  
    target_node='Exited',  
    num_samples_randomization=100  
)
```

Here, we are using 100 random samples for evaluations, as you see from the `num_samples_randomization` parameter input. Before we visualize the quantified intrinsic causal influences, we will normalize the scale so that they sum up to 100% first and then visualize them with a bar plot, using the following code:

```
total = sum([val for key, val in influence.items() if key != "Exited")  
influence_perc = {key: val/total*100 for key, val in influence.items())  
xlabels = sorted(influence_perc.keys())  
yvals = [influence_perc[x] for x in xlabels]  
plt.bar(xlabels, yvals)  
plt.xticks(rotation=45)  
plt.grid()  
plt.ylabel("Variance attribution in %")  
plt.show()
```

The output of this code should produce a bar plot like the following:

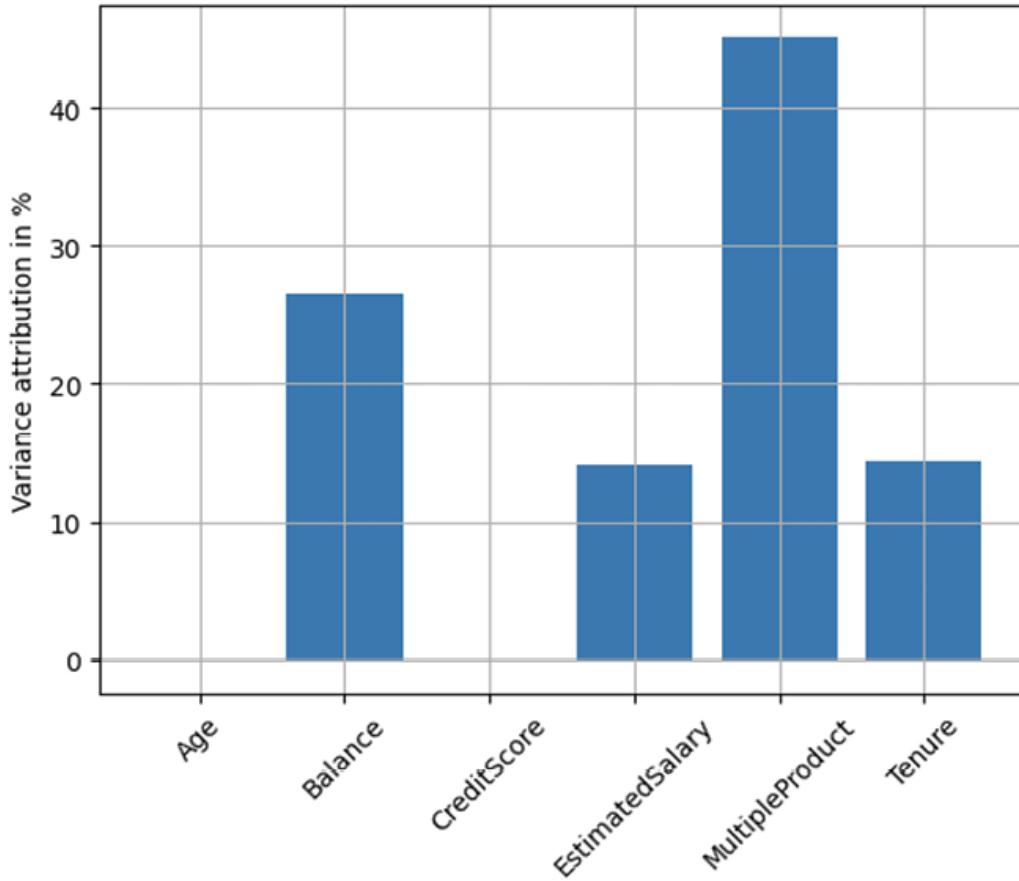


Figure 3.17: Bar plot of variance attribution for each factor

As expected, the variable `MultipleProduct` has the highest contribution to the variance of the outcome variable among all the variables. `Balance` comes next as the second most contributing variable. The variables `Age` and `creditScore` seem to have very little attribution to the variance of the outcome, `Exited`.

### What is a Shapley value?

You may notice that “Shapley values” appear while running causal inference code. Simply put, Shapley values are the estimations of the degrees of impact of individual variables on the target variable. If you would like to learn more, we





suggest you take a look at the package documentation on how it computes the feature relevance here:

[https://www.pywhy.org/dowhy/main/user\\_guide/causal\\_tasks/root\\_causaling\\_and\\_explaining/feature\\_relevance.html](https://www.pywhy.org/dowhy/main/user_guide/causal_tasks/root_causaling_and_explaining/feature_relevance.html).

## Anomaly attribution

So far, we have seen ways to analyze the overall causal effects of individual variables on the outcome variable. However, there will be cases where we want to dig deeper into understanding which variables may have contributed in what way. For example, within our dataset, one may be curious about how a certain customer's `Balance` amount may have caused them to churn or how that person's `Tenure` with the bank may have impacted their churn. In order to figure out the causal contributions of different variables for individual customer churn, we can use the `attribute_anomalies` function, as in the following:

```
attribution = gcm.attribute_anomalies(  
    scm,  
    target_node='Exited',  
    anomaly_samples=df.loc[df['Exited'] == 1].iloc[0:1]  
)
```

In our example, we chose the first customer who churned as an example, as you can see from the input for the `anomaly_samples` parameter. We can visualize the individual attributions on the outcome using the following code:

```

total = sum([abs(val[0]) for key, val in attributions.items() if
attributions_perc = {
    key: val[0]/total*100 for key, val in attributions.items()
}
xlabels = sorted(attributions_perc.keys())
yvals = [attributions_perc[x] for x in xlabels]
plt.bar(xlabels, yvals)
plt.xticks(rotation=45)
plt.grid()
plt.ylabel("Anomaly Attribution (%)")
plt.show()

```

As this code suggests, we are normalizing the attributions so that they sum up to 100% and then visualizing individual attributions with a bar plot.

The resulting bar plot will look like the following:

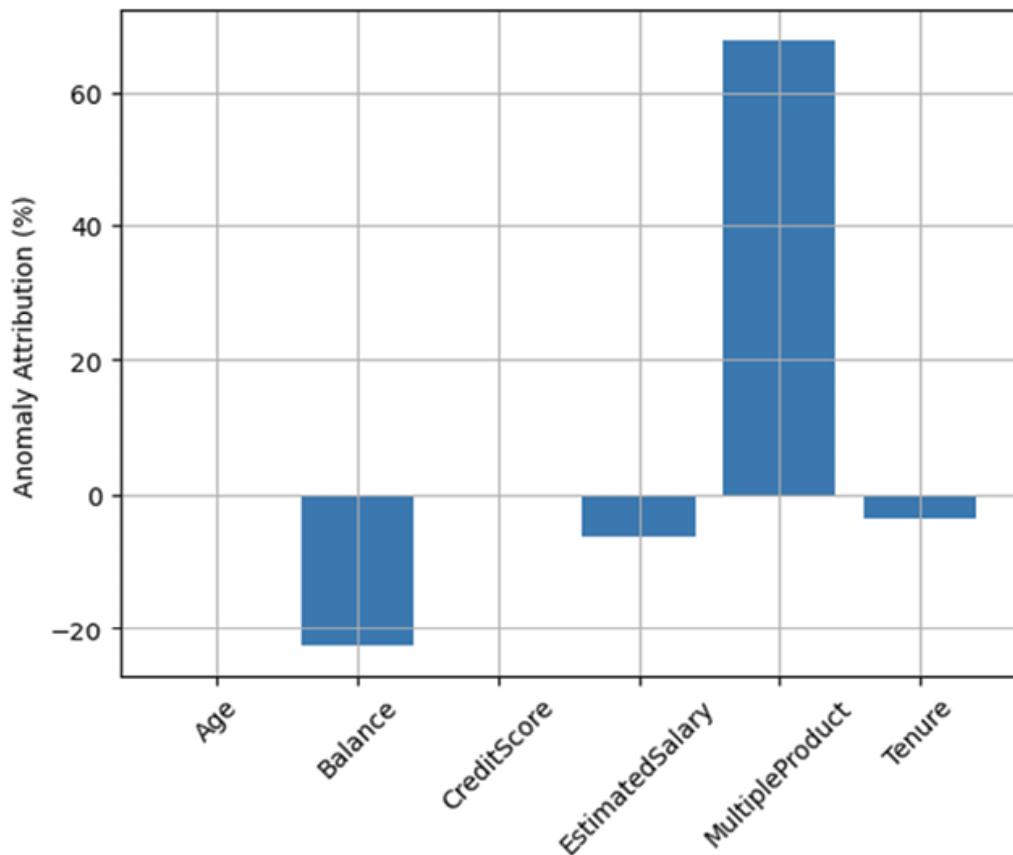


Figure 3.18: Bar plot of anomaly attribution for each factor

Before we go deeper into the interpretations of this bar plot, let's first look at the sample we chose for quantifying individual causal contributions. Take a look at the following code to take a closer look at an individual sample:

```
df.loc[df["Exited"] == 1].iloc[0][list(attributions_perc_ke
```

The output of this code will look as follows:

CreditScore	619
Age	42
Balance	0.0
EstimatedSalary	101348.88
Tenure	2
MultipleProduct	0
Name: 0, dtype: object	

Figure 3.19: Selected sample

If we combine the attribution results with this customer's info, we can dissect which factors were the most influential that ended up making this customer churn. We see from this customer's record that this person only had one product with the bank and this was the most contributing factor to why this customer churned. On the other hand, `Balance`, `EstimatedSalary`, and `Tenure` actually reduced the chance of customer churn as they have negative scores. Let's look at another sample, as follows:

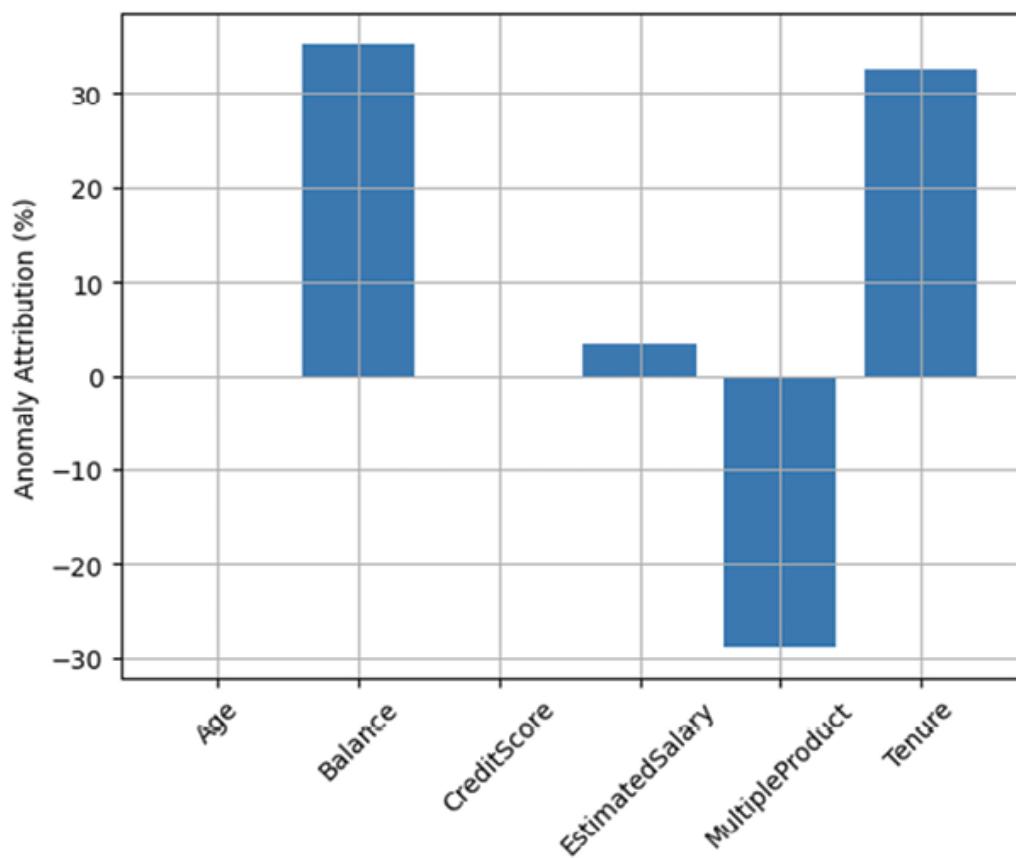
```
df.loc[df["Exited"] == 1].iloc[1][list(attributions_perc_ke
```

This has output like the following:

```
CreditScore      502
Age             42
Balance        159660.8
EstimatedSalary 113931.57
Tenure          8
MultipleProduct 1
Name: 2, dtype: object
```

*Figure 3.20: Selected sample*

This customer's causal contribution bar plot looks like the following:



*Figure 3.21: Bar plot of customer's causal contribution*

As you can see from this second example, this customer had a much higher balance, longer tenure, and more than one product with the bank, compared to the first example. For this customer, this higher balance, longer tenure,

and higher estimated salary had a positive impact on why the customer churned, while the fact that this customer had more than one product reduced the chance of this customer churning.

As we have seen from these examples, causal analysis and causal effect estimations bring deep insights into why people churn from your products. When used along with other methods and AI/ML models, this causal analysis will bring actionable insights that you can employ in your future marketing campaigns.

## Summary

In this chapter, we gained an in-depth understanding of the factors affecting certain customer behaviors. We explored how **regression analysis** can help us understand the directional relationships between various factors and the outcome of customer behavior. Using our auto insurance marketing dataset as an example, we saw how to implement the `statsmodels` package in Python to run regression analysis and unveil the successes behind engagement rate marketing campaigns. We also discussed how **decision trees** can help us identify complex interactions that result in certain outcomes. Using a bank marketing dataset as an example and the `scikit-learn` package in Python, we successfully built a decision tree that unveiled the hidden interactions among various factors that lead to customer conversions. Lastly, with the bank churn dataset and the `dowhy` package in Python, we saw how **causal analysis** can bring deep insights into the root causes and directional contributions to the outcome of an event.

In the next chapter, we are going to explore time-series data. In marketing, there are certain temporal patterns that repeat in certain ways. We will be discussing how to break down time-series data into overall trend,

seasonality, and random noise components and how to utilize the trends and seasonalities that are identified through time-series decomposition for marketing initiatives, as well as how to do time-series forecasting, which can equip you with better planning and preparedness for certain events that may happen in the future.

## **Join our book's Discord space**

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)

# 4

## **Harnessing Seasonality and Trends for Strategic Planning**

Unless it is a one-time marketing event, such as opening pop-up stores or occasional celebratory product sales, there will always be impacts on the marketing outcomes from time components. For example, companies selling umbrellas will naturally see significant sales increases during rainy seasons. Not only can there be seasonal impacts on businesses but there can also be general trends in businesses. For example, businesses selling landline phones will see a gradual decline in their sales as people use mobile phones more and more.

This chapter discusses in depth the temporal impacts on marketing campaigns and how to utilize them for the most efficient marketing strategies. We will introduce the basics of time-series analysis, such as some of the common approaches to identifying overall trends and anomalies and visualizing time-series data. We'll then move on to exploring how time-series data can be decomposed into trends and seasonalities, which will inform us of the contribution of breakdowns of certain events based on these factors and how this decomposition helps with building more efficient marketing strategies. We'll end by learning how to build time-series

forecasting models and understanding how these forecasts can be utilized for proper marketing campaigns. We will use a product sales dataset as an example and discuss how to conduct time-series analysis and modeling in Python.

More specifically, we will cover the following topics:

- Time series analysis basics
- Trend and seasonality decomposition in time-series data
- Time series forecasting models

## Time series analysis basics

Understanding natural trends within businesses is critical, not only for marketers but also for operators, sales, and most other business units.

Without a good grasp of how a business and its products are evolving and how customers are reacting and behaving around the services and products that you provide, it is very easy to fall behind, resulting in suboptimal revenue growth or even business growth slowing to a halt.

It is easy to find how businesses build their product strategies around this temporal component of the business cycle. In the fashion industry, clothing businesses typically market spring and summer clothes from late winter to early spring, while fall and winter clothes are advertised from early summer to early fall, as customers tend to shop before the actual season comes as a preparation. Then, spring and summer clothes typically go on sale during the summer, typically from mid-July, as the demand for these clothes is low during the actual season. This illustrates how businesses can optimize their business cycles for different seasons based on the customer demands that

naturally form and how they can maximize sales while minimizing the excess inventory.

**Time series analysis** comes in handy when we try to understand these natural temporal components within businesses. Time series analysis, in short, is a way of analyzing a series of data points over time and how **time** affects the changes in data values. In this chapter, we will be using a historical sales dataset as an example to discuss how to conduct time series analysis and modeling.



#### Source code and data:

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/tree/main/ch.4>

#### Data source:

<https://community.tableau.com/s/question/0D54T00000CWeX8SAL/sample-superstore-sales-excelxls>

## Basic time series trends

Let's look at overall monthly product sales using our example dataset. We will first need to load this data into a DataFrame and aggregate the data per month. Take a look at the following code (note that the CSV file used here is included in the book GitHub repository):

```
df = pd.read_csv("./data.csv", encoding="latin")
df[["OrderDate"]] = pd.to_datetime(df[["OrderDate"]])
ts_df = df[[
```

```
"OrderID", "OrderDate", "Quantity", "Sales", "Category"]].copy().set_index("OrderDate")
df.columns = [x.replace(" ", "") for x in df.columns]
```

Here, we are loading the data into a `pandas DataFrame`. One thing that is different from previous chapters is the encoding parameter. This dataset contains non-UTF characters, so we are using `latin` encoding to load the data. Then, we convert the values in the `orderDate` column into a datetime type and copy the key columns into a new `DataFrame`, `ts_df`. Lastly, for ease of use, we remove spaces in the column names by replacing all spaces with empty strings.

One way to resample and aggregate the data into monthly frequency is as follows:

```
monthly_unique_orders = ts_df[ "OrderID" ].resample( "MS" ).nunique
monthly_unique_order_changes = (
    monthly_unique_orders - monthly_unique_orders.shift()
) / monthly_unique_orders.shift() * 100
```

We are using the `resample` function within `pandas` with the argument `MS`, which will resample the data into a monthly timeframe, and then counting the number of unique `OrderID` values using the `nunique` function, which will give us monthly unique order count data. We then compute month-over-month changes in percentages by subtracting the previous month's value from the current month's value and dividing it by the previous month's value. The `shift` function will move the data by one period, resulting in giving us the previous month's value.

We can apply the same to compute monthly order quantities and sales, as in the following:

```

monthly_quantities = ts_df["Quantity"].resample("MS").sum()
monthly_quantities_changes = (
    monthly_quantities - monthly_quantities.shift()
)/monthly_quantities.shift()*100
monthly_sales = ts_df["Sales"].resample("MS").sum()
monthly_sales_changes = (
    monthly_sales - monthly_sales.shift()
)/monthly_sales.shift()*100

```

The simplest approach to visualizing time-series data is using line charts.

Take a look at the following code:

```

fig, axes = plt.subplots(
nrows=3, ncols=1, figsize=(10, 10), sharex=True
)
monthly_unique_orders.plot(
    ax=axes[0], grid=True
)
monthly_unique_order_changes.plot(
    ax=axes[0], secondary_y=True, color="silver", linestyle="dashed"
)
axes[0].set_title("Monthly Unique Orders")
axes[0].set_ylabel("# Unique Orders")
axes[0].right_ax.set_ylabel("Month over Month Change (%)")
monthly_quantities.plot(
    ax=axes[1], grid=True
)
monthly_quantities_changes.plot(
    ax=axes[1], secondary_y=True, color="silver", linestyle="dashed"
)
axes[1].set_title("Monthly Order Quantities")
axes[1].set_ylabel("Order Quantity")
axes[1].right_ax.set_ylabel("Month over Month Change (%)")
monthly_sales.plot(
    ax=axes[2], grid=True
)
monthly_sales_changes.plot(
    ax=axes[2], secondary_y=True, color="silver", linestyle="dashed"
)

```

```

axes[2].set_title("Monthly Sales")
axes[2].set_ylabel("Sales")
axes[2].right_ax.set_ylabel("Month over Month Change (%)")
plt.show()

```

In this code, we are creating three line charts. The first one is for the monthly unique order, the second one is for the monthly order quantities, and the last one is for the monthly sales. This code generates the following charts:

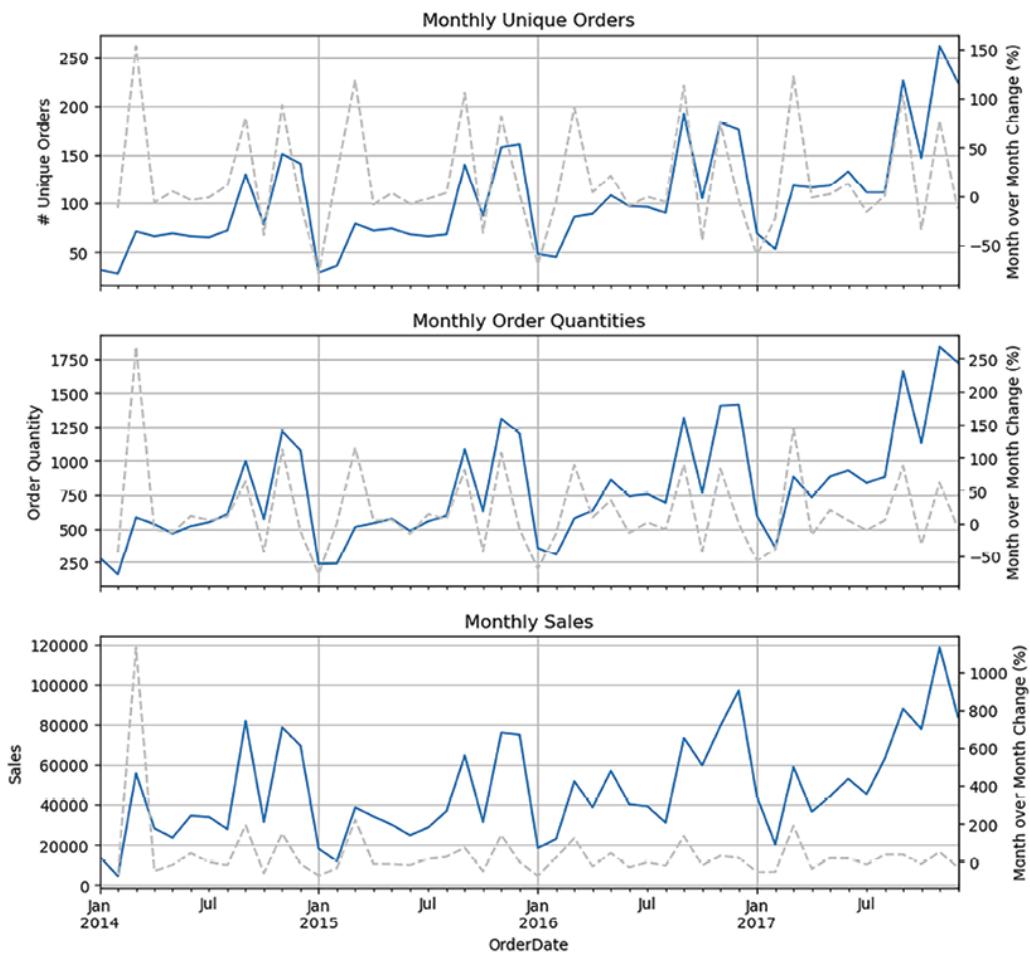


Figure 4.1: Visualization of monthly time-series data

Let's dive into these charts deeper. As you may notice, all three charts show clear repeating patterns. They spike in March, September, and November-December. This can be confirmed by both looking at the solid lines, which are overall monthly values, and looking at the dotted lines, which are month-over-month changes in the values. This suggests that this business is cyclical with a larger volume of sales in those corresponding months.

Marketers should turn this insight into actions by marketing more heavily slightly before the peak seasons to capture the greatest amount of sales potential. Also, marketers can offer discounts during the months that show weaker sales volumes in an effort to manage the inventory, as well as boost off-season sales.

As you can see, visualizing time-series data itself gives great insight into the business cycles and how to prepare for the peak and trough seasons. Another thing you may notice from these charts is they are overall in an uptrend, meaning the values are typically rising year over year. This overall trend is more easily identifiable when we look at the moving averages in the time-series charts.



You can not only sample time-series data by monthly frequency but also by daily, weekly, or yearly frequencies. There are many options you can resample by, so try different resample frequencies and see what other trends you can observe!

**Reference:**

[https://pandas.pydata.org/pandas-docs/stable/user\\_guide/timeseries.html#resampling](https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#resampling)

# Moving averages

**Moving averages**, which are the averages over specific time periods, are good at smoothing out spiky or noisy time-series data and showing overall trends. They are essential tools to answer questions like “Are our product sales in an overall uptrend or downtrend?”. Within our example data, we can do the following to compute moving averages:

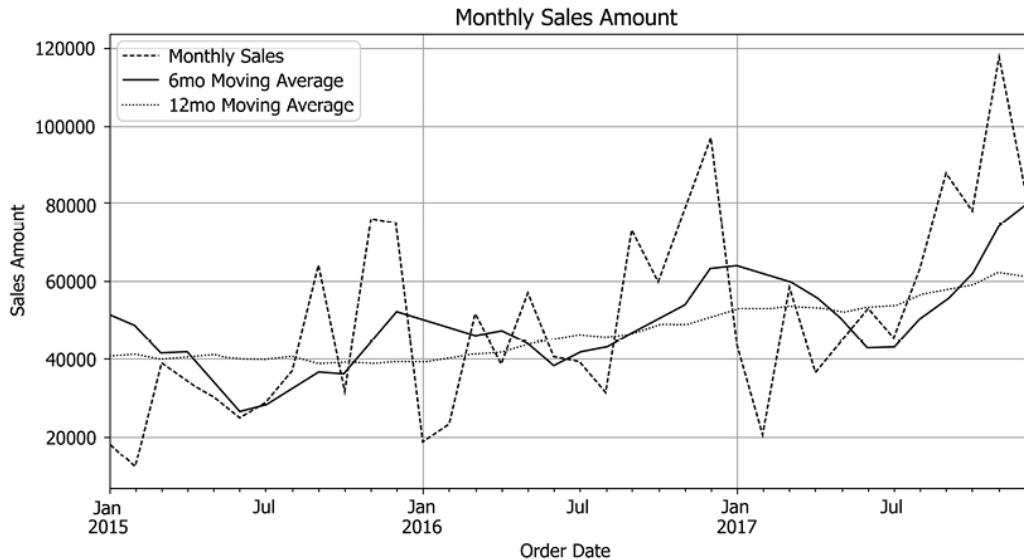
```
m6_ma_sales = monthly_sales.rolling(6).mean()  
m12_ma_sales = monthly_sales.rolling(12).mean()
```

As you can see from this code, you can use the `rolling` function within `pandas` and apply the `mean` function to get the moving average of monthly sales. The input into the `rolling` function defines how many periods you would like to average the values for. Here, we have computed the 6-period moving average and the 12-period moving average of monthly sales.

You can use the following code to visualize the moving averages:

```
ax = monthly_sales["2015-01-01":].plot(figsize=(10,5))  
m6_ma_sales["2015-01-01":].plot(ax=ax, grid=True)  
m12_ma_sales["2015-01-01":].plot(ax=ax, grid=True)  
ax.set_ylabel("Sales Amount")  
ax.set_xlabel("Order Date")  
ax.set_title("Monthly Sales Amount")  
plt.legend(["Monthly Sales", "6mo Moving Average", "12mo Moving  
plt.show()
```

The resulting chart will look as follows:



*Figure 4.2: Monthly sales with 6-month and 12-month moving averages*

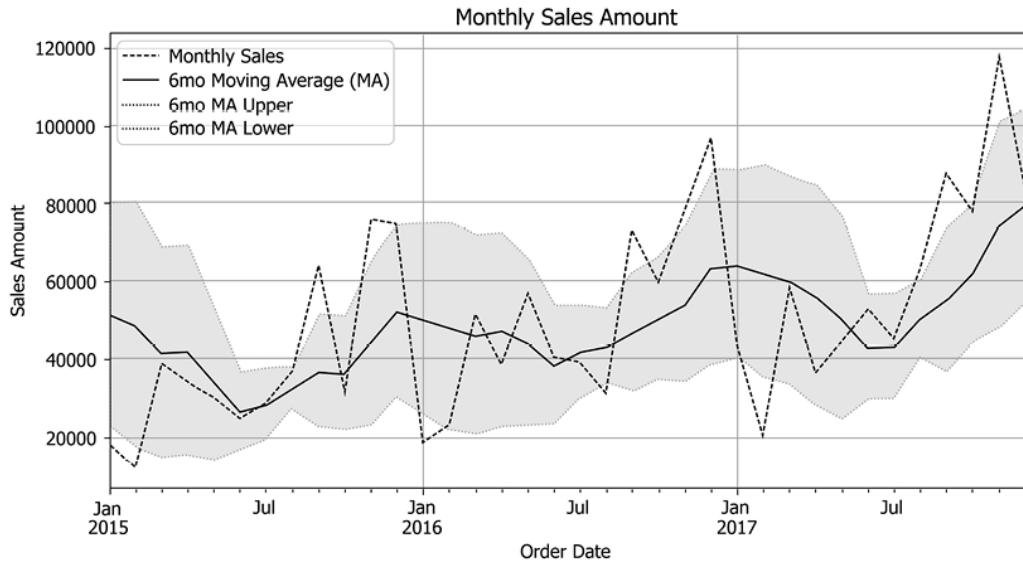
As expected, the 12-month moving average chart is smoother than the 6-month moving average chart. However, the 6-month moving average chart is more responsive or captures the monthly trends more closely than the 12-month moving average chart while still smoothing out the spikes in the monthly sales chart. As you may notice, the moving average charts show a general uptrend in the monthly sales data, which may have been harder to notice due to spikes and noises when we were only looking at the monthly chart.

Moving averages can also help you understand whether the spikes in certain months are within the normal range or are abnormal. Take a look at the following code:

```
m6_ma_sales_std = monthly_sales.rolling(6).std()
ax = monthly_sales["2015-01-01":].plot(figsize=(10,5))
m6_ma_sales["2015-01-01":].plot(ax=ax, grid=True)
(m6_ma_sales["2015-01-01"] + m6_ma_sales_std["2015-01-01"]).p
    ax=ax, grid=True, linestyle="dashed", color="silver", linewidth=2)
(m6_ma_sales["2015-01-01"] - m6_ma_sales_std["2015-01-01"]).p
```

```
        ax=ax, grid=True, linestyle="dashed", color="silver", linewidth=1)
ax.set_ylabel("Sales Amount")
ax.set_xlabel("Order Date")
ax.set_title("Monthly Sales Amount")
dates = m6_ma_sales["2015-01-01":].index
plt.fill_between(
    dates,
    m6_ma_sales["2015-01-01":] + m6_ma_sales_std["2015-01-01":],
    m6_ma_sales["2015-01-01":] - m6_ma_sales_std["2015-01-01":],
    facecolor="grey",
    alpha=0.2,
)
plt.legend(["Monthly Sales", "6mo Moving Average (MA)", "6mo MA Std Dev"])
plt.show()
```

This code calculates the moving standard deviations (which is the same as moving averages but measuring standard deviations over a certain period of time in the past) over a 6-month period by using the `std` function and visualizes data with the bands where the upper boundary is one standard deviation above the 6-month moving average and the lower boundary is one standard deviation below the 6-month moving average. The chart will look as follows:



*Figure 4.3: Monthly sales with 6-month moving average and 1-std boundaries*

This chart shows how the monthly sales volumes look compared to the moving averages and one standard deviation boundaries. Depending on your preference, you may want to consider values within one standard deviation as normal or within two standard deviations as normal. Using this example, if we assume values within one standard deviation to be normal, we can see that the months of February, September, November, and December are abnormal months and went beyond the defined boundaries, where the sales in February dropped significantly lower than expected and the sales in September, November, and December increased significantly higher than expected.

Understanding the overall trend of a business is equally as important as understanding the seasonal or cycles within a business. Depending on the general trend, you can be better informed as to whether to expect higher or lower sales and marketing potential than last year. If the overall trend is in an uptrend, you should expect the demand for a given month is likely to be higher than the demand for the same month in the previous year. Also,

having a good grasp of the sales that go above or below the normal ranges, such as during Black Fridays or the Christmas season, can help to build efficient marketing strategies to overcome the shortages and the abundance of marketing potential. Moving averages and moving standard deviations are handy tools for identifying overall trends and normal expected ranges.

## Autocorrelation

Another basic time-series analysis that comes in handy is **autocorrelation**. Autocorrelation represents the correlation with lagged versions of itself. It will be easier to discuss with an example. Take a look at the following code:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(
    monthly_sales, lags=25
)
plt.show()
```

Here, we are utilizing the `statsmodels` package, which you can install using the `pip install statsmodels` command. The `plot_acf` function will plot the autocorrelation of a given series for you. Here, we are plotting the autocorrelation of the monthly sales for the past 25 periods. The chart should look as follows:

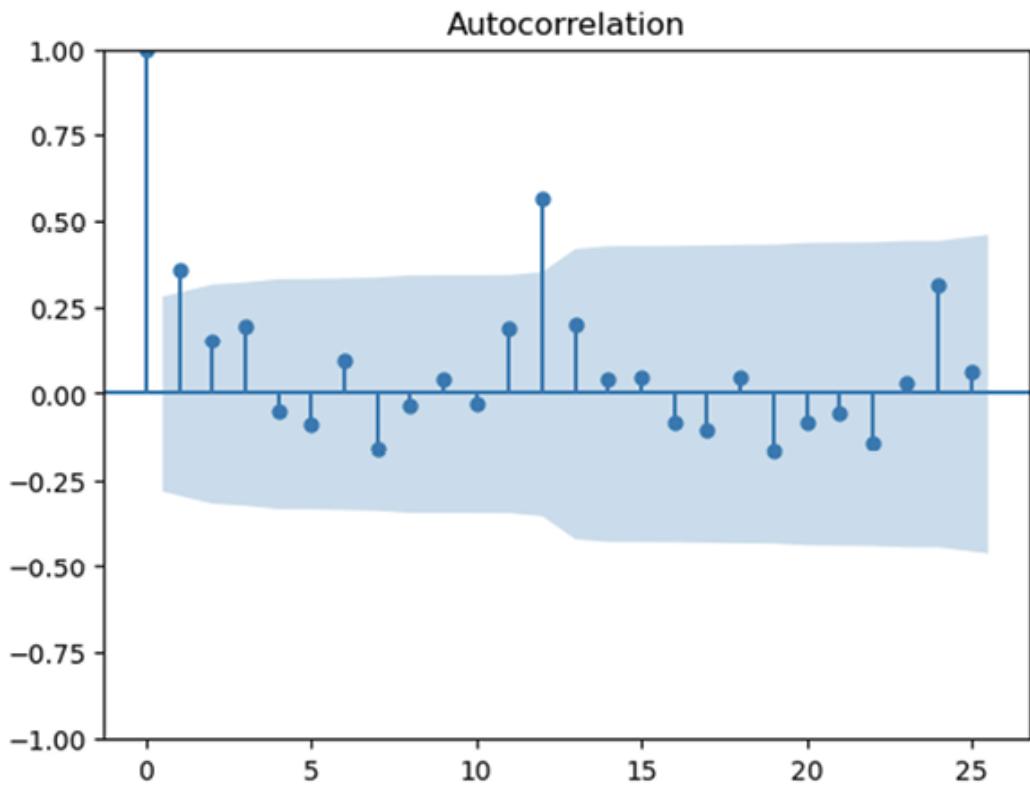


Figure 4.4: Autocorrelation plot of monthly sales data

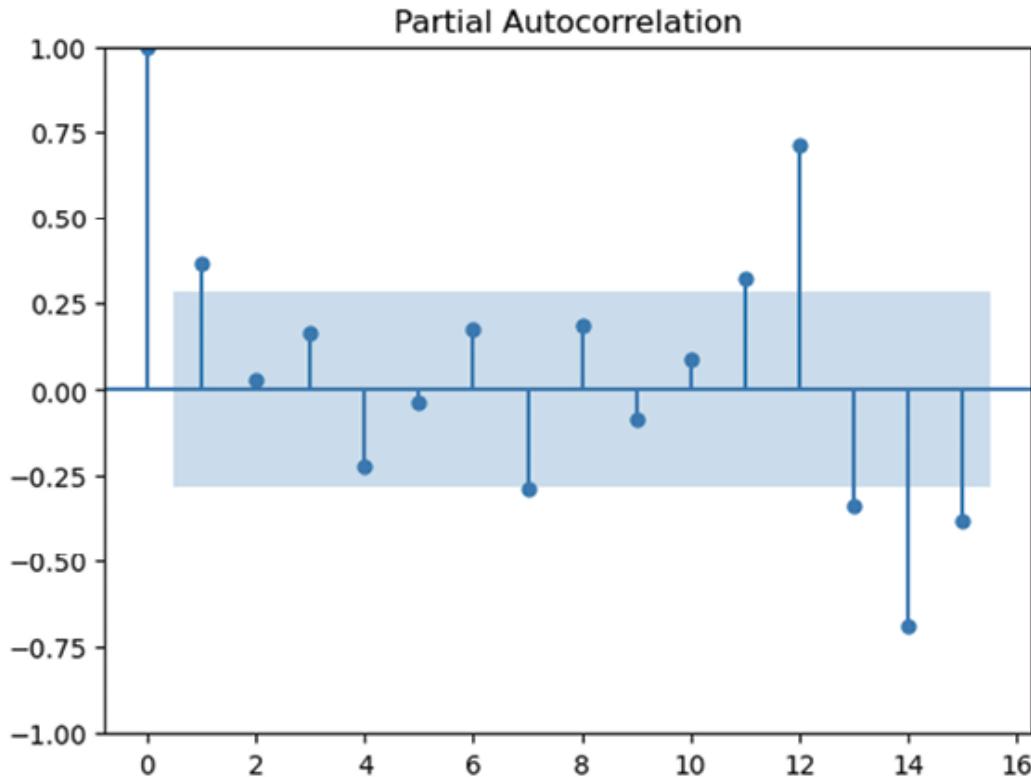
Let's analyze this chart more closely. The x-axis represents the number of lags and the y-axis is the degree of correlation. This chart shows the correlation between the current point of time against one period before, two periods before, and up to 25 periods before. Here, in the monthly sales data, we have relatively strong positive correlations up to three lagged periods and slight negative correlations with the fourth and fifth periods. This suggests that if there were increases in sales for the past 3 months, it is likely that the current month's sales will increase, and vice versa. Also, this means that observing an increase in sales 4–5 months ago likely results in a decrease in sales during the current month. As this example shows, autocorrelation is a way to see how different time periods may have affected the current time period's result. In the *Time series forecasting models* section of this chapter, we will experiment with the **autoregressive**

**integrated moving average (ARIMA)** model, and the autoregression part of it uses these lagged variables for time-series forecast modeling.

**Partial autocorrelation**, in contrast to autocorrelation, ignores the influences of intermediate lags. Unlike autocorrelation, partial autocorrelation measures the direct impact and correlation at each lag. The `plot_pacf` function can be used to plot partial autocorrelations, as in the following code:

```
plot_pacf(  
    monthly_sales, lags=15  
)  
plt.show()
```

Here, we are plotting partial autocorrelations up to period 15. The chart should look as follows:



*Figure 4.5: Partial autocorrelation plot of monthly sales data*

As you can see, the degrees of correlations at each lag are different compared to the previous autoregression chart. This is due to the fact that partial autoregression measures the direct correlation at each lag, whereas autoregression includes the influence of intermediate lags. In this example, we can see that there is a significant positive partial autocorrelation at lag 1 and a negative partial autocorrelation at lag 4. This suggests that the prior month's increase or decrease in sales is likely to result in an increase and decrease in the current month's sales, respectively, whereas the increase in sales 4 months ago is likely to result in a decrease in the current month's sales, excluding the influences of the sales results in the months between. Intuitively, these can be a result of natural business cycles, where, if your business cycle is quarterly, then you will likely see negative autocorrelations at lags 3–4, and for semi-annual cycles, you are likely to see negative autocorrelations at lags 6–7.

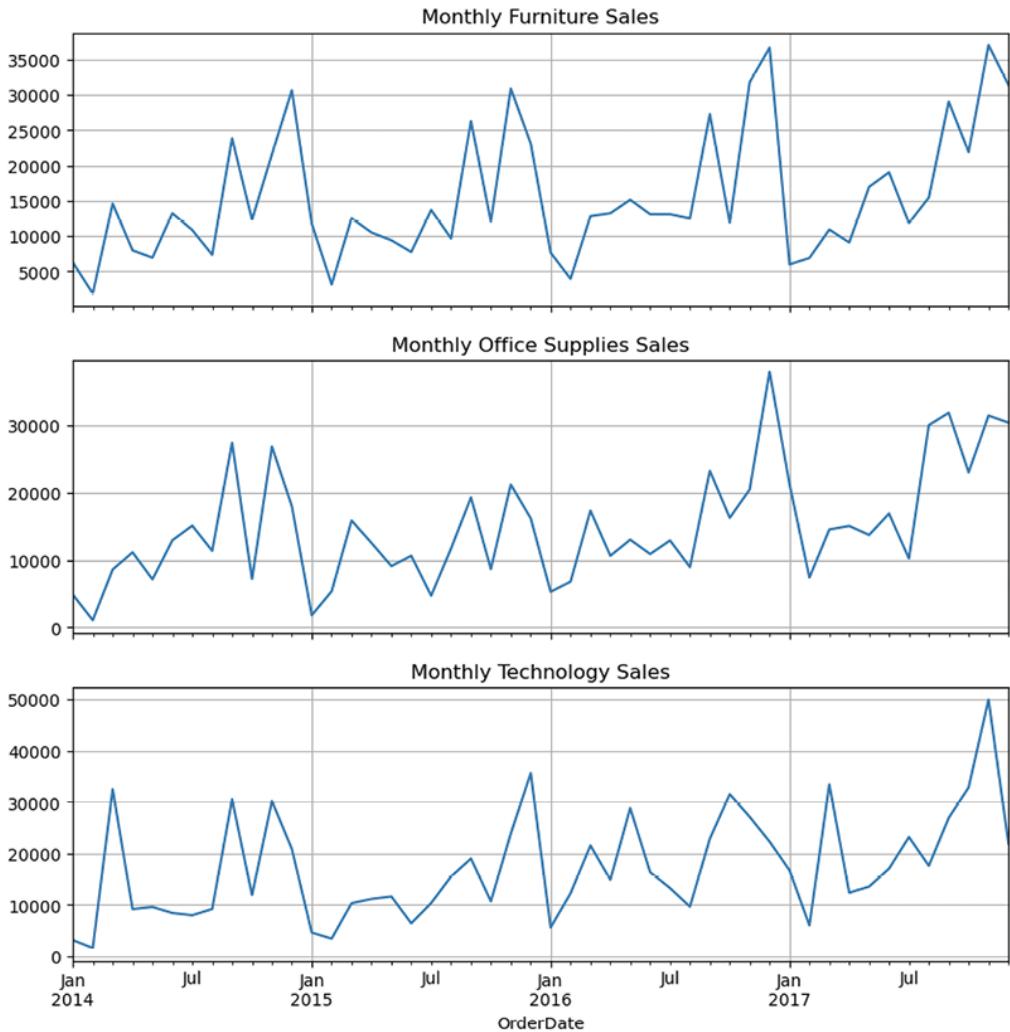
## Product trends

All of the analysis and visualizations we have discussed so far can be applied to more granular levels. We have only looked at the overall monthly sales time-series data. However, we can dissect it by different products and discover how each product may have different overall trends and cyclical nature in the demands. We can also look at how different states or geographic regions show trends and business cycles that may be different from the overall sales trends.

For illustration purposes, we will dissect the trends by different product categories. Take a look at the following code:

```
furniture_monthly_sales = ts_df.loc[
    ts_df["Category"] == "Furniture"
]["Sales"].resample("MS").sum()
office_monthly_sales = ts_df.loc[
    ts_df["Category"] == "Office Supplies"
]["Sales"].resample("MS").sum()
tech_monthly_sales = ts_df.loc[
    ts_df["Category"] == "Technology"
]["Sales"].resample("MS").sum()
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(10, 10), sharex=True)
furniture_monthly_sales.plot(ax=axes[0], grid=True)
axes[0].set_title("Monthly Furniture Sales")
office_monthly_sales.plot(ax=axes[1], grid=True)
axes[1].set_title("Monthly Office Supplies Sales")
tech_monthly_sales.plot(ax=axes[2], grid=True)
axes[2].set_title("Monthly Technology Sales")
plt.show()
```

First, we create time-series records for each product category. Largely, we are creating three time series of monthly Furniture, Office Supplies, and Technology sales, and then showing these charts side by side. The resulting charts look as follows:



*Figure 4.6: Visualization of monthly sales data for Furniture, Office Supplies, and Technology products*

The overall trends are similar across these three product lines, where the sales spike in September and November/December. However, the spike in March sales is the most prominent in Technology sales and there was a rapid growth in Furniture sales from January 2017. Also, Office Supplies sales seem to spike more frequently than the other two product lines; they seem to spike every quarter or so, which may be a result of quarterly restocking of office supplies. As these results show, different product lines tend to show slightly different behaviors across time.



Try diving deeper into segmentation and how time-series data differs by different segmentations. Try dissecting different variables, such as region, city, and sub-category!

## Trend and seasonality decomposition

We have seen how there are natural trends and cycles that are shown from the time-series data. By visualizing charts and utilizing moving averages, we were able to identify overall trends and seasonalities. However, there are more statistical approaches to decomposing time-series data into trend and seasonality components. Largely, there are two main ways to do time-series decomposition:

- **Additive:** As the name suggests, the **additive** time-series decomposition method decomposes the data into trend, seasonality, and error (which is the component that cannot be explained by the overall trend and seasonality) so that when they are summed together, it can reconstruct the original time-series data:

$$Y_t = \text{Trend}_t + \text{Seasonality}_t + \text{Error}_t$$

- **Multiplicative:** On the other hand, the **multiplicative** time-series decomposition method decomposes the data into trend, seasonality, and error in a way that when they are multiplied together, the original time-series data can be reconstructed. The equation looks as follows:

$$Y_t = \text{Trend}_t * \text{Seasonality}_t * \text{Error}_t$$

Now let's look at these methods in greater detail.

# Additive time series decomposition

Conveniently, the `statsmodels` package provides a function for easy trend and seasonality decomposition. As an example, we will decompose monthly `Furniture` sales data and see what trends and seasonality it has. Take a look at the following code:

```
import statsmodels.api as sm
decomposition = sm.tsa.seasonal_decompose(
    furniture_monthly_sales, model='additive'
)
fig = decomposition.plot()
plt.show()
```

As this code suggests, we are using the `seasonal_decompose` function for the time-series decomposition of monthly `Furniture` sales data, which is in the variable with the name `furniture_monthly_sales`. Another thing to note is the `model` parameter to the `seasonal_decompose` function, with which you can decide whether to use additive or multiplicative approaches.

This time-series decomposition can easily be visualized using the `plot` function of the output of the `seasonal_decompose` function. The chart should look something like the following:



*Figure 4.7: Time-series decomposition plot of monthly furniture sales data*

The top chart shows the original time-series data, which, in our case, is the monthly Furniture data. The second chart from the top shows the decomposed trend. As expected, there is a clear uptrend, where the sales grow year over year. The third chart is the decomposed seasonality chart. Here, it shows clear spikes in sales during September, November, and December and drops in sales during February. Lastly, the bottom chart shows the residuals or the error terms when this data is decomposed into trend and seasonality. According to the equation discussed in the previous section, these bottom three charts correspond to each decomposed component of the original time-series data.

When properly decomposed, the error terms should be stationary, meaning there should not be a noticeable pattern in the residuals that are dependent on time. In our example, there is no noticeable pattern across time in the residuals, so the decomposed series seems reasonable. We can examine how

well the decomposed series captures the original series by reconstructing the time-series data based on the trend and seasonality decomposition and how big a gap there is between the original series and the reconstructed series. Take a look at the following code:

```
reconstructed_wo_resid = decomposition.trend + decomposition.se  
corr = np.corrcoef(  
    list(furniture_monthly_sales[dates]),  
    list(reconstructed_wo_resid[dates]))  
)[0,1]  
dist = np.sqrt(  
    np.square(  
        furniture_monthly_sales[dates] - reconstructed_wo_resid  
    ).sum()  
)  
rmse = np.sqrt(  
    np.square(  
        furniture_monthly_sales[dates] - reconstructed_wo_resid  
    ).mean()  
)
```

The first line shows how we can reconstruct the time-series data using the decomposed trend and seasonality. Since we have used the additive approach for time-series decomposition in this example, we simply sum the trend and seasonality components together to get the reconstructed series.

There are three metrics that we are using to measure the similarity or dissimilarity between the original series and the reconstructed series:

- **Correlation:** Using the `numpy` package's `corrcoef` function, this metric measures the similarity between the original and reconstructed series.
- **Euclidean distance:** This is the square root of the sum of the squared errors. This metric measures how big of a gap there is between the

original and reconstructed series.

- **Root mean squared error (RMSE):** This is the square root of the mean of the squared errors. This metric measures the degree of error between the original and reconstructed series.

The following code can be used:

```
print(f"Correlation: {corr:.02f}\nEuclidean Distance: {dist:.02
```

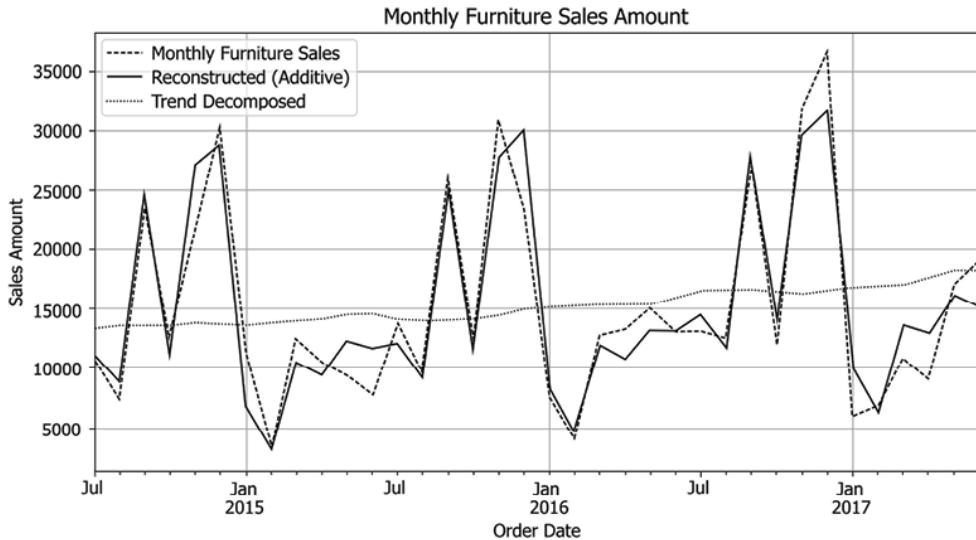
The result should look like the following:

```
Correlation: 0.95
Euclidean Distance: 15995.62
RMSE: 2665.94
```

You can also visually compare the reconstructed series against the original series with the following code:

```
dates = reconstructed_wo_resid.dropna().index
ax = furniture_monthly_sales[dates].plot(figsize=(10,5))
reconstructed_wo_resid[dates].plot(ax=ax, grid=True)
decomposition.trend[dates].plot(ax=ax, grid=True)
ax.set_ylabel("Sales Amount")
ax.set_xlabel("Order Date")
ax.set_title("Monthly Furniture Sales Amount")
plt.legend(["Monthly Furniture Sales", "Reconstructed (Additive
plt.show()
```

The chart should look as follows:



*Figure 4.8: Monthly furniture sales data along with decomposed trend and reconstructed series*

As you can see from this chart and as expected from the previous metrics, the reconstructed series does not capture 100% of the original series. However, they move closely together and the reconstructed series has a high degree of similarity with the original series.

## Multiplicative time series decomposition

We will compare the results of the additive approach against the results of the multiplicative approach results to see which one captures the original time series more closely. As you may have guessed, you can replace the `model` parameter of the `seasonal_decompose` function with `multiplicative`, as shown in the following code:

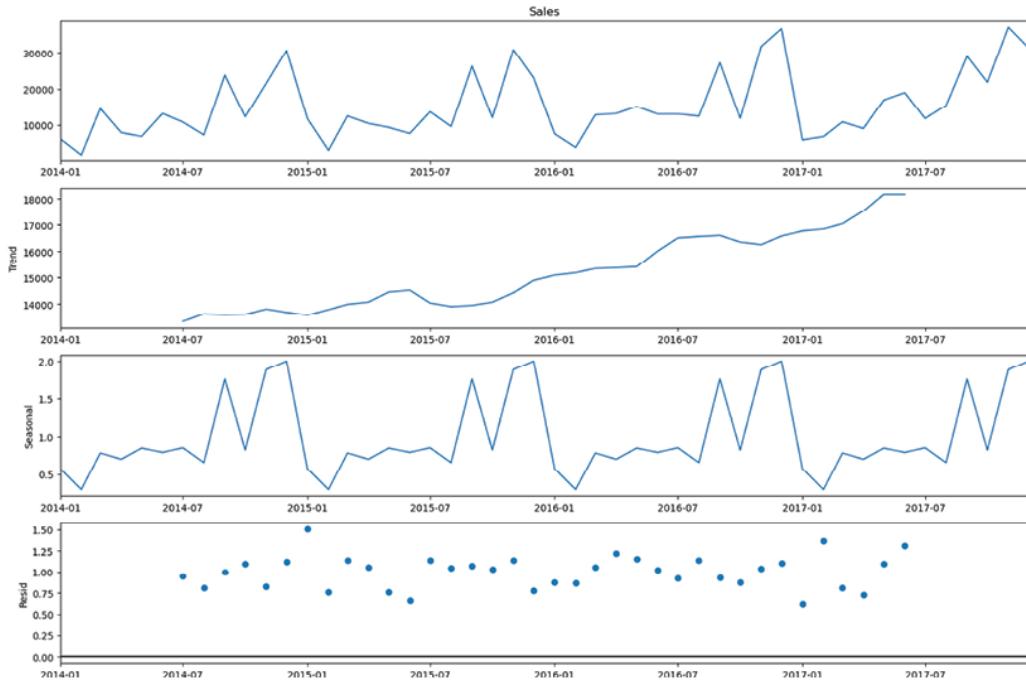
```
decomposition = sm.tsa.seasonal_decompose(
    furniture_monthly_sales, model='multiplicative'
)
```

```

fig = decomposition.plot()
plt.show()

```

The resulting chart should look as follows:



*Figure 4.9: Time-series decomposition plot of monthly furniture sales data with the multiplicative approach*

This should look very similar to the chart with the additive approach.

However, there are two key things that are noticeably different:

- The y-axis of the *Seasonal chart*, which is the second chart from the bottom, ranges from **0** to about **2.0**, whereas the y-axis of the Seasonal component of the additive chart ranges from about – 10,000 to 35,000.
- Similarly, the y-axis of the *Residual chart*, which is the bottom chart, ranges from **0.00** to about **1.5**.

As you may have guessed, this is because we used the multiplicative approach in this example as opposed to the additive approach. In order to

reconstruct the original series from the decomposed trend and seasonality components, we need to multiply these two components together, as in the following code:

```
reconstructed_wo_resid = decomposition.trend * decomposition.se
```

Similar to before, we can measure the degree of similarity between the reconstructed series from the multiplicative approach and the original series based on the three metrics we have used before (correlation, Euclidean distance, and RMSE) using the following code:

```
corr = np.corrcoef(  
    list(furniture_monthly_sales[dates]),  
    list(reconstructed_wo_resid[dates]))  
)[0,1]  
dist = np.sqrt(  
    np.square(  
        furniture_monthly_sales[dates] - reconstructed_wo_resid  
    ).sum()  
)  
rmse = np.sqrt(  
    np.square(  
        furniture_monthly_sales[dates] - reconstructed_wo_resid  
    ).mean()  
)
```

The similarity measures can be viewed using the following code:

```
print(f"Correlation: {corr:.02f}\nEuclidean Distance: {dist:.02f}
```

This gives us the following results:

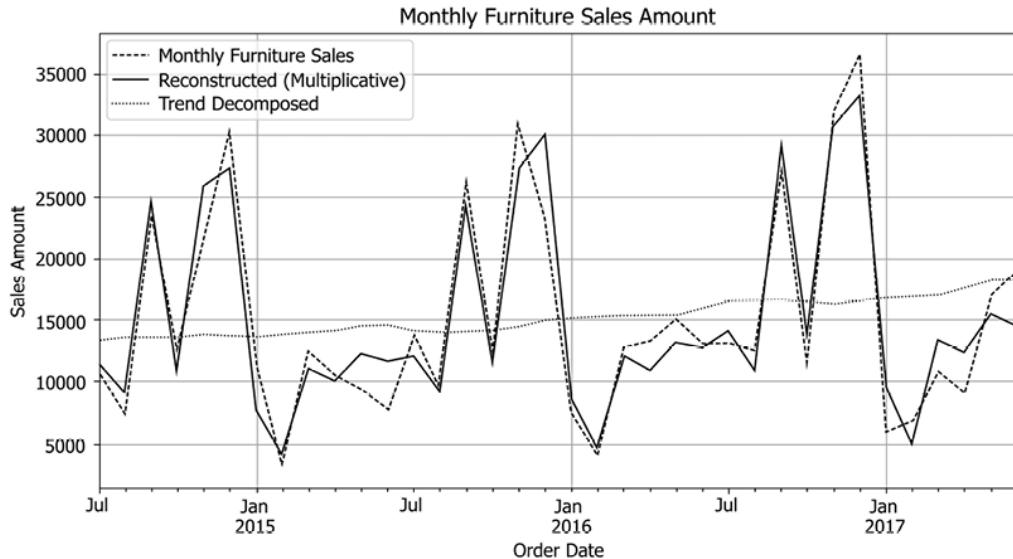
```
Correlation: 0.95
Euclidean Distance: 15307.16
RMSE: 2551.19
```

If you compare these results against the previous results with the additive approach, you will notice that the Euclidean distance and RMSE are much lower with the multiplicative approach, while the correlation measure is similar. The Euclidean distance with the multiplicative approach is about **700** less than it is with the additive approach, and the RMSE with the multiplicative approach is about **100** less than it is with the additive approach. This suggests that the multiplicative decomposition approach may capture the original time-series data better in this case than the additive approach.

Similarly, we can visually inspect the multiplicative approach's reconstruction results with the following code:

```
dates = reconstructed_wo_resid.dropna().index
ax = furniture_monthly_sales[dates].plot(figsize=(10,5))
reconstructed_wo_resid[dates].plot(ax=ax, grid=True)
decomposition.trend[dates].plot(ax=ax, grid=True)
ax.set_ylabel("Sales Amount")
ax.set_xlabel("Order Date")
ax.set_title("Monthly Furniture Sales Amount")
plt.legend(["Monthly Furniture Sales", "Reconstructed (Additive")
plt.show()
```

The resulting chart should look as follows:



*Figure 4.10: Monthly furniture sales plot with the decomposed trend and reconstructed series with the multiplicative approach*

We have seen how we can decompose time-series data statistically with the `statsmodels` package in this section. As previously mentioned, having an in-depth understanding of general trends as well as seasonal trends is critical in formulating product and target marketing strategies as the alignments of the marketing strategies with these trends and the timing of the actions ahead of the expected trends dictate the successes and failures of marketing campaigns. You do not want to be too late in the trend but you also do not want to be too early in the trend. This time-series decomposition technique should be a handy tool to strategize and time your marketing campaigns.

# Time series forecasting models

The overall trends and the seasonalities bring valuable insights for efficient and timely marketing campaigns. We have discussed how time-series decomposition can help marketers put out timely promotions in order to capture the maximum sales and marketing potential when the demands are expected to rise, as well as minimize the dips and excess inventory when the demands are expected to fall via out-of-season sales promotions or focusing on the products that have different peak and trough cycles.

We can take a step further with **machine learning (ML)** and **artificial intelligence (AI)** and build time-series forecasting models. The future forecasts with these AI/ML models can play a pivotal role not only for marketing but also for various other business units, including sales, operations, finance, supply chain, procurement, and many others. By utilizing time-series forecasts, marketers can optimize their marketing goals in numerous ways:

- If a marketing goal is to promote a *new product*, then the time-series forecast model output can inform the marketer when will be the best time to start promoting based on the expected demand rises or falls of similar product categories.
- If a marketing goal is to promote an *off-season sales* increase, then the time-series model can be built to forecast the types of off-season promotions that may result in the highest sales.
- If a marketing goal is to clean out the *excess inventory*, a time-series model that forecasts the different demands in different regions or demographics can help the marketer target certain regions or demographics for maximum efficiency in reducing the excess inventory.

There are numerous time-series forecasting algorithms that can be used to build forecast models. Ranging from traditional statistical time-series models to more modern deep learning-based time-series models, there are various algorithms that can be used to forecast time-series data. In this section, we will experiment with the two most frequently used time-series models: ARIMA and Prophet.

## ARIMA

The **ARIMA** model is a statistical model that is often used to predict future time-series data based on past values. ARIMA is a form of regression analysis that we discussed in *Chapter 3*, but there are three key components that the ARIMA model is composed of:

- **Autoregression:** The **AR** part of the ARIMA model

Similar to what we discussed for autocorrelation, autoregression is a regression on its own lagged variables, where each lagged variable is a feature of the forecasting model.

- **Integrated:** The **I** part of the ARIMA model

This is the difference between the values and their previous values to achieve stationarity that, as discussed previously, means the errors do not depend on the time component.

- **Moving Average:** The **MA** part of the ARIMA model

As discussed previously, the moving average is an average over a rolling window and the ARIMA model regresses on these moving averages.

Each of these three components – **Autoregression (AR)**, **Integrated (I)**, and **Moving Average (MA)** – in the ARIMA model has its own parameter. Typically, these parameters have notations of  $p$ ,  $d$ , and  $q$ , and are defined as follows:

- $p$ : The number of lag periods in the model for the AR component
- $d$ : The number of times data are differenced for the I component
- $q$ : The rolling window of the moving average for the MA component

## Training the ARIMA model

The `statsmodels` package in Python has a module that makes it easy for us to build ARIMA models. Take a look at the following code:

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(furniture_monthly_sales[:'2017-06-01'], order=(12
model_fit = model.fit()
```

As this code snippet shows, we are using the `ARIMA` class to build an ARIMA model in Python. For the inputs for the model, we are giving the monthly furniture sales time-series data, which is the variable

`furniture_monthly_sales` in our case, and the parameters that we have discussed previously. As an example, we are using `12` for the  $p$  or AR component, `1` for the  $d$  or I component, and `3` for the  $q$  or MA component.

We recommend you try various combinations when you actually build a forecasting model with ARIMA to find the most optimal set of parameters. Here, we are training our ARIMA model up to June 2017 and we will test our predictions for July to December 2017.

With the following command, we can view the trained model results:

```
print(model_fit.summary())
```

The model summary looks as follows:

SARIMAX Results						
Dep. Variable:	Sales	No. Observations:	42			
Model:	ARIMA(12, 1, 3)	Log Likelihood	-424.032			
Date:	Wed, 10 Apr 2024	AIC	880.065			
Time:	14:33:09	BIC	907.482			
Sample:	01-01-2014 - 06-01-2017	HQIC	890.049			
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.1478	0.136	1.083	0.279	-0.120	0.415
ar.L2	0.1186	0.122	0.969	0.332	-0.121	0.358
ar.L3	-0.0144	0.128	-0.112	0.911	-0.266	0.237
ar.L4	-0.1569	0.150	-1.047	0.295	-0.451	0.137
ar.L5	-0.0825	0.182	-0.453	0.651	-0.440	0.275
ar.L6	0.0695	0.121	0.573	0.567	-0.168	0.307
ar.L7	-0.0390	0.169	-0.231	0.818	-0.371	0.293
ar.L8	0.0734	0.170	0.431	0.666	-0.260	0.407
ar.L9	-0.0575	0.125	-0.459	0.646	-0.303	0.188
ar.L10	-0.0739	0.156	-0.474	0.636	-0.379	0.232
ar.L11	-0.0249	0.097	-0.256	0.798	-0.216	0.166
ar.L12	0.7454	0.082	9.115	0.000	0.585	0.906
ma.L1	-1.0175	0.197	-5.178	0.000	-1.403	-0.632
ma.L2	-0.1318	0.286	-0.461	0.645	-0.692	0.428
ma.L3	0.2450	0.270	0.908	0.364	-0.284	0.774
sigma2	1.824e+07	1.23e-08	1.48e+15	0.000	1.82e+07	1.82e+07
Ljung-Box (L1) (Q):	0.29	Jarque-Bera (JB):	33.45			
Prob(Q):	0.59	Prob(JB):	0.00			
Heteroskedasticity (H):	0.34	Skew:	1.23			
Prob(H) (two-sided):	0.05	Kurtosis:	6.68			

Figure 4.11: Summary of the ARIMA model fit

Some of the key things to note in this model summary output are as follows:

- **ar:** These are the AR components within the ARIMA model. As we have given 12 for the AR component, there are 12 lagged variables that this model regresses on and each `coef` shows the coefficient of each lag variable with the target variable.

- **ma:** These are the MA components within the ARIMA model. We have given 3 for our example for the MA component, so there are three variables that the model regresses on and each `coef` shows the coefficient of each MA variable with the target variable.
- **AIC/BIC:** We won't go into too much detail about these metrics, but the **Akaike information criterion (AIC)** and the **Bayes information criterion (BIC)** are the metrics that can be used to evaluate the model fit and compare among different models. The lower the values, the better the model fit is without overfitting.



To find an optimal set of parameters, you will have to run numerous simulations with different sets of `(p, q, d)` parameters. Or, you can also use a package, such as `pmdarima`, to automatically discover the optimal parameters for the ARIMA model.

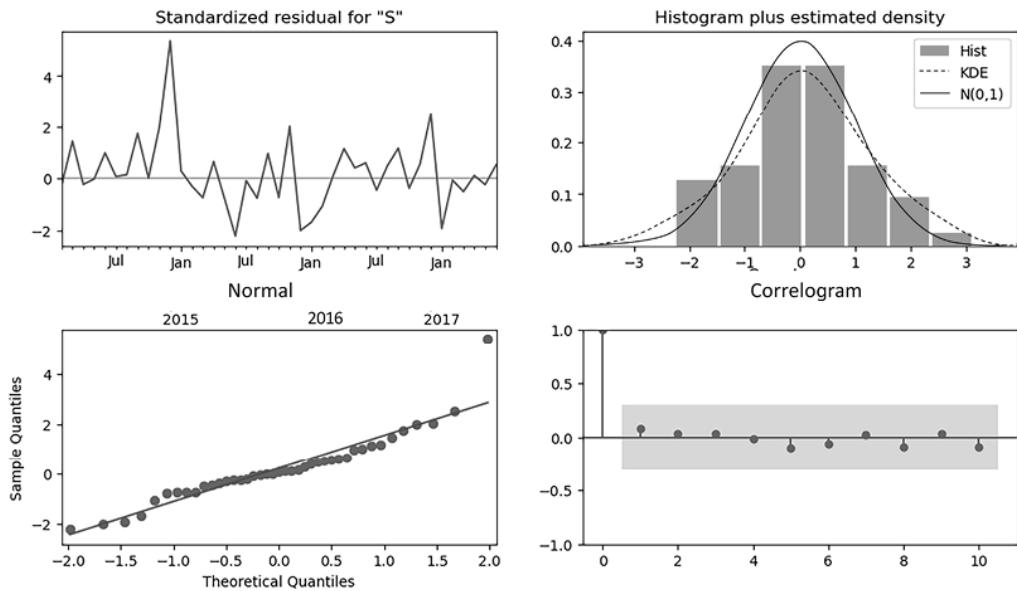
Reference: [https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto\\_arima.html](https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html)

## ARIMA model diagnostics

The `statsmodels` package provides a handy way of diagnosing the trained ARIMA model. The `plot_diagnostics` function of the `ARIMA` model can be used to visualize the key diagnostic plots, as in the following code:

```
model_fit.plot_diagnostics(figsize=(12, 6))
plt.show()
```

The chart should look as follows:



*Figure 4.12: ARIMA model diagnostics plots*

As shown in this chart, there are four components in this diagnostic plot.

Let's dive deeper into each of these:

- **Standardized residuals over time (top-left chart):** This chart shows the residuals or the errors across time. For a perfect model, we would expect it to be completely random without any noticeable pattern. However, in our case, there still are some minor seasonal patterns that are noticeable.
- **Histogram and estimated density of standardized residuals (top-right chart):** This chart shows the distribution of the standardized residuals. For a perfect model, we would expect it to show a Gaussian or normal curve with a mean of 0 and a standard deviation of 1. Our model is very close to the theoretical normal curve, which suggests that the residuals are nearly normally distributed.

- **Normal Q-Q plot (bottom-left chart):** This chart shows theoretical quantile distributions against the actual quantile distributions of the fitted model. It suggests that the residuals are normal when the dots are closely aligned with the straight line. In our case, it is not perfect but somewhat closely aligned with the straight line, which suggests that the residuals are nearly normally distributed.
- **Correlogram (bottom-right chart):** This chart shows the autocorrelation of residual terms across lag periods. The smaller the correlation values, the more random the residuals are. Our example shows minimal correlations that suggest that the residuals are not correlated with each other.

In summary, these diagnostic plots tell us that the residuals are generally normally distributed in our case.

## Forecasting with the ARIMA model

Now it is finally the time to make predictions with the ARIMA model that we have trained. Take a look at the following code first:

```
pred = model_fit.get_forecast(steps=6)
pred_ci = pred.conf_int()
```

The `statsmodels` package's `ARIMA` model provides a function named `get_forecast`. The `steps` parameter is used to define how many steps into the future you would like to make predictions for. It also provides a function named `conf_int`, which gives a confidence band of the predictions.

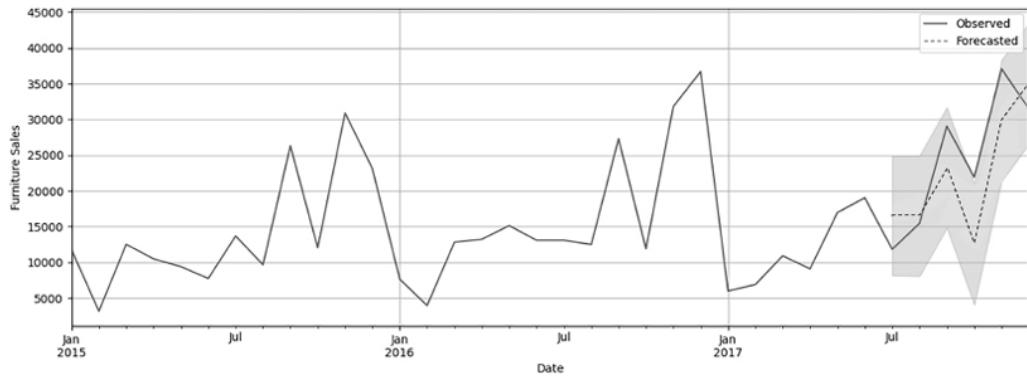
We can easily plot the prediction results with the following code:

```

ax = furniture_monthly_sales["2015-01-01":].plot(figsize=(15, 5))
pred.predicted_mean.plot(
    ax=ax, grid=True
)
ax.fill_between(
    pred_ci.index,
    pred_ci.iloc[:, 0],
    pred_ci.iloc[:, 1],
    color='cornflowerblue',
    alpha=.3
)
ax.set_xlabel('Date')
ax.set_ylabel('Furniture Sales')
plt.legend(["Observed", "Forecasted"])
plt.show()

```

When you run this code, you should get a chart similar to the following:



*Figure 4.13: ARIMA model prediction plot*

This chart shows previous monthly furniture sales data as well as the predictions and confidence band or interval for the predictions. As you may remember, we have trained the model with data up to June 2017 and we have made predictions for six steps from July 2017, which is from July to December 2017.

As you can see from these predictions, the predictions from the ARIMA model are directionally well aligned with the actual observed data. Also, the actual values fall within the confidence bands, suggesting the reliability of the usage of the predictions based on the confidence intervals. One of the frequently used metrics for measuring the accuracy of time-series forecasts is RMSE, using the following code:

```
rmse = np.sqrt(  
    np.square(furniture_monthly_sales["2017-07-01":] - pred.pre  
)  
rmse
```

```
5913.227463714012
```

This gives us the result `5913.227463714012`.

Here, the RMSE of our predictions for the next 6 months was about 5913. We will compare this value against other time-series forecasting models' values that we will experiment with in the following sections.



There are many other metrics that can be used to measure the time-series model performance, aside from RMSE. Some other commonly used metrics are **mean absolute error (MAE)**, **mean absolute percentage error (MAPE)**, and **mean absolute scaled error (MASE)**. Try some other regression metrics and see how they differ from each other!

## Prophet time-series modeling

**Prophet** is an open-source package from Meta (formerly Facebook) for time-series forecasting. Similar to the ARIMA model we have just discussed, the Prophet model also takes trend and seasonality into consideration, except there is more flexibility and more parameters you can fine-tune the time-series models with, and holiday effects are also included. So, there are largely three components in the Prophet model:

- **Growth (or trend):** Prophet models overall growth or trend. There are three assumptions you can make for your growth factor of the time-series data:
  - **Linear:** This is the default assumption of Prophet and is used when the overall trend is expected to be linear.
  - **Logistic:** This should be used when there is a cap or floor in the trend of your time-series data.
  - **Flat:** This is when you assume there is no growth over time.
- **Seasonality:** By default, this is set to *auto*, but depending on your observations within your time-series data, you can set it to model daily, weekly, or yearly seasonalities.
- **Holidays:** One of the key differentiators the Prophet has is its notion of holidays. As holidays have significant impacts on time-series outcomes, it is beneficial to be able to model holiday effects on your time-series data with Prophet.

In this section, we are going to experiment with modeling the monthly furniture sales with Prophet.



For more detailed information on the parameters that you can fine-tune with Prophet, we suggest you visit their official site



([https://facebook.github.io/prophet/docs/quick\\_start.html](https://facebook.github.io/prophet/docs/quick_start.html)) or their GitHub page (<https://github.com/facebook/prophet/blob/main/python/prophet/forecaster.py>).

## Training a Prophet model

In order to train a Prophet model, make sure you have the package installed first. You can use the following command to have the package installed:

```
pip install prophet
```

The first step to training a Prophet model is to prepare the data that it expects. Take a look at the following code:

```
from prophet import Prophet
monthly_furniture_sales_df = pd.DataFrame(
    furniture_monthly_sales[:'2016-12-01']
).reset_index()
monthly_furniture_sales_df.columns = ["ds", "y"]
```

As you can see from this code, we are using monthly furniture data up to December 2016 as our train set. The requirements for the Prophet model for the train set are the columns `ds` and `y`, where `ds` is for the dates and times of a given record and `y` is for the time-series values.

With this train set, we can easily train a Prophet model using the following code:

```
model = Prophet()
model.fit(monthly_furniture_sales_df)
```

Here, we are initializing the model with the `Prophet` class and with the default parameters, which essentially instructs the model to assume linear growth trends and seasonality with no holiday effects. Then, we use the `fit` function with the train set DataFrame that we have prepared previously to train the model.

## Forecasting with a Prophet model

Now that we have a trained Prophet model, it is time to make some predictions. One thing we need to do for a Prophet model to make predictions is to generate a series of dates it should make predictions for. Prophet provides a handy function to do this, as shown in the following:

```
dates = model.make_future_dataframe(periods=24, freq='MS')
```

Here, we are using the `make_future_dataframe` function. The `periods` parameter defines how many future dates or periods we would like to make predictions for and the `freq` parameter defines what the frequency of each future date or period should be, where we define it to be monthly with `'MS'` as an input for the parameter.

The newly created variable should now have dates ranging from January 2014 to December 2018. As you may remember, we have used monthly series up to December 2016 as our train set, so from January 2017 to December 2018 is essentially what we would like to make predictions for as out-of-sample predictions.

You can use the following code for generating predictions with the trained model:

```
forecast = model.predict(dates)
```

The `predict` function generates a DataFrame, `forecast`, which contains predicted data for each period, such as predicted value, upper and lower bound, modeled trend, and so forth. Some of the key fields within the `forecast` DataFrame that are the most relevant to us are as follows:

	ds	yhat	yhat_lower	yhat_upper
0	2014-01-01	6516.362514	4067.155620	9025.644286
1	2014-02-01	558.820965	-1888.963561	2753.614396
2	2014-03-01	12235.017313	9829.731852	14575.423549
3	2014-04-01	9042.415516	6847.255764	11440.875014
4	2014-05-01	8562.676158	6277.210776	11033.134529

Figure 14.14: Prediction results of a Prophet model

The `yhat` column is the predicted value for a given period and `yhat_lower` and `yhat_upper` are the lower and upper boundaries of the predicted confidence intervals.

We can easily visualize these predictions and prediction confidence intervals with the following code:

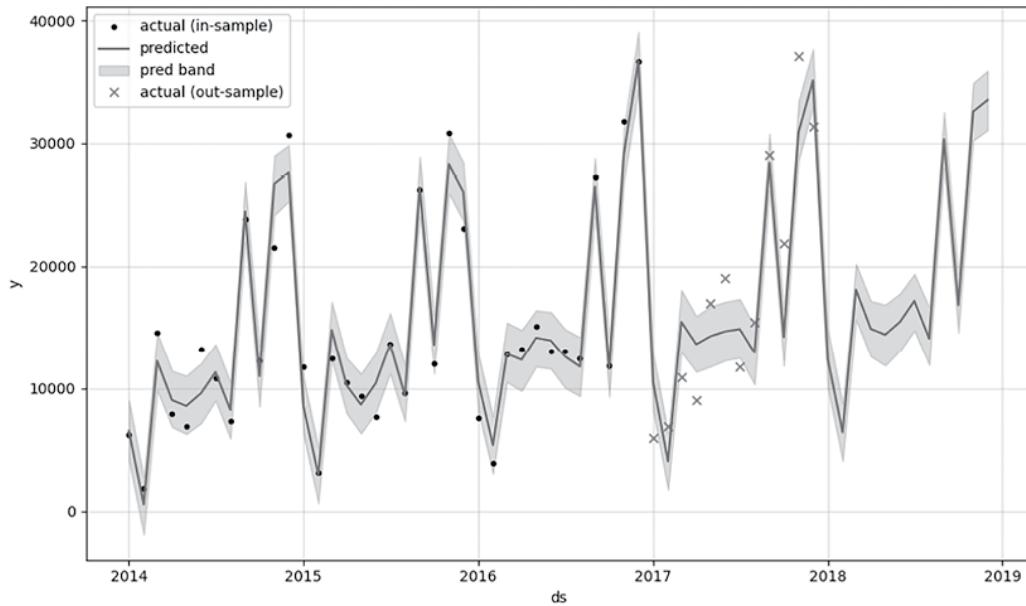
```
fig = model.plot(forecast, uncertainty=True)
ax = fig.axes[0]
outsample_dates = furniture_monthly_sales["2017-01-01":].index
ax.plot(
    outsample_dates,
    furniture_monthly_sales.loc[outsample_dates],
    "k.",
    marker="x",
    color="red"
```

```

)
plt.legend([
    "actual (in-sample)", "predicted", "pred band", "actual (ou
])
plt.show()

```

Let's take a closer look at this code. The Prophet model object has the `plot` function, which generates a plot with predictions as a line chart, prediction intervals as an area chart, and actual values as a scatter chart. Then, we add out-of-sample data points as a scatter plot with the marker `x`. This chart should look as follows:



*Figure 4.15: Prophet model prediction plot*

Similar to what we have done with the ARIMA model, we can also look at the RMSE of the Prophet model predictions using the following code:

```

rmse = np.sqrt(
    np.square(
        forecast.loc[

```

```
        forecast["ds"].isin(outsample_dates)
        ]["yhat"].to_numpy() - furniture_monthly_sales.loc[outs
    ).mean()
)
print(f"Out-Sample RMSE: {rmse:.02f}")
```

The result is as follows:

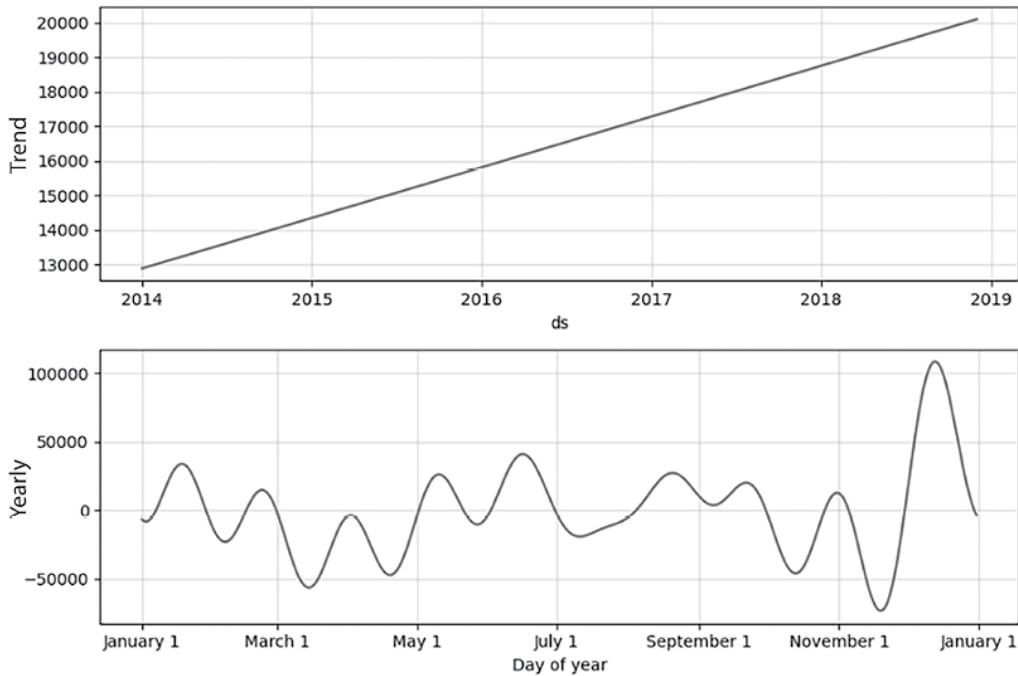
```
Out-Sample RMSE: 4295.65
```

Compared to the RMSE of the ARIMA model, which was around 5913, the Prophet model with an RMSE of about 4295 seems to have predictions that are closer to the actuals.

Lastly, Prophet also models trend and seasonality decompositions and provides an easy way to visualize them, as shown in the following code:

```
model.plot_components(forecast)
```

The resulting chart should look as follows:



*Figure 4.16: Prophet model time-series decomposition plot*

Similar to what we have seen previously when we decomposed the time-series data into trend and seasonality, the Prophet model also identified an overall uptrend and spikes in December and drops in January. As we have seen so far, Prophet provides handy and reliable tools for modeling time-series data.

With this knowledge, we will look at another way of modeling time-series data with a deep learning approach in the following section.

## Other time-series models

We have discussed time-series modeling with ARIMA and Prophet in this chapter. However, there are various other algorithms and models that can be used to model time-series data. There are largely three types of approaches to time-series modeling:

**1. Statistical models:** The statistical approaches to modeling time-series data have been around for decades. The ARIMA model is one of the most frequently used statistical models and was developed in the 1970s and is still used to date. To name a few other statistical models that are often used:

- **Exponential smoothing:** One of the oldest time-series forecasting methods, which gives more weight to recent observations for averaging time-series data points
- **Generalized AutoRegressive Conditional Heteroskedasticity (GARCH):** A frequently used model in finance that models the variance of the error terms that is dependent on time, such as increasing or decreasing volatility or variance over time
- **Seasonal ARIMA (SARIMA):** An extension of the ARIMA model that incorporates the seasonality component on top of the ARIMA components

**2. ML models:** Although not as frequently used as other statistical models and deep learning models, ML models are still used for time-series forecasting and for quick checks for predictability in time series. The advantage of ML models over statistical models is the ability to use various features when building forecasting models, whereas statistical models for time-series forecasting typically are univariate in nature. Here are some commonly used ML models:

- a. **Linear regression:** One of the most basic regression models in ML that can be used to model time-series data with various features. Feature engineering is the key to making a linear regression model powerful.

- b. **Tree-based models:** XGBoost, which learns the data by sequentially building decision trees that learn from previous trees' errors, or random forest, which learns the data by building a bag of decision trees that each learn subparts of the data, can be used for time-series forecasting. The ability to model interactions and relationships among various features provides an advantage in modeling time-series data that has complex intercorrelations among the features.
  - c. **Support vector machine (SVM):** SVM does not perform well when the dataset is noisy or the dimensionality of the dataset is large as it learns the data by finding a hyperplane that maximally separates different categories, and building such an effective hyperplane in high-dimensional space is difficult, but SVM is still used for time-series forecasting.
3. **Deep learning (DL) models:** With the rising accessibility and availability of compute resources, there have been lots of developments in DL-driven time-series modeling. Similar to other tasks, such as image recognition and **Natural Language Processing (NLP)**, DL models are used more and more often to make accurate time-series forecasts:
- a. **Recurrent neural network (RNN) models:** RNN is a type of neural network that is designed to process sequential data. Because RNN models "remember" the previous inputs for future predictions, they work well for sequential data, such as speech and time-series data. DeepAR, ESRNN, and AR-Net are some of the RNN-based models for time-series forecasting.

- b. **Multilayer perceptron (MLP) models:** MLP is a type of neural network where there are multiple layers of neurons, where each of the layers learns the data and extracts features. MLP-based models, such as N-BEATS, NHiTs, and TSMixer, typically have a deep stack of fully connected layers. This group of models is proven to work well in practice.
- c. **Transformer-based models:** With the success of transformer-based models in NLP that use a multi-head attention mechanism that allows capturing different relationships and dependencies within the input, transformer-based time-series models and architectures are also actively being developed. **Temporal Fusion Transformer (TFT)** is an example of a transformer-based time-series forecasting model and is useful for multi-horizon and multivariate time-series forecasting.

### How to build a DL model for time-series data

There is no shortage of Python packages that help you build time-series models. Darts, Kats, PyCaret, and PyTorch Forecasting are some of the frequently used Python packages that have easy-to-use implementations of DL models for time-series forecasting. If you'd like to see an example of N-BEATS model applications for this chapter, visit the following GitHub repository:

<https://github.com/yoonhwang/Machine-Learning-and-Generative-AI->

for-  
Marketing/blob/main/ch.4/TimeSer  
iesAnalysis.ipynb

# Summary

In this chapter, we have discussed the importance of time-series analysis and its applications for marketing. From fundamental analyses of how data progresses over time and what stories and insights can be gathered from such analyses to the development of advanced time-series forecasting models, we have touched on a wide range of topics in time-series analysis. We have seen how moving averages and autocorrelations and visualizations of them play a critical role in understanding the big picture of events happening over time. We have also covered how time-series data can be decomposed into trends and seasonalities that uncover the hidden insights of cycles within businesses and different product lines. Lastly, we have experimented with two of the frequently used statistical methods of modeling the time-series data and how to make forecasts for the future that can then be used for more efficient and timely marketing campaigns. Although not discussed in depth in this chapter, we have shared some of the other AI/ML models that are used and are in development. Make sure you check out this chapter's GitHub repository for an example of building a DL model for time-series forecasting. We will also go deeper into DL and Generative AI later, in Part IV of this book.

In the following chapter, we are going to explore language modeling and how you can benefit from it for your next marketing initiatives. We will be discussing how to use and apply some of the NLP techniques using

sentiment analysis as an example and how it can equip marketers to gauge the public perception of the brand or products and monitor and refine the marketing messages for better alignment with customer preferences.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](https://OceanofPDF.com)

# 5

## Enhancing Customer Insight with Sentiment Analysis

In today's digital age, understanding customer sentiment is key for shaping marketing strategies, refining brand messaging, and improving customer experience. Sentiment analysis, a subset of **natural language processing (NLP)**, empowers marketers to sort through massive amounts of unstructured text data, such as customer feedback, social media conversations, and product reviews, to gauge public sentiment. This analytical approach not only helps in monitoring brand reputation but also in tailoring marketing messages according to customer preferences, which enhances the overall customer insight.

This chapter explores the world of sentiment analysis for marketing. Using the power of Python, you will learn how to classify sentiments as positive, negative, or neutral, and identify the nuances embedded within customer feedback. We will also use hands-on examples, based on the “Twitter Airline Sentiment” dataset from Kaggle, to equip you with the skills to perform sentiment analysis, interpret the results, and apply these insights to craft more effective marketing strategies.

Overall, this chapter will give you a comprehensive walkthrough of the fundamentals of sentiment analysis in marketing and then guide you through the practical aspects of data preparation, analysis, and results visualization. By the end of this chapter, you will gain proficiency in:

- Understanding the critical role of sentiment analysis in marketing
- Preprocessing text data to prepare it for analysis
- Applying Python libraries to perform sentiment analysis using traditional natural language processing and elements from generative AI
- Interpreting and visualizing the outcomes to derive actionable marketing insights

## **Introduction to sentiment analysis in marketing**

In the fast-paced world of marketing, staying attuned to customer sentiments is not just beneficial; it's a necessity. Sentiment analysis, or the process of detecting positive, negative, or neutral tones in text data, stands at the forefront of this effort, offering a lens through which marketers can view and understand the emotional undertones of customer interactions. This approach uses NLP, machine learning, and computational linguistics to systematically identify, extract, quantify, and study patterns within text. These patterns can range from the presence of certain keywords and phrases to the structure of sentences and the context in which terms are used.

# The significance of sentiment analysis

The importance of sentiment analysis in marketing cannot be overstated. It acts as a compass, guiding brands through the vast and often turbulent sea of public opinion. By analyzing customer feedback, social media conversations, and product reviews, sentiment analysis helps marketers understand not just what people say but how they feel.

## Further reading on sentiment analysis in marketing



For an overview of approaches to automated textual analysis in marketing, refer to the article “*Uniting the Tribes: Using Text for Marketing Insight*” (<https://journals.sagepub.com/doi/full/10.1177/0022242919873106>).

More specifically, the application of sentiment analysis in marketing opens an array of opportunities:

- **Brand monitoring:** Sentiment analysis enables real-time monitoring of brand perception across various digital platforms. By tracking shifts in sentiment, marketers can anticipate and mitigate potential PR crises. For example, during the United Airlines incident on April 9, 2017, where a passenger was forcibly removed from a flight due to overbooking, sentiment analysis could have helped United detect the rapidly escalating negative sentiment around the viral social media footage and respond more proactively.

- **Campaign analysis:** Understanding emotional responses to marketing campaigns allows for agile strategy adjustments. Sentiment analysis can reveal whether a campaign resonates positively with the target audience or if it misses the mark. For instance, Pepsi's 2017 ad featuring Kendall Jenner received backlash for being perceived as trivializing social justice movements. Early sentiment analysis could have identified negative feedback and allowed for a timely campaign pivot.
- **Product feedback:** Detailed sentiment insights help pinpoint specific aspects of products or services that delight or disappoint customers. This feedback loop is invaluable for continuous improvement and innovation. Consider the case of Apple's launch of the iPhone 6, where sentiment analysis of customer feedback highlighted the "Bendgate" issue, prompting Apple to address the problem quickly.
- **Market research:** Sentiment analysis offers a window into the broader market landscape, providing a competitive edge by uncovering trends, competitor standings, and gaps in the market. For example, Netflix uses sentiment analysis to understand viewer preferences and trends, which aids in content creation and recommendation algorithms.

By harnessing the insights gained through sentiment analysis, marketers can not only monitor brand reputation but also tailor their communications to resonate more deeply with their audience, leading to more effective and impactful marketing strategies.

## **Advancements in AI and sentiment analysis**

The emergence of **large language models (LLMs)** and **Generative AI (GenAI)** has transformed sentiment analysis, offering unprecedented depth and accuracy in understanding text data. LLMs are trained on vast datasets that encompass diverse linguistic patterns, enabling them to understand and generate human-like text. For example, LLMs can grasp context and nuance in ways that were previously unattainable, with state-of-the-art models that can distinguish between sarcasm and genuine dissatisfaction, or recognize the underlying positivity in a complaint that includes a suggestion for improvement.

These advancements are particularly relevant for sentiment analysis, where the difference between a satisfied and dissatisfied customer can often be subtle and context-dependent. However, it is important to note that while LLMs offer these advantages, they also come with certain limitations. They can be computationally expensive and require significant processing power, which might not be accessible to all users or can come at a cost.

Additionally, LLMs are not infallible, and they can sometimes produce biased or inaccurate results based on the data they were trained on.

While a more in-depth discussion of GenAI models and their limitations will come in *Part IV* of the book, this chapter will touch upon the targeted application of GenAI for sentiment analysis.

## **Practical example: Twitter Airline Sentiment dataset**

Our exploration of sentiment analysis will be based on the `Twitter Airline Sentiment` dataset. This collection of tweets directed at various airlines

provides a rich dataset for understanding how sentiment analysis can be applied to real-world marketing challenges.

Here, sentiments are classified as positive, negative, or neutral via human annotation, reflecting a range of customer emotions from satisfaction to frustration.



**Source code and data:**

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/blob/main/ch.5/SentimentAnalysis.ipynb>

**Data source:**

<https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment>

Conveniently, this dataset contains not only tweets and their sentiment classifications but also, in some cases, explanations for tweets with negative sentiment. These will provide useful benchmarks for us to evaluate the approaches we will develop in this chapter.

Before diving into the data, below are some sample tweets, highlighting the sentiment classifications, the topics present, and the nature of the tweet content. As you can see, the first couple of rows of the dataset contain terms that are a result of real-world data entry errors, which will be addressed during the data preprocessing stage.

text	airline_sentiment	negativereason	negativereason_confidence
@JetBlue I'm disappointed my flight was Cancelled Flighted, mostly because I was excited to listen to the song "I'm Blue" while flying on JetBlue.	negative	Cancelled Flight	1.0000
@JetBlue but by Cancelled Flighting my flight and pushing me to the next day I'd lose \$150 hotel which was why I was trying to get a same-day flight.	negative	Cancelled Flight	1.0000
@AmericanAir Okay, I think 1565 has waited long enough for a gate at DFW...	negative	Late Flight	0.4904
@southwestair - kind of early but any idea when dates through the first week of sept will come available?	neutral	NaN	0.0000
@DeltaAssist now at 57 minutes waiting on Silver Elite line for someone to pick up! Help!	negative	longlines	0.4686
@United So what do you offer now that my flight was Cancelled Flighted and I'm stranded away from home and work?	negative	Cancelled Flight	1.0000
@AmericanAir over the last year 50% of my flights have been delayed or Cancelled Flighthed. I'm done with you.	negative	Cancelled Flight	0.7918
@SouthwestAir I've been on hold for over an hour and counting...just need a simple name change <a href="http://t.co/5SICzxt0ez">http://t.co/5SICzxt0ez</a>	negative	Customer Service Issue	0.9256
@SouthwestAir If a travel advisory is posted for DEN, can I then make changes to my ticket? Anything I need to know before I call?	neutral	NaN	0.0000
@JetBlue I am heading to JFK now just on principle alone to deal w my lost & damaged bag. #jetblue #jetbluesucks #fjk #badservice #fail	negative	Lost Luggage	0.8436

Figure 5.1: Sample tweets from the Twitter Airline Sentiment dataset

In the subsequent sections, we'll tackle the data preparation, model building, analysis, and visualization stages, employing both conventional NLP techniques and a more modern approach using LLMs. Initially, we'll apply NLP tools for data cleaning and structuring, laying the groundwork for traditional sentiment analysis.

Recognizing the challenges of class imbalance, we'll demonstrate how GenAI can be used as a tool to augment our dataset with additional examples, as necessary. We will then use the models we've built to derive actionable insights that can help guide marketing campaigns.

# Preparing data for sentiment analysis

Before diving into sentiment analysis, it's crucial to prepare your data effectively. Data preparation is a process that involves cleaning, structuring, and enhancing data to improve analysis outcomes. The goal of these steps is

to ensure that the data is in a form that is directly usable for analysis and to remove any inaccuracies or irregularities.

Let's begin by loading the [Twitter Airline Sentiment](#) dataset:

```
import pandas as pd
df = pd.read_csv('Tweets.csv')
df.head(5)
```

Using `df.columns`, we can see a number of columns, such as `text`, which contains the tweet itself, along with several valuable metadata and sentiment-related fields. The following is a summary of the columns, along with a short description of their meaning:

Column	Description
<code>tweet_id</code>	ID of tweet
<code>airline_sentiment</code>	Class label of tweets (positive, neutral, or negative)
<code>airline_sentiment_confidence</code>	Confidence level in sentiment classification
<code>negative_reason</code>	Reason for negative sentiment
<code>airline</code>	Official name of the airline
<code>airline_sentiment_gold</code>	Gold standard for airline sentiment classification
<code>name</code>	Name of the user
<code>negativerationale_reason_gold</code>	Gold standard for the rationale behind the negative reason

<code>retweet_count</code>	Numerical value representing the number of retweets
<code>text</code>	Text of the tweet as typed by the user
<code>tweet_coord</code>	Latitude and longitude of the Twitter user
<code>tweet_created</code>	Creation date of the tweet
<code>tweet_location</code>	Location from which the tweet was sent
<code>user_timezone</code>	Timezone of the user

*Table 5.2: Columns and their description for the Twitter Airline Sentiment dataset*

## Traditional NLP techniques for data preparation

In traditional NLP techniques, careful text preparation—encompassing cleaning, tokenization, stop word removal, and lemmatization—is important to structure the input text for effective analysis. We’ll discuss these processes in detail in this section.

In contrast, modern techniques such as word embeddings (e.g., Word2Vec and GloVe) and contextual embeddings (e.g., BERT and GPT-4) offer more advanced ways to represent and process text data. These modern techniques will also be explained in greater detail in *Part IV* of the book. Unlike traditional methods that rely on manual feature extraction, modern techniques automatically learn dense representations of words and context from their pre-training on other texts.

For illustration, we will take a sample of five tweet texts and use them as examples to see the impact of traditional data preparation steps. We will

also use the column width setting via `pd.set_option` to show the full column width in our DataFrame, thus displaying the full tweet text:

```
pd.set_option("max_colwidth", None)
examples_idx = df.sample(5).index # [1106, 4860, 6977, 8884, 91
df_sample = df.loc[examples_idx]
```

## Cleaning text data

Text cleaning enhances the quality of data analysis by removing noise and making the format of text uniform. The primary advantages include improved model accuracy and faster computation. However, it's essential to approach cleaning carefully to avoid removing contextually important information. Cleaning is especially useful for the Twitter dataset, due to the informal and diverse nature of social media text. Tweets often contain URLs, mentions, emojis, and hashtags that can sometimes detract from the primary sentiment analysis.

Our approach targets these specifics to preserve the core message while eliminating extraneous elements:

```
!python -m spacy download en_core_web_sm
import re
import spacy
nlp = spacy.load("en_core_web_sm")
def clean_text(text):
    text = re.sub(r'@\w+|\#\w+|https?://\S+', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    return text.lower()
df_sample['cleaned_text'] = df_sample['text'].apply(clean_text)
df_sample[['text", "cleaned_text"]]
```

This yields the following output:

text	cleaned_text
@united noooooooooooooooooooooope.	noooooooooooooooooooooope
@SouthwestAir Thank you SWA and Shannon G. @LASairport (C22) for being a miracle worker! #awesome	thank you swa and shannon g c22 for being a miracle worker
"@JetBlue: Our fleet's on fleek. http://t.co/tEjrN09tfw" NOOOOOOOOOOOOOOOOOOO	our fleets on fleek noooooooooooooo
@JetBlue Great thanks	great thanks
@USAirways We're back at a gate. Opposite of wheels up. Im sure we'll get thete eventually. So thanks.	were back at a gate opposite of wheels up im sure well get thete eventually so thanks

Figure 5.2: Examples of tweet text before and after cleaning

Note that the removal of (@) mentions from the text does little to detract from the context, as our dataset already has the subject of the tweet captured by the `airline` column.

### Additional packages for text preprocessing

Other useful Python tools for NLP and text preprocessing not used in this section include:



- **NLTK** (<https://www.nltk.org/>)
- **TextBlob**  
(<https://textblob.readthedocs.io>)
- **Gensim**  
(<https://radimrehurek.com/gensim/>)

## Tokenization and stop word removal

Tokenization divides text into smaller units, such as words or phrases, making it easier for algorithms to understand language structure. Stop words are commonly used words like “is,” “and,” or “the,” and they are

often removed because they add little semantic value, allowing models to focus on more meaningful content.

In this implementation, we use spaCy's model to apply both tokenization and stop word removal and show the results of each step of the process:

```
def tokenize_and_remove_stopwords(row):
    doc = nlp(row['cleaned_text'])
    all_tokens = [token.text for token in doc]
    tokens_without_stop = [token.text for token in doc if not token.is_stop]
    processed_text = ' '.join(tokens_without_stop)
    row['all_text_tokens'] = all_tokens
    row['without_stop_words_tokens'] = tokens_without_stop
    row['processed_text'] = processed_text
    return row
df_sample = df_sample.apply(tokenize_and_remove_stopwords, axis=1)
df_sample[['cleaned_text', 'all_text_tokens', 'without_stop_words_tokens', 'processed_text']]
```

This gives us the following output:

cleaned_text	all_text_tokens	without_stop_words_tokens	processed_text
nooooooooooooooooooooope	[, nooooooooooooooooooooope]	[, nooooooooooooooooooooope]	[, nooooooooooooooooooooope]
thank you swa and shannon g c22 for being a miracle worker	[, thank, you, swa, and, shannon, g, , c22, for, being, a, miracle, worker]	[, thank, swa, shannon, g, , c22, miracle, worker]	thank swa shannon g c22 miracle worker
our fleets on fleek noooooooooooooooo	[, our, fleets, on, fleek, , noooooooooooooooo]	[, fleets, fleek, , noooooooooooooooo]	fleets fleek
great thanks	[, great, thanks]	[, great, thanks]	great thanks
were back at a gate opposite of wheels up im sure well get thete eventually so thanks	[, were, back, at, a, gate, opposite, of, wheels, up, i, m, sure, well, get, thete, eventually, so, thanks]	[, gate, opposite, wheels, m, sure, thete, eventually, thanks]	gate opposite wheels m sure thete eventually thanks

Figure 5.3: Examples of tweet texts after tokenization and stop word removal

## Lemmatization

Lemmatization reduces words to their dictionary form, ensuring the outcome is a valid word. This process aims to consolidate different forms of a word to analyze them as a single item, enhancing the efficiency and accuracy of the downstream task.

Looking at the last row of the below code, we see that the final lemmatized text standardizes `wheels` to its singular form, `wheel`, and converts `thanks` to its base verb form, `thank`:

```
def lemmatize_text(text):
    doc = nlp(text)
    lemmatized = [token.lemma_ for token in doc]
    return ' '.join(lemmatized)
df_sample['final_text'] = df_sample['processed_text'].apply(lemmatize_text)
df_sample[['processed_text', 'final_text']]
```

This gives us the following output:

processed_text	final_text
nooooooooooooooooooooope	nooooooooooooooooooooope
thank swa shannon g c22 miracle worker	thank swa shannon g c22 miracle worker
fleets fleek noooooooooooooooo	fleet fleek noooooooooooooooo
great thanks	great thank
gate opposite wheels m sure thete eventually thanks	gate opposite wheel m sure thete eventually thank

Figure 5.4: Examples of the tweet text after lemmatization

## Class imbalance

We will now introduce why it is important to understand class balance through exploratory data analysis on our dataset. Class balance directly impacts a model's ability to learn from data effectively. Unaddressed class imbalances can obscure insights and lead to models that do not perform as intended in real-world applications.

In the following sections, we will not only quantify class imbalance but also discuss both simple and more advanced strategies to address it.

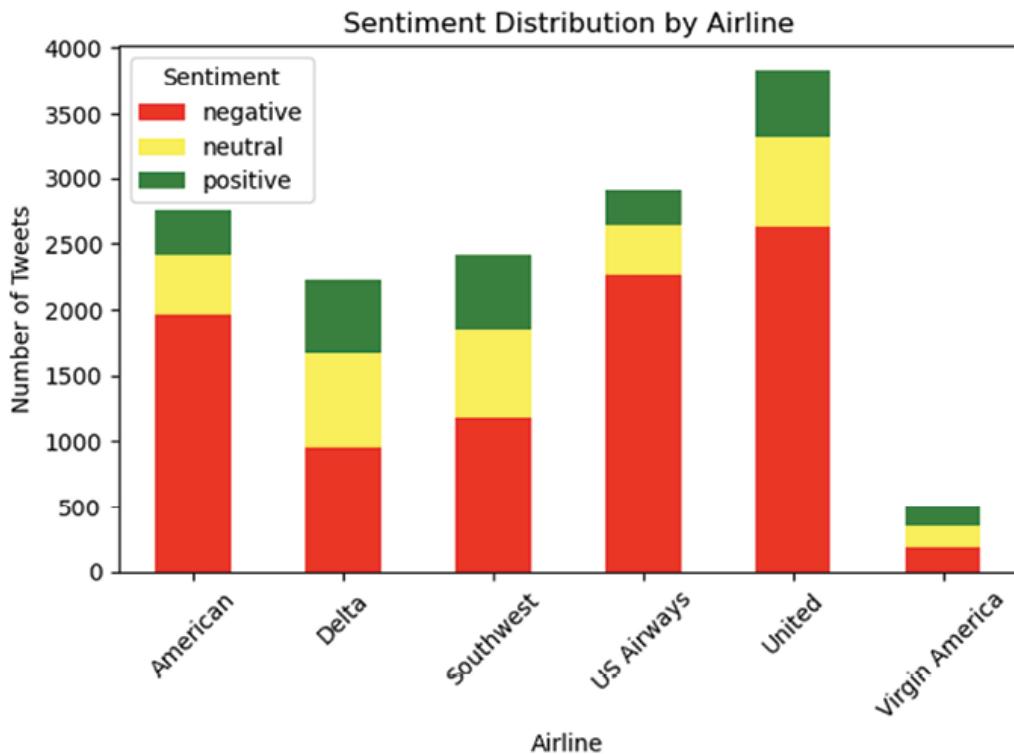
# Evaluating class balance

Identifying the overall sentiment distribution is crucial, as it helps us understand the general mood and opinions expressed in a dataset. This understanding is also essential for understanding any biases that may be present due to class imbalance.

The following code groups tweets by airlines and sentiment, calculates the size of each group, and generates a bar plot to visualize the result:

```
import matplotlib.pyplot as plt
from datetime import datetime
sentiment_by_airline = df.groupby(['airline', 'airline_sentimen
plt.figure(figsize=(14, 6))
sentiment_by_airline.plot(kind='bar', stacked=True, color=['red
plt.title('Sentiment Distribution by Airline')
plt.xlabel('Airline')
plt.ylabel('Number of Tweets')
plt.xticks(rotation=45)
plt.legend(title='Sentiment')
plt.tight_layout()
plt.show()
```

This code generates the following graph, illustrating how the airline tweets captured in this dataset are primarily negative in sentiment:



*Figure 5.5: Distribution of tweets, grouped by airline and sentiment*

If we look at the class balance across the dataset, we can see there are 9,178 negative examples, 3,099 neutral examples, and 2,363 positive tweets. This can be shown via:

```
df['airline_sentiment'].value_counts()
```

Datasets with class imbalance are frequently encountered in real-world datasets, and therefore, we will retain this characteristic of the data throughout the chapter, allowing us to understand the impact that class imbalance can have on modeling results.

## Addressing class imbalance

The following traditional strategies can be employed to address class imbalance:

- **Undersampling** reduces the size of the majority class to match the minority class. This helps balance the dataset but may result in the loss of valuable majority class examples.
- **Oversampling** increases the size of the minority class to match the majority class by duplicating existing examples. This improves balance but can lead to model overfitting on the repeated data examples.
- **Synthetic Minority Over-sampling TEchnique (SMOTE)** is similar to oversampling, except that it generates synthetic examples instead of simply duplicating existing ones. It does this by generating new instances based on examples that are similar in feature space.

As an exercise, you can undersample the majority class (negative sentiment) to achieve a more balanced dataset. This can be achieved with the following code, where we first divide our training data into the negative, neutral, and positive classes and then downsample the negative class to match the number of examples in the minority (positive) class, resulting in a more balanced starting dataset:

```
from sklearn.utils import resample
negative = df[df.airline_sentiment == 'negative']
neutral = df[df.airline_sentiment == 'neutral']
positive = df[df.airline_sentiment == 'positive']
negative_downsampled = resample(negative, n_samples=len(positiv
df_downsampled = pd.concat([negative_downsampled, neutral, posi
```

## Experimenting with undersampling



As an exercise, run the downsampling code given in the text and substitute `df_downsampled` for `df` in the remainder of the chapter to see how your results differ with a more balanced dataset.

## GenAI for data augmentation

For this section, we will demonstrate a strategy to augment the underrepresented positive class by generating new examples, using a seed text via GenAI. This more novel approach complements traditional techniques to address class imbalance but with greater potential diversity in the new examples. However, using AI-generated data can introduce risks, such as overfitting to generated patterns or reflecting potential biases from the generative model itself. Ensuring a variety of seed texts can help mitigate these issues.

We will utilize the `distilgpt2` model from Hugging Face's Transformers library to augment our dataset. This model, a simplified version of GPT-2, is tailored toward resource efficiency, thus making this example accessible to users with varying computational resources:

```
from transformers import pipeline
generator = pipeline('text-generation', model='distilgpt2')
def augment_text(text, augment_times=2):
    augmented_texts = []
    for _ in range(augment_times):
        generated = generator(text, max_length=60, num_return_sequences=1)
        new_text = generated[0]['generated_text'].strip()
        augmented_texts.append(new_text)
    return augmented_texts
seed_text = "Fantastic airline service on this flight. My favorite"
augmented_examples = augment_text(seed_text)
def remove_extra_spaces(text):
```

```
words = text.split()
return ' '.join(words)
for example in augmented_examples:
    print("-----\n", remove_extra_spaces(example))
```

Remember that GenAI's probabilistic nature means that output may vary with each execution. By starting with a carefully chosen seed text of

"Fantastic airline service on this flight. My favorite part of the flight was" we are able to generate varied positive sentiments about airline services. When using this seed text in the `text-generation` pipeline above, we generate outputs such as the following:

- **Fantastic airline service on this flight. My favorite part of the flight was** *enjoying the fantastic view of all the flight attendants and the runway.*
- **Fantastic airline service on this flight. My favorite part of the flight was** *that it was the first time I ever flew in it, and it was worth it.*

By default, most LLMs will produce language that's reflective of what you'd expect to hear in everyday life. For example, try removing the end part of the seed statement "My favorite part of the flight was" and see how much more difficult it is to have the LLM produce examples with positive sentiment.

### Importance of GenAI model selection and prompt sensitivity

Exploring more sophisticated models available from Hugging Face, such as `gpt-neo`, `gpt-j`, and `EleutherAI/gpt-neo-2.7B`, will yield more impressively nuanced and realistic augmentations.



The choice of prompt also plays a crucial role in steering the generative model's output. A subtle change in the seed text can lead to dramatically different results, underscoring the importance of prompt design in GenAI applications.

This topic is explored in detail in *Chapter 9* of this book.

While this augmentation step enhances the representation of positive sentiment tweets within our dataset, achieving true class balance would require more extensive data augmentation, as well as a variety of seed texts, to ensure that the model does not overfit on the beginning part of the tweet. We can increase the number of augmentation examples generated by changing the `augment_times` parameter in the above `augment_text()` function:

```
augmented_data = pd.DataFrame({  
    'text': augmented_examples,  
    'airline_sentiment': ['positive'] * len(augmented_examples)  
})  
df_augmented = pd.concat([df, augmented_data], ignore_index=True)
```

By carefully employing GenAI for data augmentation, we now have `df_augmented` as a dataframe, with additional data that can be added to our existing dataset to mitigate class imbalance and enhance the dataset with varied expressions of positive sentiment. However, in order to illustrate the impact of the class imbalance in the original dataset on our results, we will refrain from adding these examples to our dataset.

# **Performing sentiment analysis**

The power of sentiment analysis lies in its ability to uncover the emotions behind text data, providing invaluable insights into customer sentiments. While the focus of the Twitter Airline Sentiment dataset is on categorizing sentiments into positive, negative, and neutral classes, sentiment analysis can also extend beyond these basic categories. Depending on the application, sentiments can be analyzed to detect more nuanced emotional states or attitudes, such as happiness, anger, surprise, or disappointment.

## **Building your own ML model**

A fundamental aspect of training sentiment analysis models, especially with traditional NLP techniques, is the necessity for pre-labeled data. These labels are typically derived from human annotations, a process that involves individuals assessing the sentiment of a piece of text and categorizing it accordingly. The sentiment scores in this Twitter dataset were collected with the help of volunteers, and some of the negative tweets were also broken down based on specific issues they highlighted, such as flight delays or poor service, providing a more nuanced view of customer dissatisfaction.

Using `scikit-learn`, we will construct sentiment analysis models that leverage this pre-labeled dataset. These models will utilize the text preprocessing demonstrated in the previous section to extract TF-IDF features from the text, which, in turn, serve as inputs for machine learning inferences that can predict the sentiment of unseen tweets.

## **Feature engineering**

Before training a model, we need to convert the text data into numerical features. One common approach is to use the **TF-IDF (Term Frequency-Inverse Document Frequency)** technique. TF-IDF is a statistical measure used to evaluate the importance of a word to a document in a collection or corpus. We will utilize the text preprocessing steps we performed earlier and apply the `tfidf` vectorizer directly to the processed text, limiting the features to the top 1,000 terms (`max_features=1000`). This step is important because reducing dimensionality helps to simplify the model, making it faster to train and reducing the risk of overfitting. By focusing on the most relevant words, we ensure that the model captures the most significant patterns in the data while ignoring less important details:

```
from sklearn.feature_extraction.text import TfidfVectorizer
df['cleaned_text'] = df['text'].apply(clean_text)
df = df.apply(tokenize_and_remove_stopwords, axis=1)
df['processed_text'] = df['cleaned_text'].apply(tokenize_and_re
df['final_text'] = df['processed_text'].apply(lemmatize_text)
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
X = tfidf_vectorizer.fit_transform(df['final_text'])
y = df['airline_sentiment']
```

## Exploring feature engineering with TF-IDF

Another parameter to explore on your own during TF-IDF feature engineering is `ngram_range`.

N-grams allow you to go beyond individual words and consider pairs (bigrams) or triples (trigrams) of consecutive words as single features. This can capture more context and the relationship between words – for example, while “not”



and “good” individually might not be very informative, the bigram “not good” carries a clear sentiment.

## Model training

With our features ready, we can proceed to train a simple model using scikit-learn. Logistic regression is a simple yet powerful algorithm that works well with the dimensionality of the data contained in these short tweets. It models the probabilities for classification problems with two possible outcomes, but it can be extended to handle multiple classes, which is applicable in our case:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

With our logistic regression model trained, we can now interpret the model’s coefficients to obtain insight into how specific words influence sentiment classification. For each sentiment category, the coefficients represent the influence of these terms on the likelihood of a text being classified within that particular sentiment, and the magnitude represents the importance of each term in the model’s decision-making process.

To accomplish this, we iterate over each class label to extract and display the most influential features, sorted by the absolute value of their coefficients:

```
feature_names = tfidf_vectorizer.get_feature_names_out()
class_labels = model.classes_
```

```

for index, class_label in enumerate(class_labels):
    coefficients = model.coef_[index]
    coefficients_df = pd.DataFrame({
        'Feature': feature_names,
        'Coefficient': coefficients
    })
    coefficients_df['Absolute_Coefficient'] = coefficients_df[''
    coefficients_df = coefficients_df.sort_values(by='Absolute_'
    print(f"Class: {class_label}")
    print(coefficients_df[['Feature', 'Coefficient']].head(10))

```

This yields the following output:

Class: negative			Class: neutral			Class: positive		
	Feature	Coefficient		Feature	Coefficient		Feature	Coefficient
880	thank	-3.881871	234	customer	-2.240415	880	thank	4.356471
454	hour	3.609952	325	experience	-1.911483	416	great	3.540018
104	bad	2.964692	361	fix	-1.882879	101	awesome	3.175967
252	delay	2.837971	438	helpful	-1.846837	63	amazing	3.065953
147	cancel	2.633174	454	hour	-1.827311	542	love	2.572124
612	not	2.329509	50	agent	-1.803310	318	excellent	2.405343
444	hold	2.312044	91	attendant	-1.765099	415	good	2.379571
548	luggage	2.097095	416	great	-1.756973	886	thx	2.178217
105	bag	2.042137	55	airline	-1.754047	500	kudo	2.156888
539	lose	2.030467	970	well	-1.689587	428	happy	1.845353

Figure 5.6: Most influential features by sentiment class for the logistic regression model

We can see from the above coefficients the influence of the following words on the three sentiment classes:

- **Negative sentiment:** The word `thank` (-3.88) decreases the likelihood of a tweet being classified as negative, whereas words like `hour` (3.61), `bad` (2.96), `delay` (2.84), and `cancel` (2.63) increase this likelihood and are characteristic of complaints.

- **Neutral sentiment:** Words such as `customer` (-2.24), `experience` (-1.91), and `fix` (-1.88) decrease the likelihood of neutral classification, indicating that these terms are more often used in non-neutral contexts.
- **Positive sentiment:** Terms like `thank` (4.36), `great` (3.54), `awesome` (3.18), and `amazing` (3.07) significantly increase the likelihood of a tweet being classified as positive.

## Model evaluation

Evaluating our model's performance is a critical step after training. Without proper evaluation, we risk deploying a model that may perform poorly in real-world scenarios, potentially leading to incorrect conclusions based on its predictions.

## Classification report

The classification report from scikit-learn gives us the precision, recall, and F1 score for each class. These metrics are crucial, as they tell us not just about the overall accuracy but also how well the model performs for each sentiment class:

```
from sklearn.metrics import accuracy_score, classification_report
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
negative	0.82	0.92	0.87	2814
neutral	0.61	0.49	0.54	884
positive	0.79	0.61	0.69	694
accuracy			0.78	4392
macro avg	0.74	0.67	0.70	4392
weighted avg	0.77	0.78	0.77	4392

Figure 5.7: Classification report metrics evaluating logistic regression model performance

The macro average and weighted average scores give us an understanding of the model's performance across all classes. The macro average treats all classes equally, while the weighted average takes the class imbalance into account. The differences between these scores highlight the impact of the class imbalance on the model's performance.

Let's look more closely at the results for each class:

- The negative sentiment, as the majority class, has high precision (0.82) and recall (0.92), indicating that the model is particularly good at identifying negative tweets. This is an expected side effect of our class imbalance, since the model has more examples of this class to learn from, leading to a higher likelihood of predicting this class correctly.
- The neutral sentiment, being less represented than the negative but more than the positive, shows a significantly lower precision (0.61) and recall (0.49). The precision is reasonably good, meaning that when the model predicts a tweet as neutral, it's correct a little over half the time.

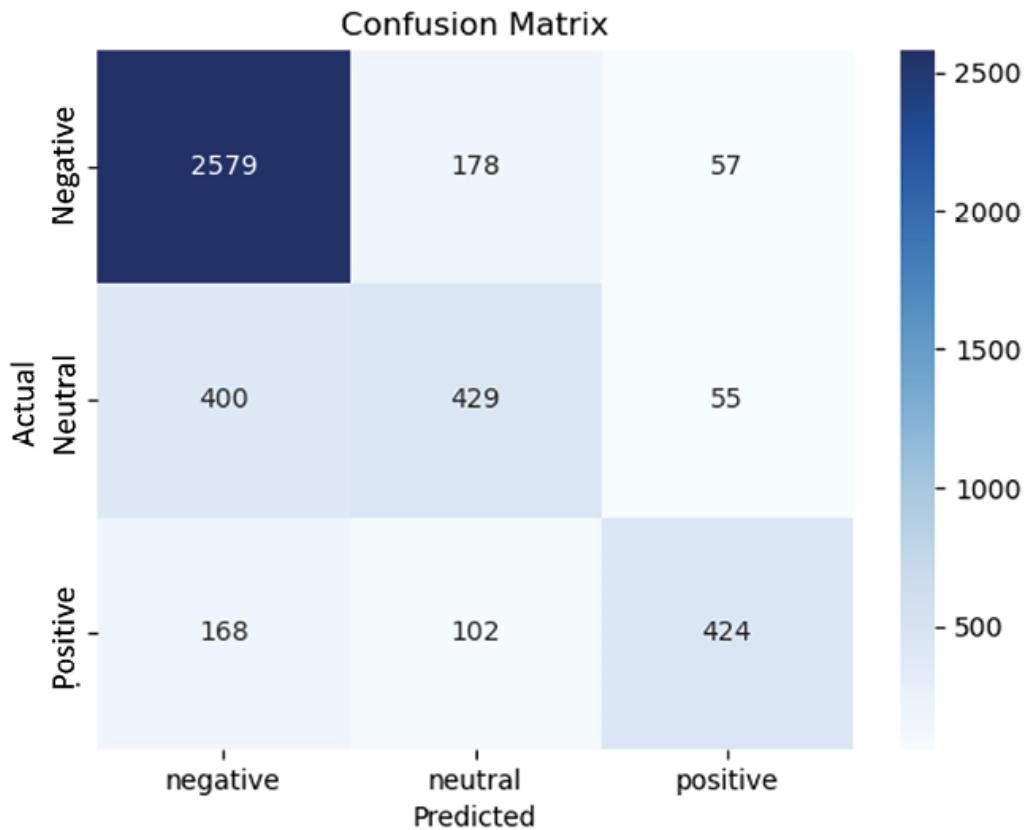
- The positive sentiment, the least represented class, has a relatively high precision (0.79) but lower recall (0.61) than the negative class. High precision here indicates that most tweets predicted as positive are indeed positive, but the model fails to catch many positive sentiments (low recall).

## Confusion matrix

A confusion matrix is an excellent next step to further understanding a model's performance. It shows a matrix with the actual classes on one axis and the predicted classes on the other. By analyzing the confusion matrix, we can see which classes are being confused with one another. For example, if many neutral tweets are misclassified as negative, this might suggest that the model is biased toward predicting negative and that the features for neutral tweets are not distinctive enough. We can calculate and visualize the confusion matrix using the following:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test, y_pred, labels=['negative', 'neutral'])
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[],
            yticklabels=[],
            xlabel('Predicted'),
            ylabel('Actual'))
plt.title('Confusion Matrix')
plt.show()
```

We can then see the following confusion matrix:



*Figure 5.8: Confusion matrix of tweet sentiment classes for the logistic regression model*

Let's break this down to understand it better:

- Consistent with our observations from the classification report, the model shows strong performance in correctly identifying negative tweets, as captured by the first row in the confusion matrix.
- When it comes to neutral sentiments, the second row indicates a tendency of the model to confuse neutral tweets with negative ones.
- Lastly, the third row corresponds to positive sentiments. While the model correctly identifies positive tweets more than half the time, there is still a considerable portion that is confused with negative or neutral sentiments.

# Understanding misclassifications

After assessing our model with various metrics, we can investigate examples when the model fails, giving insight into its limitations and opportunities for improvement.

Let's look at the instances where the model's predictions clash with the trusted classifications provided by `airline_sentiment_gold` data labels:

```
gold_df = df[df['airline_sentiment_gold'].notnull()]
X_gold = tfidf_vectorizer.transform(gold_df['final_text'])
y_gold = gold_df['airline_sentiment_gold']
y_gold_pred = model.predict(X_gold)
gold_df['predicted_sentiment'] = y_gold_pred
misclassified = gold_df[gold_df['airline_sentiment_gold'] != go
misclassified[['airline_sentiment_gold', 'predicted_sentiment',
```

We get the following output:

airline_sentiment_gold	predicted_sentiment	text	final_text
neutral	negative	@southwestair - kind of early but any idea when dates through the first week of sept will come available?	kind early idea date week sept come available
negative	positive	@JetBlue I'm disappointed my flight was Cancelled Flighted, mostly because I was excited to listen to the song "I'm Blue" while flying on JetBlue.	m disappoint flight cancel flight excited listen song m blue fly jetblue
positive	negative	@AmericanAir Hopefully you ll see bad ones as opportunity to get better and not dwell in it... and the good ones as encouragement words!	hopefully ll bad one opportunity well dwell good one encouragement word

*Figure 5.9: Example tweets where the dataset labels disagree with model predictions from the logistic regression model*

An analysis of the misclassified examples demonstrates where our model's `predicted_sentiment` falls short of accurately capturing the context of the full tweet. For instance, the second misclassification example of a tweet expressing disappointment over a canceled flight, cloaked in a humorous tone, highlights the inherent challenge in detecting sarcasm and underscores the importance of model training that encompasses a wider spectrum of

sentiment expressions. Then, for the last example, we have an encouraging tweet, acknowledging the opportunity for improvement from negative experiences, which is predicted to be negative. This illustrates the difficulty of TF-IDF features in a model to interpret nuanced positive sentiments, especially when intertwined with negative words. These mischaracterizations could benefit from more advanced NLP models that are capable of better discerning context and nuance, a topic we will explore in the next section on pre-trained LLMs.

### Enhancing the performance of traditional sentiment models

- **Address training data gaps:** Ensure that your dataset includes a wide range of sentiment expressions such as sarcasm, humor, and conditional positivity. A model's ability to accurately interpret sentiments is directly tied to the diversity of examples it has learned from. Sampling techniques, such as stratified sampling, can be used to ensure that all sentiment types are adequately represented in the training set.
- **Understand feature representation limitations:** Traditional feature representation methods like TF-IDF may not fully capture complex sentiment nuances, especially where overall sentiment is not the sum of its parts.
- **Enhance models with contextual features:** Enrich feature sets by incorporating contextual cues like n-grams or part-of-speech tagging. These additions help



capture sentiment nuances by considering word order and grammatical structure.

- **Explore more robust ML algorithms:** Beyond simpler methods such as logistic regression and naive Bayes, ensemble methods like random forests and boosting algorithms (XGBoost) capture complex patterns better. Additionally, hyperparameter tuning on logistic regression (e.g., adjusting regularization strength) can significantly improve performance. Deep learning methods such as CNNs and RNNs often provide the best performance, provided there are enough training examples present.

Having explored various techniques to enhance traditional sentiment models, including addressing data gaps, feature representation, and appropriate algorithms, we now turn our attention to a more modern advancement in sentiment analysis, using pre-trained LLMs.

## Using pre-trained LLMs

Before applying pre-trained LLMs for sentiment analysis, it is important to understand the concept of embeddings, which serve as the foundation for these advanced models. At their core, embeddings are dense vector representations of data, which could be anything, from words and entire documents to even images and relational data. These vectors are designed to capture the key features of data in a high-dimensional space.

Early examples of NLP embeddings include Word2Vec and GloVe, which generate **static embeddings**. In Word2Vec, the embeddings are influenced

by local context through techniques like skip-grams and **continuous bag of words (CBOW)**, but once trained, the same word has the same vector representation regardless of the broader context. However, state-of-the-art LLMs like BERT and GPT introduced true **contextual embeddings**, where the representation of a word dynamically changes based on its contextual usage. The key attribute of an effective NLP embedding is that it preserves the original data's semantic relationships in its vector space, meaning that similar vectors (words, phrases, or documents) are closer together than less similar data.

Incorporating LLMs for sentiment analysis marks a significant advancement in the field of NLP, streamlining the process of understanding complex textual data. These models excel in capturing the subtleties of human language through extensive pre-training on diverse datasets, thus bypassing the need for elaborate text preprocessing, hyperparameter tuning – or even the need for pre-labeled data. For marketing professionals, this translates into a more efficient way to gauge customer sentiment across different platforms.

## Implementing pre-trained models

To demonstrate the efficacy of a pre-trained LLM, the `sentiment-analysis` pipeline from the `distilbert-base-uncased-finetuned-sst-2-english` model in the Transformers library will be used. DistilBERT is a smaller, faster version of BERT that retains 95% of its contextual embedding performance. This particular variant has been fine-tuned on the **Stanford Sentiment Treebank (SST-2)** dataset, a standard benchmark for sentiment analysis, consisting of movie reviews with human-annotated `positive` or `negative` sentiments.

Given its binary classification nature, `neutral` sentiments are excluded from our test set to align the model's available predictions between `positive` and `negative`. For this example, we will also incorporate the `time` and `tqdm` modules into our code to track the execution time. After the model loads, inference on all of the test texts may take a few minutes to complete using the `sentiment-analysis` pipeline:

```
from tqdm.auto import tqdm
import time
filtered_df = df[df['airline_sentiment'] != 'neutral']
X = filtered_df['text']
y = filtered_df['airline_sentiment']
X_train_texts, X_test_texts, y_train, y_test = train_test_split
sentiment_pipeline = pipeline("sentiment-analysis", model="distilbert-base-uncased")
start_time = time.time()
results = []
for text in tqdm(X_test_texts, desc="Analyzing sentiments"):
    result = sentiment_pipeline(text)
    results.append(result[0]['label'].lower())
end_time = time.time()
total_time = end_time - start_time
print(f"Total time for analyzing {len(X_test_texts)} tweets: {total_time} seconds")
```

Utilizing the `'sentiment-analysis'` pipeline for DistilBERT simplifies the sentiment analysis process by encapsulating several complex steps into one efficient process. Without this, the tokenization, text embeddings, inference, and post-processing would all need to be handled separately.

## Evaluating model performance

The classification report from the LLM inference can be obtained using the following code:

```
print(classification_report(y_test, results))
```

This yields the following results:

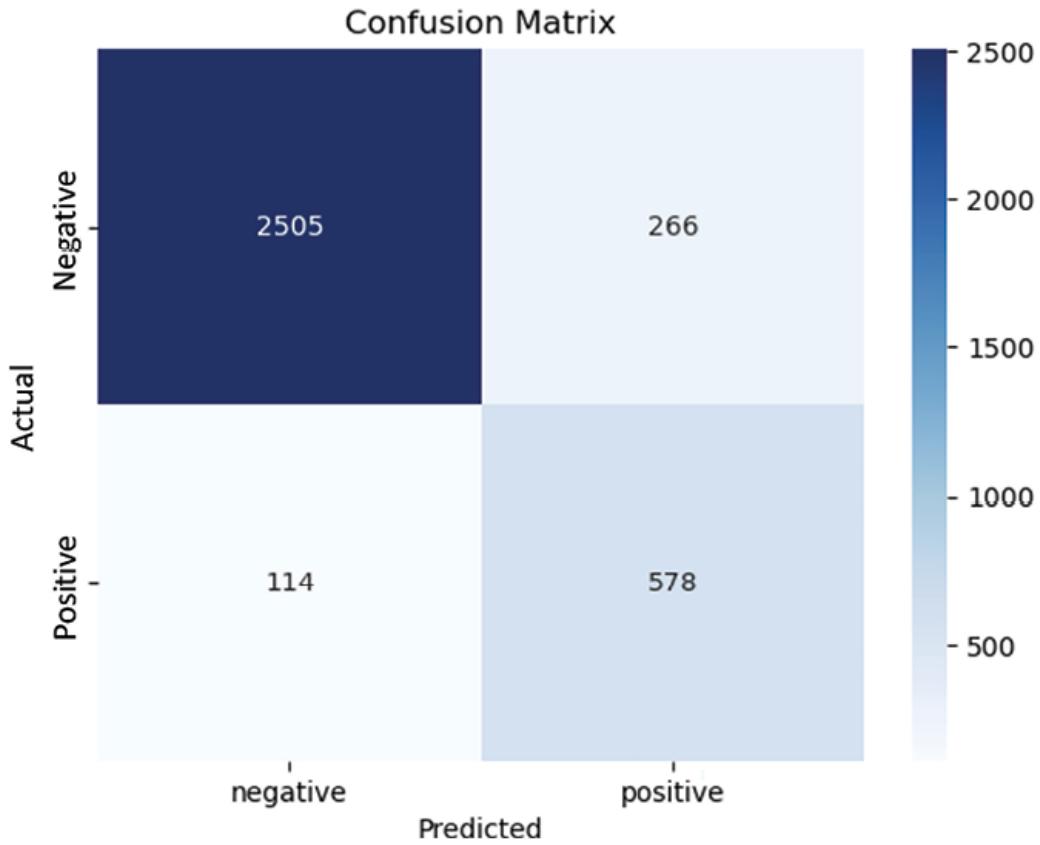
	precision	recall	f1-score	support
negative	0.96	0.90	0.93	2771
positive	0.68	0.84	0.75	692
accuracy			0.89	3463
macro avg	0.82	0.87	0.84	3463
weighted avg	0.90	0.89	0.89	3463

Figure 5.10: Classification report metrics evaluating LLM performance

Next, let's generate the confusion matrix:

```
cm = confusion_matrix(y_test, results, labels=['negative', 'pos
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```

This gives us the following output:



*Figure 5.11: Confusion matrix of tweet sentiment classes for the LLM*

When comparing the performances of the previous logistic regression model utilizing TF-IDF features and our pre-trained LLM, several observations emerge. The logistic regression model, while providing a strong baseline for sentiment analysis, has limitations in handling the nuances of natural language. While class imbalance likely played some role in the model's performance issues, it is important to note that the LLM's capabilities to understand language nuances, derived from its extensive pre-training and fine-tuning, likely played a larger role in its superior performance. Additionally, the logistic regression model only used unigrams for TF-IDF, which fails to capture context and contributes to mischaracterizations. For these reasons, it particularly struggles at

classifying positive (and neutral) sentiments amid a dominant negative sentiment class.

The LLM – keeping in mind we excluded the neutral class from its classification task to stay consistent with the nature of its fine-tuning procedure – demonstrates a more robust performance.

This is showcased by its heightened accuracy in distinguishing between positive and negative sentiments. It is important to note that the logistic regression model started with imbalanced data to illustrate its impact on results, whereas the LLM did not undergo any task-specific training on the Twitter data and was trained only on the SST-2 movie reviews. The key takeaway here is that the LLM's knowledge is generalized effectively, from movie reviews to Twitter data, highlighting its robust language understanding capabilities. With a precision of `0.96` for negative sentiments and a recall rate of `0.90`, the model underscores the potential of leveraging pre-trained neural networks for sentiment analysis. The improvement in positive sentiment detection, achieving a precision of `0.68` and a recall of `0.84`, further emphasizes the model's predictive power.

One drawback of using advanced machine-learning models like LLMs is their lack of explainability. Understanding what exactly creates negative sentiment can be challenging with these black-box models. Techniques such as **LIME (local interpretable model-agnostic explanations)** can be used to improve explainability. For a more detailed discussion on model transparency and techniques to elucidate LLM decisions, please refer to *Chapter 13*.

While the LLM performance is noteworthy, fine-tuning the LLM on the specific sentiment labels present in our airline tweets dataset would further improve its performance. Fine-tuning adapts a model more closely to a

task's unique context, allowing the model to leverage and understand the nuances of the classification task.

In addition to fine-tuning, transfer learning and few-shot learning are powerful methodologies that further refine a model's ability to classify sentiments with high accuracy. Transfer learning involves adapting a pre-trained model on a related task to perform well on the target task, even with minimal additional training data. Conversely, few-shot learning trains the model to make accurate predictions with only a few examples of the target task available.



### Improving LLMs: transfer and few-shot learning

In *Part 4* of this book, we will explore these cutting-edge methodologies in more depth. We will explore how fine-tuning, transfer learning, and few-shot learning can be applied to pre-trained models, transforming them into highly specialized tools for domain-specific tasks like sentiment classification.

When we arrive at Part 4 of this book, we will delve deeper into related methodologies at the cutting edge of adapting pre-trained models for domain-specific tasks.

## Translating sentiment into actionable insights

So far in this chapter, we have explored the tools and strategies needed to understand and apply sentiment analysis to your data, from the foundational

techniques of data preparation and prediction using traditional NLP methods to the advanced capabilities of GenAI. In this final part of the chapter, we will discuss how these insights can be analyzed to generate actionable strategies that can guide a brand to success across all stages of a marketing campaign.

## Creating your own dataset

Before applying this analysis to your use case, we need an approach to collecting the data that captures the underlying customer sentiment related to your brand. While this chapter utilizes the Twitter Airline dataset as an example, the techniques we've explored are applicable regardless of the industry or data source. This section will present the general steps you can take to curate your own proprietary dataset for analysis, whether it be from Twitter or another major data platform.

### Ethics and governance in AI-enabled marketing



Ethics and governance in AI-enabled marketing is the topic of *Chapter 13*, and it is crucial to adhere to ethical guidelines and governance frameworks that respect consumer privacy and data protection laws when collecting data. Ethical practices in marketing include obtaining consent for data collection, ensuring data anonymization to protect individual identities, and providing clear opt-out mechanisms. Companies should establish robust data governance policies that define data handling procedures and compliance with legal standards such as the EU's

**General Data Protection Regulation (GDPR) or the California Consumer Privacy Act (CCPA).**

## Collecting twitter data

To start, ensure you have a Twitter Developer account and access to the Twitter (now rebranded as X) API. You can follow these steps:

1. You'll first need to create a project and obtain your API keys and tokens, and then, using the credentials from your Twitter Developer account (<https://developer.twitter.com/en/portal/petition/essential/basic-info>), authenticate your session to access the Twitter API. The following are general instructions to do this:

```
!pip install tweepy
import tweepy
# Replace these with your API keys and tokens
consumer_key = 'YOUR_CONSUMER_KEY'
consumer_secret = 'YOUR_CONSUMER_SECRET'
access_token = 'YOUR_ACCESS_TOKEN'
access_token_secret = 'YOUR_ACCESS_TOKEN_SECRET'
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
```

2. Now that we have access to the Twitter API, you can use the Twitter handle or relevant hashtags associated with your brand and combine this with the `search_tweets` method to find relevant tweets. This example collects the latest 100 tweets mentioning

```
"@YourBrandHandle":
```

```
query = "@YourBrandHandle -filter:retweets"
tweets = api.search_tweets(q=query, lang="en", count=100)
```



Due to recent changes, the Twitter (X) API may have limited access, and certain endpoints may require elevated access levels. If you encounter a 403 Forbidden error, you may need to upgrade your access level or use alternative endpoints available in the API documentation. More details can be found on the Twitter developer portal (<https://developer.twitter.com/en/portal/petition/essential/basic-info>).

- With the tweets collected, we can extract relevant information such as the tweet ID, text, creation time, and location. We can then structure this data into a pandas DataFrame for easier analysis:

```
data = [
    'tweet_id': tweet.id,
    'text': tweet.text,
    'tweet_created': tweet.created_at,
    'tweet_location': tweet.user.location,
] for tweet in tweets]
your_brand_df = pd.DataFrame(data)
```

There are many further metadata fields available in the API response documentation that are worth considering, including retweet count (tweet.retweet\_count), hashtags (tweet.entities['hashtags']), and mentions (tweet.entities['user\_mentions']). As discussed in the sections below, these fields can provide valuable insight into topics that are

pivotal for understanding your brand's sentiment narrative, including salient topics, tweet engagement, and virality.

## Collecting data from other platforms

Analyzing brand sentiment extends beyond Twitter, encompassing a variety of social media platforms including Facebook, Instagram, Google reviews, and more. Each platform presents unique challenges and opportunities to gather and analyze data. The approach to collecting data differs, based on each platform's API capabilities and data availability. For platforms like Google reviews, APIs may allow you to directly access reviews and ratings. On platforms like Facebook and Instagram, you might rely on posts, comments, and hashtags to gauge sentiment.

### Accessing developer APIs

To collect data from different social media platforms, you need to access their respective developer APIs. Here are some useful links to get you started:

- **Facebook:**  
<https://developers.facebook.com/docs/graph-api>
- **Instagram:**  
<https://developers.facebook.com/docs/instagram-api>
- **Reddit:** <https://www.reddit.com/dev/api/>
- **YouTube:**  
<https://developers.google.com/youtube>



/v3

- **TikTok:**<https://developers.tiktok.com/products/research-api/>

Once the data is obtained, a primary challenge can be accurately identifying mentions and discussions related to your brand. This is where **named entity recognition (NER)** and entity mapping techniques come into play. NER can help identify proper nouns, like brand names or products, within texts, while entity mapping can link these mentions to your brand across datasets.

## Performing NER on a dataset for a fictional retailer

For example, let's consider a series of customer reviews from a fictional online retailer, Optimal Hiking Gear, which sells outdoor equipment. To extract mentions of the brand, we can use the built-in capabilities for NER in the spaCy language model and look for its `ORG` tag to identify relevant mentions:

```
nlp = spacy.load("en_core_web_sm")
reviews = [
    "I recently purchased a sleeping bag from Optimal Hiking Ge
    "The tent I bought from Optimal Hiking was damaged on arriv
    "The Optimal Hiking company makes a backpack that's the bes
]
for review in reviews:
    doc = nlp(review)
    for ent in doc.ents:
        print(f"Entity: {ent.text}, Label: {ent.label_}")
```

This yields the following result:

```
Entity: Optimal Hiking Gear, Label: ORG  
Entity: Optimal Hiking, Label: ORG  
Entity: Optimal Hiking, Label: ORG  
Entity: years, Label: DATE
```



### Enhancing NER with custom training

To tailor NER models more closely to your needs, consider training them with your data. This involves providing examples of texts with manually labeled entities that are specific to your brand or industry. By doing so, a model learns to recognize and categorize these custom entities more accurately. Tools like spaCy offer functionalities to train your NER models.

## Understanding topics and themes

This section delves into various tools to extract insights from your dataset to provide an overview of the key topics and themes present. Such insights are crucial for grasping the broader context of customer sentiment within your data. As a starting point, we can use, for reference, the number of different reasons for negative sentiment that were tagged by the annotators of this dataset:

```
df.negativereason.value_counts()
```

This gives us the following output:

```

negativereson
Customer Service Issue      2910
Late Flight                  1665
Can't Tell                  1190
Cancelled Flight             847
Lost Luggage                 724
Bad Flight                   580
Flight Booking Problems      529
Flight Attendant Complaints 481
longlines                     178
Damaged Luggage              74
Name: count, dtype: int64

```

*Figure 5.12: Counts of different negative sentiment tweets' reasons given in the Twitter dataset*

## Using word clouds

**Word clouds**, where the size of each word in a plot corresponds to its frequency of occurrence, are a valuable starting tool to visualize text data. In order to enrich our word clouds and ensure that visualizations reflect not only the most frequent terms but also those that are most indicative of unique sentiments and topics, we will incorporate TF-IDF analysis to produce our plot. By introducing the `ngram_range=(1, 2)` argument into our `tfidf_vectorizer`, we create both unigrams and bigrams. Leveraging the `wordcloud` library, we can create word clouds via:

```

from wordcloud import WordCloud
tfidf_vectorizer = TfidfVectorizer(max_features=1000, ngram_ran
tfidf_matrix = tfidf_vectorizer.fit_transform(df['final_text'])
tfidf_scores = dict(zip(tfidf_vectorizer.get_feature_names_out(
wordcloud_tfidf = WordCloud(width=800, height=400, background_c
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud_tfidf, interpolation='bilinear')

```

```
plt.axis('off')  
plt.show()
```

We'll then see something similar to the following word cloud:



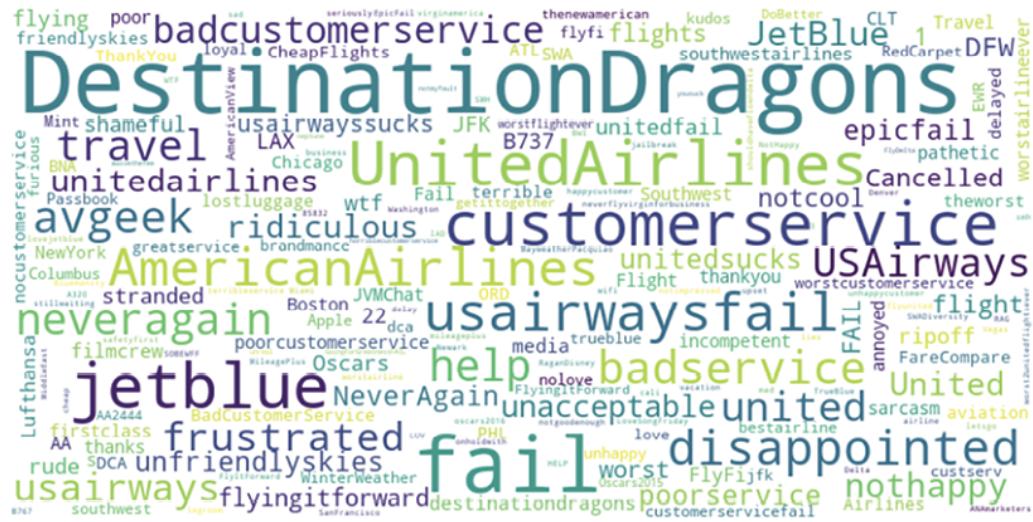
*Figure 5.13: Word clouds showing the frequency of occurrence of different terms from TF-IDF analysis*

The analysis so far reveals pivotal themes through prevalent words such as `delay`, `cancel`, and `help`. Through the inclusion of bigrams, key terms such as `customer service` and `cancel flight` can also be captured. In the context of Twitter data, hashtags offer a direct glimpse into the core topics and sentiments expressed. To dive deeper, we will proceed to extract hashtags from the tweets, creating a word cloud to visualize these key phrases differently. This approach aims to provide a different lens to view the themes within our dataset:

```
import nltk
def extract_hashtags(text):
    return re.findall(r"#(\w+)", text)
hashtags = sum(df['text'].apply(extract_hashtags).tolist(), [])
hashtag_freq_dist = nltk.FreqDist(hashtags)
```

```
wordcloud_hashtags = WordCloud(width=800, height=400, background_color='white')
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud_hashtags, interpolation='bilinear')
plt.axis('off')
plt.show()
```

We then see the following:



*Figure 5.14: Word clouds showing the frequency of occurrence of Twitter hashtags*

Excluding direct airline mentions, the word cloud distinctly highlights prevalent negative sentiments through hashtags like #Fail, #Disappointed, and #BadCustomerService. In contrast to these, #DestinationDragons emerges prominently as well, signifying a well-publicized tour by Southwest Airlines, showcasing the duality of customer feedback captured in our dataset.

# Discovering latent topics with LDA

**Latent Dirichlet allocation (LDA)** is an advanced technique that goes beyond simple frequency metrics, like those seen in word clouds, by

identifying the underlying topics within a text corpus through unsupervised machine learning. Unlike word clouds, which only highlight the most frequent words, LDA discerns the hidden thematic structure by considering documents as mixtures of topics, where each topic is defined by a particular set of words. This process involves Bayesian inference, using the Dirichlet distribution to estimate not just the presence but also the proportion of topics within documents. For example, in a collection of hotel reviews, LDA might identify topics related to location, parking, bathroom cleanliness, and check-in experience. The sophistication of LDA lies in its ability to capture the context and co-occurrence of words across the corpus, providing a more nuanced understanding of the text's thematic content without prior labeling.

We can implement this approach via:

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(max_df=0.95, min_df=2, stop_words=
doc_term_matrix = count_vect.fit_transform(df['final_text'])
LDA = LatentDirichletAllocation(n_components=5, random_state=42
LDA.fit(doc_term_matrix)
for i, topic in enumerate(LDA.components_):
    print(f"Top words for topic #{i}:")
    print([count_vect.get_feature_names_out()[index] for index
    print("\n")
```

This gives us the following output:

```
Top words for topic #0:
['fly', 'follow', 'send', 'dm', 'flight', 'hour', 'sit', 'gate',
Top words for topic #1:
['tomorrow', 'change', 'amp', 'time', 'late', 'delay', 'fly', 'f
Top words for topic #2:
```

```
['response', 'bad', 'good', 'time', 'flight', 'bag', 'great', 'c  
Top words for topic #3:  
['wait', 'need', 'help', 'problem', 'delay', 'luggage', 'hour',  
Top words for topic #4:  
['service', 'number', 'customer', 'minute', 'email', 'hour', 'he
```

Analyzing the clusters generated by LDA allows us to pinpoint important themes in the tweets, such as flight delays and customer service issues, represented in topics #1 and #2, respectively. By changing the `n_components` parameter, we can alter the algorithm's assumption around the number of topics present, leading to more granular topics as the parameter increases in value.

It's important to recognize that not all topics identified by LDA will directly align with key sentiment expressions, and some may simply reflect common, non-specific language pervasive throughout the dataset, as seen in topic #0.

To extract meaningful insights, additional scrutiny is often required to discern which topics are most relevant for analysis. This process can be facilitated either through human review or by employing a carefully designed prompt for a language model, helping to summarize the word groupings according to their possible topics.

## Temporal trends: tracking the brand narrative

In the dynamic realm of social media, the viral nature of tweets often acts as a barometer for the sentiments that are most impactful to a brand's perception. This is especially true for negative sentiments, which carry a

substantial risk to a brand's image. By scrutinizing the Twitter activity around airlines, such as JetBlue, we can unearth insights into tweets that may significantly influence brand reputation.



This paper presents a social media-based brand reputation tracker that monitors brand events in real time and connects them to specific drivers of brand reputation:  
[https://ora.ox.ac.uk/objects/uuid:00e9fc\\_b7-9bf1-486a-b4dd-3c1d086af24e/files/rz316q188f](https://ora.ox.ac.uk/objects/uuid:00e9fc_b7-9bf1-486a-b4dd-3c1d086af24e/files/rz316q188f).

This analysis involves sifting through tweets mentioning the `@JetBlue` handle. The code below categorizes the sentiment of these tweets over time, where each data point's bubble size corresponds to that day's aggregate retweets, providing a visual scale of engagement:

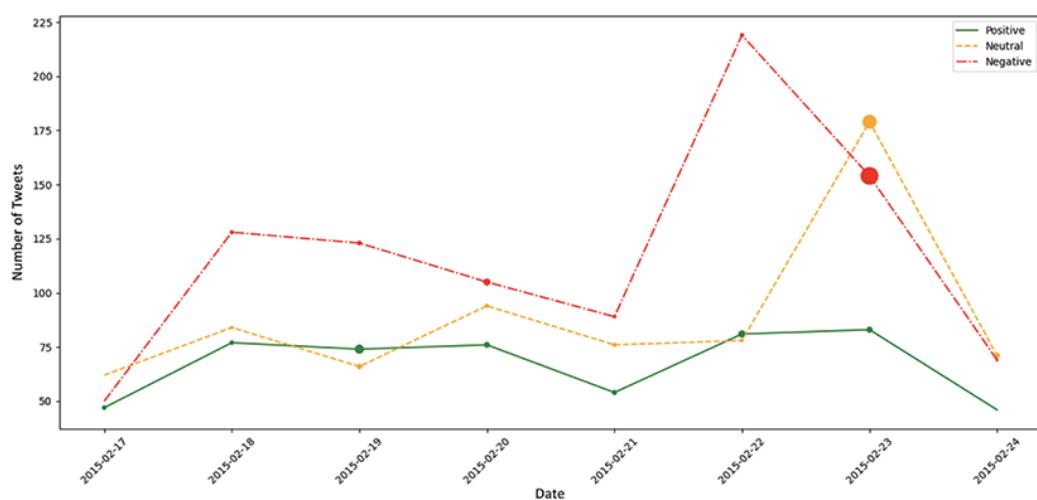
```
df['tweet_created'] = pd.to_datetime(df['tweet_created']).dt.tz  
df['date'] = df['tweet_created'].dt.date  
airline_handle = "@JetBlue"  
airline_tweets = df[df.text.str.contains(airline_handle)]  
grouped = airline_tweets.groupby(['airline_sentiment', 'date'])  
positive_tweets = grouped[grouped['airline_sentiment'] == 'posi  
neutral_tweets = grouped[grouped['airline_sentiment'] == 'neutr  
negative_tweets = grouped[grouped['airline_sentiment'] == 'nega  
plt.figure(figsize=(14, 7))  
scale_factor = 3  
for tweets, sentiment, color, linestyle in zip(  
    [positive_tweets, neutral_tweets, negative_tweets],  
    ['Positive', 'Neutral', 'Negative'],  
    ['green', 'orange', 'red'],  
    [':', '--', '-.'])  
:  
    scaled_retweet_count = tweets['retweet_count'] * scale_fact  
    plt.plot(tweets['date'], tweets['tweet_id'], linestyle=linest
```

```

plt.scatter(tweets['date'], tweets['tweet_id'], scaled_retw
plt.title(f'Daily Sentiment Trend for {airline_handle} with Bub
plt.xlabel('Date')
plt.ylabel('Number of Tweets')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```

This yields the following graph:



*Figure 5.15: Sentiment of JetBlue tweets over time, where each data point's bubble size corresponds to that day's aggregate retweets*

Significant peaks in negative sentiment tweets are observed on February 22 and 23, with an unusual surge in retweets on the latter day indicating widespread engagement. Those focusing on brand reputation for JetBlue may find it valuable to delve into the specifics of these tweets.

We can aggregate and rank the tweets in this date range, as well as the day before, according to those with the highest retweets, using the following code:

```

dates_of_interest = [pd.to_datetime('2015-02-22').date(), pd.to_datetime('2015-02-24').date()]
filtered_df = airline_tweets[(airline_tweets['date'].isin(dates_of_interest))]
top_tweets_per_date = filtered_df.groupby('date').apply(lambda df: df[['text', 'retweet_count', 'negativereason']])

```

This gives us the following result:

			text	retweet_count	negativereason
		date			
2015-02-22	7436	2015-02-22	@JetBlue Good perspective. If only this safety concern had been expressed at some point before I arrived at the airport. #communicationiskey	1	Can't Tell
	7451		@JetBlue hey awesome peeps, what's up with flight 1159 from Boston? Delayed 3hrs?	1	Late Flight
	7508		@JetBlue 5hrs on Tarmac yest b4 Cancelled Flight @12am.Today 3+ hr delay. Why not send txt/email re: JFK closed & saved all trip 2 sit @airport again?	1	Late Flight
2015-02-23	7132	2015-02-23	STOP. USING.THIS.WORD. IF. YOU'RE. A. COMPANY. RT @JetBlue: Our fleet's on fleek. <a href="http://t.co/Fd2TNyCtRb">http://t.co/Fd2TNyCtRb</a>	31	Can't Tell
	7115		can you not? RT @JetBlue Our fleet's on fleek. <a href="http://t.co/413GiAL0yl">http://t.co/413GiAL0yl</a>	22	Can't Tell
	7027		Just in case you needed confirmation that "on fleek" is dead & gone. RT @JetBlue: Our fleet's on fleek. <a href="http://t.co/G4O6yX7TMJ">http://t.co/G4O6yX7TMJ</a>	18	Can't Tell
2015-02-24	6887	2015-02-24	@JetBlue absolutely no worries. I'm just never flying with you again. Ever. Even I have to hitch hike cross country #passengersarepeople	1	Can't Tell
	6749		@JetBlue They weren't on any flight, they just came Late Flight. Your JetBlue employee just informed us!	0	Flight Attendant Complaints
	6750		@JetBlue everyone is here but our pilots are no where to be found and my last flight the plane was dirty that I had to clean my area & seat!	0	Bad Flight

Figure 5.16: Analysis of peak negative sentiment tweets for JetBlue on February 22–24, 2015

The first and last day of the result reveal typical grievances, like late flights and poor service, that are common in the airline industry. However, the backlash on February 23 highlights a controversy around a marketing campaign by JetBlue, encapsulated by the phrase “Our fleet’s on fleek.” As discussed in the introduction to this chapter, real-time monitoring of brand perception would have enabled them to track this shift in campaign sentiment early on, enabling them to anticipate and potentially mitigate the PR issues that ensued.

The potential long-term repercussions of this negative publicity would necessitate further monitoring and analysis – using metrics such as the KPIs discussed in *Chapter 2* – to assess the impact on brand reputation.

Nonetheless, this analysis illustrates the importance of monitoring social

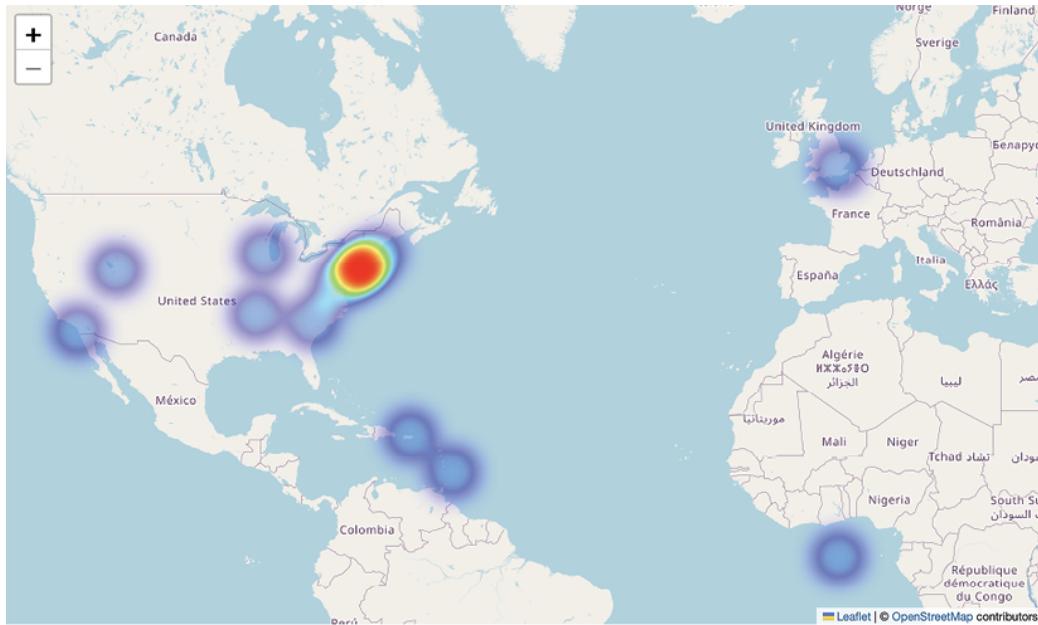
media trends over time to preemptively address issues that could adversely affect brand perception.

## Mapping sentiments using geospatial analysis

Geospatial analysis offers a powerful lens through which to view customer sentiment, enabling companies to identify areas of concern – from customer service issues to poorly designed marketing campaigns — potentially even before such issues become apparent to on-site staff. To illustrate how to generate such insights, let's utilize the `folium` package to create a heat map that pinpoints the origins of negative sentiments, based on tweet coordinates:

```
!pip install folium
import folium
from folium.plugins import HeatMap
filtered_df = df[(df['text'].str.contains('@JetBlue') & (df['ai
filtered_df = filtered_df.dropna(subset=['tweet_coord']))
valid_coords = []
for coord in filtered_df['tweet_coord']:
    try:
        lat, long = eval(coord)
        valid_coords.append((lat, long))
    except (TypeError, SyntaxError, NameError):
        continue
if valid_coords:
    map_center = [sum(x)/len(valid_coords) for x in zip(*valid_
else:
    map_center = [0, 0]
tweet_map = folium.Map(location=map_center, zoom_start=4)
HeatMap(valid_coords).add_to(tweet_map)
tweet_map
```

This code yields the following map:



*Figure 5.17: Heat map that pinpoints the origins of negative JetBlue sentiments, based on tweet coordinates*

The resulting heat map reveals concentrations of negative sentiment in regions where JetBlue predominantly operates – a correlation that is expected, given that a greater volume of flights naturally leads to more reports of negative experiences. However, by looking at the time evolution of these patterns, we can see that this method can serve as a real-time tool to spot unusual patterns of sentiment.

For instance, a sudden influx of negative tweets from a new location could signal service issues and foreshadow potential PR challenges to follow. Conversely, identifying areas with a high number of positive tweets might highlight strengths within a company. Coupling geospatial analysis with the topic modeling approaches introduced earlier can also unlock further insights, revealing not only what is discussed but also where these

conversations take place, providing a valuable trove of actionable marketing intelligence.

# Summary

This chapter underscored the importance of sentiment analysis in modern marketing strategies. It introduced sentiment analysis as a key tool to interpret vast quantities of unstructured text data, such as social media conversations, to refine marketing strategies, brand messaging, or customer experience. By utilizing the Twitter Airline dataset, we covered the end-to-end process needed to classify sentiment as positive or negative, using both traditional NLP and more advanced GenAI methods involving pre-trained LLMs. We then covered an array of tools for the visualization and interpretation of these results to derive actionable marketing insights. This chapter should leave you equipped with the necessary skills to harness sentiment analysis effectively, for applications ranging from brand reputation monitoring to aligning marketing messages with customer preferences.

Looking ahead to the next chapter, we will progress from understanding customer sentiment to actively shaping customer engagement using predictive analytics, with a focus on the empirical validation of marketing strategies through A/B testing. We will discuss identifying features to predict customer engagement, training machine learning models, model evaluation, and the implementation of A/B testing. The chapter is designed to advance your knowledge by providing skills in feature selection, building predictive models, optimizing model performance, conducting A/B tests, and integrating insights into effective marketing strategies. This next step will equip you with the ability to not only forecast customer behaviors but

also empirically validate and refine marketing strategies for heightened effectiveness.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](https://OceanofPDF.com)

# 6

## Leveraging Predictive Analytics and A/B Testing for Customer Engagement

There are various ways that we can benefit from data-driven and AI/ML-driven marketing techniques. To name a few, you can optimize your marketing strategy based on the key drivers behind the success and failures of your previous marketing campaigns, as we discussed in *Chapters 2 and 3*. You can also optimize your marketing strategy based on the trend and seasonality within your business or based on the customer sentiments around your products and business, as we have discussed in *Chapters 4 and 5*. Targeted product recommendation (*Chapter 7*) and optimizing the marketing content with Generative AI (*Chapters 9 and 10*) are some other key benefits of applying AI/ML in marketing.

Among those mentioned, we are going to experiment with predictive analytics in this chapter and how you can utilize these predictive models in your next marketing campaign. By intelligently predicting the expected behaviors of customers, you can target subgroups of the customer base that are likely to result in your favor. This way, instead of mass marketing to the entire potential customer base, you can better custom-tailor your marketing messages and also save on marketing costs as you are only targeting the

group with the higher chance of success. We will also discuss how A/B testing can help decide the best predictive model for your next marketing effort.

In this chapter, we will cover the following topics:

- Predicting customer conversion with tree-based algorithms
- Predicting customer conversion with deep learning algorithms
- Conducting A/B testing for optimal model choice

# Predicting customer conversion with tree-based algorithms

Predictive analytics or modeling can be applied at various stages of the customer life cycle. If you recall from *Chapter 2*, there are largely five stages that we can break down a customer life cycle into: **Awareness**, **Engagement**, **Conversion**, **Retention**, and **Loyalty**, as shown in the following diagram:

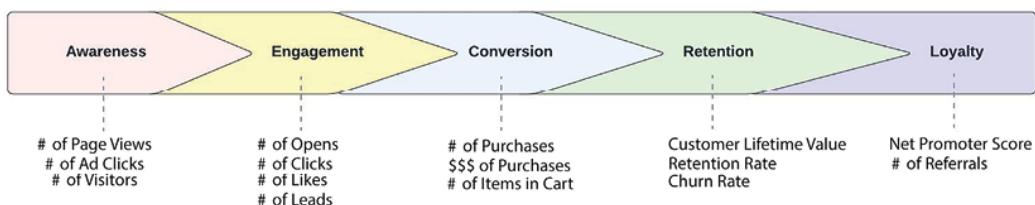


Figure 6.1: Customer life cycle diagram from Chapter 2

The applicability of predictive modeling is broad, depending on your marketing goal. For example, if you have a new brand or product launch and would like to improve new product awareness via ads on social media,

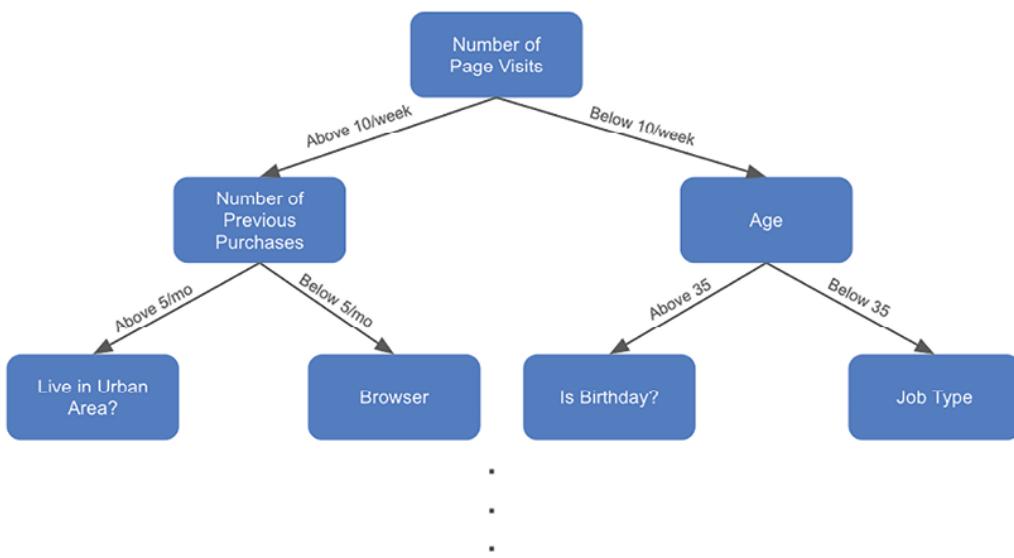
you can build predictive models that can help you identify the target customers who are likely to click on the ads. On the other hand, if you would like to improve product purchase conversion rates, you can build predictive models that can identify customers who are more likely to make purchases in the next X number of days and target them. This results in more effective marketing, as you can avoid creating fatigue among the customers, which happens when they are exposed to your marketing campaigns too frequently with irrelevant content. This happens often when you do mass marketing without targeting the right subgroup of customers. Also, you can reduce marketing costs by sending marketing materials only to a specific subgroup of customers. This will help you repurpose the remaining marketing budget for other marketing campaigns.

Not only can you utilize predictive analytics for better brand awareness, engagement, and conversion, but predictive analytics can also be used to improve retention rates. Often, customer churn likelihood models are built to identify who is at risk of turning away from your business. Through these customer churn predictions, you can build marketing strategies and marketing content that is customized for this high-churn risk group to bring them back to engaged customers. Discounts, free subscription trials, or free plan upgrades are often offered to this high-churn risk group as part of the retention strategies.

## **Tree-based machine learning algorithms**

Numerous AI/ML algorithms can be used for predictive modeling, such as linear regression, logistic regression, and decision tree models, which we have discussed in previous chapters, as well as deep learning models that

are rising in usage. In this chapter, we are going to build predictive models with tree-based models, such as random forest and gradient boosted trees, and neural network models, which are the backbones of deep learning models. Underneath any tree-based ML model, there is a decision tree. As we have discussed in *Chapter 3*, a decision tree is like a flowchart, where it splits into child nodes based on the information gained. Each node represents a question or criteria for a split and each branch or edge represents the outcome of the question posited at the node. The following diagram shows a high-level overview of how a decision tree may be built:



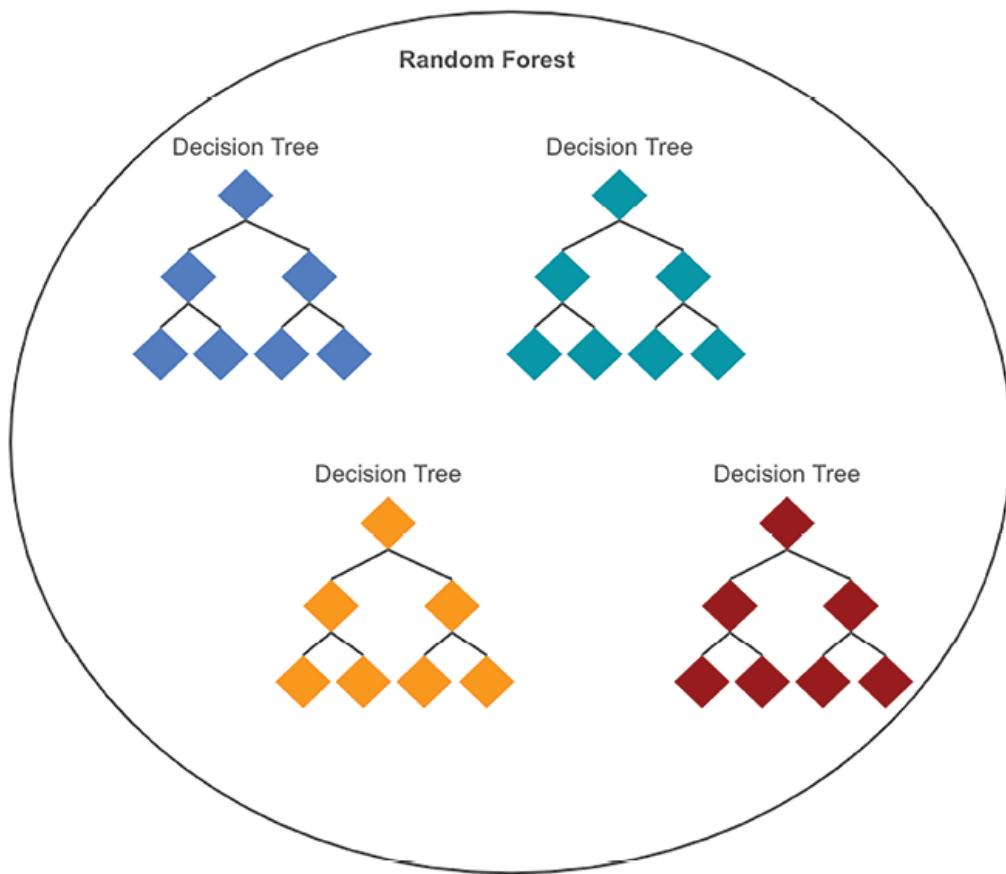
*Figure 6.2: Illustration of a decision tree*

Among numerous tree-based ML models, the **gradient-boosted decision tree (GBDT)** and random forest are the two most popular models that are frequently used for predictive modeling. Both GBDT and random forest models are built with multiple decision trees. However, the main difference is how these decision trees are built.

Simply put, a random forest model is one with lots of decision trees, where each decision tree is built with a random subsample of the dataset and a

random subset of features. This way, each decision tree within a random forest learns the information or relationships within the data slightly differently and with different focus areas.

The final prediction is the average of all the outcomes or predictions of these individual decision trees. The following shows an illustration of a random forest:



*Figure 6.3: Illustration of random forest*

A GBDT model, on the other hand, also consists of lots of decision trees, but each decision tree is built sequentially, and each subsequent decision tree is trained based on the errors that the previous decision tree makes. The final prediction of a GBDT model is the weighted average of all of the

individual decision trees' predictions. The following shows an illustration of a GBDT model:

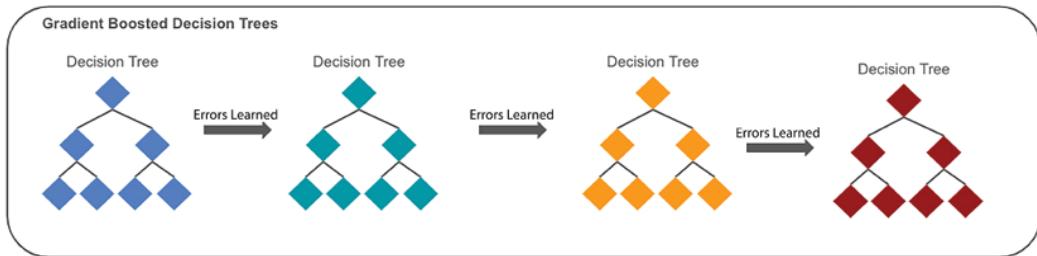


Figure 6.4: Illustration of GBDTs

## Building random forest models

In this chapter, we will be using an online purchase dataset as an example to build a predictive model to predict whether a customer will convert or not. First, we will discuss how we can build a random forest model in Python using the `scikit-learn` package.



**Source code and data:**  
<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/tree/main/ch.6>

**Data source:**  
<https://archive.ics.uci.edu/dataset/468/online+shoppers+&+purchasing+intention+dataset>

## Target and feature variables

We need to first define the target and feature variables, where the target variable is the factor that we want to predict, and the feature variables are the factors that will be learned by the models to make the predictions or decisions. To do this, you can follow these steps:

1. Let's first load the data into a DataFrame and examine what features we can use for our random forest model:

```
import pandas as pd
df = pd.read_csv("./data.csv")
df.info()
```

When you run this code, you should see the following output for the information about this data:

```
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   Administrative    12330 non-null   int64  
 1   Administrative_Duration 12330 non-null   float64 
 2   Informational     12330 non-null   int64  
 3   Informational_Duration 12330 non-null   float64 
 4   ProductRelated    12330 non-null   int64  
 5   ProductRelated_Duration 12330 non-null   float64 
 6   BounceRates       12330 non-null   float64 
 7   ExitRates         12330 non-null   float64 
 8   PageValues        12330 non-null   float64 
 9   SpecialDay        12330 non-null   float64 
 10  Month            12330 non-null   object  
 11  OperatingSystems  12330 non-null   int64  
 12  Browser          12330 non-null   int64  
 13  Region           12330 non-null   int64  
 14  TrafficType      12330 non-null   int64  
 15  VisitorType      12330 non-null   object  
 16  Weekend          12330 non-null   bool   
 17  Revenue          12330 non-null   bool
```

Figure 6.5: Summary of the example dataset

2. The first thing to note in the preceding output is the column, `Revenue`, which is the target variable that tells us whether a customer made a purchase or converted or not, has a type of `Boolean`. The other column, `Weekend`, also has a `Boolean` data type. We are going to encode them as `0` for False and `1` for True with the following code:

```
df["Revenue"] = df["Revenue"].astype(int)
df["Weekend"] = df["Weekend"].astype(int)
```

3. Then, two columns have an `object` as the data type, `Month` and `VisitorType`. If you look closer, the column, `Month`, has string values for the months, which we will convert into corresponding month numbers. The column, `visitorType`, has three unique values, `New_Visitor`, `Returning_Visitor`, and `Other`. We are going to encode each as `0`, `1`, and `2` respectively, as shown in the following code:

```
from time import strftime
df["MonthNum"] = df["Month"].apply(lambda x: strftime(x[:3])
df["VisitorTypeNum"] = df["VisitorType"].apply(
    lambda x: 0 if x == "New_Visitor" else 1 if x == "F"
)
```

As you can see in this code, we are using the `strftime` function of a `time` module to encode three-letter month string values into corresponding month numbers. Then, we use the `apply` function of a `pandas` DataFrame to encode each value of the `VisitorType` column into corresponding integer values.

4. Now that we have converted all the column values into numeric values, we are going to define the target and feature variables as in the following:

```
TARGET = "Revenue"
FEATURES = [
    'Administrative',
    'Administrative_Duration',
    'BounceRates',
    'Browser',
    'ExitRates',
    'Informational',
    'Informational_Duration',
    'MonthNum',
    'OperatingSystems',
    'PageValues',
    'ProductRelated',
    'ProductRelated_Duration',
    'Region',
    'SpecialDay',
    'TrafficType',
    'VisitorTypeNum',
    'Weekend'
]
X = df[FEATURES]
Y = df[TARGET]
```

As you can see from this code, we have defined the target variable, `TARGET`, to use the `Revenue` column and the rest columns as the feature variables, `FEATURES`. Then, we create a DataFrame, `X`, which is the feature set, and `Y`, which is the target series.

5. Lastly, we will split these target and feature sets into train and test sets with the following code:

```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(
    X, Y, test_size=0.2
)
```

We are using the `train_test_split` function within the `sklearn.model_selection` module. As you can see from the `test_size` parameter, we are using 80% of the dataset for training and the other 20% for testing.

With these train and test sets, we are now ready to train a random forest model.

## Training a random forest model

Python's `scikit-learn` package provides a handy way to a random forest model. Take a look at the following code:

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(
    n_estimators=250, max_depth=5, class_weight="balanced", n_j
)
rf_model.fit(train_x, train_y)
```

Let's take a closer look at this code. We are using the `RandomForestClassifier` class from the `sklearn.ensemble` module and initiated a Random Forest model with `n_estimators`, `max_depth`, `class_weight`, and `n_jobs` parameters:

- The `n_estimators` parameter defines how many individual decision trees to build.

- The `max_depth` parameter defines how deep each decision tree can grow. Along with other parameters, such as `min_samples_split`, `min_samples_leaf`, and `max_features`, `max_depth` helps prevent overfitting issues by limiting how much a decision tree can grow.
- The `class_weight` parameter defines weights for each class. This parameter is useful when the dataset is imbalanced. In our example dataset, only about 15% are in the positive class, meaning only 15% of the target variable, Revenue, has a value of 1, or only 15% of customers have converted. You can give custom weights for each class as a dictionary or use the `"balanced"` option to automatically adjust the weights that are inversely proportional to the actual class frequencies.
- Lastly, the `n_jobs` parameter defines how many jobs to run in parallel. If you recall from our discussion of random forest and GBDTs, random forest is a bag of decision trees, so individual trees can be built in parallel without any dependency on other trees. By giving `-1` as the input for this parameter, you are instructing it to use all available resources to train this random forest model.

With these parameters, we can now train this random forest model, using the `fit` function with the train set.

### Overfitting versus underfitting?

Overfitting refers to when models are fit to the training set so closely that it does well on the training data but poorly on the data that the models have not seen before. Underfitting, on the other hand, is when models are over-generalized or do not tune well enough to the training set that they did not



learn the relationships between the feature variables and the target variable. Multiple iterations of hyperparameter tuning are often required to find the sweet spot for minimal overfitting.

## Predicting and evaluating random forest model

The `RandomForestClassifier` object provides handy functions for making predictions from the trained random forest model. Take a look at the following code:

```
rf_pred = rf_model.predict(test_x)
rf_pred_proba = rf_model.predict_proba(test_x)[:, 1]
```

As the names suggest, the `predict` function makes predictions on the given input. In our case, the results will be a list of 0s and 1s for each record in the test set, as we are predicting whether a customer has converted or not.

The `predict_proba` function also makes predictions on the given input, but the difference is it gives predicted probabilities running between `0` and `1`. It returns predicted probabilities for each record and for each class, so, in our case, it returns two values for each record, where the first element is a predicted probability to be a class of `0` and the second element is a predicted probability to be a class of `1`. Since we are only interested in the predicted probability of class 1, we are slicing it with `[:, 1]` so that we have a list of predicted probabilities of conversion.

Now that we have the predicted conversion probabilities, we need to evaluate how good our predictions are.

There are multiple ways to evaluate the accuracy and effectiveness of a predictive model, but we will mainly look at the overall accuracy, precision, recall, **area under the curve (AUC) - receiver operating characteristics (ROC)** curve, and confusion matrix. We will go deeper into these metrics with examples.

As the name suggests, the **accuracy** is the percentage of correct predictions or **true positives (TP)** among all the predictions. The **precision** is the percentage of correct predictions among those predicted positive or the percentage of TPs among those predicted to be positive that include **false positives (FP)**. The **recall** is the percentage of positive cases identified by the model or the percentage of TPs among the actual positives, which are TPs and false negatives. The equations for the accuracy, precision, and recall are as follows:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Predictions}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

In Python, the scikit-learn package provides a handy tool for computing these metrics, as shown in the following code:

```
from sklearn import metrics
accuracy = (test_y == rf_pred).mean()
precision = metrics.precision_score(test_y, rf_pred)
recall = metrics.recall_score(test_y, rf_pred)
```

In our example, when these codes are run, the results of these key metrics look as in the following:

```
** Key Metrics:  
Accuracy: 86.3%, Precision: 53.9%, Recall: 84.9%
```

*Figure 6.6: Summary of random forest model performance metrics*

These results suggest the random forest model we have trained has decent overall accuracy and recall rates but does not seem to do well with precision. This suggests that of those customers this model predicted to be positive or likely to convert, only about 54% of them have actually converted. However, if you recall, the actual overall conversion rate is 15%; in other words, if you randomly guess who will convert, you may only be right about 15% of the time.

Thus, since this model has predicted converted customers 54% of the time, it proves to be way much more effective in selecting customers that are more likely to convert than random guesses. Also, the high recall rate suggests that about 85% of the customers who have actually converted are among those who are predicted to be highly likely to convert by this model.

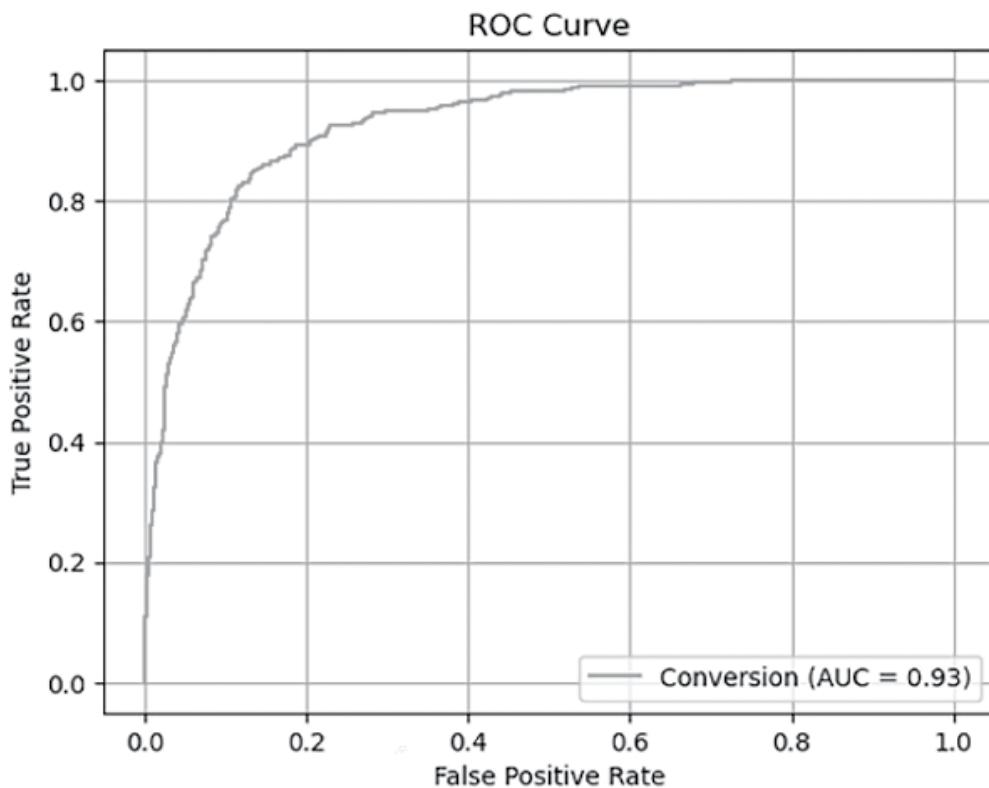
From an actual marketing perspective, if you have marketed only to the customers who have been predicted to be likely to convert by this model, you would have still captured most of those conversions. Also, if you have marketed to your entire customer base, 85% (100% minus 15%, which is the overall conversion rate) of your marketing spend would have been wasted. But if you have marketed only to these highly likely customers, only about 46% (100% minus 54%, which is the precision of this model) of the marketing spend would have been wasted.

The other key evaluation metric we are going to look at is the **AUC - ROC curve**. The **ROC curve**, simply put, shows the trade-offs between gains in **true positive rates (TPRs)** for each sacrifice you make for a **false positive rate (FPR)**. The **AUC** is, as the name suggests, the area under the ROC curve and tells us how well the model separates the positive cases from negative cases. The AUC ranges from `0` to `1` and the higher it is, the better the model is. At the AUC of `0.5`, it suggests that the model performs the same as random guessing.

The following code can be used to plot the ROC curve:

```
dp = metrics.RocCurveDisplay.from_predictions(  
    test_y,  
    rf_pred_proba,  
    name="Conversion",  
    color="darkorange",  
)  
_ = dp.ax_.set(  
    xlabel="False Positive Rate",  
    ylabel="True Positive Rate",  
    title="ROC Curve",  
)  
plt.grid()  
plt.show()
```

Here, we are using the `metrics` module again to plot the ROC curve. The main difference between computing the accuracy, precision, and recall and computing the AUC - ROC curve is how we use the predicted probabilities, `rf_pred_proba`, instead of predicted labels, `pred`. This is because the ROC curve examines how TPRs and FPRs change at different probability levels. By using predicted probabilities, we can calculate how the TPR and FPR change as the decision threshold varies. The resulting chart looks like the following:



*Figure 6.7: AUC - ROC curve of the random forest model predictions*

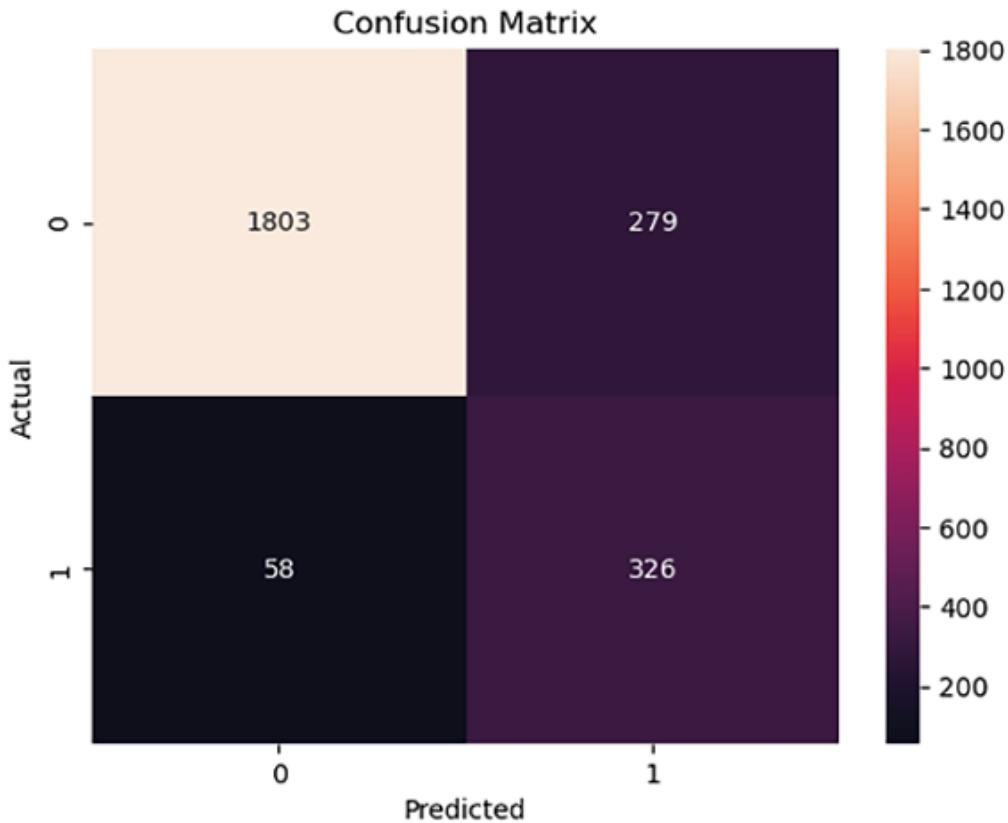
As you can see from this chart, you can easily evaluate how each sacrifice in the FPR affects the TPR. For example, in this chart, at 20% FPR, we already achieve about 90% TPR, which suggests that this model does well separating positive cases from negative cases. The AUC here is 0.93, which also suggests that the model does well in identifying positive cases from negative cases.

Lastly, we will look at the **confusion matrix**. As the name suggests, the confusion matrix is a good way to look at how and where the model gets confused the most. It will be easier to understand with an example. Take a look at the following code:

```
import seaborn as sns
cf_matrix = metrics.confusion_matrix(test_y, rf_pred)
```

```
ax = plt.subplot()
sns.heatmap(
    cf_matrix,
    annot=True,
    annot_kws={"size": 10},
    fmt="g",
    ax=ax
)
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
ax.set_title(f"Confusion Matrix")
plt.show()
```

Similar to before, we are using the `metrics` module to build a confusion matrix. The `confusion_matrix` function takes the actual and predicted values and builds a confusion matrix. Then, we are using the `heatmap` function of the `seaborn` Python package to plot a heat map. The resulting chart looks like the following:



*Figure 6.8: Confusion matrix of the random forest model predictions*

As you can see from this plot, the y-axis represents the actual classes and the x-axis represents the predicted classes. For example, the top left box is where the actual class was 0 or no conversion, and the predicted class was also 0. The top right box is where the actual class was 0, but the model predicted them to be a class of 1 or conversion.

As you can see, the confusion matrix shows you where the model is the most confused. A model that predicts the outcome with high accuracy will have large numbers or percentages in the diagonal boxes and small numbers in the other boxes.

We have experimented with predicting the customer conversion likelihood with a random forest model. Here, we have observed and discussed how this random forest predictive model can help target the subset of the

customer base without losing too many converted customers and how this can result in much more cost-effective marketing strategies.

## Gradient boosted decision tree (GBDT) modeling

We will use the same dataset and train/test sets for building a GBDT model and compare its performance against the random forest model we have just built. XGBoost is the most commonly used library in Python for training a GBDT model. You can install this package using the following command in the terminal or Jupyter Notebook:

```
pip install xgboost
```

## Training GBDT model

The XGBoost package follows the same pattern as the `scikit-learn` package. Take a look at the following code:

```
from xgboost import XGBClassifier
xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    scale_pos_weight=1/train_y.mean(),
)
xgb_model.fit(train_x, train_y)
```

As you can see from this code, we initiate an `XGBClassifier` with the `n_estimators`, `max_depth`, and `scale_pos_weight` parameters:

- Similar to the case of the random forest model, the `n_estimators` parameter defines how many individual decision trees are to be built. You may have noticed we are using a much lower number for this parameter. If you recall, a GBDT model is sequentially built where each subsequent decision tree learns the errors the previous decision tree makes. This often results in a smaller number of individual trees performing as well as or better than the random forest model.

Also, because a GBDT model is sequentially built, a large number of decision trees results in much longer training time than a random forest model, which can build individual decision trees in parallel.

- The other parameter, `max_depth`, is the same as in the case of the random forest model. This parameter restricts how deep each decision tree can grow.
- Lastly, the `scale_pos_weight` parameter defines the balance of the positive and negative class weights. As you may recall we have an imbalanced dataset where the positive class is only about 15% of the data. By giving inversely proportionate weight for the positive class with `1/train_y.mean()`, we are instructing the model to adjust for the imbalanced dataset. This enforces the GBDT model to penalize more for incorrect predictions of the positive class, which makes the model more sensitive to the positive class.

There are various ways to handle potential overfitting issues that often occur when you have large individual decision trees within a GBDT model. On top of the `max_depth` parameter, the XGBoost package provides various other



parameters, such as `max_leaves`, `colsample_bytree`, `subsample`, and `gamma`, that can help reduce overfitting.

We suggest that you look at the documentation and experiment with various parameters and see how they help you prevent overfitting.

Here is the documentation:

[https://xgboost.readthedocs.io/en/latest/python/python\\_api.html#xgboost.XGBClassifier](https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBClassifier).

## Predicting and Evaluating GBDT Model

Similar to the `RandomForestClassifier` of the random forest model, the `XGBClassifier` object also provides the same syntax for predicting with the trained GBDT model. Take a look at the following code:

```
xgb_pred = xgb_model.predict(test_x)
xgb_pred_proba = xgb_model.predict_proba(test_x)[:, 1]
```

As you can see from this code, you can use the `predict` function to predict the positive versus negative label for each record of the test set and the `predict_proba` function to predict the probabilities for each class of individual records of the test set. As before, we are retrieving the second column of the predicted probabilities by slicing the output with `[:, 1]` to get the predicted probabilities of the positive class, which is the predicted probability of a conversion, for each record of the test set.

To compare the performance of this GBDT model against the random forest model, we will use the same evaluation metrics and approaches. As you may recall, we can use the following code to get the key metrics of accuracy, precision, and recall:

```
accuracy = (test_y == xgb_pred).mean()
precision = metrics.precision_score(test_y, xgb_pred)
recall = metrics.recall_score(test_y, xgb_pred)
```

Due to some randomness in building these trees, there can be some variances and differences each time a GBDT model is trained, but at the time of this writing, the results look as follows:

```
** Key Metrics:
Accuracy: 86.6%, Precision: 54.9%, Recall: 79.2%
```

*Figure 6.9: Summary of GBDT model performance metrics*

These key metrics look very similar to the random forest model. The overall accuracy and precision of this GBDT model are slightly higher than the random forest model. However, the recall is slightly lower than the random forest model.

Similarly, the AUC-ROC curve can be plotted using the following code:

```
dp = metrics.RocCurveDisplay.from_predictions(
    test_y,
    xgb_pred_proba,
    name="Conversion",
    color="darkorange",
)
_ = dp.ax_.set(
    xlabel="False Positive Rate",
    ylabel="True Positive Rate",
    title="ROC Curve",
```

```
)  
plt.grid()  
plt.show()
```

The resulting AUC-ROC Curve looks like the following:

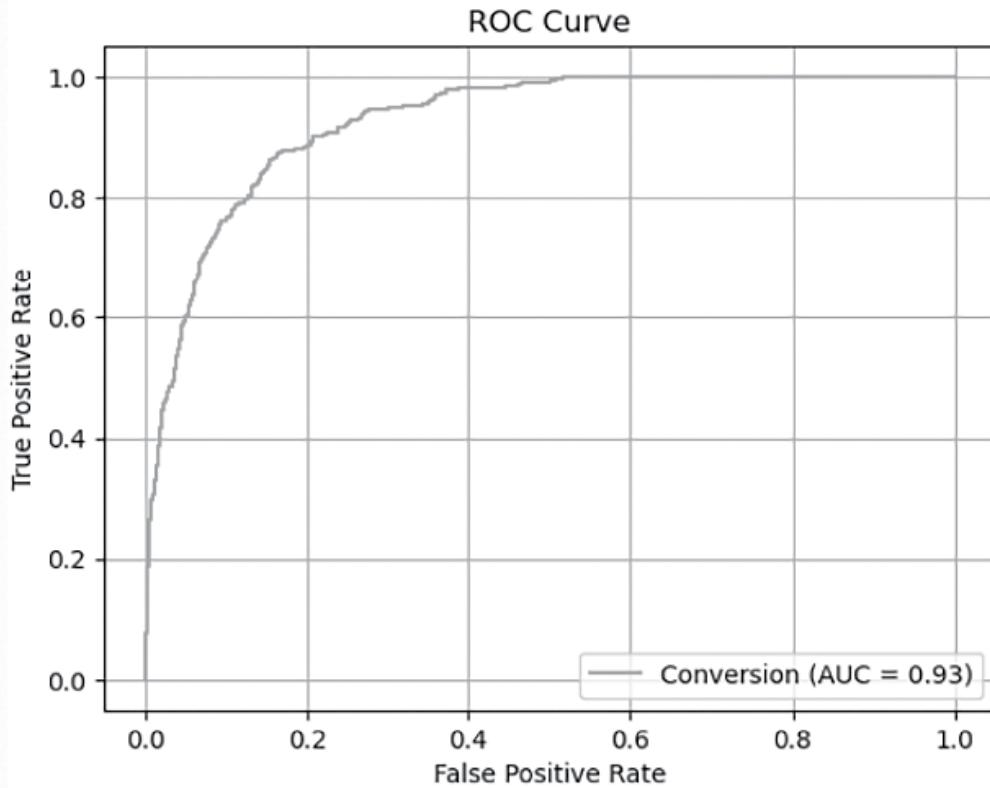


Figure 6.10: AUC- ROC Curve of the GBDT model predictions

When you compare this against the random forest model, the results are almost identical. AUC is about the same with `0.93` and at the FPR of `0.2`, the TPR of the GBDT model is also about `0.9`, which was also the case of the previously built random forest model.

Lastly, let's take a look at the confusion matrix with the following code:

```
import seaborn as sns  
cf_matrix = metrics.confusion_matrix(test_y, xgb_pred)
```

```
ax = plt.subplot()
sns.heatmap(
    cf_matrix,
    annot=True,
    annot_kws={"size": 10},
    fmt="g",
    ax=ax
)
ax.set_xlabel("Predicted")
ax.set_ylabel("Actual")
ax.set_title(f"Confusion Matrix")
plt.show()
```

The resulting confusion matrix looks like the following:

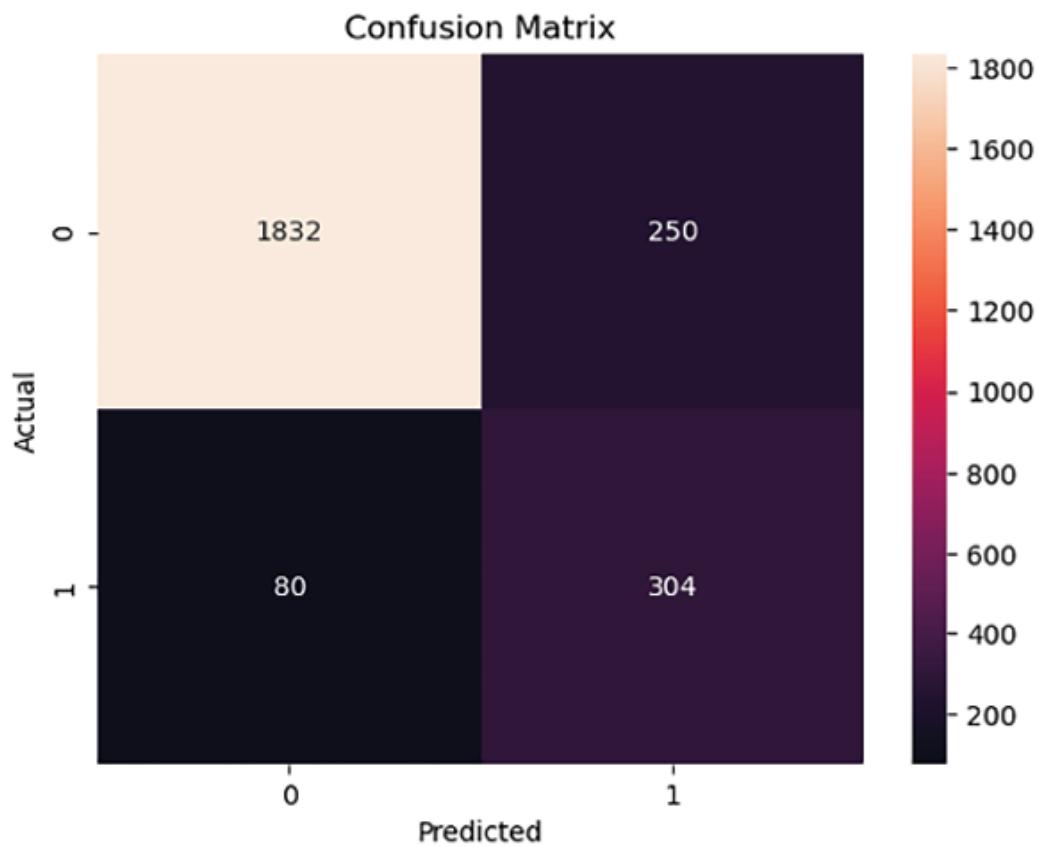


Figure 6.11: Confusion matrix of the GBDT model predictions

When you compare this against the confusion matrix of the random forest model, you will notice that the TPs are lower with the GBDT model, but false negatives are also lower with the GBDT model. This is expected as we have seen that the precision of the GBDT model is higher, but recall is lower compared to the random forest model.

Overall, the performances of random forest and GBDT models are very similar and hard to distinguish from these examples. Depending on how you fine-tune your model, you may end up with a better random forest or GBDT model.



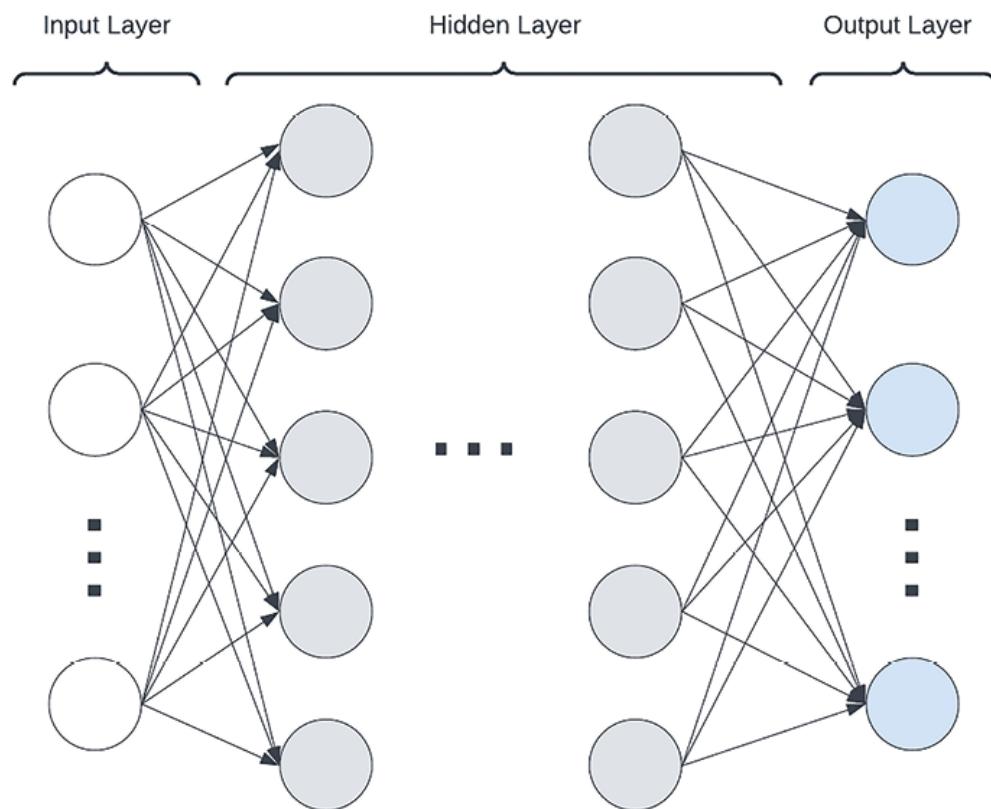
There are numerous ways and parameters you can fine-tune the tree-based models. We suggest you experiment with various sets of parameters and see how they affect the model's performance!

## Predicting customer conversion with deep learning algorithms

Deep learning has become a hot topic and its popularity and usage are rising, as deep learning models are proven to work well when data have complex relationships within the variables and learn or extract features autonomously from the data, even though tree-based models are also very frequently used and powerful for predictive modeling. We touched on deep learning in *Chapter 5* when we used pre-trained language models for sentiment analysis and classification. In this section, we are going to build

on that knowledge and experiment with developing deep learning models for predictive modeling and, more specifically, for making predictions on which customers are likely to convert.

Deep learning is basically an **artificial neural network (ANN)** model with lots of hidden and complex layers of neurons, or, in other words, a deep ANN. An ANN is a model inspired by the biological neural networks in animal and human brains. An ANN learns the data through layers of interconnected neurons that resemble animal and human brains. The following diagram shows the high-level structure of an ANN:



*Figure 6.12: Example ANN architecture*

As this diagram shows, there are three layers in any deep learning or ANN model: the input layer, hidden layer, and output layer. Detailed explanations of how ANN models learn or build the hidden layer weights are beyond the

scope of this book, but at a high level, they go through iterations of forward propagations and backward propagations:

- Forward propagation is where the data is fed through a network from the input layer to the output layer and the activation function is applied regardless of whether each neuron is activated or not and returns the output of each node.
- Backward propagation is the process of moving from the output layer to the input layer and adjusting the weights of the network by analyzing the losses or errors from the previous iteration.

Through iterations of these forward and backward propagations, a neural network learns the weights of each neuron that minimizes the error of the predictions.

A number of hidden layers and a number of neurons in each layer should be experimented with to find the right neural network architecture that works best for each case of predictive models. In this section, we are going to experiment with how wide neural network architecture performs against deep neural network architecture.

### **When to use wide versus deep neural networks?**

A wide neural network refers to an ANN model that has fewer layers but more neurons in each layer, whereas a deep neural network refers to an ANN model with lots of hidden layers. Aside from the model performance comparisons to see which architecture works better for your case, there are some key factors you may want to consider or limit which approach to take:



- **Training time and compute resources:** As the number of layers grows, it takes more time to train as backpropagation through each layer is computationally intensive, which also results in higher compute costs. The wide neural network may have an advantage in shortening the training time and reducing the compute costs.
- **Model interpretability:** Shallower architecture may offer better interpretability compared to deep neural networks. If explainability is a requirement, shallow architecture may be a better fit.
- **Ability to generalize:** Deep neural network models tend to be able to capture more abstract patterns through more layers and learn higher-order features compared to wide neural networks. This may result in the deep architecture model having better performance on new and unseen data.

## Train and test sets for deep learning models

For the best performances of neural network models, you need to normalize the data before you train the models. We are going to use the same train and test sets that we used previously for tree-based models, but normalize them. Take a look at the following code:

```
mean = train_x.mean()  
std = train_x.std()
```

```
normed_train_x = (train_x - mean)/std  
normed_test_x = (test_x - mean)/std
```

As this code suggests, we first get the mean and standard deviation from the train set and normalize both train and test sets by subtracting the mean and dividing by the standard deviation. After the standardization, the `normed_train_x` DataFrame should have means of `0` and standard deviations of `1` for all the columns.

## Wide neural network modeling

We will first look at building **wide neural network** models using the `keras` package in Python.



To install `keras` on your machine, run the following command in your terminal:

```
pip install keras
```

## Training the wide neural network model

With the normalized train and test sets, let's start building neural network models. Take a look at the following code to see how a neural network model can be initiated:

```
import keras  
model = keras.Sequential(  
    [  
        keras.Input(shape=normed_train_x.shape[1:]),
```

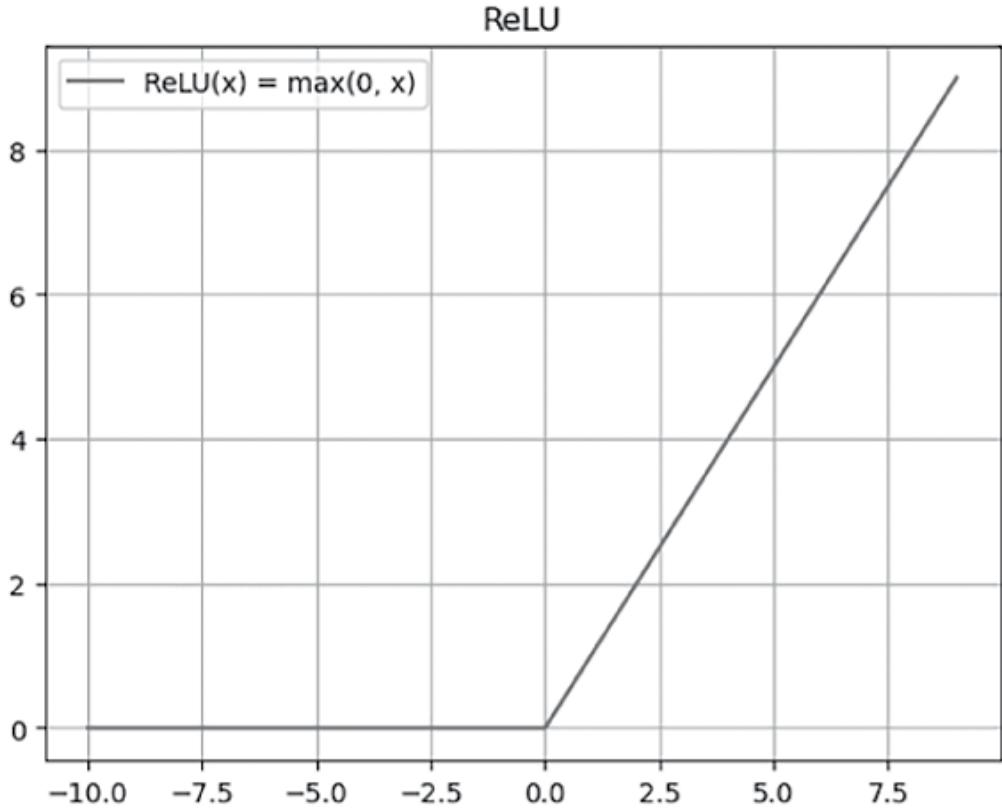
```
        keras.layers.Dense(2048, activation="relu"),
        keras.layers.Dropout(0.2),
        keras.layers.Dense(1, activation="sigmoid"),
    ]
)
```

As you can see from this code, we create a sequence of layers with the `Sequential` class. We start with the input layer with the `Input` class, where we define the shape or the number of input neurons to match the number of columns or features of the train set. Then, we have one wide hidden layer with 2,048 neurons. We added a `Dropout` layer, which defines what percentage of neurons to drop and this helps reduce the overfitting issues. We are instructing the neural network model to drop 20% of the neurons between the hidden and output layers. Lastly, we have one `Dense` layer for the output layer with one output neuron, which will output the predicted probability of the positive case or the likelihood of a conversion.

You may have noticed there are two activation functions we use for the hidden layer and the output layer. The **rectified linear unit (ReLU)** activation function is one of the most frequently used activation functions, which only activates the positive values and deactivates all the negative values. The equation for the ReLU activation function looks as follows:

$$\text{ReLU}(x) = \max(0, x)$$

The behavior of the ReLU activation function looks like the following:



*Figure 6.13: ReLU activation function*

The **sigmoid** function, on the other hand, turns values into a range of  $0$  and  $1$ . This makes the sigmoid function the top choice to predict the probability as an output. The equation for the sigmoid function is as follows:

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

The behavior of the sigmoid function looks as follows:

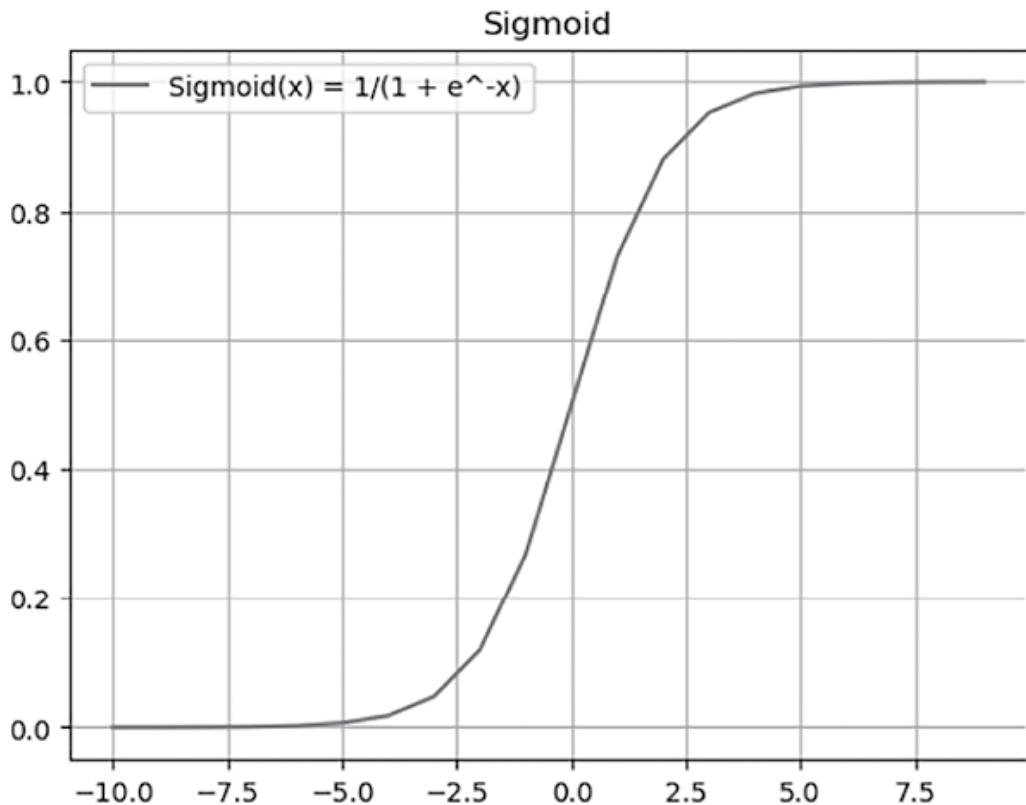


Figure 6.14: Sigmoid activation function



There are various other activation functions other than ReLU and sigmoid, such as tanh, leaky ReLU, and softmax. We suggest you do some research and experiment with how different activation functions work and affect the neural network models' performances!

There are a few more steps involved before we can train a neural network model:

1. First, we need to define the key metrics we will track through the iterations of model training and compile the model. Take a look at the following code:

```
model_metrics = [
    keras.metrics.Accuracy(name="accuracy"),
    keras.metrics.Precision(name="precision"),
    keras.metrics.Recall(name="recall"),
]
model.compile(
    optimizer=keras.optimizers.Adam(0.001),
    loss="binary_crossentropy",
    metrics=model_metrics
)
```

As we have discussed previously with the tree-based models, we are going to track accuracy, precision, and recall. We will be using the Adam optimizer for this exercise. Simply put, optimizers are the algorithms that are used to update the weights of the network to minimize the errors based on the loss function, which in this case is `binary_crossentropy`, which measures the difference between the predicted and actual binary outcomes. Since our output or target variable for the prediction is a binary variable of conversion versus no conversion, `binary_crossentropy` will be the right fit for the loss function. For multi-class predictions where the target variable is not a binary variable and has more than two possible outcomes, `categorical_crossentropy` will be a better-suited loss function.

2. Next, we are going to have a checkpoint to save the best model among all the iterations or epochs that we will run while training the model. Take a look at the following code:

```
best_model_path = "./checkpoint.wide-model.keras"
model_checkpoint_callback = keras.callbacks.ModelCheckpoint
    filepath=best_model_path,
    monitor="val_precision",
    mode="max",
```

```
        save_best_only=True  
    )
```

Here, we are using precision as the metric to monitor and store the best model to a local drive based on that metric value. The best model is going to be stored as defined in the `best_model_path` variable.

3. Finally, it is now time to train this **wide neural network** model with the following code:

```
class_weight = {0: 1, 1: 1/train_y.mean()}  
history = model.fit(  
    normed_train_x.to_numpy(),  
    train_y.to_numpy(),  
    batch_size=64,  
    epochs=30,  
    verbose=2,  
    callbacks=[  
        model_checkpoint_callback,  
    ],  
    validation_data=(normed_test_x.to_numpy(), test_y.to_nu  
    class_weight=class_weight,  
)
```

Similar to the class weights we have given to our tree-based models previously, we are also going to define the class weights that are inversely proportional to the class composition. As you can see from the code, we instruct the model to run for 30 iterations or epochs with a batch size of 64. We register the callback that we created to store the best model previously with the custom class weights. When you run this code, it will run for 30 epochs and report the accuracy, precision, and recall metrics that we defined previously. The output of this code should look something like the following:

```
Epoch 1/30
155/155 - 1s - loss: 0.8298 - accuracy: 9.1241e-04 - precision: 0.4052 - recal
l: 0.8091 - val_loss: 0.4735 - val_accuracy: 8.1103e-04 - val_precision: 0.4235
- val_recall: 0.8438 - 1s/epoch - 7ms/step
Epoch 2/30
155/155 - 0s - loss: 0.7282 - accuracy: 3.0414e-04 - precision: 0.4585 - recal
l: 0.8294 - val_loss: 0.4113 - val_accuracy: 0.0000e+00 - val_precision: 0.4868
- val_recall: 0.8177 - 338ms/epoch - 2ms/step
Epoch 3/30
155/155 - 0s - loss: 0.6947 - accuracy: 0.0000e+00 - precision: 0.4652 - recal
l: 0.8301 - val_loss: 0.4247 - val_accuracy: 0.0000e+00 - val_precision: 0.4744
- val_recall: 0.8438 - 305ms/epoch - 2ms/step
Epoch 4/30
155/155 - 0s - loss: 0.6772 - accuracy: 0.0000e+00 - precision: 0.4796 - recal
l: 0.8425 - val_loss: 0.4238 - val_accuracy: 0.0000e+00 - val_precision: 0.4670
- val_recall: 0.8464 - 308ms/epoch - 2ms/step
```

Figure 6.15: Sample output of the neural network model

## Predicting and evaluating wide neural network model

For making predictions from the model we have just trained, we first need to load the best model. You can use the following code to load the best model:

```
wide_best_model = keras.models.load_model(best_model_path)
```

Making predictions with this model is similar to the `scikit-learn` package's syntax. Take a look at the following code:

```
wide_pred_proba = wide_best_model.predict(normed_test_x).flatten
wide_preds = [1 if x > 0.5 else 0 for x in wide_pred_proba]
```

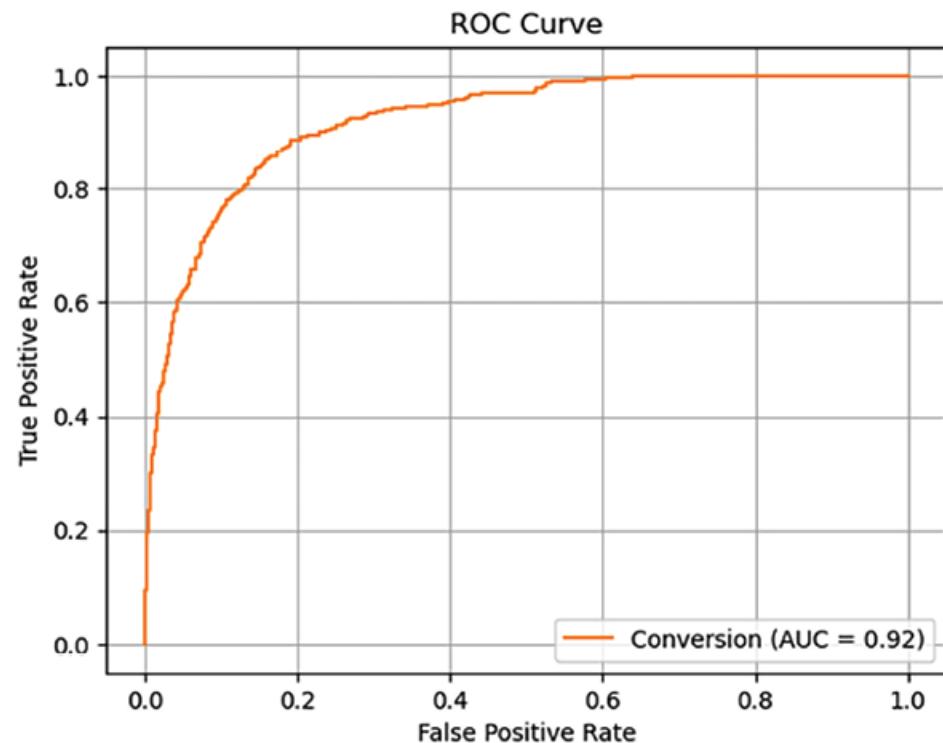
As you may notice from this code, you can use the `predict` function of the `keras` model to get the predicted probabilities for each record. We use the `flatten` function to make the predicted probability output a list of predicted probabilities from a two-dimensional array. Then, for illustration purposes,

we are going to assume any record with a predicted probability above `0.5` is considered to have a high chance of converting and assume they are positive cases. The probability threshold is another factor you can fine-tune for the final predictions of conversions versus no conversions.

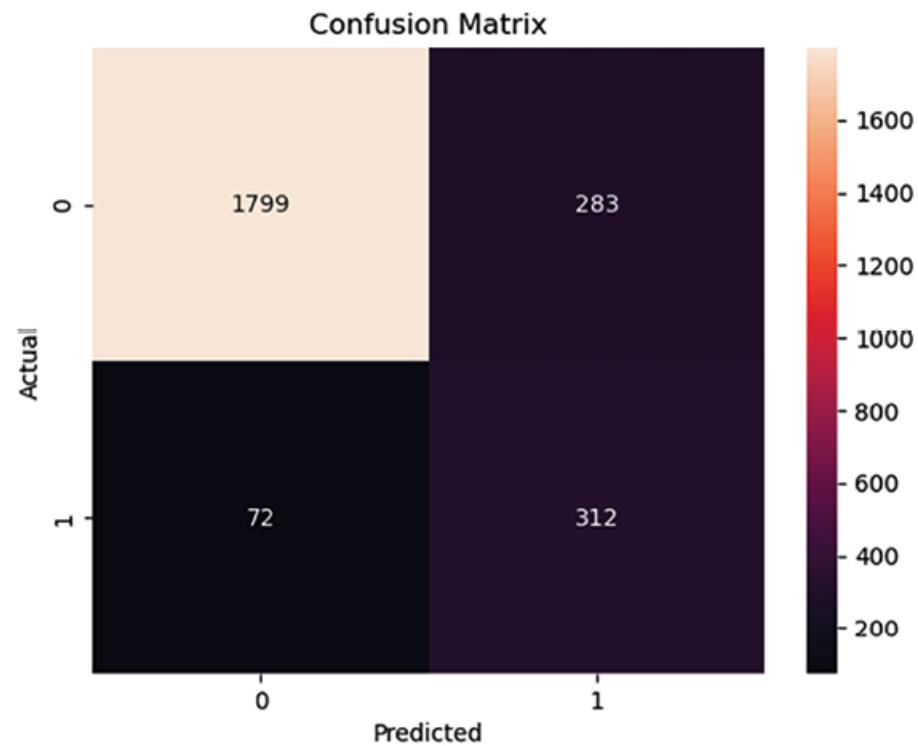
We are going to use the same codes that we have used previously for the tree-based models for key metrics, the AUC-ROC curve, and the confusion matrix. The results look as follows:

\*\* Key Metrics:  
Accuracy: 85.6%, Precision: 52.4%, Recall: 81.2%

\*\* AUC/ROC:



\*\* Confusion Matrix:



*Figure 6.16: Summary of the evaluation results for the wide neural network model*

If you compare these results against the tree-based models, you will notice the wide neural network model performed a little worse than the tree-based models. The precision and recall are slightly lower and the AUC is also a little lower compared to those of the tree-based models. We will go deeper into the practical uses of these models and how to choose the best model to use for marketing campaigns later when we discuss A/B testing in more depth, but the model performances based on the train and test sets look better for the tree-based models.

## Deep neural network modeling

In this section, we will discuss how to build deep neural network models and compare the results against the wide neural network model that we built in the previous section. The only difference between the two neural network models is how we structure or architect them.

### Training deep neural network model

Take a look at the following code:

```
model = keras.Sequential(  
    [  
        keras.Input(shape=normed_train_x.shape[1:]),  
        keras.layers.Dense(128, activation="relu"),  
        keras.layers.Dense(128, activation="relu"),  
        keras.layers.Dropout(0.2),  
        keras.layers.Dense(128, activation="relu"),  
        keras.layers.Dropout(0.2),  
        keras.layers.Dense(1, activation="sigmoid"),  
    ]  
)
```

As before, we are using `keras.Sequential` to build a model, but this time we have more layers between the first `Input` layer and the final output `Dense` layer:

- We have four hidden layers with 128 neurons each with the ReLU activation function.
- There are also two `Dropout` layers between the last and the second last hidden layers and between the last hidden layer and the output layer.

Compared to the previous wide neural network model, each layer has a smaller number of neurons but it has a larger number of layers. As you can see from this example, the model architecture is up to the person building the model and should be based on the performance. We suggest experimenting with different architectures of the model and analyzing how it changes the model performances.

Using the same Keras model training code we used previously for the wide neural network model, you can train this deep neural network model as well. Make sure you have the callback function to save the best model so that we can use it to evaluate the model's performance.

## Predicting and evaluating the deep neural network model

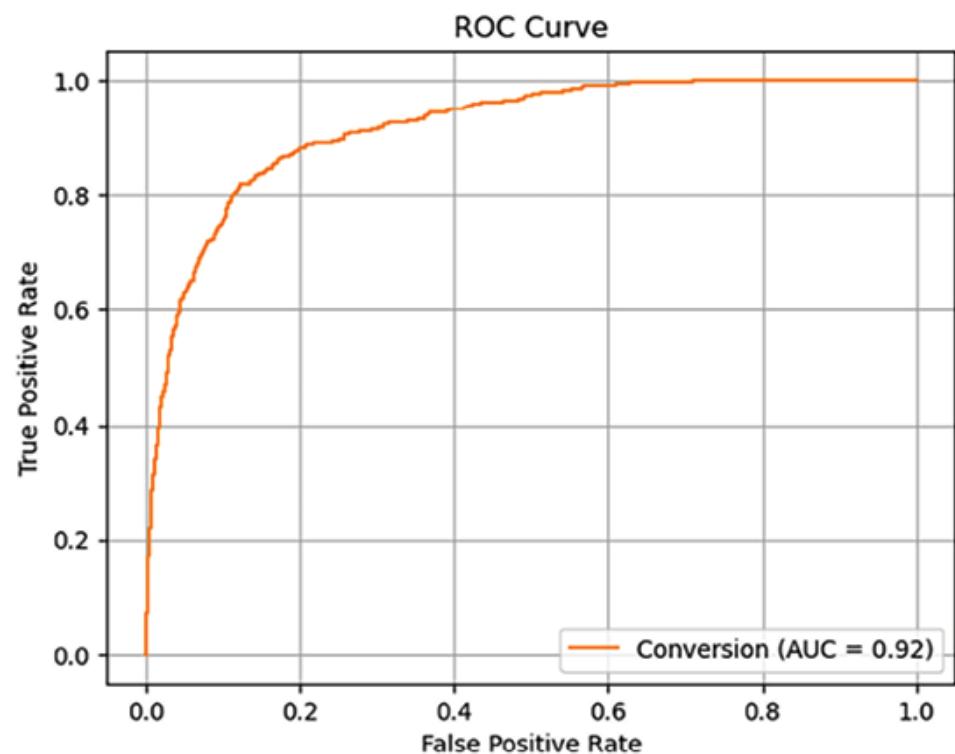
As before, you can use the following code to make predictions based on the best model found from training the deep neural network model:

```
deep_best_model = keras.models.load_model(best_model_path)
deep_pred_proba = deep_best_model.predict(normed_test_x).flatte
```

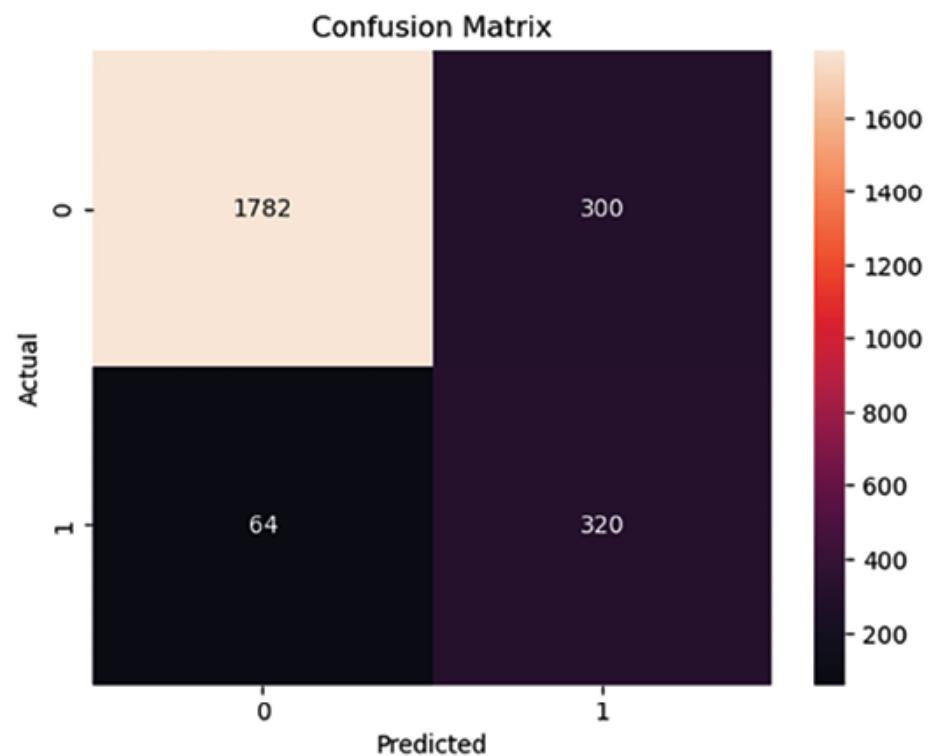
With the predicted probabilities, you can run the same evaluation metrics and charts. Due to the randomness in the models, the results may vary slightly each time you train the models. At the time of this run, the results looked as in the following:

**\*\* Key Metrics:**  
Accuracy: 85.2%, Precision: 51.6%, Recall: 83.3%

**\*\* AUC/ROC:**



**\*\* Confusion Matrix:**



*Figure 6.17: Summary of the evaluation results for the deep neural network model*

Compared to the wide neural network model we have built previously, the deep neural network model performs slightly worse. The AUC is about the same, but the precision and recall are lower than those with the wide neural network. These evaluation metrics and plots help examine the model performances at the facial value and at the training time. For more practical evaluations between models, you may want to consider running A/B testing to choose the final model for your marketing campaigns.

## **Conducting A/B testing for optimal model choice**

**A/B testing**, simply put, compares two versions of features or models to identify which one is better. It plays a critical role in decision-making processes across various industries. Web developers may use A/B testing to test which of the two versions of the app performs better. Marketers may use A/B testing to test which version of the marketing messages may do better in engaging potential customers. Similarly, A/B testing can be used to compare two different models in terms of their performance and effectiveness. In our example in this chapter, we can use A/B testing to choose which of the models we have built based on our train and test sets may work the best in the actual real-world setting.

A/B testing is typically conducted across a predefined set of periods or until a predefined number of samples are collected. This is to ensure you have enough samples collected to make your decisions based on. For example, you may want to run A/B testing for two weeks and collect the results for the duration of the test at the end of the two weeks and analyze the results.

Or, you may want to set the target of 2,000 samples for both A and B scenarios without setting a certain period.

You may also want to set both the period and the number of samples as the target, whichever comes first. For example, you set the sample target at 2,000 samples and a 2-week timebox; if you end up collecting 2,000 samples before 2 weeks, you may want to terminate the test sooner as you have collected enough samples to analyze the results. The period and sample size for A/B testing should be determined based on the business needs or limitations and confidence in the test results.

Examining the results of A/B testing is typically done with statistical hypothesis testing. The **two-sample t-test** is frequently used to determine whether the differences observed in A and B cases of the A/B testing are statistically significant or not. There are two important statistics in a t-test – the **t-value** and **p-value**:

- The **t-value** measures the degree of difference between the two means relative to the variances of the two populations. The larger the **t-value** is, the more different the two groups are.

The equation to get the **t-value** for the two-sample **t-test** is as follows:

$$t = \frac{M_1 - M_2}{\sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}}}$$

$M_1$  and  $M_2$  are the averages,  $S_1$  and  $S_2$  are the standard deviations, and  $N_1$  and  $N_2$  are the number of samples in groups 1 and 2. As you may infer from this equation, the large negative **t-value** will suggest the average of group 2 is significantly larger than the average of group 1 and the large positive t-

value will suggest the average of group 1 is significantly larger than the average of group 2. This is also called a two-tailed **t-test**.

- The **p-value**, on the other hand, measures the probability that the results occur by chance. Thus, the smaller the **p-value** is, the more statistically significant that the two groups are different and the difference is not by a mere chance.

## Simulating A/B Testing

We are going to simulate A/B testing as if we are running this experiment live with our models. For illustration purposes, we are going to run A/B testing on the XGBoost or GBDT model and the wide neural network model that we have built previously. We are going to run this A/B test for 1,000 samples for each scenario and analyze the results to see which one of the two models may work better for our marketing campaign to maximize customer conversion rates.

First, we are going to randomly route the incoming traffic to group A (GBDT model) and group B (wide neural network model). Take a look at the following code:

```
group_a_indexes = sorted(list(np.random.choice(2000, 1000, replace=False)))
group_b_indexes = [x for x in range(2000) if x not in group_a_indexes]
```

Here, we are using the `numpy` package's `random.choice` function to randomly select 1,000 items from 2,000 items. We route these to group A and the rest to group B. Then, we simulate the A/B test as in the following code:

```
group_a_actuals = test_y.iloc[group_a_indexes].to_numpy()
group_b_actuals = test_y.iloc[group_b_indexes].to_numpy()
group_a_preds = []
group_b_preds = []
for customer in range(2000):
    if customer in group_a_indexes:
        # route to XGBoost
        conversion = test_y.iloc[customer]
        pred_prob = xgb_model.predict_proba(
            np.array([test_x.iloc[customer].to_numpy(),]))
        )[0][1]
        pred = 1 if pred_prob > 0.5 else 0
        group_a_preds.append(pred)
    elif customer in group_b_indexes:
        # route to Wide Net
        conversion = test_y.iloc[customer]
        pred_prob = wide_best_model.predict(
            np.array([normed_test_x.iloc[customer].to_numpy(),]))
        )[0][0]
        pred = 1 if pred_prob > 0.5 else 0
        group_b_preds.append(pred)
```

As you can see from this code, for the first 2,000 customers, we route half to group A and the rest to group B. The group A customers are predicted with their likelihood of conversions by the XGBoost or GBDT model. The group B customers are predicted with their likelihood of conversions by the wide neural network model.

Then, we are going to examine the actual outcomes of these predictions. In the actual test setting, these outcomes may come days or weeks after the predictions were made, as it takes time for customers to decide whether to purchase or not. We are going to evaluate the actual conversions of those customers who are predicted to convert and the missed opportunities, which are the conversions of the customers who are predicted not to convert. Take a look at the following code:

```

def get_cumulative_metrics(actuals, preds):
    cum_conversions = []
    missed_opportunities = []

    customer_counter = 0
    cum_conversion_count = 0
    missed_opp_count = 0
    for actual, pred in zip(actuals, preds):
        customer_counter += 1
        if pred == 1:
            if actual == 1:
                cum_conversion_count += 1
        else:
            if actual == 1:
                missed_opp_count += 1
    cum_conversions.append(cum_conversion_count/customer_co
    missed_opportunities.append(missed_opp_count/customer_c
return cum_conversions, missed_opportunities

```

As you can see, we are counting the cumulative number of conversions for those customers who are predicted to convert and also counting the number of conversions for those customers who are predicted not to convert but have converted. These are essentially missed opportunities as we would not have sent our marketing messages to these customers but they would have converted. We are going to aggregate these results for each group, using the following code:

```

a_cum_conv_rates, a_missed_opp_rates = get_cumulative_metrics(
    group_a_actuals, group_a_preds
)
b_cum_conv_rates, b_missed_opp_rates = get_cumulative_metrics(
    group_b_actuals, group_b_preds
)
ab_results_df = pd.DataFrame({
    "group_a_cum_conversion_rate": a_cum_conv_rates,
    "group_a_cum_missed_opportunity_rate": a_missed_opp_rates,
}

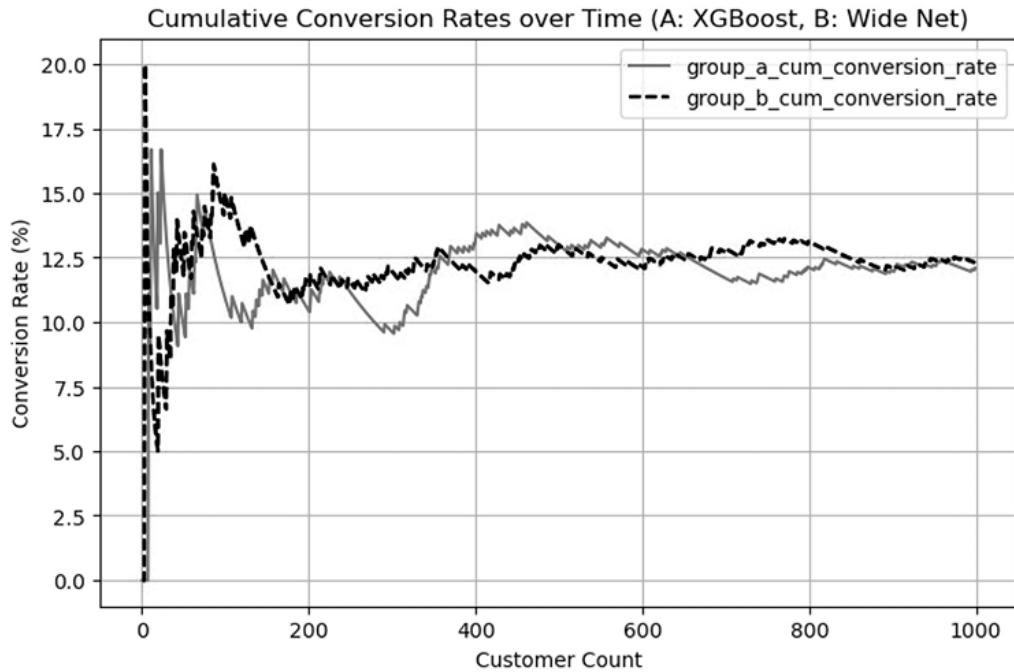
```

```
        "group_b_cum_conversion_rate": b_cum_conv_rates,
        "group_b_cum_missed_opportunity_rate": b_missed_opp_rates,
    })
```

Now, let's look at the cumulative conversion rates over time with the following code:

```
ax = (
    ab_results_df[[
        "group_a_cum_conversion_rate", "group_b_cum_conversion_
    ]]*100
).plot(
    style=['-', '--'],
    figsize=(8, 5),
    grid=True
)
ax.set_ylabel("Conversion Rate (%)")
ax.set_xlabel("Customer Count")
ax.set_title(
    "Cumulative Conversion Rates over Time (A: XGBoost, B: Wide
)
plt.show()
```

This will produce a chart that looks like the following:



*Figure 6.18: Cumulative conversion rates over time for groups A and B*

As this chart suggests, group B, who are treated with the wide neural network model predictions, shows overall higher conversion rates, compared to group A, who are treated with the XGBoost model predictions. Let's also take a look at how each group does for the missed opportunities with the following code:

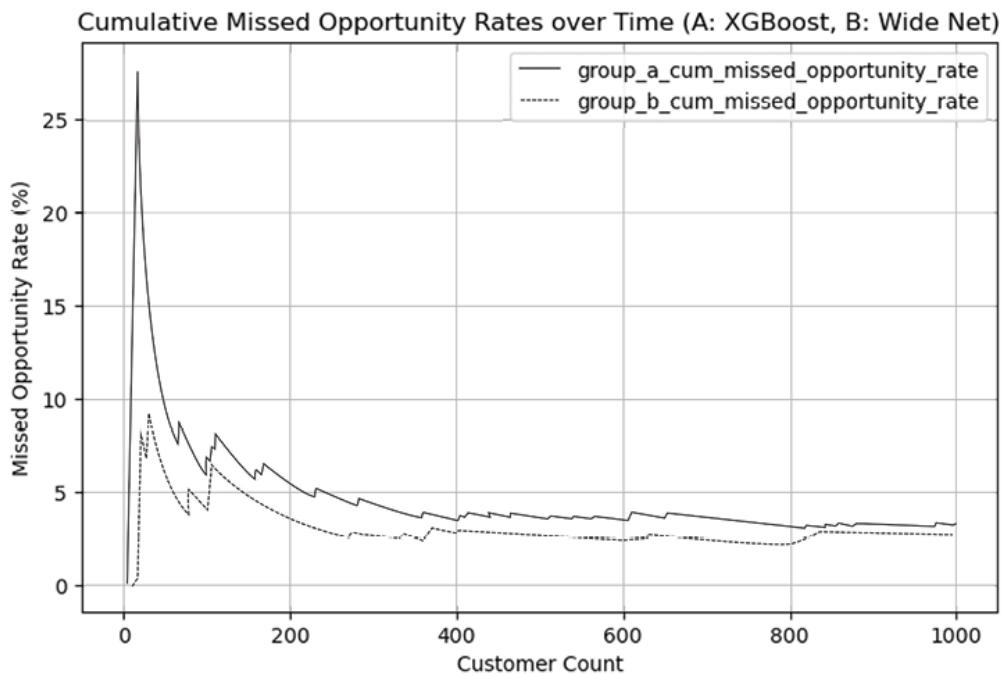
```

ax = (
    ab_results_df[[
        "group_a_cum_missed_opportunity_rate",
        "group_b_cum_missed_opportunity_rate"
    ]]*100
).plot(
    style=['-', '--'],
    figsize=(8, 5),
    grid=True
)
ax.set_ylabel("Missed Opportunity Rate (%)")
ax.set_xlabel("Customer Count")
ax.set_title(

```

```
"Cumulative Missed Opportunity Rates over Time (A: XGBoost,  
)  
plt.show()
```

When you run this code, you should get a chart that looks similar to the following:



*Figure 6.19: Cumulative missed opportunity rates over time for groups A and B*

This chart suggests that the missed opportunities are higher for group A with the XGBoost model predictions than group B with the wide neural network model predictions.

These visuals are a great way to examine the performances of different groups or models. However, to validate whether these differences are statistically significant, we will have to run the t-test.

# Two-tailed T-test

The `scipy` package in Python provides a handy tool to compute the t-value and p-value for a two-tailed t-test.



To install `scipy` on your machine, run the following command in your terminal:

```
pip install scipy
```

Once you have installed the `scipy` package, you can run the following commands to examine the statistical significance of the differences in the conversion and missed opportunity rates between groups A and B:

```
from scipy.stats import ttest_ind
t, p = ttest_ind(a_cum_conv_rates, b_cum_conv_rates)
print(
    f"Conversion Rate Difference Significance -- t: {t:.3f} & p: {p:.3f}"
)
t, p = ttest_ind(a_missed_opp_rates, b_missed_opp_rates)
print(
    f"Missed Opportunity Rate Difference Significance -- t: {t:.3f} & p: {p:.3f}"
)
```

As you can see from this code, the `ttest_ind` function from the `scipy.stats` module lets you easily get the t-value and p-value for the two-tailed t-test between two groups. The output of this code should look similar to the following:

```
Conversion Rate Difference Significance -- t: -13.168 & p: 0.000
Missed Opportunity Rate Difference Significance -- t: 2.418 & p: 0.016
```

*Figure 6.20: Two-tailed t-test results*

Let's take a closer look at this output. First, the t-value and p-value for the conversion rate t-test are `-13.168` and `0`. This suggests that the difference in the conversion rates between the two groups is statistically significant. The negative value of the t-test suggests that the mean of group A is smaller than that of group B.

Since from the p-value, we have concluded that the difference is significant, this translates to the result that group A's conversion rate is significantly lower than that of group B. In other words, the wide neural network model performs significantly better in capturing the high conversion likely customers than the XGBoost model.

Secondly, the t-value and p-value for the missed opportunity rate t-test are `2.418` and `0.016`. This suggests that the mean of group A is significantly larger than that of group B. This translates to the result that group A's missed opportunity is significantly larger than that of group B. In other words, the XGBoost model results in missing more opportunities than the wide neural network model.

As you can see from this A/B testing simulation results, A/B testing provides powerful insights into which model performs better in the real-world setting. Everyone building data science or AI/ML models should develop a habit of not only evaluating the models based on the train and test sets but also evaluating the models via A/B testing so that they can tell how realistic and applicable the models being built are in a real-world setting. Through a short period of small sample A/B testing, you can choose the optimal model to use for the upcoming marketing campaign.

# Summary

In this chapter, we have covered a lot about building predictive models using an Online Purchase dataset. We have explored two different tree-based models, random forest and GBDT, and how to build predictive models to forecast who is likely to convert. Using the same example, we have also discussed how we can build neural network models that are the backbone of deep learning models. There is great flexibility in how you architect the neural network model, such as wide network, deep network, or wide and deep network. We have briefly touched on the activation functions and optimizers while building neural network models, but we suggest you do some more in-depth research into how they affect the performances of neural network models. Lastly, we have discussed what A/B testing is, how to conduct A/B testing, and how to interpret the A/B testing results. We have simulated A/B testing with the models we built for a scenario where we want to choose the best model for capturing the most amount of customer conversions.

In the following chapter, we are going to expand further on targeted marketing using AI/ML. More specifically, we will discuss how to build personalized product recommendations in various ways and how this can lead to micro-targeted marketing strategies.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)

# 7

## Personalized Product Recommendations

With the advancements in technology and the rising amount of data being collected, personalization is everywhere. From streaming services, such as Netflix and Hulu, to marketing messages and advertisements you see on your phones, most of the content shown to you is personalized nowadays. In marketing, personalized or targeted marketing campaigns have proven to work significantly better for driving customer engagements and conversions compared to generic or mass marketing campaigns.

In this chapter, we are going to discuss building personalized recommendation models with which we can better target customers with the products that interest them the most. We will be examining how to conduct **market basket analysis** in Python, which helps marketers better understand which items are frequently bought together, how to build **collaborative filtering** algorithms in two approaches for personalized product recommendations, and what other approaches are taken for recommending products that are personalized to individual customers.

In this chapter, we will cover the following topics:

- Product analytics with market basket analysis
- User-based versus item-based collaborative filtering

- Other frequently used recommendation methods

## Product analytics with market basket analysis

The classic example of **market basket analysis** is that researchers have found that customers who buy diapers tend to buy beer as well. Although beer and diapers seem too distant items to go together, this finding shows that there are hidden associations between different products that we can find from data. The goal of market basket analysis is to find these hidden relationships between products and efficiently arrange for or recommend the products to customers. Market basket analysis helps answer questions like the following:

- Which products should be recommended to customers who are buying product X?
- What products should be close together in the store so that customers can easily find the products that they want to buy?

To answer these questions, association rules of the market basket analysis should be found. An association rule, simply put, shows how confidently or significantly an itemset occurs in a transaction using a rule-based machine learning approach. An association rule has two parts: the antecedent, which is the condition of a rule, and the consequent, which is the result of a rule. Consider the following statement:



*“Customers who buy diapers are likely to buy beer.”*

In this statement, diapers are the antecedent and beer is the consequent. Five metrics are used to evaluate the strengths of these association rules:

- **Support:** Support signals how frequently an itemset occurs in the dataset. The equation to calculate the support for an itemset  $(X, Y)$  is:

$$Support(X, Y) = \frac{Frequency\ of\ X\ \&\ Y}{Number\ of\ Transactions}$$

For example, if there are 3 transactions of diapers in all 10 transactions in the data, the support of diapers is  $3/10$  or 0.3. If there are 2 transactions of diapers and beer in all 10 transactions in the data, the support of diapers and beer is  $2/10$  or 0.2.

- **Confidence:** Confidence is a conditional probability of the itemset  $(X, Y)$  occurring given that the antecedent  $X$  has occurred. The equation to calculate the confidence for the consequent  $Y$  given the antecedent  $X$  is:

$$Confidence(X \rightarrow Y) = \frac{Frequency\ of\ X\ \&\ Y}{Frequency\ of\ X}$$

For example, if there are 3 transactions of diapers and 2 transactions of diapers and beer in all 10 transactions, the confidence of buying beer given diapers is  $2/3$  or 0.67. The confidence value will be 1 if the consequent always occurs with the antecedent.

- **Lift:** Lift is a metric to measure how much more often the itemset  $(X, Y)$  occurs together than if the antecedent and consequent were independent events. The equation for the lift for the antecedent  $X$  and consequent  $Y$  is:

$$Lift(X \rightarrow Y) = \frac{Confidence(X \rightarrow Y)}{Support(Y)}$$

For example, if the confidence of diapers and beer was 2/3 and the support of beer was 5/10, then the lift of diapers to beer would be 2/3 divided by 5/10, which is 20/15 or 4/3. If the antecedent and the consequent are independent, the lift score will be 1.

- **Leverage:** Leverage is a metric used to measure the difference between the frequency of the antecedent and the consequent occurring together and the frequency of the antecedent and the consequent if they were independent. The equation for the leverage is:

$$Leverage(X \rightarrow Y) = Support(X, Y) - Support(X) \times Support(Y)$$

For example, if diapers and beer occurred in 4 transactions out of 10 total transactions and the support of diapers and beer were 0.6 and 0.5 respectively, then the leverage will be  $0.4 - (0.6 * 0.5)$ , which is 0.1. Leverage values range between –1 and 1 and if the antecedent and the consequent were independent, the leverage value will be 0.

- **Conviction:** Conviction measures how much the consequent depends on the antecedent. The equation for the conviction is:

$$Conviction(X \rightarrow Y) = \frac{1 - Support(Y)}{1 - Confidence(X \rightarrow Y)}$$

For example, if the confidence between diapers and beer is 0.67 and the support of beer is 0.2, meaning there were 2 transactions of beer in 10 total transactions, the conviction would have been about 2.42. If the antecedent and the consequent are independent, then the conviction will be 1. On the other hand, if the confidence of the antecedent and consequent is 1 or if the

consequent always occurs together with the antecedent, then the denominator becomes 0 and the conviction value becomes infinite.

With this foundation, let's dive into finding the association rules within an e-commerce dataset. We will be using an online retail dataset and the `mlxtend` package to show how market basket analysis can be performed with Python.

### **Source code and data:**

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/tree/main/ch.7>

### **Data source:**

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/blob/main/ch.7/data.csv>



Here is the link for the original data source:

<https://archive.ics.uci.edu/dataset/352/online+retail>

The following is the command for installing the `mlxtend` package:

```
pip install mlxtend
```

# Apriori algorithm - finding frequent itemsets

The Apriori algorithm is used to identify and generate the association rules that we discussed previously. We will discuss how to run the Apriori algorithm for market basket analysis in Python with an example. Let's first load the data into a DataFrame using the following code:

```
import pandas as pd
df = pd.read_csv("./data.csv")
df = df.dropna(subset=['CustomerID'])
```

Once we run this code, we now have all the purchases that the customers made. If you recall what we have discussed previously about the market basket analysis, the two key components are finding the itemsets and finding the rules among the itemsets. In this section, we are going to focus on finding the itemsets, which are the combinations of items.

When you run the `df["StockCode"].nunique()` command, you will see that there are 3,684 unique items or products in this data. As you can imagine, the number of combinations of the items you can create grows exponentially with the number of items. The number of all possible combinations of items is:

$$\# \text{of itemsets} = 2^N - 1, \text{where } N \text{ is the number of items}$$

With over 3,000 items in our dataset, it will be unrealistic to check all the possible itemsets, as there are  $2^{3684}-1$  itemsets to check. This is where the **Apriori** algorithm comes in. Intuitively, the items or itemsets with low transaction frequency or low support are going to have less value in the

findings. Thus, the Apriori algorithm extracts the itemsets that are considered frequent enough by the predefined threshold of support.

To find the frequent itemsets using the Apriori algorithm, we need to convert the DataFrame in a way that it is a matrix where each row represents the customers and each column represents the items or products. Take a look at the following code:

```
customer_item_matrix = df.pivot_table(  
    index='CustomerID',  
    columns='StockCode',  
    values='Quantity',  
    aggfunc='sum'  
)
```

Here, we are using the `pivot_table` function to create a customer-to-item matrix and each value will be the total quantity of each item that each customer bought. With this customer-to-item matrix, we can run the Apriori algorithm to find the frequent itemsets using the following code:

```
from mlxtend.frequent_patterns import apriori  
frequent_items = apriori(  
    customer_item_matrix,  
    min_support=0.03,  
    use_colnames=True  
)  
frequent_items["n_items"] = frequent_items["itemsets"].apply(  
lambda x: len(x)  
)
```

As you can see from this code, we are using the `apriori` function within the `mlxtend` package. We have defined the minimum support required for each itemset to be `0.03` with the `min_support` parameter, which means we

are going to ignore those itemsets that occur in less than 3% of the transactions. Then, we add a column, `n_items`, which simply counts the number of items in each itemset.

The final DataFrame looks like the following:

	<b>support</b>	<b>itemsets</b>	<b>n_items</b>
<b>0</b>	0.044145	(15036)	1
<b>1</b>	0.032479	(15056BL)	1
<b>2</b>	0.042086	(15056N)	1
<b>3</b>	0.030878	(16156S)	1
<b>4</b>	0.043001	(16161P)	1
...	...	...	...
<b>1951</b>	0.034309	(20727, 22383, 20725, 20728, 22384)	5
<b>1952</b>	0.032022	(20727, 22383, 20725, 22382, 22384)	5
<b>1953</b>	0.033166	(22383, 20725, 22382, 20728, 22384)	5
<b>1954</b>	0.030192	(22383, 23207, 20725, 22382, 20728)	5
<b>1955</b>	0.031565	(20727, 22383, 22382, 20728, 22384)	5

1956 rows x 3 columns

*Figure 7.1: The frequent\_items DataFrame*

If you recall, the total number of possible combinations of itemsets was  $2^{3684}-1$ . However, as you can see from this result of applying the Apriori algorithm, we now have 1,956 items, which is a manageable and reasonable amount of itemsets that we can examine the association rules for.

# Association rules

With the frequent itemsets we have built, we can now generate the association rules that will tell us which itemsets are frequently bought together. Take a look at the following code:

```
from mlxtend.frequent_patterns import association_rules
rules = association_rules(
    frequent_items,
    metric="confidence",
    min_threshold=0.6,
    support_only=False
)
```

Here, we are using the `association_rules` function in the `mlxtend` package. There are a few key parameters to consider:

- `metric`: This parameter defines which metric to use for selecting the association rules. Here, we are using `confidence` to choose association rules with high confidence, but you can also use `lift` if you are interested in finding the rules with high lift.
- `min_threshold`: This parameter defines the threshold for the selection of association rules based on the metric you have chosen. Here, we are only interested in finding the rules with confidence above the 60% threshold.
- `support_only`: You can set this parameter to `True` if you are only interested in computing the support of the rules or if other metrics cannot be computed due to some data missing. Here, we set this to `False` as we are interested in finding the rules with high confidence.

The output of these rules should look like the following:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(20712)	(85099B)	0.056267	0.144556	0.038655	0.686992	4.752418	0.030521	2.732976	0.836657
1	(20713)	(85099B)	0.047118	0.144556	0.034309	0.721655	5.037176	0.027498	3.146811	0.841107
2	(20719)	(20724)	0.054209	0.076624	0.037283	0.687764	8.975830	0.033129	2.957299	0.939520
3	(20723)	(20724)	0.052150	0.076624	0.037740	0.723684	9.444619	0.033744	3.341742	0.943313
4	(20723)	(22355)	0.052150	0.056496	0.031336	0.600877	10.635770	0.028390	2.363944	0.955824
...	...	...	...	...	...	...	...	...	...	...
2019	(22383, 20727, 22384)	(22382, 20728)	0.043916	0.064273	0.031565	0.718750	11.182829	0.028742	3.327031	0.952403
2020	(22382, 20727, 20728)	(22383, 22384)	0.044145	0.055124	0.031565	0.715026	12.971341	0.029131	3.315657	0.965530
2021	(22382, 20728, 22383)	(20727, 22384)	0.047575	0.063586	0.031565	0.663462	10.434007	0.028539	2.782486	0.949324
2022	(22382, 20727, 22383)	(20728, 22384)	0.046661	0.064730	0.031565	0.676471	10.450634	0.028544	2.890834	0.948573
2023	(20727, 20728, 22383)	(22382, 22384)	0.044831	0.057868	0.031565	0.704082	12.166976	0.028970	3.183756	0.960888

2024 rows × 10 columns

*Figure 7.2: Association rules based on confidence above 60%*

The association rule output contains antecedents, consequents, and key metrics, such as `support`, `confidence`, and `lift`. You can sort by each metric and generate insights on which itemsets have close relationships. For example, you may want to find the top 20 itemsets that have the highest lift or the top 10 itemsets that have the highest confidence. You can also visualize these relationships so that you can easily view the relationships and the strengths of the relationships. Take a look at the following code:

```
most_lift = rules.sort_values(
    by="lift", ascending=False
).head(20).pivot_table(
    index='antecedents',
    columns='consequents',
    values='lift',
    aggfunc='sum'
)
most_lift.index = [
    " + ".join([
        df.loc[df["StockCode"] == item]["Description"].unique()
        for item in list(x)
    ]) for x in most_lift.index
]
most_lift.columns = [
    " + ".join([

```

```
        df.loc[df["StockCode"] == item]["Description"].unique()
        for item in list(x)
    )) for x in most_lift.columns
]
```

Here, we first select the top 20 rules with the highest lift by using the `sort_values` function. Then, we pivot the table so that each row is the antecedent, each column is the consequent, and the value of a cell is the lift. A heatmap can be used to visualize the strengths of these rules easily. The following code can be used to visualize these 20 rules with the highest lift:

```
ax = plt.subplot()
sns.heatmap(
    most_lift,
    annot=True,
    annot_kws={"size": 6},
    # fmt=".1f",
    ax=ax
)
ax.set_title("Top 20 Rules by Lift")
ax.set_xlabel("Consequent")
ax.set_ylabel("Antecedent")
ax.yaxis.set_ticklabels(list(most_lift.index), fontsize=6)
ax.xaxis.set_ticklabels(list(most_lift.columns), fontsize=6)
plt.show()
```

This will generate a heatmap that looks like the following:

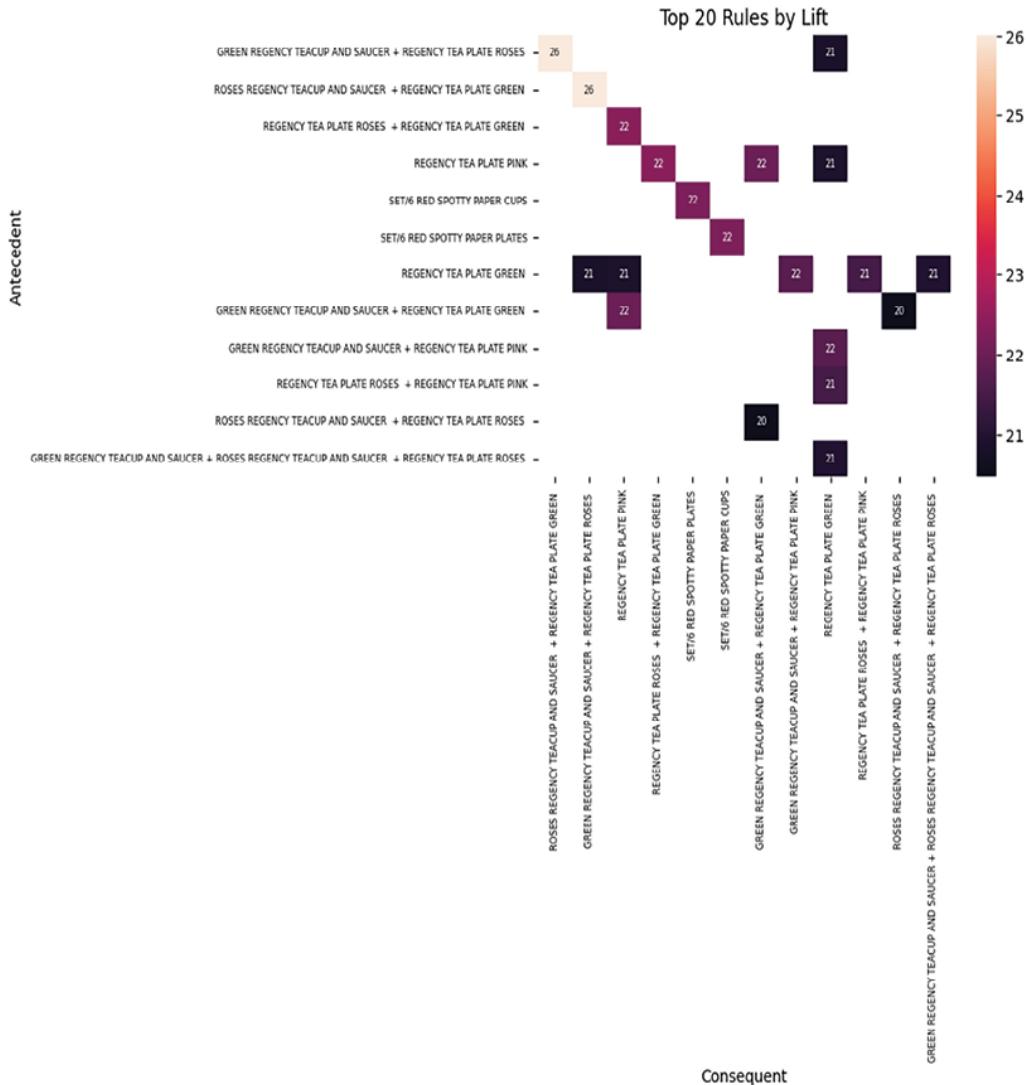


Figure 7.3: The top 20 association rules with the highest lift

In this chart, the lighter the color is, the higher the lift is for a given rule. For example, when the antecedent itemset is **GREEN REGENCY TEACUP AND SAUCER, REGENCY TEA PLATE ROSES** and the consequent itemset is **ROSES REGENCY TEACUP AND SAUCER, REGENCY TEA PLATE GREEN**, the lift is about 26 and the highest. On the other hand, when the antecedent itemset is **ROSES REGENCY TEACUP AND SAUCER, REGENCY TEA PLATE ROES** and the consequent itemset is **GREEN REGENCY TEACUP AND SAUCER, REGENCY TEA PLATE GREEN**, the lift is about 20 and the lowest among the top 20 rules.

As you can see from this example, with the association rules, we can easily find out which products are bought together and their relationships with one another. This can be used in personalized marketing, where certain products are recommended based on these rules. If a customer purchased itemset A, then naturally you can recommend itemset B, which has high confidence or lift as the association rule suggests that A and B are frequently bought together with significance. This is one step closer to recommending individualized product sets in marketing messages rather than blindly recommending random product sets in the hopes that customers express interest in purchasing them.

Lastly, if you recall, lift has a linear relationship with confidence and an inverse relationship with support, as lift is defined as:

$$Lift(X \rightarrow Y) = \frac{Confidence(X \rightarrow Y)}{Support(Y)}$$

This relationship can be easily shown from the rules we have found with the `mlxtend` package. Take a look at the following code:

```
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax2 = ax1.twiny()
rules[["support","lift"]].plot(
    kind="scatter", x="support", y="lift", ax=ax1
)
rules[["confidence","lift"]].plot(
    kind="scatter", x="confidence", y="lift", ax=ax2, color="orange"
)
ax1.legend(["lift vs. support"], loc="upper right")
ax2.legend(["lift vs. confidence"], loc="upper left")
plt.grid()
plt.show()
```

Here, we have lift values on the y-axis and support and confidence values on the x-axis. The chart will look as the following:

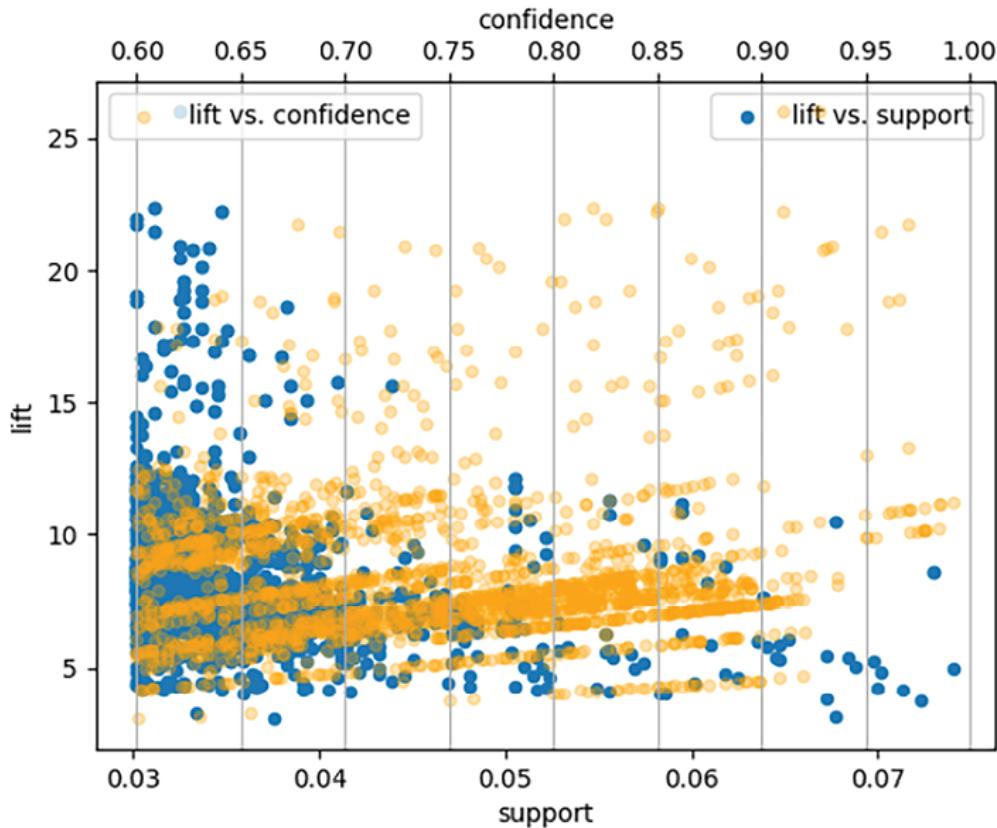


Figure 7.4: The relationships between lift and confidence and between lift and support

As you can see from this chart, the top x-axis shows the confidence values and the scatter plot of lift against confidence shows a linear relationship, where the lift values increase as the confidence values increase. On the other hand, the bottom x-axis shows the support values and the scatter plot of lift against support shows an inverse relationship, where the lift values decrease as the support values increase.

## Collaborative filtering

There are various approaches in building recommendation systems and often multiple approaches take part. Some of the frequently used AI/ML-driven approaches in building recommendation systems are:

- **Collaborative filtering:** This method uses previous user behaviors, such as pages they viewed, products they purchased, or ratings they have given previously. This algorithm leverages this kind of data to find similar products or content to those that the users have shown interest in previously. For example, if a user viewed a few thriller movies on a streaming platform, some other thriller movies may be recommended to this user. Or, if a customer bought dress shirts and dress shoes on an e-commerce platform, dress pants may be recommended to this customer. The collaborative filtering algorithms are often built based on the similarities between users or items:
  - **User-based** collaborative filtering uses data to find similar users based on the pages viewed or products purchased previously.
  - **Item-based** collaborative filtering uses data to find items that are often bought or viewed together.
- **Content-based filtering:** As the name suggests, this method uses the characteristics of products, contents, or users. Where the collaborative filtering algorithm focuses on user-to-user or item-to-item similarities, the content-based filtering algorithm focuses on characteristic similarities, such as genres, keywords, and metadata. For example, if you are recommending movies to watch, this approach may look at the descriptions, genres, or actors of movies and recommend those that match a user's preferences. On the other hand, if you are recommending fashion items, then this approach may look at the product categories, descriptions, and brands to recommend certain items to users.

- **Predictive modeling:** As discussed in *Chapter 6*, predictive models can be built for recommendation systems as well. Previous content that the users viewed, products that they have purchased, or web session history can be the features to identify the items that the users are highly likely to be interested in seeing or purchasing. The probability output of these predictive models can be used to rank the items so that more likely items are shown to the users before other less likely items.
- **Hybrid models:** Oftentimes, recommendation systems are built with all of the previously mentioned approaches. Blending different approaches for recommendations can help improve the recommendation accuracy that single-approach recommendation systems may miss.

In this chapter, we are going to focus on the *collaborative filtering* algorithm as it is a frequently used backbone of lots of recommender systems, and more specifically, how to build recommendation systems using user-based collaborative filtering and item-based collaborative filtering algorithms. This technique is frequently used in recommendation systems as it captures the interactions among the users and items very well in a very intuitive way. Lots of organizations where recommendations are critical in their businesses, such as Netflix, Amazon, and Spotify, use collaborative filtering algorithms as part of their recommendation systems.

The key to the collaborative filtering algorithms is to find similar users or items. There can be various metrics that can be used to measure the similarities between users or items, such as Euclidean distance, Manhattan distance, or Jaccard distance. However, cosine similarity is one of the most frequently used similarity metrics in collaborative filtering, as it focuses more on the directional similarity over the magnitude of the distance.

**Cosine similarity** is simply the cosine of the angle between two vectors.

Here, vectors can be user vectors or item vectors in our case of collaborative filtering algorithms. As cosine similarity has strength in high-dimensional space, it is often chosen as the distance metric for collaborative filtering algorithms. The equation for computing the cosine similarity between two vectors looks as follows:

$$\text{Cos}(V_1, V_2) = \frac{\sum_{i=1}^n V_{1i} V_{2i}}{\sqrt{\sum_{i=1}^n V_{1i}^2} \sqrt{\sum_{i=1}^n V_{2i}^2}}$$

In this equation,  $V_{1i}$  and  $V_{2i}$  represent each item in each vector, which can be a user vector or an item vector. These cosine similarity values range between -1 and 1. If the 2 vectors are similar, the cosine similarity value will be close to 1; if the 2 vectors are independent, then the cosine similarity value will be 0; if the 2 vectors are the opposite vectors, then the cosine similarity value will be close to -1.

To build collaborative filtering algorithms, we will first need to build the customer-to-item matrix. We will be using the same code that we have used for the market basket analysis, which looks like the following:

```
customer_item_matrix = df.pivot_table(  
    index='CustomerID',  
    columns='StockCode',  
    values='Quantity',  
    aggfunc='sum'  
)
```

As before, this matrix shows the quantities purchased for each item (column) by each customer (row). Instead of using the raw quantities of

items purchased, we are going to one-hot encode so that the values are 1 if a given customer bought a given item or 0 if not, as in the following code:

```
customer_item_matrix = customer_item_matrix.map(  
    lambda x: 1 if x > 0 else 0  
)
```

The `customer_item_matrix` matrix should look like the following:

StockCode	10002	10080	10120	10123C	10124A	10124G	10125	10133	10135	11001	...	90214Y	90214Z	BANK CHARGES	C2	CRUK	D	DOT	M	PADS	P
CustomerID																					
12346.0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12347.0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12348.0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12349.0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12350.0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 7.5: The customer-to-item matrix

As expected, each row represents whether a given user bought each item or not. This matrix will be the foundational matrix that we will use to build user-based collaborative filtering and item-based collaborative filtering algorithms in the following sections.

## User-based collaborative filtering

The first step to building a user-based collaborative filtering model is to build a matrix of similar users. Take a look at the following code:

```
from sklearn.metrics.pairwise import cosine_similarity  
user_user_sim_matrix = pd.DataFrame(  
    cosine_similarity(customer_item_matrix)  
)  
user_user_sim_matrix.columns = customer_item_matrix.index
```

```
user_user_sim_matrix['CustomerID'] = customer_item_matrix.index  
user_user_sim_matrix = user_user_sim_matrix.set_index('Customer
```

If you recall, the cosine similarity measure is one of the most frequently used similarity metrics for collaborative filtering algorithms. Here, we are using the scikit-learn package's `cosine_similarity` function to compute cosine similarities between users. The newly created variable, `user_user_sim_matrix`, will be a matrix where each row and column represents a user and the similarities between two users.

Take a look at the following example:

```
user_user_sim_matrix.loc[  
    [12347.0, 12348.0, 12349.0, 12350.0, 12352.0],  
    [12347.0, 12348.0, 12349.0, 12350.0, 12352.0]  
]
```

CustomerID	12347.0	12348.0	12349.0	12350.0	12352.0
CustomerID					
12347.0	1.000000	0.063022	0.046130	0.047795	0.038814
12348.0	0.063022	1.000000	0.024953	0.051709	0.027995
12349.0	0.046130	0.024953	1.000000	0.056773	0.138314
12350.0	0.047795	0.051709	0.056773	1.000000	0.031846
12352.0	0.038814	0.027995	0.138314	0.031846	1.000000

*Figure 7.6: A sample of user-to-user similarities*

This example shows the user-to-user cosine similarity metrics. As expected, the diagonal values are 1 since they represent the similarities between a user and itself. For example, users `12349` and `12350` have a cosine similarity of about `0.057` and users `12349` and `12352` have a cosine similarity of `0.138`.

This suggests that user 12349 is more similar to user 12352 than to user 12350. This way, you can identify and rank the users by how similar they are to a target user.

Let's pick one customer and see how we may be able to build a recommendation system. We will select the user with ID 14806 as an example for this exercise:

```
TARGET_CUSTOMER = 14806.0
print("Similar Customers to TARGET")
user_user_sim_matrix.loc[
TARGET_CUSTOMER
].sort_values(ascending=False).head(10)
```

Here, we select the top 10 users that have the highest cosine similarity measures, by selecting TARGET\_CUSTOMER in the user\_user\_sim\_matrix matrix and sorting the values in descending order. The output should look like the following:

```
Similar Customers to TARGET

CustomerID
14806.0    1.000000
13919.0    0.251478
12561.0    0.226134
13711.0    0.181818
12618.0    0.181818
13116.0    0.165145
12497.0    0.161165
17475.0    0.161165
12372.0    0.155126
14840.0    0.150756
Name: 14806.0, dtype: float64
```

*Figure 7.7: Similar customers to the target customer*

Here, you can see that customer `13919` is the most similar to the target customer, `14806`, customer `12561` comes second, and customer `13711` follows in third.

## **Recommending by the most similar customer**

The simplest approach to recommend products to the target customer is to see which items the target customer has bought already and compare them against the most similar customer, `13919`. Then, recommend the products that the target customer has not bought yet that the most similar customer has bought. This is based on the assumption, as discussed previously, that customers similar to each other are likely to behave similarly or share similar interests and they are likely to purchase similar products.

To do this, we can follow these steps:

1. We will first find out the items that the target customer has bought already, using the following code:

```
items_bought_by_target = set(  
    df.loc[  
        df["CustomerID"] == TARGET_CUSTOMER  
    ]["StockCode"].unique()  
)
```

This will get all the items that the target customer has bought and store them in the `items_bought_by_target` variable.

2. Next, we need to get all the items the most similar customer, `13919`, has bought. Take a look at the following code:

```
items_bought_by_sim = set(  
    df.loc[  
        df["CustomerID"] == 13919.0  
    ]["StockCode"].unique()  
)
```

This finds out all the items that the most similar customer, `13919`, has bought and stores them in the `items_bought_by_sim` variable.

3. Using the `set` operation, it is easy to find out the items that the most similar customer has bought but not bought by the target customer, as shown in the following code:

```
items_bought_by_sim_but_not_by_target = items_bought_by_sim - items_bought_by_target
```

As you can see, we simply subtract the `items_bought_by_target` set from the other set, `items_bought_by_sim`. This will get all the items that the target customer has not yet bought but were bought by the most similar customer and store them in `items_bought_by_sim_but_not_by_target`.

4. We can get the details about these items using the following code:

```
df.loc[
    df["StockCode"].isin(items_bought_by_sim_but_not_by_target)
][["StockCode", "Description"]].drop_duplicates()
```

This output should look like the following:

<b>StockCode</b>		<b>Description</b>
<b>39</b>	21731	RED TOADSTOOL LED NIGHT LIGHT
<b>45</b>	POST	POSTAGE
<b>105</b>	22961	JAM MAKING SET PRINTED
<b>1217</b>	21498	RED RETROSPOT WRAP
<b>1296</b>	22666	RECIPE BOX PANTRY YELLOW DESIGN
<b>1299</b>	22567	20 DOLLY PEGS RETROSPOT
<b>2116</b>	22847	BREAD BIN DINER STYLE IVORY
<b>5016</b>	16161P	WRAP ENGLISH ROSE
<b>9525</b>	16161U	WRAP SUKI AND FRIENDS
<b>9780</b>	22847	BREAD BIN, DINER STYLE, IVORY
<b>37655</b>	22707	WRAP MONSTER FUN
<b>41750</b>	22046	TEA PARTY WRAPPING PAPER
<b>58513</b>	23233	WRAP POPPIES DESIGN
<b>269898</b>	23546	WRAP PAISLEY PARK
<b>272071</b>	23547	WRAP FLOWER SHOP
<b>273724</b>	16169E	WRAP 50'S CHRISTMAS
<b>353509</b>	23511	EMBROIDERED RIBBON REEL EMILY
<b>354136</b>	23512	EMBROIDERED RIBBON REEL ROSIE
<b>354137</b>	23515	EMBROIDERED RIBBON REEL DAISY
<b>354140</b>	23518	EMBROIDERED RIBBON REEL RACHEL
<b>354254</b>	23513	EMBROIDERED RIBBON REEL SUSIE

Figure 7.8: Items to recommend to the target customer

This is the list of items that the most similar customer, 13919, has bought but have not been bought by the target customer, and their descriptions. In the most simple approach to recommending products, this list can be shown to the target customer as the recommended items.

Since these two customers have shown similar interests by having bought similar items in the past, it is more likely for the target customer to purchase some of these items rather than a random selection of products. You can apply the same process for each customer and build sets of products to recommend based on the most similar customers' historical purchases.

## Recommending by the top products bought by similar customers

Another approach is to rank the products by how frequently they were bought by the most similar customers.

We can start with the following code:

```
top10_similar_users = user_user_sim_matrix.loc[
    TARGET_CUSTOMER
].sort_values(
    ascending=False
).head(11).to_dict()
potential_rec_items = {}
for user, cos_sim in top10_similar_users.items():
    if user == TARGET_CUSTOMER:
        continue

    items_bought_by_sim = list(set(
        df.loc[
            df["CustomerID"] == user
        ]["StockCode"].unique()
    ))
    for each_item in items_bought_by_sim:
```

```
        if each_item not in potential_rec_items:
            potential_rec_items[each_item] = 0
            potential_rec_items[each_item] += cos_sim
potential_rec_items = [(key, val) for key, val in potential_rec_items.items()]
potential_rec_items = sorted(
    potential_rec_items, key=lambda x: x[1], reverse=True
)
```

Here, we first get the top 10 most similar customers by the cosine similarity. Then, for each similar customer, we iterate through the items that were bought by the similar customer but not by the target customer. We count the number of times similar customers bought the given product, sort by the purchase frequency in the reverse order, and store it in the `potential_rec_items` variable. The result should look like the following:

`potential_rec_items[:10]`

```
[('21500', 7),
 ('21499', 7),
 ('21498', 4),
 ('POST', 4),
 ('22704', 3),
 ('16161U', 3),
 ('22138', 3),
 ('22711', 2),
 ('22631', 2),
 ('22326', 2)]
```

Figure 7.9: Top products purchased by similar customers

This suggests that 7 out of 10 customers have bought the `21500` and `21499` products, 4 out of 10 customers have bought the `21498` and `POST` products,

and so forth. We can query to get the product descriptions from these product IDs using the following code:

```
top_10_items = [x[0] for x in potential_rec_items[:10]]
df.loc[
    df["StockCode"].isin(top_10_items)
][["StockCode", "Description"]].drop_duplicates().set_index(
    "StockCode"
).loc[top_10_items]
```

This output should look like the following:

StockCode	Description
21500	PINK POLKADOT WRAP
21499	BLUE POLKADOT WRAP
21498	RED RETROSPOT WRAP
POST	POSTAGE
22704	WRAP RED APPLES
16161U	WRAP SUKI AND FRIENDS
22138	BAKING SET 9 PIECE RETROSPOT
22711	WRAP CIRCUS PARADE
22631	CIRCUS PARADE LUNCH BOX
22326	ROUND SNACK BOXES SET OF4 WOODLAND

Figure 7.10: Top products purchased by similar customers

This approach, however, does not account for how similar or dissimilar each customer was to the target customer. It just counted the number of top 10 similar customers for each product and recommended based on the simple sum.

Since we have similarity measures for each of the top 10 similar customers, we may also want to consider that when we recommend products. Instead of simple counts, we can do weighted counts. Take a look at the following code:

```
potential_rec_items = []
for user, cos_sim in top10_similar_users.items():
    if user == TARGET_CUSTOMER:
        continue

    items_bought_by_sim = list(set(
        df.loc[
            df["CustomerID"] == user
        ]["StockCode"].unique()
    ))
    for each_item in items_bought_by_sim:
        if each_item not in potential_rec_items:
            potential_rec_items[each_item] = 0
            potential_rec_items[each_item] += cos_sim
potential_rec_items = [(key, val) for key, val in potential_rec_items.items()]
potential_rec_items = sorted(
    potential_rec_items, key=lambda x: x[1], reverse=True
)
```

As you may notice, the only part that is different from the previous code is `potential_rec_items[each_item] += cos_sim`. Instead of simply counting, we are now adding the cosine similarity metric that measures how similar or dissimilar each customer is to the target customer.

When we query the top 10 items weighted by the cosine similarity, the results look as in the following:

StockCode	Description
21500	PINK POLKADOT WRAP
21499	BLUE POLKADOT WRAP
21498	RED RETROSPOT WRAP
POST	POSTAGE
22704	WRAP RED APPLES
16161U	WRAP SUKI AND FRIENDS
22138	BAKING SET 9 PIECE RETROSPOT
22711	WRAP CIRCUS PARADE
22631	CIRCUS PARADE LUNCH BOX
22326	ROUND SNACK BOXES SET OF4 WOODLAND

Figure 7.11: Top products purchased by similar customers (weighted by cosine similarity)

In this example, the output from the simple count is the same as the output from the weighted count. Not only can you do the weighted sum of cosine similarities but you can also do the weighted average. There are many other ways you can aggregate the scores and recommend products, so being creative is the key to building the final recommendation output from the cosine similarity measures.

As you can see from these examples, we can build product sets to recommend based on the customers who have shown similar behaviors in the past. Oftentimes, people with similar tastes buy similar items, so this user-based collaborative filtering works well when recommending products based on the similarities among your customer base. On top of recommending based on customer similarities, we can also recommend based on how similar individual items are to others. We are going to discuss how to build recommendation systems based on item-based collaborative filtering in the following section.

## Item-based collaborative filtering

As in the case of user-based collaborative filtering, the key starting point of building an item-based collaborative filtering algorithm is to build an item-to-item similarity matrix. From the customer-item matrix we built previously, we can use the following code to build an item-to-item similarity matrix:

```
item_item_sim_matrix = pd.DataFrame(  
    cosine_similarity(customer_item_matrix.T)  
)
```

As shown in this code, we transpose the customer-item matrix, `customer_item_matrix`, first, which will make it an item-to-customer matrix where each row is an item, each column is a customer, and each value is 0 or 1 representing whether a given item is bought by a given column. Then, we compute cosine similarity by using the `cosine_similarity` function and

save the results as a pandas DataFrame in an `item_item_sim_matrix` variable.

The results should look like the following:

StockCode	10002	10080	10120	10123C	10124A	10124G	10125	10133	10135	11001	...	90214Y	90214Z	BANK CHARGES
StockCode														
10002	1.000000	0.000000	0.094868	0.091287	0.0	0.000000	0.090351	0.063246	0.098907	0.095346	...	0.000000	0.0	0.000000
10080	0.000000	1.000000	0.000000	0.000000	0.0	0.000000	0.032774	0.045883	0.047836	0.000000	...	0.000000	0.0	0.000000
10120	0.094868	0.000000	1.000000	0.115470	0.0	0.000000	0.057143	0.060000	0.041703	0.060302	...	0.000000	0.0	0.000000
10123C	0.091287	0.000000	0.115470	1.000000	0.0	0.000000	0.164957	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000
10124A	0.000000	0.000000	0.000000	0.000000	1.0	0.447214	0.063888	0.044721	0.000000	0.000000	...	0.000000	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
D	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000
DOT	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.104257	0.150756	...	0.000000	0.0	0.000000
M	0.059423	0.017244	0.075165	0.000000	0.0	0.000000	0.075165	0.067648	0.054855	0.101983	...	0.000000	0.0	0.071307
PADS	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.050000	0.000000	0.000000	...	0.000000	0.0	0.000000
POST	0.070057	0.000000	0.000000	0.000000	0.0	0.000000	0.071209	0.016615	0.028871	0.050098	...	0.031976	0.0	0.017514

3684 rows × 3684 columns

Figure 7.12: Item-to-item similarity matrix

## Recommending by the most similar items

The simplest approach to generating recommendations based on item similarities is to find the most similar items for a given item and recommend them. When a person is viewing a certain product and you want to show related or similar items, this approach can be used. For example, when you search for a product on Amazon and click on one of the products shown on the search page, Amazon will show related or similar items at the bottom of the page that are often bought together. This is where item-based collaborative filtering can be used and, more specifically, when you want to recommend products based on one given product.

Using our example dataset, let's assume a customer is looking at a product, `MEDIUM CERAMIC TOP STORAGE JAR`, which has a product code of `23166`.

From the item-to-item matrix, `item_item_sim_matrix`, that we have just built, we can find the top 10 most similar items with the following code:

```
most_similar_items = item_item_sim_matrix.loc[
    "23166"
].sort_values(ascending=False).head(10)
```

As you can see in this code, we are selecting the top 10 most similar items by ordering the items by the cosine similarity in descending order. The output should look like the following:

```
StockCode
23166    1.000000
23165    0.672199
23167    0.659394
22993    0.277381
23307    0.264449
22722    0.251841
23243    0.249567
22666    0.249185
22720    0.248604
22961    0.244451
Name: 23166, dtype: float64
```

Figure 7.13: Top 10 similar items to product 23166

We can also query the descriptions based on these product codes as in the following:

```
rec_items = [
    x for x in most_similar_items.index if x != "23166"
]
```

```

df.loc[
    df['StockCode'].isin(rec_items),
    ['StockCode', 'Description']
].drop_duplicates().set_index('StockCode').loc[rec_items]

```

As you can see, we are excluding product 23166 as it is the target item or the item that the customer is currently viewing and showing the rest of the 10 most similar items. The output should look like the following:

StockCode	Description
23165	LARGE CERAMIC TOP STORAGE JAR
23167	SMALL CERAMIC TOP STORAGE JAR
22993	SET OF 4 PANTRY JELLY MOULDS
23307	SET OF 60 PANTRY DESIGN CAKE CASES
22722	SET OF 6 SPICE TINS PANTRY DESIGN
23243	SET OF TEA COFFEE SUGAR TINS PANTRY
22666	RECIPE BOX PANTRY YELLOW DESIGN
22720	SET OF 3 CAKE TINS PANTRY DESIGN
22961	JAM MAKING SET PRINTED

Figure 7.14: Top 10 items similar to product 23166, including itself

As you can imagine, we can show these 10 items, including itself, on the page while the customer is viewing the target product. This way, we are showing the most similar or the most frequently bought together items so that the customer has more options to choose from or more items to purchase.

# Recommending by the purchase history

Expanding on the previous example of recommending products based on a single item, we can also build an item-based collaborative filtering recommendation system for multiple items. Especially when you may be sending out marketing emails or newsletters, you may want to send product recommendations along with them. In this case, it will be useful to consider the purchase history of customers, such as what kinds of products they have purchased, what product pages they have viewed online, or what products they have put in the cart. Using this information, we can build personalized product recommendations that are different for each customer using the item-based collaborative filtering algorithm.

Using our example dataset as an example, let's assume a customer has bought three items with product codes `23166`, `22720`, and `23243`. These items on our dataset are `SET OF 3 CAKE TINS PANTRY DESIGN, MEDIUM CERAMIC TOP STORAGE JAR`, and `SET OF TEA COFFEE SUGAR TINS PANTRY`. You can get the top 10 most similar items for these items using the following code:

```
item_item_sim_matrix[[
    "23166", "22720", "23243"
]].mean(axis=1).sort_values(
    ascending=False
).head(10)
```

As you can see from this code, we query the item-to-item matrix for those three items that the customer has bought and take the mean of the cosine similarities. Then, we sort these average values in descending order to get

the top 10 most similar items to the given 3 items. The output should look like the following:

```
StockCode
23243    0.593115
22720    0.592794
23166    0.499390
22722    0.417434
23165    0.403306
22666    0.369518
23167    0.364973
23245    0.322518
22993    0.320600
22961    0.305328
dtype: float64
```

Figure 7.15: Top 10 similar items to the given 3 items

Excluding the items that were already bought, product 22722 comes first and is the most similar item to the given itemset, and 23165 follows the second. We can query the data to get the product descriptions of these top 10 similar items. Take a look at the following code:

```
rec_items = [
    x for x in item_item_sim_matrix[
        "23166", "22720", "23243"
    ].mean(axis=1).sort_values(ascending=False).head(13).index
    if x not in ["23166", "22720", "23243"]
]
df.loc[
    df['StockCode'].isin(rec_items),
    ['StockCode', 'Description']
].drop_duplicates().set_index('StockCode').loc[rec_items]
```

Here, we exclude the 3 items that the customer has already bought from the recommendation list and retrieve the top 10 items that are the most similar or the most frequently bought together with these 3 items. The output of this query should look like the following:

StockCode	Description
22722	SET OF 6 SPICE TINS PANTRY DESIGN
23165	LARGE CERAMIC TOP STORAGE JAR
22666	RECIPE BOX PANTRY YELLOW DESIGN
23167	SMALL CERAMIC TOP STORAGE JAR
23245	SET OF 3 REGENCY CAKE TINS
22993	SET OF 4 PANTRY JELLY MOULDS
22961	JAM MAKING SET PRINTED
22960	JAM MAKING SET WITH JARS
23198	PANTRY MAGNETIC SHOPPING LIST
23307	SET OF 60 PANTRY DESIGN CAKE CASES

Figure 7.16: Top 10 similar items to the given 3 items

As you can imagine, these product recommendations based on the item-based collaborative filtering algorithm results can be used for any marketing campaign. If you are building an email marketing campaign, you can run this item-based collaborative filtering algorithm to select the top similar products based on previous purchases, page views, or cart items and

include them as part of the marketing emails. You can also have pop-up windows on web pages for when this user logs into your online page next time. Giving discounts or some other promotions on these items can also result in higher chances of conversions.

There are numerous ways you can apply user-based and item-based collaborative filtering for recommendation systems. It will be wise to keep track of how your recommendations perform and tune how you use them as you progress with your recommendation systems. You will most likely want to track, of those items that you have recommended, how many of them have converted.

As discussed in *Chapter 2* when we discussed KPIs, tracking conversion and other KPIs among the recommended items can tell you the effectiveness of your recommendation systems and which approach works better.

## **Other frequently used recommendation methods**

We have discussed the market basket analysis and collaborative filtering in depth for building personalized recommendation systems. However, there are various other ways these recommendation systems can be built. As previously mentioned, some of the common AI/ML-based approaches are association rules and collaborative filtering algorithms, which we have covered in this chapter; predictive modeling approaches are often used as well, and nowadays a hybrid of all these approaches is a typical method of building more comprehensive recommendation systems.

Not only are there AI/ML-driven approaches for recommendation systems but there can be various other ways to recommend products or content without even using AI/ML. The following are some common methods used for making recommendations:

- **Bestsellers or top views:** As the name suggests, recommendations based on the bestselling products or the most frequently viewed content are still used frequently. This helps new users or customers acquaint themselves with your products or platforms by easily viewing what others view or purchase the most on your platform. These can also be a great way to build your introductory marketing content.
- **Trending:** Similar to bestsellers, trending items show what is rising in popularity at a given moment. There can be some events, such as disasters in certain regions, breaking news at certain moments, or holiday events happening around the neighborhoods, that spike customer interest sporadically. It's wise to adjust for these events as these special events trigger certain customer behaviors and can be great marketing opportunities.
- **New arrivals:** As marketers, you most likely would not want to miss an opportunity to promote new products. Customers, existing or new, are often attracted to new arrival items, and this can be a great marketing opportunity to retain existing customers, as well as gain new customers. By recommending new arrivals to customers who have shown interest in similar items, the marketing campaign can result in great success.
- **Promotions:** Not only can you recommend products or content by their popularity or relevance but you can also recommend products or content that are going on promotion. Customers are often attracted to special deals and you would want to expose and market special

promotions as actively as possible. These are great ways to clear out overstocked inventories, raise brand awareness, bring in new customers, and retain inactive customers.

As you can see, there can be various ways you can build recommendation systems on top of utilizing AI/ML-driven methodologies. Given the competitive landscape and abundance of marketing campaigns basically by all businesses around the world, using only one approach in recommending products or content is not going to be that successful. The more comprehensive recommendation system you have, the more successful you will be in attracting and retaining customers. When you are designing recommendation systems, it would be wise to consider all the approaches mentioned in this chapter and apply them at the right moments and places.

## Summary

In this chapter, we discussed building personalized product recommendation systems. First, we discussed how to identify interrelationships between products by analyzing which itemsets are frequently bought together. We covered how to conduct market basket analysis in Python using the Apriori algorithm and association rules. Then, we dove deep into one of the AI/ML-driven approaches to building recommendation systems. We saw how user-based and item-based collaborative filtering algorithms can be used to identify similar users or items and products that are frequently bought together. In turn, these findings can then be used to recommend certain products that other customers are highly likely to be interested in and purchase. Lastly, we discussed various other approaches that can be used for recommending

products and content. We showed how combining non-AI/ML approaches can result in a more comprehensive and diverse recommendation system.

In the following chapter, we will continue our discussion around personalized and targeted marketing efforts. More specifically, we are going to cover how customer segmentations can be done in Python and why having an in-depth understanding of different customer segments within your customer base helps and affects the successes and failures of your marketing efforts.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](https://OceanofPDF.com)

# 8

## Segmenting Customers with Machine Learning

With the rising availability of data on customer characteristics and behaviors, it has become more accessible to approach customers with more informed insights. From analyzing the drivers behind customer engagements that we discussed in *Chapter 3* to understanding which specific products that individual customers may like, which we touched on in *Chapter 7*, the assumptions behind these approaches were based on the fact that there are certain groups of similar customers that behave in similar fashions.

Targeted marketing approaches are proven to work significantly better than mass marketing, due to which **customer segmentation** has been a frequently discussed topic in this domain. Also, with the upcoming cookieless world, first-party data is expected to play an even more critical role. Consequently, targeted strategies based on customer segments, such as geographic segments, demographic segments, or interest topic segments that will still be available without browser cookies, are going to be critical for success.

In this chapter, we will cover the following topics:

- One-time versus repeat customers

- Customer segmentation with K-means clustering and purchase behaviors
- Customer segmentation with **large language models (LLMs)** and product interests

## One-time versus repeat customers

According to *BIA Advisory Services*, which provides some very clear metrics, repeat customers spend 67% more than new customers on average. Also, a survey conducted by *BIA Advisory Services* says that over half of the surveyed businesses' revenue comes from repeat customers rather than new customers. This signifies the fact that retaining existing customers is as important as growing the customer base. However, businesses often sacrifice customer service to gain new customers.



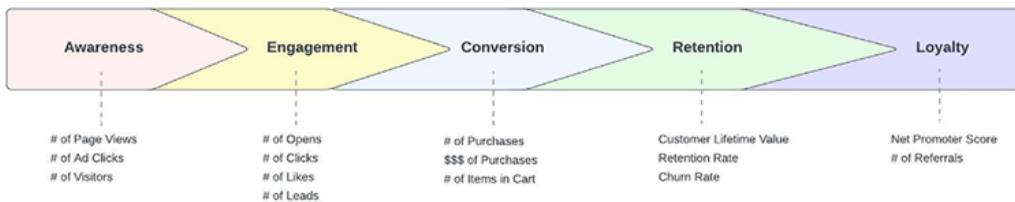
**BIA Advisory Services report**

<https://www.bia.com/small-business-owners-shift-investment-from-customer-acquisition-to-customer-engagement-new-report-by-manta-and-biakelsey/>

## The need to retain customers

There are several benefits that indicate why retaining customers is so important to businesses:

- **Less investment:** The first obvious reason is that new customers cost more. If you recall a typical customer life cycle from *Chapter 2*, you need to spend capital and marketing resources to promote brand awareness among potential new customers, engage prospects with your business, and then finally convert them into paying customers.



*Figure 8.1: The customer life cycle*

For existing customers, you can skip these steps and focus your efforts on retaining them as repeat customers by providing great customer service and introducing products that interest them enough. Obtaining new customers often costs multiple times more than keeping existing customers.

- **Greater reliability:** Repeat customers bring more reliable and recurring revenue to your business. As previously mentioned, repeat customers often contribute more than half of the revenue for businesses and spend more than new customers. As you provide products or services that existing customers like and as you provide better customer services, your customers may become loyal to your brand and continue to purchase products.

For example, if you are selling pet products, such as pet food or pet toys, and they like your products, they may come back next month to purchase more or even subscribe to get monthly pet foods delivered to them. This results in a reliable and recurring revenue stream that is going to strengthen the cash flow of your business, which you can use to invest more into your

products, which will bring in more revenue. This starts the positive cycle for your business.

- **Building brand loyalty:** Repeat customers who are loyal to your business bring in more new customers. As you may recall from *Chapter 2*, loyal customers act as your brand ambassadors and marketing agents, where they spread the word about your business and bring in new customers. Without having to spend more marketing dollars, these loyal customers promote your business and attract new customers.

There are many more subtle benefits to having a strong repeat customer base, but these are the three most obvious benefits of having repeat customers, and demonstrate how they help businesses significantly.

## Analyzing the impact of retaining customers

Let's take a look at a practical example of the kinds of impacts these repeat customers have on the business compared to new customers:



**Source code and data:**  
<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/tree/main/ch.8>

**Data source:**  
<https://archive.ics.uci.edu/dataset/352/online+retail>

As always, we will start by importing the data into a `pandas` DataFrame.

Take a look at the following code:

```
import pandas as pd
df = pd.read_csv("./data.csv")
df = df.dropna()
df = df.loc[
    df["Quantity"] > 0
]
```

Here, we load the data into a `pandas` DataFrame, `df`. As we are only interested in comparing new versus repeat customers, we are going to drop the rows with `NaN` values using the `dropna` function and only take the customers who have purchased at least one or more items by filtering with `df["Quantity"] > 0`.

1. Then, we are going to create the following additional variables:
  - **Sales amount:** By simply multiplying the quantity that the customers have purchased by the individual price of the items, we can get the total sales amount for each order. In the following code, we create a new column named `Sales`, which has the total sales amount for each order:

```
df["Sales"] = df["Quantity"] * df["UnitPrice"].
```

- **Month variable:** In order to decide whether a given customer is a new customer or not, we need to consider the time horizon of the data. If a given customer has not purchased any items previously, then this customer will be considered new. On the other hand, if a given customer has purchased an item previously, then we will consider this customer a repeat customer. For this exercise, we are going to look at

month-over-month data, which will require us to create a variable for what month the invoice was created. In the following code, we convert the column, `InvoiceDate`, into the `datetime` type and cast `InvoiceDate` into each month. For example, the date `2011-02-23` will be cast to `2011-02-01`:

```
df["InvoiceDate"] = pd.to_datetime(df["InvoiceDate"])
df["month"] = df["InvoiceDate"].dt.strftime("%Y-%m-01")
```

- With these two variables, we can now start dissecting whether the sales are from the new or repeat customers. Take a look at the following code:

```
monthly_data = []
for each_month in sorted(df["month"].unique()):
    up_to_last_month_df = df.loc[
        df["month"] < each_month
    ]
    this_month_df = df.loc[
        df["month"] == each_month
    ]
    curr_customers = set(this_month_df["CustomerID"].unique())
    prev_customers = set(up_to_last_month_df["CustomerID"])

    repeat_customers = curr_customers.intersection(prev_customers)
    new_customers = curr_customers - prev_customers

    curr_sales = this_month_df["Sales"].sum()

    sales_from_new_customers = this_month_df.loc[
        this_month_df["CustomerID"].isin(new_customers)
    ]["Sales"].sum()
    sales_from_repeat_customers = this_month_df.loc[
        this_month_df["CustomerID"].isin(repeat_customers)
    ]["Sales"].sum()

    avg_sales_from_new_customers = this_month_df.loc[
```

```
        this_month_df["CustomerID"].isin(new_customers)
    ]["Sales"].mean()
avg_sales_from_repeat_customers = this_month_df.loc[
    this_month_df["CustomerID"].isin(repeat_customers)
]["Sales"].mean()

monthly_data.append({
    "month": each_month,

    "num_customers": len(curr_customers),
    "repeat_customers": len(repeat_customers),
    "new_customers": len(new_customers),

    "curr_sales": curr_sales,
    "sales_from_new_customers": sales_from_new_customer
    "sales_from_repeat_customers": sales_from_repeat_cu
    "avg_sales_from_new_customers": avg_sales_from_new_
    "avg_sales_from_repeat_customers": avg_sales_from_r
})
```

Let's take a closer look at this code. We first iterate through each month in the `month` variable. For each iteration, we find the unique customers in the given month and store them as a set into a variable, named `curr_customers`. We do the same for the past customers by getting the unique customers up to the given month and storing them as a set in a variable named `prev_customers`. Based on these two variables, we can identify the new customers from the repeat customers by using some set operations:

1. First, we find the intersection between `curr_customers` and `prev_customers`, which represent the repeat customers as we have seen these customers in our sales data already.
2. Next, we subtract the `prev_customers` set from the `curr_customers` set, which gives us the new customers as these are the customers that we have not seen before. From these operations, we have successfully identified new versus repeat customers.

Based on these `prev_customers` and `curr_customers` sets, we can find the revenue from the new and repeat customers. Using the `isin` function, we select the `CustomerIDs` that match the IDs in the set of `new_customers` and compute the total sales from the new customers by summing all the sales amounts from these customers and the average sales amount for the new customers by taking a mean of all the sales amounts from these customers. Similarly, using the `isin` function, we select the `customerIDs` that match the IDs in the set of `repeat_customers` and compute the total sales from the repeat customers. We do this by summing all the sales amounts from these customers and the average sales amount for the repeat customers by taking a mean of all the sales amounts from these customers. We save this data into a variable, `monthly_data`.

3. We do one last set of computations as in the following and will be ready to look at the differences between the new and repeat customers:

```
monthly_data_df = pd.DataFrame(monthly_data).set_index("month")
monthly_data_df["repeat_customer_percentage"] = monthly_data["repeat_customers"] / monthly_data["customers"]
monthly_data_df["repeat_sales_percentage"] = monthly_data["repeat_sales"] / monthly_data["sales"]
```

This code simply converts the data into a `pandas` DataFrame and computes what percentage of customers are actually repeat customers and what percentage of sales are from the repeat customers.

Now that we're done, let's take a look at the monthly customer counts and the breakdowns between the new and repeat customers using the following code:

```
ax = monthly_data_df[["new_customers", "repeat_customers"]].plot(kind="bar", grid=True, figsize=(15, 5))
```

```
(monthly_data_df["repeat_customer_percentage"]*100).plot(
    ax=ax, secondary_y=True, color="salmon", style="-o"
)
ax.right_ax.legend()
ax.right_ax.set_ylim([0, 100.0])
ax.right_ax.set_ylabel("repeat customer percentage (%)")
ax.set_ylabel("number of customers")
ax.set_title("number of new vs. repeat customers over time")
plt.show()
```

This should produce the following chart:

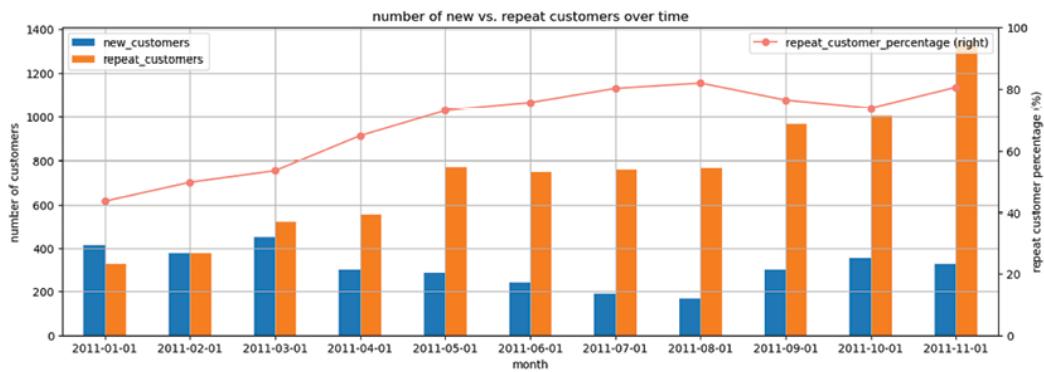


Figure 8.2: The number of new versus repeat customers

The bars on the left for each time period represent the number of new customers for each month and the bars on the right represent the number of repeat customers. As you can see, the composition of the repeat customers grows over time and there is some cycle in terms of the influx of the new customers. We see more new customers at the beginning and end of the year and dips during the summertime from June to August.

The line chart shows what percentage of the customers in each month were repeat customers and as this chart suggests, it grew from around 40% in January 2011 to around 80% in November 2011.

This indicates a very healthy business. It shows continuous demand and purchases from the customers who have made previous purchases from this business. The number of repeat customers continuously grows, which suggests that the products and services this business provides continuously attract customers who have once interacted with this business. For a business that lacks attractive products and/or good customer service, the number of repeat customers will typically decrease. One thing to note here though is that the rate of new customer influx is relatively steady and not growing. This, of course, is better than a decreasing new customer count over time, but this shows that there is a growth potential to capture more new customers. Given that there is a healthy repeat and recurring customer base, the marketers can focus on new customer acquisition more for this business.

Similarly, let's take a look at the sales amount from the new and repeat customers. Take a look at the following code:

```
ax = (monthly_data_df[[  
    "sales_from_new_customers", "sales_from_repeat_customers"  
]]/1000).plot(kind="bar", grid=True, figsize=(15,5))  
(monthly_data_df["repeat_sales_percentage"]*100).plot(  
    ax=ax, secondary_y=True, color="salmon", style="-o"  
)  
ax.set_ylabel("sales (in thousands)")  
ax.set_title("sales from new vs. repeat customers over time")  
ax.right_ax.legend()  
ax.right_ax.set_ylim([0, 100.0])  
ax.right_ax.set_ylabel("repeat customer percentage (%)")  
plt.show()
```

This code produces the following chart:

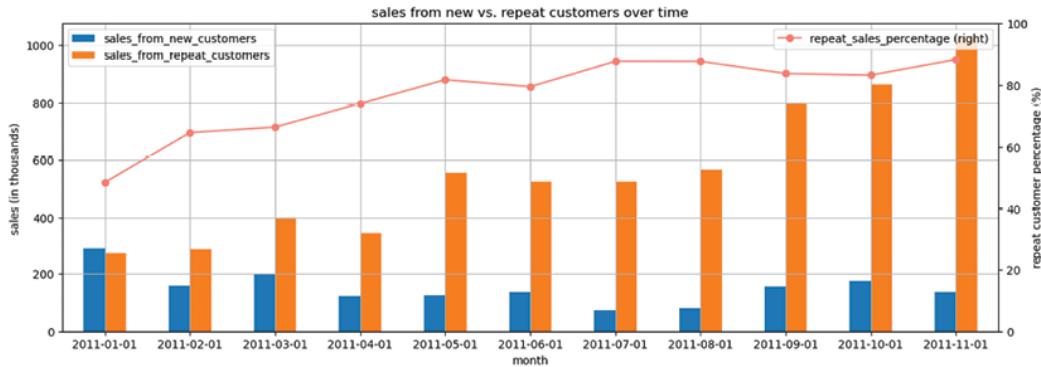


Figure 8.3: Sales from new versus repeat customers

Similar to before, the bars on the left represent the sales amount from new customers and the bars on the right represent the sales amount from repeat customers. As was the case with a number of customer comparisons before, the sales amount from repeat customers outweighs the sales amount from new customers. This suggests that there is a strong continuous recurring revenue from repeat customers. The percentage of sales from the repeat customers reached above 80% in November 2011. This is somewhat expected as we have discussed previously how repeat customers often take up more than half of the revenue for the businesses. Another point discussed previously and reported by *BIA Advisory Services* was that repeat customers typically spend 67% more than new customers.

Now let's see what our data says about this business by comparing average monthly sales from new customers against repeat customers. Take a look at the following code:

```
monthly_data_df["repeat_to_new_avg_sales_ratio"] = (
    monthly_data_df["avg_sales_from_repeat_customers"]
    /
    monthly_data_df["avg_sales_from_new_customers"]
)
ax = monthly_data_df[[
    "avg_sales_from_new_customers", "avg_sales_from_repeat_cust
```

```

]].plot(kind="bar", grid=True, figsize=(15,5), rot=0)
ax.set_ylabel("average sales")
ax.set_title("sales from new vs. repeat customers over time")
monthly_data_df["repeat_to_new_avg_sales_ratio"].plot(
    ax=ax, secondary_y=True, color="salmon", style="-o"
)
ax.right_ax.set_ylim([0, 2.0])
ax.right_ax.set_ylabel("repeat to new customer avg sales ratio")
plt.show()

```

This code should produce a chart like the following:

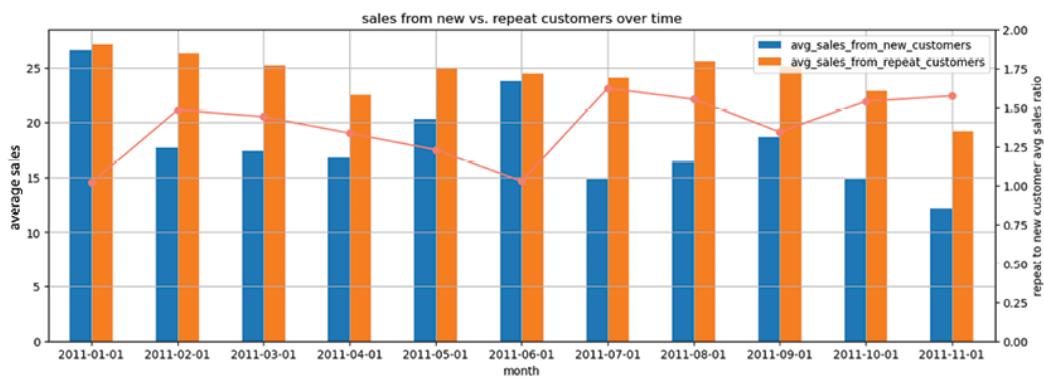


Figure 8.4: Average sales from new versus repeat customers

Similarly, the bars on the left are the average sales from the new customers and the bars on the right are the average sales from the repeat customers for each month. The line plot shows the ratios between the average sales of the new customers versus those of the repeat customers. For example, if the ratio is 1.5, it means that the repeat customers spent 1.5 times more than the new customers on average during that month. Based on this example dataset, we see that, on average, repeat customers spent more than the average customers for all months reported. In November 2011, the ratio was 1:1.58, which suggests that the repeat customers spent about 60% more than the new customers on average. This aligns with the *BIA Advisory*

*Services* report that repeat customers spend significantly more than new customers on average.

In this exercise, we have dissected the customer base into two simple segments: new versus repeat. As you may have noticed, this is a simple analytical exercise to conduct, but this produces powerful insights into the dynamics and health of the business and also tells the marketers which group of customers to focus on.

If there is a strong recurring customer base with continuous recurring revenue, but it shows steadiness or a decreasing new customer base, it suggests that marketers should focus more on new customer acquisition.

On the other hand, if there is a decline in repeat customers, it suggests that the products and services may not be attractive for customers to come back or that the customer service does not meet the customer expectations. In this case, the marketers should focus on improving the product marketing strategies, customer satisfaction strategies, or other marketing strategies to attract customers to come back for repeated purchases.

## **Customer segmentation with purchase behaviors**

Segmenting customers based on new versus repeat customers is one of the basic and critical analyses to conduct. However, oftentimes, we would like to segment customers based on multiple factors, which can be demographic factors, such as age, geolocation, and occupation, or purchase history, such as how much they spent in the past year, how many items they have purchased, and how many returns they have requested. You can also segment based on customer web activities, such as number of logins in the

past X number of days, how long they stay on your webpage, and what pages they look at.

There are still challenges to segmenting customers based on these factors, as there are an infinite number of ways and values you can segment the customers by. For example, if you are segmenting your customers by age, some of the questions that may arise are “`how many buckets should I create?`” or “`what age cutoff thresholds should I choose?`”. Similarly, if you want to segment your customers by past sales volume, you will still have to choose what thresholds to use to break down the customer base and how many segments you want to create.

Furthermore, when you combine these segments across multiple factors, the number of segments grows exponentially. For example, if you have 2 segments based on the sales volume and combine it with the other 2 segments from purchase quantity, you will end up with 4 segments. If you had 3 segments for both factors, then you would end up with 9 segments in total. In summary, there are mainly three key questions to be answered in conducting customer segmentation:

1. What factor(s) should be used for customer segmentation?
2. How many segments should be created?
3. What thresholds should be used to break down into segments?

We are going to use the online retail dataset that we have used previously as an example and discuss how to answer these key questions with the K-means clustering algorithm and silhouette score for measuring the effectiveness of clusters.

## K-means clustering

K-means clustering is one of the most frequently used machine learning algorithms for clustering and segmentation. It is a method to partition the data into  $k$  clusters. In short, the algorithm iteratively finds the centroids and groups the data points to the nearest centroids until the data points are closer to their centroids than to their neighboring centroids. As you can imagine, the data points in our case will be the factors of interest, such as sales amount, quantity, and refund, and the “ $k$ ” in K-means clustering is how many clusters or customer segments we would like to create.

In order to create customer segments based on the total sales amount, order quantity, and refunds, we need to do some prep work. Take a look at the following code:

```
# Net Sales & Quantity
customer_net_df = df.groupby('CustomerID')[["Sales", "Quantity"]]
customer_net_df.columns = ['NetSales', 'NetQuantity']
# Total Refunds
customer_refund_df = df.loc[
    df["Quantity"] < 0
].groupby("CustomerID")[["Sales", "Quantity"]].sum().abs()
customer_refund_df.columns = ['TotalRefund', 'TotalRefundQuantity']
customer_df = customer_net_df.merge(
    customer_refund_df, left_index=True, right_index=True, how="left"
).fillna(0)
```

Here, we first get the net sales and quantity for each customer. This will subtract any refunds and returned items as there are some records with negative `Sales` and `Quantity` values. Next, we get the information about the refunds. We assume any quantity that is negative is a refund. Thus, we get the total refund amount by summing all the sales with the negative quantity values and the total refund quantity by summing all the order quantities with the negative quantity values. Lastly, we merge these two

DataFrames by the index, which is the customer ID. The resulting DataFrame should look like the following:

CustomerID	NetSales	NetQuantity	TotalRefund	TotalRefundQuantity
12346.0	0.00	0	77183.60	74215.0
12347.0	4310.00	2458	0.00	0.0
12348.0	1797.24	2341	0.00	0.0
12349.0	1757.55	631	0.00	0.0
12350.0	334.40	197	0.00	0.0
...	...	...	...	...
18280.0	180.60	45	0.00	0.0
18281.0	80.82	54	0.00	0.0
18282.0	176.60	98	1.45	5.0
18283.0	2094.88	1397	0.00	0.0
18287.0	1837.28	1586	0.00	0.0

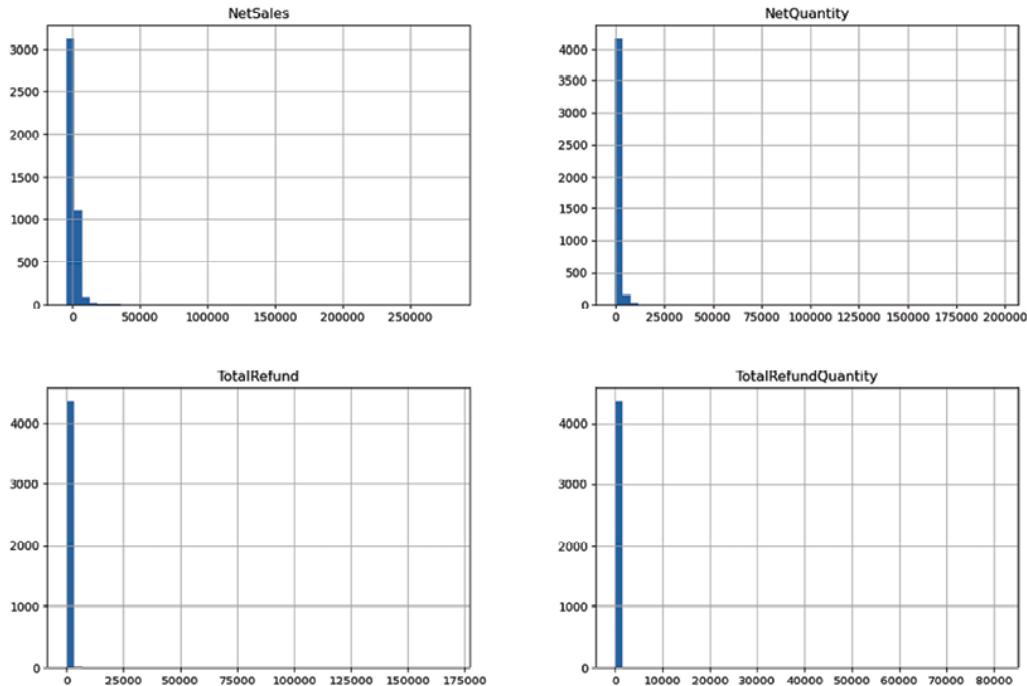
4372 rows × 4 columns

Figure 8.5: The resulting net sales, net quantity, refund amount, and refund quantity

One thing to note here is that the data is highly skewed. Take a look at the following code, which we will be using to generate histograms:

```
customer_df.hist(bins=50, figsize=(15,10))  
plt.show()
```

This gives us the following plots:



*Figure 8.6: Data distribution*

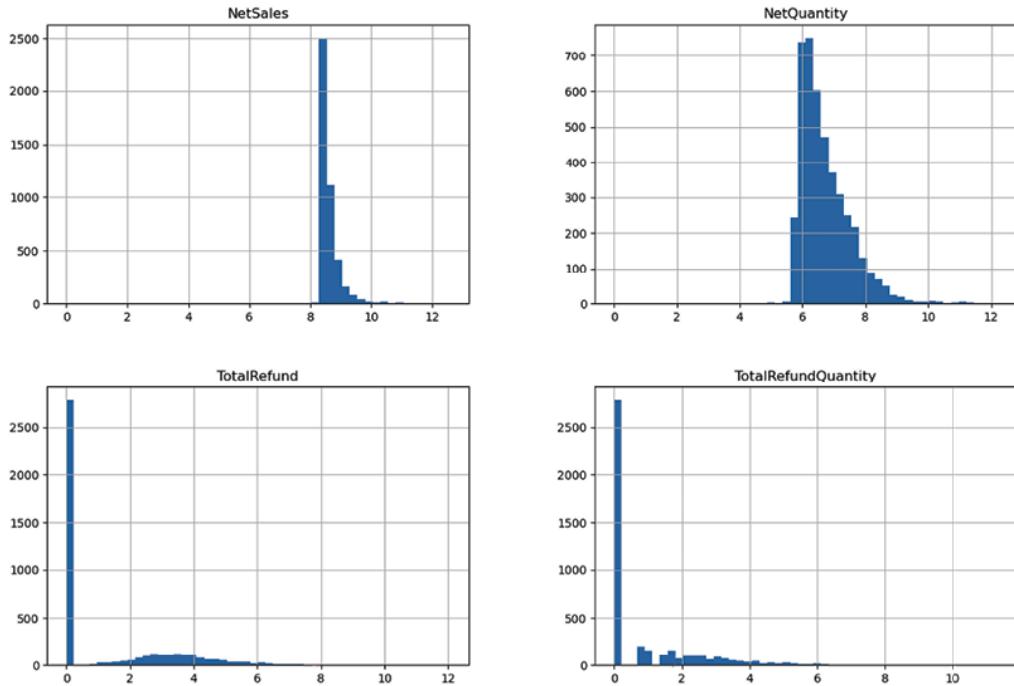
As you can see from these histograms, the data is highly skewed to the right. This occurs often, especially when the data has monetary or quantity values. Skewness in data causes disproportionate clusters with suboptimal segmentation of data. One simple approach to overcome this skewness is to log transform the data, using the following code:

```
log_customer_df = np.log(customer_df - customer_df.min() + 1)
```

We can generate the log-transformed data's histograms using the following code:

```
log_customer_df.hist(bins=50, figsize=(15, 10))
plt.show()
```

The following are the histograms that are generated:



*Figure 8.7: Log-transformed data distribution*

As expected, the log-transformed data is more centered around the mean and closer to a bell curve. We are going to examine clustering both with and without log transformation and see how it affects the clustering results.

## Without log transformation

Training a K-means clustering algorithm in Python is straightforward. Take a look at the following code, where we import the `KMeans` module in the `scikit-learn` package to build a K-means clustering algorithm:

```
from sklearn.cluster import KMeans
COLS = ['NetSales', 'NetQuantity', 'TotalRefundQuantity']
kmeans = KMeans(
    n_clusters=4, n_init="auto"
).fit(
    customer_df[COLS]
)
```

In this example, we are building customer segments based on the three columns, `NetSales`, `NetQuantity`, and `TotalRefundQuantity`. Then, we are building four clusters by using the parameter `n_clusters`. The `labels_` attribute of the trained `KMeans` model has the assigned labels (0 through 3) for each row or customer and the `cluster_centers_` attribute shows the centroids of each cluster.



## Randomness in K-means clustering

As K-means clustering is an algorithm with iterative approaches to updating centroids from the original randomly selected centroids, there is a randomness in K-means clustering that results in slightly different results each time it is run. If you would like to get the same results each time, you would want to set the `random_state` variable.

The examples in this chapter do not use the `random_state` variable, so your results may look slightly different from the charts you see here.

Now, let's visualize the clusters so that we can visually inspect how the K-means clustering algorithm has segmented the customer base using the three factors we are interested in:

```
import matplotlib.colors as mcolors
def plot_clusters(c_df, col1, col2):    colors = list(mcolors.T
clusters = sorted(c_df["cluster"].unique())
for c in clusters:
    plt.scatter(
        c_df.loc[c_df['cluster'] == c][col1],
        c_df.loc[c_df['cluster'] == c][col2],
        c=colors[c]
```

```
        )
plt.title(f'{col1} vs. {col2} Clusters')
plt.xlabel(col1)
plt.ylabel(col2)
plt.legend(clusters)
plt.show()
cluster_df = customer_df[COLS].copy()
cluster_df["cluster"] = kmeans.labels_
plot_clusters(cluster_df, "NetSales", "NetQuantity")
plot_clusters(cluster_df, "NetSales", "TotalRefundQuantity")
plot_clusters(cluster_df, "NetQuantity", "TotalRefundQuantity")
```

Here, we first define a function, `plot_clusters`, to 2D plot each cluster, where it takes the DataFrame as an input with the two columns for x- and y-axes. Then, we create three plots: one to visualize the clusters based on `NetSales` and `NetQuantity`, another for `NetSales` and `TotalRefundQuantity`, and a third for `NetQuantity` and `TotalRefundQuantity`. The three resulting plots should look like the following:

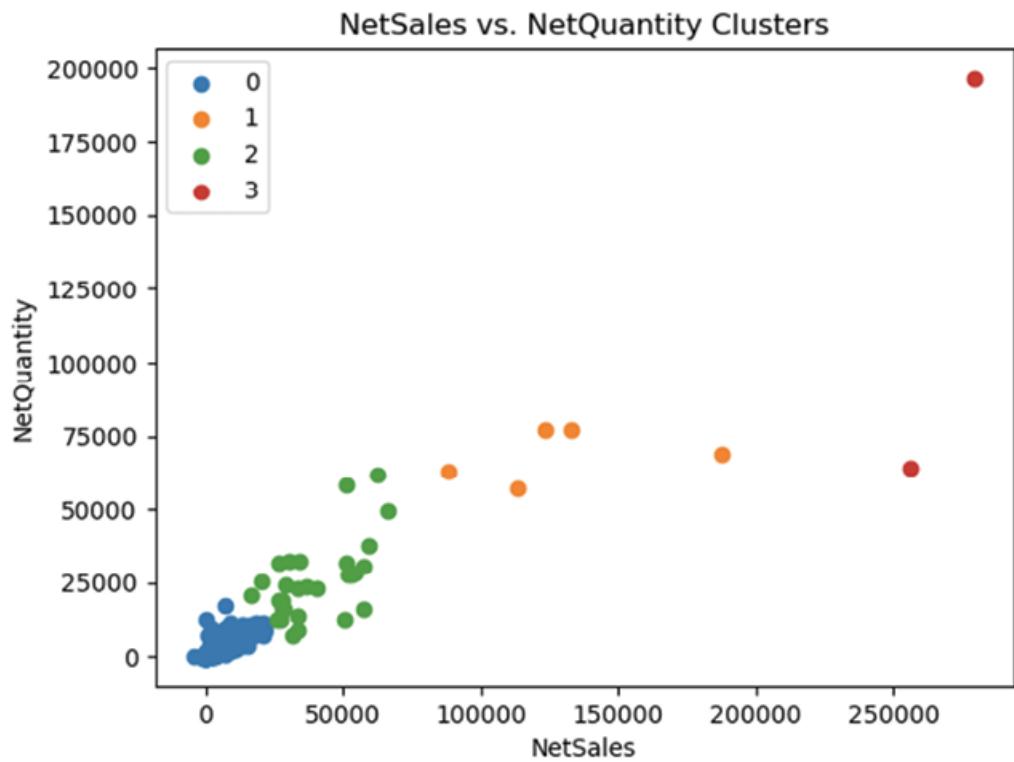
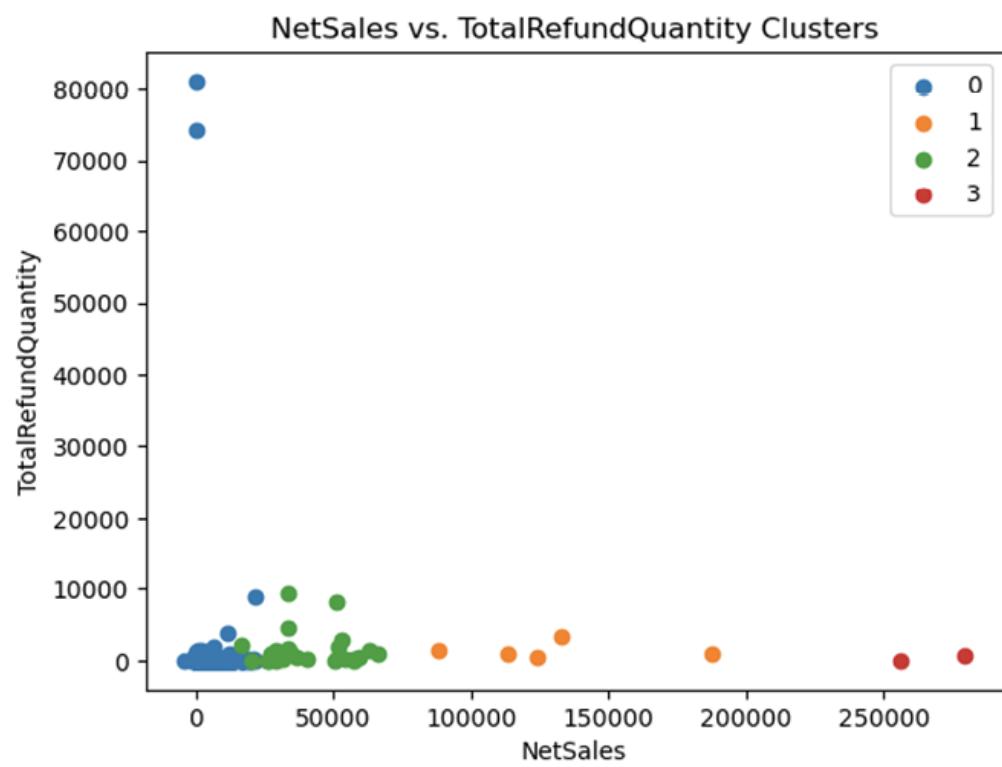
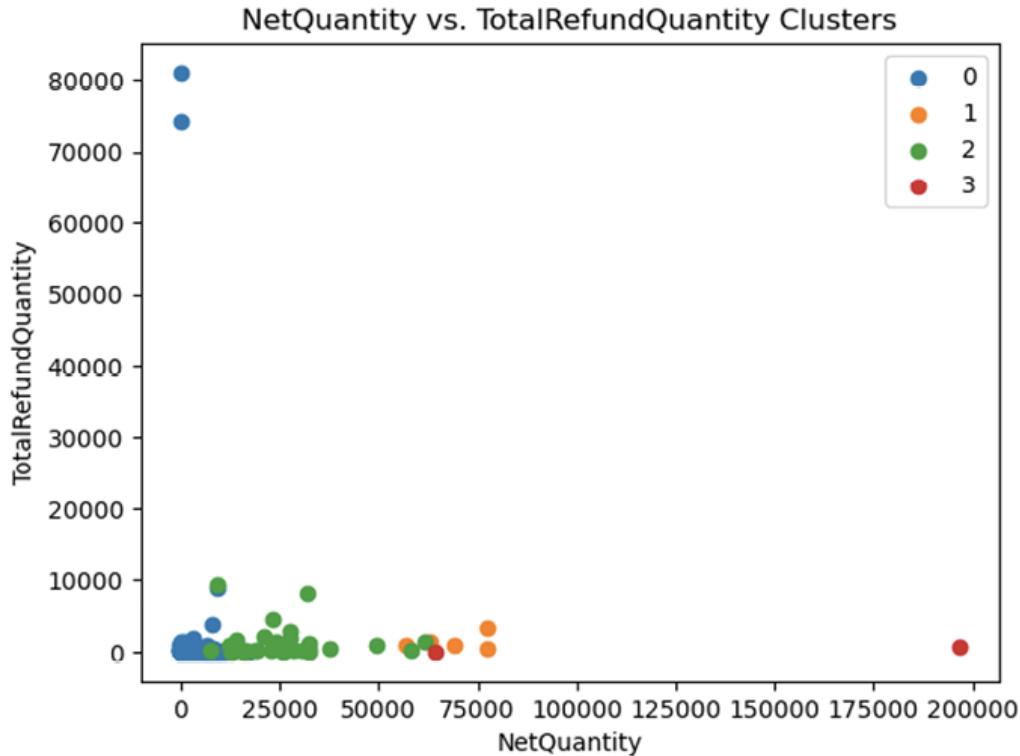


Figure 8.8: Clusters based on NetSales versus NetQuantity



*Figure 8.9: Clusters based on NetSales versus TotalRefundQuantity*



*Figure 8.10: Clusters based on NetQuantity versus TotalRefundQuantity*

In *Figures 8.8, 8.9, and 8.10*, we can easily see how clusters are formed based on the pairs. For instance, cluster 0 seems to be the customers who have low net sales, low net quantity, and low refunds, and cluster 1 seems to be the customers who have low-mid net sales, low-mid net quantity, and low refunds. However, there are two things that stand out in this example:

- Some clusters have a wide range of points. Cluster 0, as an example, has a wide range of refund quantity that ranges from 0 to 80,000 if you look at *Figure 8.10*. This makes it difficult to describe what cluster 0 actually signifies and how it is different from other clusters.
- Another thing to note here is that most of the data points are in cluster 0 and very few are in other clusters. Cluster 3 seems to contain only

two data points. These large imbalances in cluster sizes make generalizations from these clusters less reliable as insights based on a few data points are not so trustable.

You can take a look at the number of data points in each cluster using the following code:

```
cluster_df.groupby('cluster')['NetSales'].count()
```

Here are the details of each cluster:

```
cluster
0    4336
1      5
2     29
3      2
Name: NetSales, dtype: int64
```

*Figure 8.11: The number of data points in each cluster*

Data skewness often causes these problems as it makes the K-means clustering algorithm difficult to find or effectively cluster the data points. This is the reason why we need to normalize the data before applying clustering algorithms when there is a skewness in the data.

## With log transformation

Let's see how log transformation may help customer segmentation using K-means clustering. Take a look at the following code:

```
COLS = ['NetSales', 'NetQuantity', 'TotalRefundQuantity']
kmeans = KMeans(
    n_clusters=4, n_init="auto"
).fit()
```

```
    log_customer_df[COLS]
)
cluster_df = log_customer_df[COLS].copy()
cluster_df["cluster"] = kmeans.labels_
```

Here, we use the previously defined variable, `log_customer_df`, which is the log-transformed data, using the `np.log` function. We then fit a K-means clustering model with four clusters. We can see what the cluster sizes look like now:

```
cluster_df.groupby('cluster')['NetSales'].count()
```

This gives us the following output:

```
cluster
0    2371
1     311
2     884
3     806
Name: NetSales, dtype: int64
```

*Figure 8.12: The number of data points in each cluster after log transformation*

Compared to earlier in the chapter, when we fit the clustering model without the log transformation, the clusters are more balanced, with each cluster having a significant number of customers. This is going to give better insights into how customers are segmented based on the three factors of our interest: net sales, net quantity, and total refunds. Let's visualize the clusters with the following code:

```
plot_clusters(cluster_df, "NetSales", "NetQuantity")
plot_clusters(cluster_df, "NetSales", "TotalRefundQuantity")
```

```
plot_clusters(cluster_df, "NetQuantity", "TotalRefundQuantity")
```

This code should create three charts similar to the following:

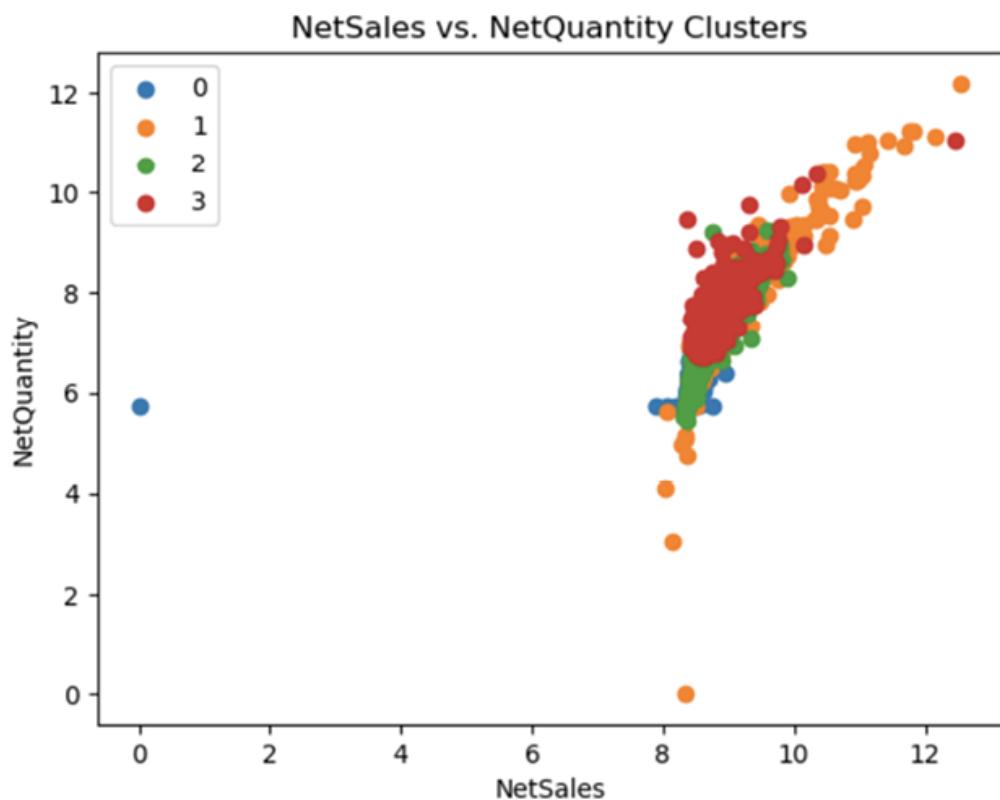


Figure 8.13: Clusters on NetSales versus NetQuantity after transformation

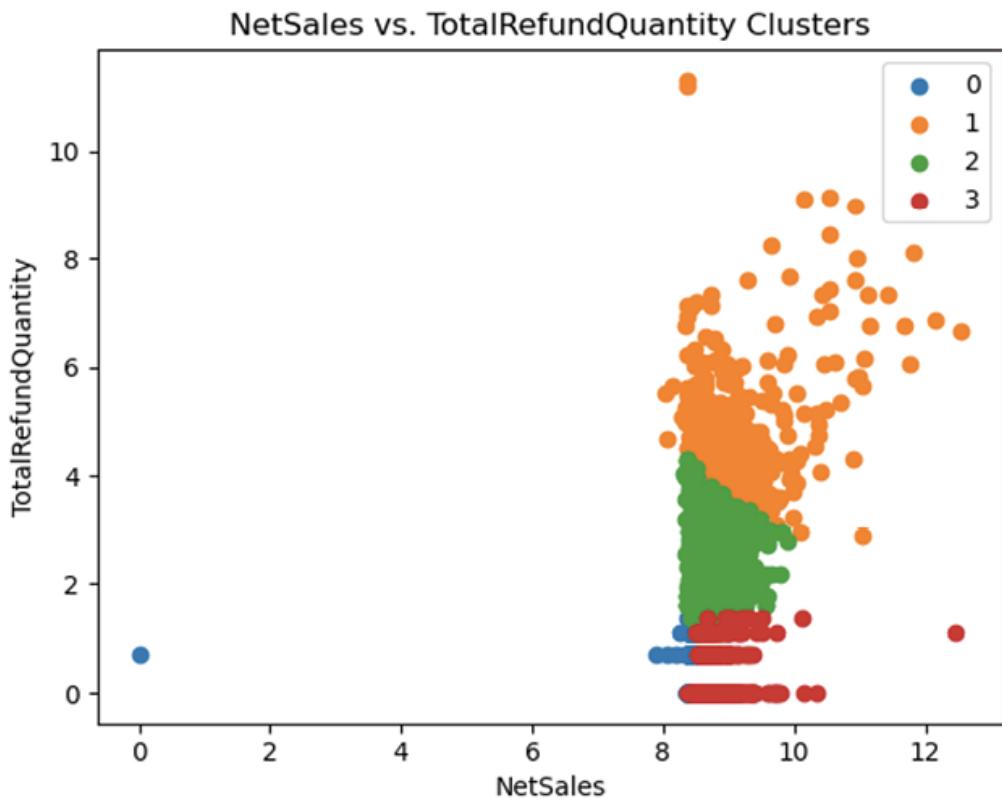


Figure 8.14: Clusters on NetSales versus TotalRefundQuantity after transformation

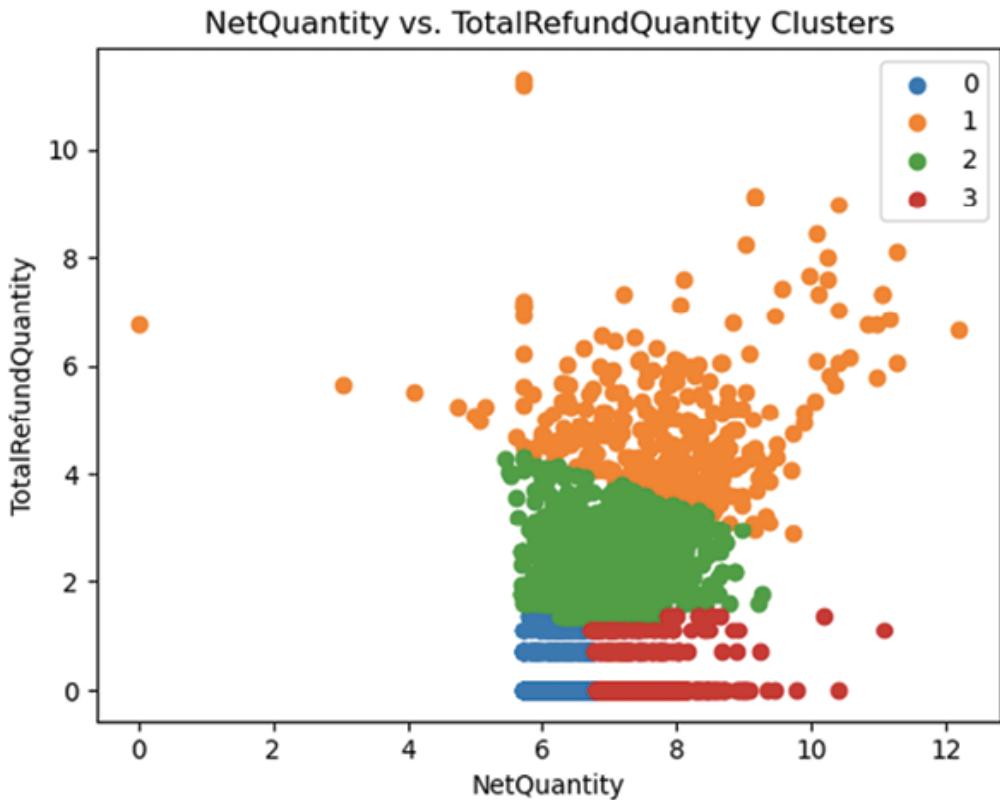


Figure 8.15: Clusters on NetQuantity versus TotalRefundQuantity after transformation

Let's dive deeper into these cluster visualizations. There is no clear separation between clusters based on the net sales and net quantity if you look at *Figure 8.13*. Cluster 3 seems more centered around the mean, while cluster 1 tends more toward higher net sales and higher net quantity, and the clusters 0 and 2 more toward lower net sales and lower net quantity. Despite these subtle differences, all clusters seem to cluster around the means of net sales and net quantity.

The distinctions between clusters are more clear in *Figures 8.14* and *8.15*. Cluster 1 seems to be the customer segment with high total refunds, with the cutoff threshold around 4 in the log scale. You can convert this log-transformed value by reverting the transformation we have applied previously or by using the following code:

```
np.exp(X) + customer_df.min()[COLUMN] - 1
```

The `X` in this code is the value in the log scale and `COLUMN` in this code is the factor of interest. Let's take a closer look at these clusters:

- **Cluster 0:** This cluster seems to be the cluster with low total refunds and low net quantity. The thresholds for `TotalRefundQuantity` in the log scale are around `2` and `6.7` for `NetQuantity`. These numbers equate to `6.4` in total refunds and `811.4` in net quantity, using the preceding equation. This suggests that the customers in this cluster have less than `6.4` total refunds and have a net quantity less than `811.4`.
- **Cluster 1:** This cluster seems to be the group of customers who have got refunds the most frequently. The cutoff threshold for cluster `1` seems to be around `4` in the log scale of `TotalRefundQuantity` and using the code above, this equates to `54 (np.exp(4) + customer_df.min()["TotalRefundQuantity"] - 1)`. Thus, cluster `1` is the segment of customers with above `54` total refunds.
- **Cluster 2:** This cluster seems to be the customer segment with mid-total refunds with a threshold of around `2` in the log scale of `TotalRefundQuantity`, which equates to about `6` in actual total refunds. Thus, cluster `2` is the segment of customers with total refunds between `6` and `54`.
- **Cluster 3:** The customers in cluster `3` seem to be the ones with low total refunds but high net quantity. This customer segment may be the sweet spot for the business as it suggests that they buy frequently from the business but do not get as many refunds as other customers in different segments. Given that their cutoff thresholds for total refunds

in the log scale are around 2 and 6.7 for net quantity, this group of customers are the ones with less than 6 in total refunds and more than 811 in net quantity.

As you can see in this exercise, the K-means clustering algorithm is helpful in segmenting the customers in an ML way without having to define the thresholds for each customer segment manually for yourselves. This programmatic approach helps you define customer segments more dynamically and in a more data-driven way. This way, you can better understand how different customer segments are grouped and what the separating factors are, which then can be used to strategize and prioritize for your future marketing efforts.

## Silhouette score

We have seen how to build customer segments with the K-means clustering algorithm. One of the key arguments to build the clusters was the number of clusters. However, you may wonder how to decide or how you would know the right number of clusters before you build such clusters. In a practical setting, you would want to build multiple clusters with different numbers of clusters and decide which one works the best. This is where the **silhouette score** comes in. Simply put, the silhouette score is a metric that quantifies how well a given data point fits into its cluster and how distinguishable it is from other clusters. The formula for the silhouette score looks as follows:

$$\text{Silhouette Score}_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Here,  $a_i$  is the average distance of the  $i^{\text{th}}$  data point to all other points in the same cluster and  $b_i$  is the minimum average distance of the  $i^{\text{th}}$  data point to all other points in the other clusters.

In order to get the silhouette scores for a clustering algorithm, you need to get the average of all individual silhouette scores for each data point.

Silhouette scores range between `-1` and `1`. The closer the score is to `1`, the better the data points are clustered together and the more distinct they are from adjacent clusters.

Silhouette scores can easily be computed in Python. The `scikit-learn` package in Python has a function, `silhouette_score`, which computes the average silhouette score for the clusters:

```
from sklearn.metrics import silhouette_score
silhouette_score(
    log_customer_df[COLS],
    kmeans.labels_
)
```

As a higher silhouette score value suggests better clusters, we can utilize this to decide what the ideal number of clusters is. Take a look at the following code, where we are experimenting with clusters of sizes from `4` to `8`:

```
COLS = ['NetSales', 'NetQuantity', 'TotalRefundQuantity']
f, axes = plt.subplots(2, 3, sharey=False, figsize=(12, 7))
for i, n_cluster in enumerate([4, 5, 6, 7, 8]):
    kmeans = KMeans(n_clusters=n_cluster, n_init="auto").fit(
        log_customer_df[COLS]
    )
    silhouette_avg = silhouette_score(
        log_customer_df[COLS],
        kmeans.labels_
    )

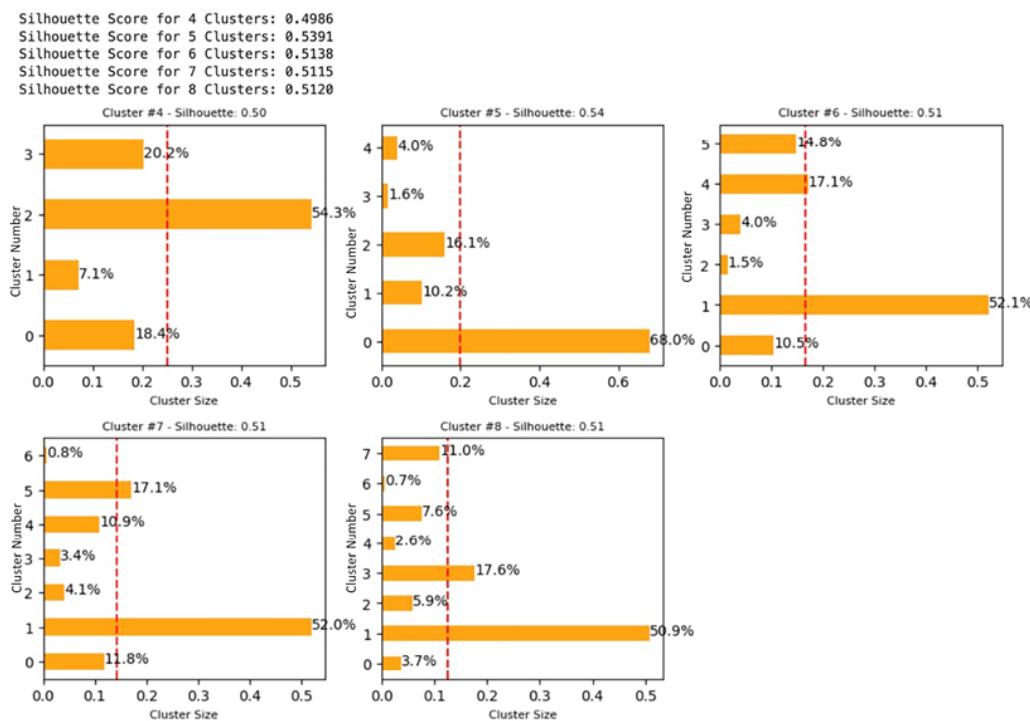
    print('Silhouette Score for %i Clusters: %0.4f' % (n_cluste
each_cluster_size = [
```

```

        (kmeans.labels_ == i).sum()/len(kmeans.labels_) for i in
    ]
ax = axes[i//3][i%3]
pd.DataFrame(each_cluster_size).plot(ax=ax, kind="barh", color="orange")
for p in ax.patches:
    ax.annotate(f'{p.get_width() * 100:.01f}%', (p.get_width(),
ax.axvline(x=(1/n_cluster), color="red", linestyle="--")
ax.set_xlabel("Cluster Size", size=8)
ax.set_title(f"Cluster #{n_cluster} - Silhouette: {silhouette:.2f}")
ax.title.set_size(8)
f.subplots_adjust(hspace=0.3)
plt.show()

```

For each cluster, we compute the silhouette scores and check what percentage of data points are in each cluster. Ideally, the best cluster is the one with the highest silhouette score and the highest number of data points evenly distributed among each cluster. This code should generate the output that follows:



*Figure 8.16: The cluster size experimentation results*

This chart shows a couple of important decision factors when you are figuring out what the best cluster size should be:

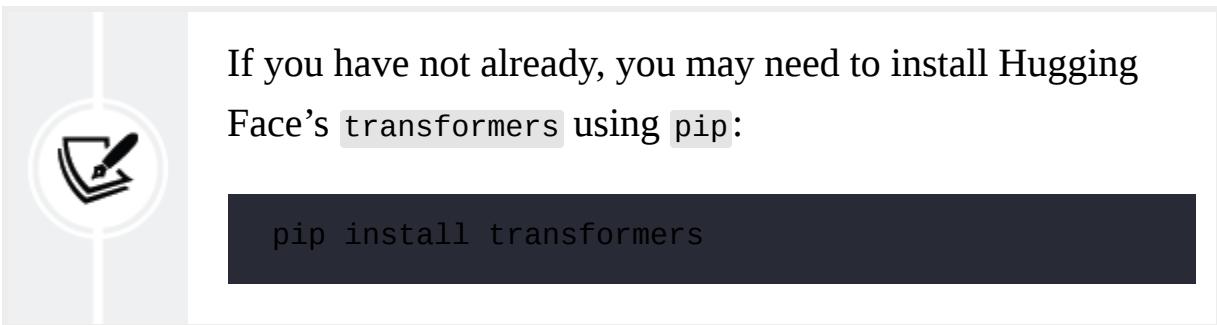
- **Silhouette scores:** First, we can see the silhouette scores for each cluster size, where it is `0.4986` for the cluster of size `4`, `0.5391` for the cluster of size `5`, and so forth. Here, the silhouette score for the cluster of size `5` seems to be the highest with `0.5391`.
- **Cluster sizes and compositions:** We should also evaluate if the data points in each cluster are somewhat evenly distributed, as large imbalances among the clusters may not be reliable enough to generate the best insights. The horizontal bar charts in *Figure 8.16* show the cluster compositions for each cluster size. The dotted vertical line shows where the bars should be if the composition is completely evenly distributed. As can be seen in these charts, clusters with high silhouette scores do not necessarily mean the best clusters. Cluster `5`, for example, has the highest overall silhouette score, but has a large imbalance where the  $0^{\text{th}}$  cluster has close to 70% of data points and the rest shares a small portion of the data. This is not an ideal cluster as too much of the data is in one cluster.

In this example, a cluster of size `4` may be the best choice as the data points are more evenly distributed across the clusters compared to others, even though the silhouette score is not the highest.

## Customer segmentation with product interests

We have discussed how we can build customer segments based on their purchase history in the previous section and how this can inform marketers on which segment to prioritize and strategize for the next marketing effort. Not only can we segment customers based on their purchase history, or more specifically with numerical values, but we can also find customer segments based on their product interests.

The items that customers purchase have hidden insights into what types of items each customer is interested in and what they are likely to purchase more of. There are multiple approaches to segmenting customers based on the products that they have purchased in the past, such as simply grouping by the product categories that they have purchased from. However, in this exercise, we are going to expand on the topic of the embedding vectors that we touched on in *Chapter 5*.



As previously discussed in *Chapter 5*, modern LLMs like **BERT** and **GPT** introduced contextual embeddings, where the words and sentences are transformed into numerical values or vectors that represent the contextual meanings. We will be using the pre-trained LLM `all-MiniLM-L6-v2` from Hugging Face to encode past product purchases for each customer in our example dataset and build customer segments using these embedding vectors. Take a look at the following code:

```
import os
os.environ["TOKENIZERS_PARALLELISM"] = "false"
from sentence_transformers import SentenceTransformer, util
customer_item_df = pd.DataFrame(
    df.groupby("CustomerID")["Description"].apply(
        lambda x: ", ".join(list(set(x)))
    )
)
embedding_model = SentenceTransformer(
    "sentence-transformers/all-MiniLM-L6-v2"
)
encoded = embedding_model.encode(
    list(customer_item_df["Description"]),
    show_progress_bar=True
)
with open('tmp.npy', 'wb') as f:
    np.save(f, encoded)
```

Here, we first get all the product descriptions of products that the customers have bought and then create a comma-separated list for each of the customers, which gets stored in the variable `customer_item_df`. Then, we load the pre-trained LLM, `sentence-transformers/all-MiniLM-L6-v2`, and encode the list of product descriptions for each customer into numerical vectors by using the `encode` function of the pre-trained LLM. This will result in a vector of 384 values for each customer. We then save this array into `tmp.npy` for future usage.

In high-dimensional space, the distances between data points become less meaningful as the volume increases due to larger dimensions making data too sparse. As the K-means clustering algorithm uses distance metrics to cluster data points together, this becomes an issue. In order to overcome this, we need to apply some dimensionality reduction techniques. In this exercise, we will simply apply **principal component analysis (PCA)** to

reduce the dimensions of the embedding vectors while retrieving most of the variance within the data:

1. Take a look at the following code:

```
from sklearn.decomposition import PCA
with open('tmp.npy', 'rb') as f:
    encoded = np.load(f)
pca = PCA(n_components=5)
transformed_encoded = pca.fit_transform(encoded)
```

Here, we import the `pca` module in the `scikit-learn` package. We import the previously built embedding vectors from the temporary location, `tmp.npy`, and fit and transform the vector by using the `fit_transform` function. We have defined it to return 5 components, as you can see from the `n_components` parameter. The resulting vector, `transformed_encoded`, should have a size of 5 vectors for each customer.

## Other dimensionality reduction approaches

There are numerous dimensionality reduction techniques other than PCA. In our exercise, we used PCA for its simplicity, but T-SNE and UMAP are two others that are frequently used when dealing with high-dimensional data. Be sure to check them out and see if they may be a better fit for this exercise!



**T-SNE:** <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>

**UMAP:** <https://umap-learn.readthedocs.io/en/latest/>

2. Now it's finally the time to build customer segments or clusters based on these embedding vectors, which have contextual understandings of the products that each customer has bought. Take a look at the following code:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_
for n_cluster in [4,5,6,7,8]:
    kmeans = KMeans(n_clusters=n_cluster, n_init="auto").fi
        transforemd_encoded
    )

    silhouette_avg = silhouette_score(
        transforemd_encoded,
        kmeans.labels_
    )

    print('Silhouette Score for %i Clusters: %0.4f' % (n_c]
```

This code should look familiar, as this is almost the exact same code as before when we built clusters using purchase history with the K-means clustering algorithm. The main difference here is instead of the sales metrics, we use the embedding vectors as the input to `KMeans` to build clusters. The output of this code should look something like the following:

```
Silhouette Score for 4 Clusters: 0.1881
Silhouette Score for 5 Clusters: 0.2120
Silhouette Score for 6 Clusters: 0.2149
Silhouette Score for 7 Clusters: 0.2152
Silhouette Score for 8 Clusters: 0.2105
```

Figure 8.17: The silhouette scores for different cluster sizes

The actual values may differ as there is some randomness in fitting a K-means clustering algorithm, but the trend should be similar.

3. Based on this, we are going to build 7 clusters based on the embedding vectors, as in the following code:

```
n_cluster = 7
kmeans = KMeans(n_clusters=n_cluster, n_init="auto").fit(
    transforemd_encoded
)
customer_item_df["cluster"] = kmeans.labels_
from collections import Counter
n_items = 5
common_items = []
for i in range(n_cluster):
    most_common_items = Counter(list(df.set_index("Customer"
        customer_item_df.loc[
            customer_item_df["cluster"] == i
        ].index
    )["Description"])).most_common(n_items)

    common_items.append({
        f"item_{j}": most_common_items[j][0] for j in range(
    5
)})
common_items_df = pd.DataFrame(common_items)
```

In this code, we build 7 clusters. Then, for each cluster, we get the top 5 most common items that customers in each cluster have purchased by using

the `collections` library in Python. We then store these top 5 most commonly bought items for each cluster in a DataFrame named `common_items_df`. This DataFrame should give us insights into what types of products interest each customer segment the most.

Let's take a closer look at this DataFrame:

	common_items_df				
	item_0	item_1	item_2	item_3	item_4
0	WHITE HANGING HEART T-LIGHT HOLDER	REGENCY CAKESTAND 3 TIER	ASSORTED COLOUR BIRD ORNAMENT	WOODEN PICTURE FRAME WHITE FINISH	WOODEN FRAME ANTIQUE WHITE
1	POSTAGE	JUMBO BAG RED RETROSPOT	BAKING SET 9 PIECE RETROSPOT	PARTY BUNTING	VINTAGE SNAP CARDS
2	REGENCY CAKESTAND 3 TIER	WHITE HANGING HEART T-LIGHT HOLDER	ROSES REGENCY TEACUP AND SAUCER	GREEN REGENCY TEACUP AND SAUCER	VICTORIAN GLASS HANGING T-LIGHT
3	JUMBO BAG RED RETROSPOT	LUNCH BAG RED RETROSPOT	LUNCH BAG BLACK SKULL	POSTAGE	LUNCH BAG SPACEBOY DESIGN
4	HOT WATER BOTTLE KEEP CALM	Manual	CHILLI LIGHTS	CHOCOLATE HOT WATER BOTTLE	HOT WATER BOTTLE TEA AND SYMPATHY
5	WHITE HANGING HEART T-LIGHT HOLDER	PAPER CHAIN KIT 50'S CHRISTMAS	REGENCY CAKESTAND 3 TIER	ASSORTED COLOUR BIRD ORNAMENT	PARTY BUNTING
6	PAPER CHAIN KIT 50'S CHRISTMAS	WHITE HANGING HEART T-LIGHT HOLDER	PAPER CHAIN KIT VINTAGE CHRISTMAS	WOODEN STAR CHRISTMAS SCANDINAVIAN	WOODEN HEART CHRISTMAS SCANDINAVIAN

Figure 8.18: The top 5 common items for each cluster

This gives us the following insights:

- The customers in the first cluster or the cluster indexed at 0 seem to have the most interest in some decoration items, such as ornaments and wooden frames.
- The customers in the second cluster or the cluster indexed at 1 seem to show interest in party-related items, such as baking sets, party bunting, and cards.
- The customers in the third cluster or the cluster indexed at 2 seem to be into teas or tea ceremonies, as they bought teacups and saucers.
- The fourth cluster's customers seem to like purchasing bags.
- The fifth cluster's customers seem to like water bottles, and so forth.

As you can see, these customer clusters show what they are mostly interested in and how different customers can be grouped together by their product interests. These are going to be powerful insights as you build your

next marketing strategies and campaigns. You may not want to promote tea sets to those who are interested in purchasing water bottles and vice versa. This misaligned targeting will result in wasteful marketing campaigns with low success rates in engagements and conversions. You would want to take these findings about different customer segments based on their product interests and build more targeted marketing campaigns for each segment with the product categories they are most interested in.

This way, you are more likely to have successful marketing campaigns with more success in drawing customer engagements and conversions.

## Summary

In this chapter, we discussed different ways to segment the customer base. We first looked at how new versus repeat customers contribute to revenue, as well as how monthly progressions of new and repeat customer numbers can tell us which segment or group of customers to focus on during the next marketing campaigns. Then, we discussed how the K-means clustering algorithm can be used to programmatically build and identify different customer segments. Using the sales amount, order quantity, and refunds, we experimented with how these factors can be used to build different customer segments. In lieu of doing it, we touched on silhouette scores as a criterion for finding the best number of clusters and how log transformation can be beneficial when dealing with highly skewed datasets. Lastly, we used word and sentence embedding vectors to convert the product descriptions into numerical vectors with contextual understanding and further built customer segments based on their product interests.

In the following chapters, we are going to explore LLMs even further. From creating compelling content using pre-trained zero-shot models to more

advanced few-shot and RAG approaches, we will touch more on LLMs and generative AI in the next chapter.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](https://OceanofPDF.com)

# 9

# Creating Compelling Content with Zero-Shot Learning

Having introduced the promise of large language models in *Chapter 5*, we will go deeper into related topics in this chapter, extending our analysis from their role in data augmentation and sentiment analysis to their broader impact across different domains. This chapter introduces **zero-shot learning (ZSL)**, a method in machine learning where a model can correctly make predictions for new, unseen classes without having received any specific training examples for those classes. It discusses the potential of ZSL and its application within the area of generative AI to create marketing copy. The discussion highlights how ZSL, an efficient tool to complement traditional marketing content creation processes, can revolutionize the generation of marketing copy.

We will start with an in-depth discussion of the core principles of generative AI and navigate through the capabilities and limitations of these technologies, which will set the stage for our subsequent exploration of the importance of pre-trained models. We will finish the chapter with a practical walkthrough of ZSL, using hands-on examples to illustrate the flexibility of this approach and how we can use it to generate marketing content. This will

equip you with the skills to understand and leverage this technique to elevate your marketing strategies to new heights.

By the end of the chapter, you will be well versed in:

- The fundamentals of generative AI and its versatile applications in marketing
- The principles of ZSL and its value in improving the efficiency of traditional content creation processes
- Practical strategies and considerations when applying ZSL to create marketing copy

## Fundamentals of generative AI

**Generative AI (GenAI)** refers to a subset of AI capable of generating new content, be it text, images, videos, or even synthetic data, that mirrors real-world examples. Unlike traditional AI models, which are designed to interpret, classify, or predict data based on inputs, GenAI takes it a step further by creating new, previously unseen outputs. It does this by understanding and learning from existing data patterns to produce novel outputs that maintain a logical continuity with the input data.

We were introduced to GenAI in *Chapter 1*, and we further touched upon it and its applications for sentiment analysis in *Chapter 5*. Before beginning our discussion of pre-trained models and ZSL, we will explore the fundamental technical considerations of GenAI, what it is (and is not), and why it's so impactful for generating marketing content. While the focus of the hands-on examples in this chapter will involve text generation,

important concepts that power GenAI's capabilities in other applications such as images and video will also be discussed.

## A probabilistic approach

At the heart of GenAI is a probabilistic approach to modeling data distributions. This involves learning the underlying probability distribution of a dataset to generate new samples from that same distribution. A cornerstone of this approach is Bayesian inference, a principle that updates the probability of a hypothesis as more evidence or information becomes available. For instance, consider a simple equation that forms the mathematical foundation on which Bayesian inference is built, Bayes' Theorem:

$$P(A | B) = P(B | A) \cdot \frac{P(A)}{P(B)}$$

where:

- $P(A|B)$  is the posterior probability of hypothesis  $A$ , given the evidence  $B$
- $P(B|A)$  is the likelihood of observing evidence  $B$ , given that hypothesis  $A$  is true
- $P(A)$  is the prior probability of hypothesis  $A$ , or how likely we believe  $A$  to be true before seeing the evidence
- $P(B)$  is the probability of observing the evidence under all possible hypotheses

**Bayes' Theorem – a pillar of probabilistic reasoning**



Bayes' Theorem is not just a cornerstone of GenAI but also a fundamental principle across a wide range of disciplines, from statistics and computer science to philosophy and medicine. At its core, Bayes' Theorem allows us to refine our hypotheses in light of new evidence, offering a rigorous mathematical approach to the concept of learning from experience.

When extending the principles of Bayesian inference and incorporating deep learning models such as **recurrent neural networks (RNNs)**, **long short-term memory networks (LSTMs)**, or transformers to the generation of sequences, we enter the realm of conditional probabilities. This sequence generation process can be viewed through the lens of predicting each element based on its predecessors, a concept foundational not just to video and audio but also to time series modeling and other forms of sequential data generation applications, including text.

Training GenAI models involves vast amounts of data, and in the case of text, these must be broken down into smaller units known as tokens. These tokens often consist of subword units or phrases, making the models more efficient in understanding and generating natural language. The process of tokenizing text is crucial because it allows a model to learn the probability distribution of different sequences of words or subwords.

When we tokenize text, we break it down into manageable pieces that a model can process. Each token is then used as an input to the model during training. The model learns to predict the probability of the next token in a sequence, given the previous tokens. This probabilistic approach is where Bayesian principles come into play. By continuously updating the

probability distribution of tokens as new data is introduced, the model becomes better at generating coherent and contextually relevant outputs.

For example, in text generation, a model might predict the next word in a sentence based on the preceding words. This prediction process involves calculating the conditional probability:

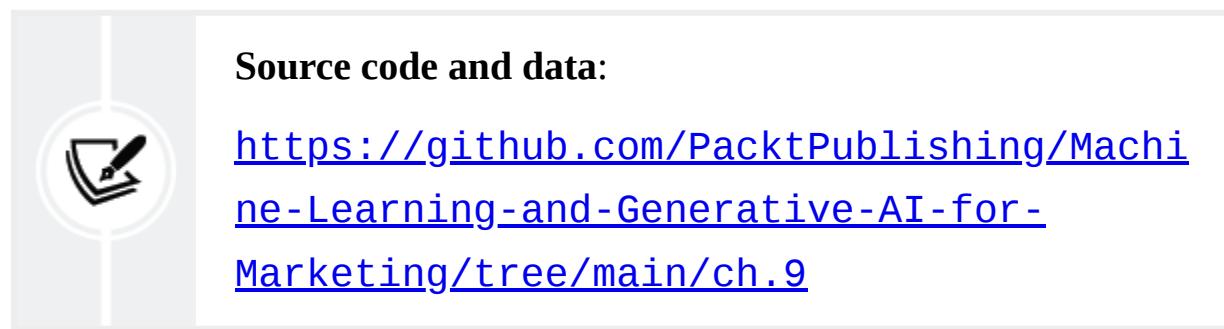
$$P(x_t | x_1, x_2, \dots, x_{t-1})$$

where  $x_t$  represents the token at time  $t$  and  $P(x_t | x_1, x_2, \dots, x_{t-1})$  denotes the probability of generating  $x_t$ , given the sequence of all preceding tokens.

In the case of video and audio generation, leveraging deep learning models informed by Bayesian principles helps in understanding and predicting temporal progression. Each frame or audio sample at time  $t$  ( $x_t$ ) is predicated on the sequence of all previous frames or samples ( $x_1, x_2, \dots, x_{t-1}$ ).

Mathematically, this relationship is captured by the previous equation:

where  $x_t$  represents the frame or audio sample at time  $t$  and  $P(x_t | x_1, x_2, \dots, x_{t-1})$  denotes the probability of generating  $x_t$ , given the sequence of all preceding samples.



#### Source code and data:

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/tree/main/ch.9>

## Foundational models

There are several important foundational models in GenAI, each contributing uniquely to applications in image, text, and sequence

generation. This section will cover some of the most important and widely used cases, such as:

- **Generative adversarial networks (GANs)**
- **Variational autoencoders (VAEs)**
- **Long short-term memory networks (LSTMs)**
- Transformer-based models like the **Generative Pre-Trained Transformer (GPT)**

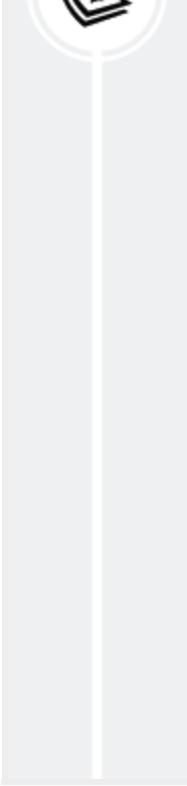
While comprehensive implementation examples and theory for each of these models are outside the scope of this chapter, we will discuss the core concepts for each model type, as well as provide simplified, illustrative examples of their architectures, in order to understand the importance of these models for marketing applications. In *Chapter 12*, we will extend our discussion to mention further model advances that have garnered more recent attention for their promise in advancing the field of GenAI.

### Exploring ML models with Google Colab notebooks

Training your own state-of-the-art ML model can be computationally expensive. However, using Google Colab notebooks, you can train and tweak models without any setup on your own machine. The following are links to get started:

- **GANs** for high-quality image generation:  
[https://colab.research.google.com/drive/1uwP1Y-4P\\_6fJ59SFRTgZLebVGgwGrUQu](https://colab.research.google.com/drive/1uwP1Y-4P_6fJ59SFRTgZLebVGgwGrUQu)
- **VAEs** for image reconstruction:  
<https://colab.research.google.com/git>





[hub/smartergeometry-ucl/dl4g/blob/master/variational\\_autoencoder.ipynb](https://hub.docker.com/r/smartergeometry/ucl/dl4g/blob/master/variational_autoencoder.ipynb)

- **GPTs** for language processing:  
[https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c\\_fvtXnx-?usp=sharing](https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing)
- **LSTMs** for time series forecasting:  
[https://colab.research.google.com/github/d2l-ai/d2l-pytorch-colab/blob/master/chapter\\_recurrent-modern/lstm.ipynb](https://colab.research.google.com/github/d2l-ai/d2l-pytorch-colab/blob/master/chapter_recurrent-modern/lstm.ipynb)

## Generative adversarial networks

GANs have found applications across a wide range of domains, from image generation and style transfer to data augmentation and beyond. They are particularly impactful in applications where realistic image generation is crucial, and they have been used by NVIDIA and Adobe in their photo editing software to generate and modify images. Their applications include the following:

- **Content creation:** GANs can generate high-quality, realistic images, artwork, and videos, enabling new forms of creative content production
- **Image-to-image translation:** Applications like photo enhancement, photo-realistic rendering from sketches, and domain adaptation, such as day-to-night and summer-to-winter transformations, leverage GANs to transform images from one domain to another while preserving contextual details

At their core, GANs consist of two neural networks that are trained simultaneously through a competitive process:

- The generator ( $G$ ) aims to generate data that is indistinguishable from real data
- The discriminator ( $D$ ) aims to accurately classify data as real or generated

This can be illustrated by the following figure:

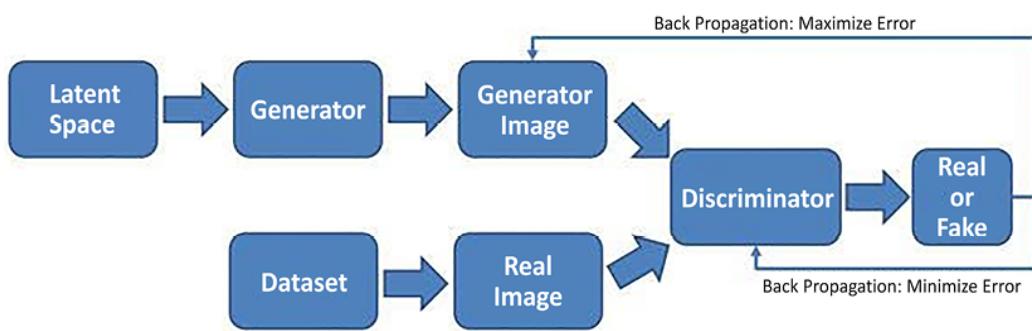


Figure 9.1: GAN workflow

The objective function for a GAN encapsulates the training dynamics between the generator and discriminator, creating a dynamic where both models improve in response to each other's performance. This is similar to a two-player game where each player's success is based on outsmarting their opponent. The game, in GAN's case, reaches equilibrium when the generator produces perfect replicas of the real data, making it impossible for the discriminator to distinguish real from fake, which ideally results in a 0.5 probability of guessing correctly by the discriminator.

For GANs to be highly effective at tasks such as high-resolution image generation, both the generator and discriminator architectures must be carefully designed. This can involve incorporating advanced architectures

that are effective at handling spatial hierarchy data, such as **convolutional neural networks (CNNs)**.



### What are CNNs?

CNNs are a cornerstone of machine learning for processing spatial data, such as images. They identify patterns using convolutional filters, excelling in tasks that require an understanding of spatial hierarchies. This makes CNNs indispensable in many GAN applications for image generation and recognition.

In image-based GAN applications, the generator uses techniques to expand latent representations into detailed images, while the discriminator applies methods to reduce the dimensionality of the input image to assess its authenticity efficiently. The latent dimension, serving as the seed to generate new data instances, is a compact, high-dimensional space, encapsulating potential data variations in a compressed format.

The following code shows the process for building the core structure of a simplified GAN for images, using Python, with the key steps described here:

1. Import the libraries needed to build the generator and discriminator:

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense
```

2. Define the generator model, which takes a latent space vector and produces a 28x28 image through a series of dense layers:

```
def build_generator(latent_dim):
    model = Sequential([
        Dense(128, activation='relu', input_dim=latent_dim)
        Dense(256, activation='relu'), # expands representation
        Dense(784, activation='tanh') # produces 28x28 image
    ])
    return model
```

3. Define the discriminator model, which takes an image and classifies it as real or generated through a series of dense layers:

```
def build_discriminator(input_shape):
    model = Sequential([
        Dense(256, activation='relu', input_shape=(input_shape))
        Dense(128, activation='relu'), # reduces dimension
        Dense(1, activation='sigmoid') # classifies input
    ])
    return model

Define the latent dimension and initialize the models:
latent_dim = 100 # size of input noise vector
generator = build_generator(latent_dim)
discriminator = build_discriminator(784) # for 28x28 image
```

Further technical considerations are necessary to address common challenges, such as limited output diversity and ensuring that the generated data is varied while still closely mirroring the real data distribution. These considerations include the choice of the activation function (`relu` for non-linear transformations), techniques to ensure consistent input distribution across layers, and strategies such as randomly omitting units during training to prevent overfitting.

For more details on this, you can refer to the recent paper on GANs:

<https://www.researchgate.net/publication/380573076>

[Understanding GANs fundamentals variants training challenges applications and open problems.](#)

## Variational autoencoders

VAEs present a different approach to generative modeling as compared to GANs. VAEs offer a probabilistic way of learning latent representations of data and consist of two main components:

- The encoder compresses input data into a latent space representation
- The decoder reconstructs the data from this latent space

Unlike traditional autoencoders, VAEs introduce a probabilistic twist where, rather than encoding input as a single point, they encode it as a distribution over the latent space.

### The versatile applications of VAEs



VAEs are instrumental in understanding and modeling complex distributions of data. One key area where VAEs excel is in data imputation, where they can predict missing information or forecast future trends in time-series data.

The loss function for VAEs combines reconstruction loss with the **Kullback-Leibler (KL)** divergence between the learned latent variable distribution and the prior distribution. The reconstruction loss measures how well the generated outputs match the original inputs, ensuring that a model creates accurate reproductions of the data, and the KL divergence acts as a form of regularization during training that ensures the model learns efficient and meaningful data representations. This regularization prevents the model from overfitting by encouraging it to generate outputs that are not just

accurate but also generalize well to new, unseen data. By combining these two components, the VAE learns to produce high-quality, diverse outputs and encourages the model to learn efficient encodings of the data.

The following is a simplified example of constructing a VAE using Keras:

1. We will use a flattened 28x28 image input as another example, and we will first sample from the latent distribution in the `sampling()` function:

```
from tensorflow.keras.layers import Input, Lambda
from tensorflow.keras import Model, backend as K
def sampling(args):
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon
```

2. We then use the encoder to map the inputs into latent space:

```
inputs = Input(shape=(784,)) # Encoder
h = Dense(256, activation='relu')(inputs)
z_mean = Dense(2)(h)
z_log_var = Dense(2)(h)
z = Lambda(sampling, output_shape=(2,))([z_mean, z_log_var]
```

3. We then use the decoder to reconstruct the image from the latent space:

```
decoder_h = Dense(256, activation='relu') # Decoder
decoder_mean = Dense(784, activation='sigmoid')
h_decoded = decoder_h(z)
x_decoded_mean = decoder_mean(h_decoded)
vae = Model(inputs, x_decoded_mean)
```

For the effective application of VAEs, selecting the right architecture for the encoder and decoder is crucial, often involving densely connected layers for basic tasks or more sophisticated structures, like CNNs, for image data. The dimensionality of the latent space is also vital – it must be large enough to capture relevant data variations but not so large that it leads to overfitting or meaningless reconstructions. When designed correctly, VAEs offer a principled approach to generative modeling, balancing the need for accurate data reconstruction with the flexibility to generate new, diverse samples from learned data distributions.

## **Long short-term memory networks**

LSTMs are a specialized type of RNN designed to learn long-term dependencies in sequential data. RNNs are a class of neural networks that include loops, allowing information to persist by passing it from one step of the network to the next. This looping mechanism makes RNNs suitable for processing sequences of data such as time series or text. However, standard RNNs often struggle with learning long-range dependencies, due to issues like vanishing and exploding gradients. This occurs because, during backpropagation, gradients can become exponentially small (vanish) or large (explode), making it difficult to update the network weights effectively. LSTMs address these challenges with a more complex internal structure that allows them to remember information for longer periods effectively.

The defining feature of LSTMs that enables them to remember information more effectively is their cell state, along with three types of gates: input, output, and forget gates. These components work together to regulate the flow of information, allowing a network to remember important information over long periods and to forget irrelevant data.



## Key components of an LSTM

Core to the LSTM are its cell state and gates with the following functions:

- **Input gate:** How much new information to store in the cell state
- **Forget gate:** What information is discarded from the cell state
- **Output gate:** The output of the cell state to the next layer

The following code sets up a simple LSTM network for time-series prediction that could be used to predict the next day's sales, based on different features from the previous day. In this architecture, we allow for 10 days to be captured by `sequence_length` and then 5 features by `num_feature`, which could include data points such as web traffic or previous sales. The LSTM layer with 50 units learns to recognize patterns in the sequence data, while the `dense` layer outputs the prediction. Finally, the model is compiled with the Adam optimizer and mean squared error (`mse`) loss function, common choices for regression tasks:

```
from tensorflow.keras.layers import LSTM
sequence_length = 10
num_features = 5
model = Sequential([
    LSTM(50, activation='relu', input_shape=(sequence_length, n),
         Dense(1)
])
model.compile(optimizer='adam', loss='mse')
```

To demonstrate the training process, we can generate synthetic time-series data that mimics sales prediction data. The synthetic data includes a base sine wave with added noise, simulating daily patterns with random fluctuations:

```
import numpy as np
import matplotlib.pyplot as plt
def generate_synthetic_data(num_samples, sequence_length, num_features):
    X = []
    y = []
    for i in range(num_samples):
        base = np.array([np.sin(x) for x in range(sequence_length)])
        features = np.tile(base, (num_features, 1)).T + np.random.normal(0, 0.1)
        target = np.sum(base) + np.random.normal(0, 0.1)
        X.append(features)
        y.append(target)
    return np.array(X), np.array(y)
X_train, y_train = generate_synthetic_data(100, sequence_length, num_features)
model.fit(X_train, y_train, epochs=10, verbose=1)
```

After training, we can evaluate the model on test data by comparing the predicted sales values with the actual values:

```
X_test, y_test = generate_synthetic_data(10, sequence_length, num_features)
y_pred = model.predict(X_test)
plt.plot(y_test, label='Actual')
plt.plot(y_pred, label='Predicted')
plt.xlabel('Days')
plt.ylabel('Sales Prediction')
plt.title('Actual vs Predicted Sales Over Time')
plt.legend()
plt.show()
```

The code produces the following plot, showing how a simple LSTM model with training can quickly provide useful predictions for metrics, such as sales, if given appropriate input features:

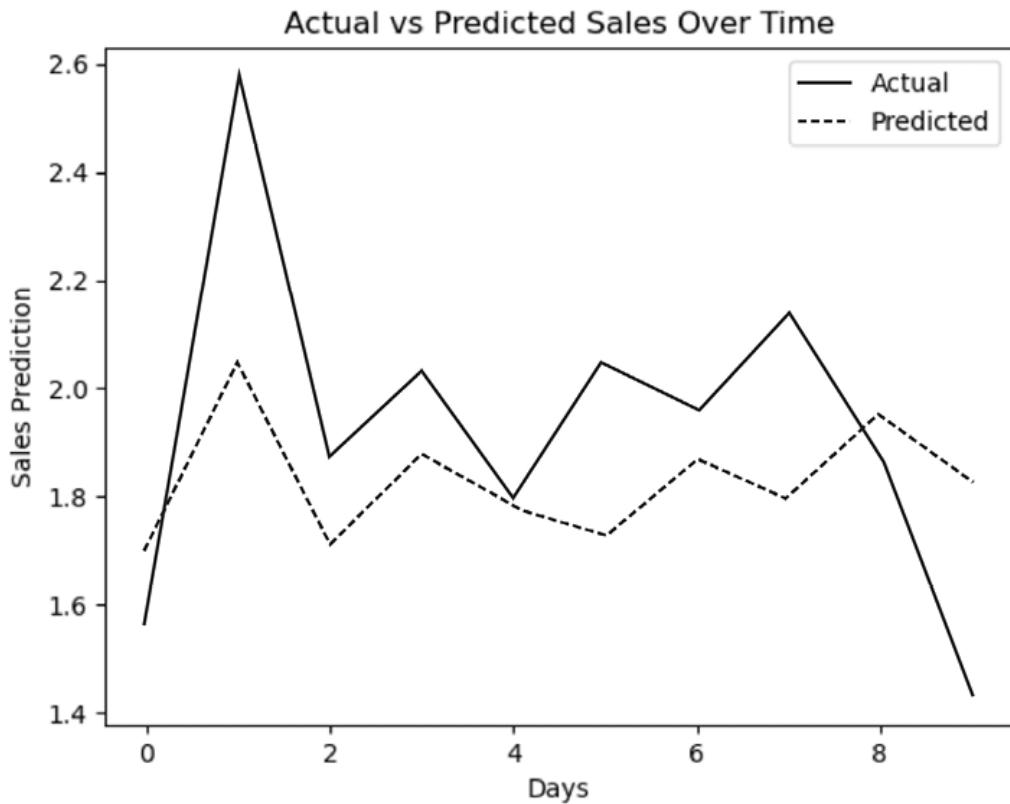


Figure 9.2: Output of an LSTM sales prediction model

## Transformers

Transformer-based models, such as the GPT series, have revolutionized natural language processing by introducing a model architecture that excels in capturing context and relationships within data. The core innovation of transformer models is the attention mechanism, which enables a model to weigh the importance of different parts of the input data differently. This mechanism allows GPT and similar models to understand the context and generate text that is coherent and contextually relevant.

GPT models leverage the transformer architecture for generative tasks, trained on vast amounts of text data to understand language patterns, grammar, and context. This pre-training enables GPT models to generate text that is highly coherent and contextually relevant to the input prompts. While building a GPT from scratch is a formidable task, at the conceptual level, there are some key components within a GPT architecture:

- **Embedding layer:** Converts token indices to dense vectors of fixed size
- **Multi-head attention:** Allows a model to focus on different parts of the input sequence simultaneously, capturing various contextual relationships
- **Layer normalization and residual connections:** Help stabilize and optimize the training process, ensuring that gradients flow smoothly throughout a network

The following code shows how a simplified GPT-like architecture can be created using Keras with the aforementioned components:

```
from tensorflow.keras.layers import Embedding, LayerNormalization
def simplified_gpt_model(vocab_size=10000, embed_dim=256, max_length=100):
    inputs = Input(shape=(max_length,)) # input layer for sequence
    embedding_layer = Embedding(input_dim=vocab_size, output_dim=embed_dim)
    # self-Attention layer with multiple heads
    attn_output = MultiHeadAttention(num_heads=num_heads, key_dim=key_dim)(inputs)
    # normalization and residual connection for the attention output
    attn_output = LayerNormalization(epsilon=1e-6)(attn_output + inputs)
    # feed Forward network to processes attention output
    ff_network = Dense(ff_dim, activation="relu")(attn_output)
    ff_network_output = Dense(embed_dim)(ff_network)
    # second normalization and residual connection
    sequence_output = LayerNormalization(epsilon=1e-6)(ff_network_output + inputs)
    # output layer to predict the next token in the sequence
    outputs = Dense(vocab_size, activation="softmax")(sequence_output)
    model = Model(inputs=inputs, outputs=outputs)
```

```
    return model
gpt_model = simplified_gpt_model()
gpt_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
```

 **The power of self-attention in GPT**

Self-attention, the key innovation behind GPT models, allows a network to weigh the importance of different words in a sentence, enhancing its understanding of context and relationships between words.

To learn more, check out the paper *Attention Is All You Need* by Illia Polosukhin et al. (<https://arxiv.org/pdf/1706.03762.pdf>).

## When GenAI is the right fit

While GenAI brings new realms of possibility to content creation and digital marketing, understanding its limitations is also crucial. GenAI shines in environments that demand innovation, creativity, and the ability to scale personalized content dynamically. More generally, GenAI can be a great fit for your marketing campaign in the following cases:

- **Brainstorming for creative campaigns:** Generating unique and compelling content, be it text, image, or video, to facilitate brainstorming for creative marketing campaigns that stand out in a crowded digital landscape. For example, we will use GenAI to generate text for a new product launch in this chapter and *Chapter 10*.
- **Dynamic content personalization:** Enabling marketers to tailor content at scale while still addressing individual user preferences and

behaviors, in order to increase engagement and conversion rates. For instance, we will show how GenAI combined with **retrieval-augmented generation (RAG)** can be used to create personalized recommendations and email content, based on individual browsing history and purchase behavior, in *Chapter 11*.

- **Efficiency in content production:** Automating the content generation process, significantly reducing the time and resources needed to produce marketing materials.

However, GenAI isn't a one-size-fits-all solution, particularly in marketing scenarios where accuracy and deep contextual understanding are needed. For example, its application in highly regulated industries or in sensitive campaigns around social issues, where a misstep can significantly impact a brand's reputation, must be done with caution. While GenAI can help with brainstorming in these cases, the unpredictability of GenAI content could pose significant risks if deployed without careful monitoring. Further discussion of this topic will be presented in *Chapter 13*, along with strategies to improve its contextual understanding in *Chapters 10 and 11*.

### GenAI in highly regulated industries



When applying GenAI in sectors like healthcare, financial services, insurance, and legal services, adherence to stringent regulatory standards is crucial. Marketing content for these applications requires extra scrutiny, as they must not only be accurate and transparent but also align with industry-specific compliance measures.

# Introduction to pre-trained models and ZSL

Building on the foundations of GenAI discussed in the chapter so far, we will now introduce some core concepts related to pre-trained models and **zero-shot learning (ZSL)**. These concepts underly how models can take vast amounts of existing data to create realistic, new outputs for scenarios that have not yet been encountered, with little to no additional training. With a focus on text data, we will discuss how contextual embeddings and semantic proximity are two key concepts that facilitate this capability. With this knowledge, you will be equipped to understand and apply these concepts in this chapter and the ones to come.

## Contextual embeddings

Contextual embeddings, enabled by advancements such as the LSTM and GPT models discussed earlier, are fundamental to how **large language models (LLMs)** interpret and generate language. As discussed in *Chapter 5*, embeddings are dense vector representations of data that capture key features in a high-dimensional space. Early models like Word2Vec and GloVe generate static embeddings where the same word always has the same vector. In contrast, advanced models like BERT and GPT create contextual embeddings, where word representations change based on their usage in context. Effective NLP embeddings preserve the semantic relationships of the original data, meaning similar vectors are closer together in vector space. This adaptability is foundational for applications such as ZSL, which rely on a model's ability to apply learned knowledge to new tasks without specific training data.

Earlier in the chapter in the exploration of GenAI’s probabilistic nature, we noted how text generation is analogous to sequence prediction in video or audio, in that the relevance of each piece of data depends on its predecessors. As an analogy, consider how Google’s auto-suggest feature adapts suggestions based on the context of the words already entered. This same concept underpins the transformative potential of models like BERT, which analyzes text from both preceding and subsequent contexts to enhance language comprehension and prediction accuracy, via their contextual embeddings.

GPT models take it a step further and adopt an autoregressive framework. The term “autoregressive” means that a model makes predictions based on its own previous outputs, meaning that it anticipates the subsequent word based on all preceding outputs of the model as context. For example, when developing a content calendar for a marketing blog, a GPT model can analyze past articles and trending topics to suggest new posts that align with the brand’s voice and audience interests. This is unlike transformer-based models, which can look at both preceding and following words simultaneously, as discussed earlier in the chapter. However, such autoregressive models can offer more nuanced text generation, enabling them to create narratives with a level of coherence that bidirectional models may not achieve as seamlessly.

### The importance of contextual embeddings

Contextual embeddings from LSTM or GPT models allow for nuanced understanding by evaluating more of the text in its entirety.





For instance, in filling in the blank in “The stormy seas calmed as the \_\_\_ sailed into the harbor,” a model leveraging both prior and subsequent context could infer “ship” with greater accuracy, whereas a more naive model with only prior context might inaccurately predict the word “day.”

## Semantic proximity

Transitioning from our discussion on contextual embeddings and their critical role in language models, we will now explore **semantic proximity**. Contextual embeddings not only enhance the understanding of text by considering its dynamic contexts; they also serve as a fundamental tool in evaluating the semantic relationships between words or phrases within that text. This nuanced understanding is pivotal when we examine the concept of semantic proximity, which involves quantifying how closely related or distant two linguistic items are in meaning.

For example, consider the phrases “limited-time offer” and “exclusive deal.” These phrases have close semantic proximity because they both relate to targeted promotions for potential customers. Conversely, the phrases “limited-time offer” and “customer feedback” would have a much larger semantic distance.

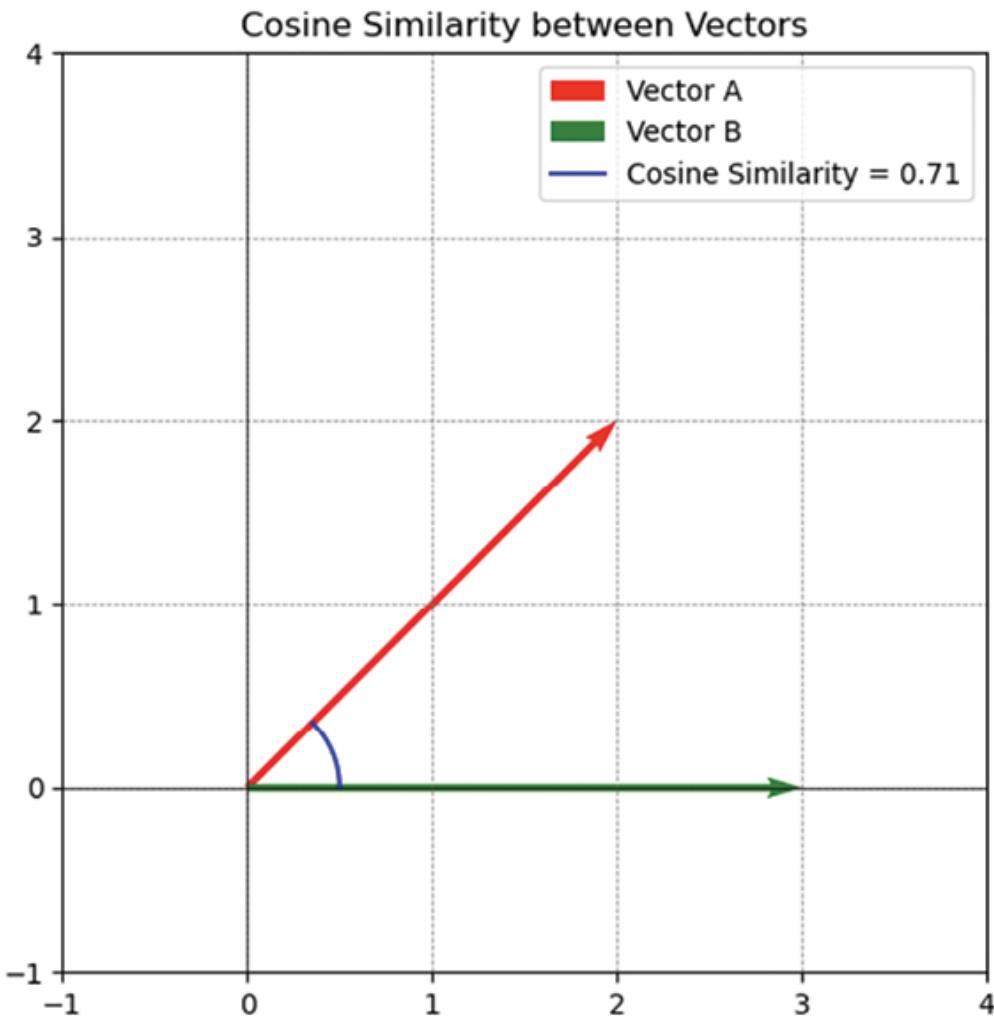
Semantic proximity is effectively assessed through methods such as cosine similarity, which measures the angle between vectors representing these items in a high-dimensional space. This metric, rooted in the geometry of vector spaces, provides a clear, mathematical way to capture and compare the meanings encoded by embeddings. Mathematically, cosine similarity is given by:

$$\text{Cosine Similarity} = A \cdot B / \|A\| \|B\|$$

where  $A$  and  $B$  are vectors (semantic embeddings for a word, phrase, document, etc),  $A \cdot B$  denotes their dot product, and  $\|A\|$  and  $\|B\|$  represent their magnitudes. The value of cosine similarity ranges from -1 to 1. A value of 1 implies that the vectors are identical in direction, indicating maximum similarity. A value of 0 implies that the vectors are orthogonal, indicating no similarity.

While, in practice, many sophisticated text embeddings are high-dimensional and can range from hundreds to thousands of dimensions, we can more easily illustrate the concept of cosine similarity in 2D space via two vectors. In the following code, we illustrate the calculation of cosine similarity using vectors  $A$  and  $B$ , with the angle associated with their cosine similarity:

```
A, B = np.array([2, 2]), np.array([3, 0])
cos_sim = np.dot(A, B) / (np.linalg.norm(A) * np.linalg.norm(B))
angle = np.arccos(cos_sim)
plt.figure(figsize=(6,6))
for vector, color, label in zip([A, B], ['red', 'green'], ['Vector A', 'Vector B']):
    plt.quiver(0, 0, vector[0], vector[1], angles='xy', scale_units='xy', scale=1, color=color)
plt.plot(0.5 * np.cos(np.linspace(0, angle, 100)), 0.5 * np.sin(np.linspace(0, angle, 100)))
plt.axis([-1, 4, -1, 4])
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.title('Cosine Similarity between Vectors')
plt.legend()
plt.show()
```



*Figure 9.3: Visualization of cosine similarity between two vectors, A and B, showing their angular relationship in 2D space*

Continuing from the mathematical foundation of cosine similarity, we can apply this same concept to explore the polysemous nature of words in different contexts. Consider the word “light” used in two different sentences:

*He turned on the **light** to read.*

*The **light** fabric was perfect for summer.*

These sentences demonstrate different semantic instances of the word “light,” and by employing contextual embedding models like BERT, we can quantify the semantic differences of “light” in these cases using cosine similarity.

As an example, in the following code, we:

1. Import libraries and load the pre-trained BERT model and tokenizer.
2. Tokenize the sentences, converting them into the format expected by the BERT model.
3. Pass the tokenized sentences through the BERT model to obtain the embeddings.
4. Extract embeddings for the word “light” by finding the index of the word in each tokenized sentence and extracting its corresponding embedding from the model output.
5. Compute cosine similarity and print the result.

To do this, use the following code:

```
import tensorflow as tf
from transformers import BertTokenizer, TFBertModel
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = TFBertModel.from_pretrained('bert-base-uncased')
sentence_1 = "He turned on the light to read."
sentence_2 = "The light fabric was perfect for summer."
tokens_1 = tokenizer(sentence_1, return_tensors="tf")
tokens_2 = tokenizer(sentence_2, return_tensors="tf")
outputs_1 = model(tokens_1)
outputs_2 = model(tokens_2)
light_index_1 = tokens_1['input_ids'][0].numpy().tolist().index(0)
light_index_2 = tokens_2['input_ids'][0].numpy().tolist().index(0)
embedding_1 = outputs_1.last_hidden_state[0, light_index_1]
embedding_2 = outputs_2.last_hidden_state[0, light_index_2]
cosine_similarity = tf.keras.losses.cosine_similarity(embedding_1, embedding_2)
```

```
cosine_similarity = -cosine_similarity.numpy()
print("Cosine similarity between 'light' embeddings in the two sentences")
```

```
Cosine similarity between 'light' embeddings in the two sentences
```

A cosine similarity value ranges from -1 to 1. As mentioned earlier, a value of 1 indicates identical vectors, while a value of 0 implies orthogonal vectors with no similarity. In this case, a cosine similarity of 0.48 suggests that the embeddings for “light” in the two sentences are somewhat similar but not identical.

## Pre-trained models

Pre-trained models are machine learning algorithms that have been previously trained on large datasets to perform general tasks, such as understanding natural language or recognizing objects in images. The utility of pre-trained models is fundamentally rooted in their use of the embeddings they were trained on. For text, these embeddings not only enable models to grasp context dynamically but also serve as the foundation to adapt these models for specific tasks, such as sentiment analysis, as discussed in *Chapter 5*, with minimal to no additional training. This adaptability is critical for applications such as ZSL.

The advent of pre-trained models democratized access to state-of-the-art AI by offering a base model that can be fine-tuned or used for inference directly. These models not only reduce the cost and time to deployment for AI-driven solutions but also the need for computational resources and energy consumption, making AI both more accessible and environmentally friendly. In the marketing domain, pre-trained models offer significant

advantages. They enable marketers to quickly deploy advanced AI solutions for tasks such as personalized content creation, customer sentiment analysis, and targeted advertising.

Sophisticated AI models that were previously attainable only for large or highly specialized technology companies are now a possibility for smaller-sized businesses, and even individual consumers, at a fraction of the cost. For perspective, training a model such as GPT-3 from scratch was estimated to be in the millions of dollars at the time of its release in 2020, a figure that encapsulates computational costs and human expertise. Today, a user can perform inference to generate text using this same (or a more advanced) model through the company's API, at the cost of a few cents for hundreds of words of content.

The key components of a pre-trained model are:

- **Weights:** They represent the learned parameters from training datasets and encode a wide range of knowledge and patterns needed for transfer learning
- **Architecture:** The model's structure, detailing how inputs are processed through various layers to generate outputs
- **Pre-processing steps:** Procedures like tokenization and normalization to ensure data compatibility with the model's training

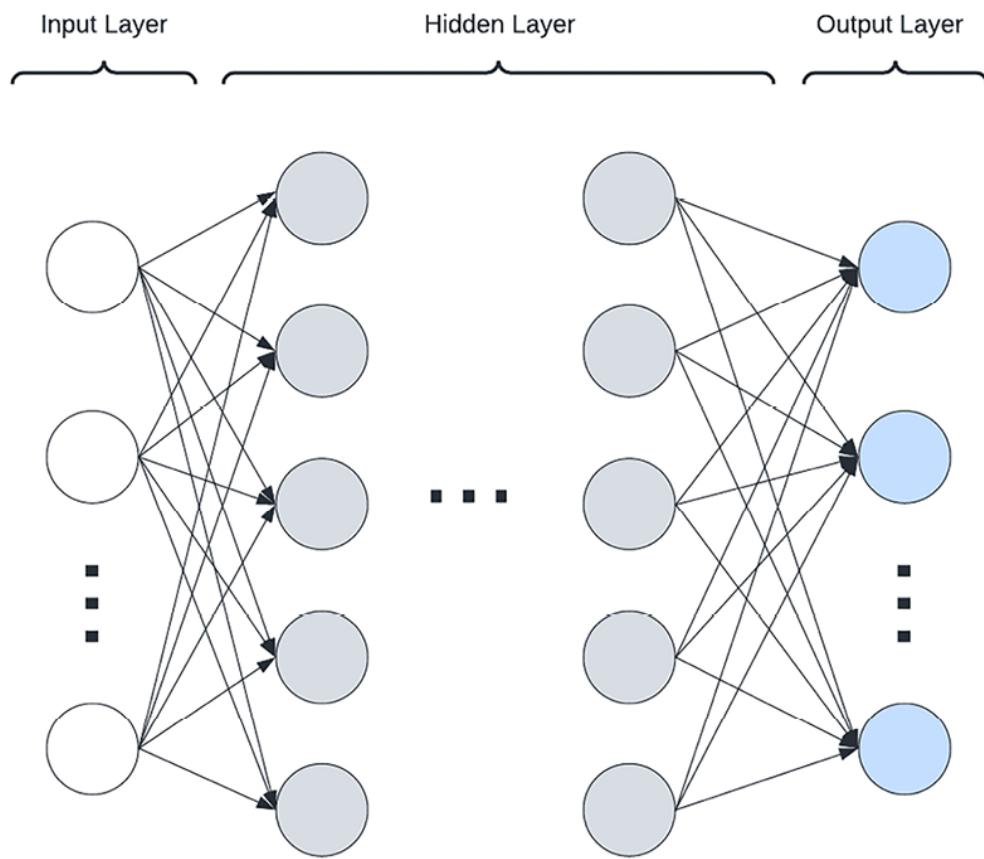
Let's look at these components in detail.

## Model weights

Model weights are at the core of neural networks, including pre-trained models. Weights are the refined parameters developed through extensive training to minimize loss, acting as a repository of a model's learned knowledge. For instance, in language models like GPT, these weights

capture the intricacies of language, such as grammar and context, enabling the generation of text that's not only coherent but also contextually rich. The effectiveness of pre-trained models in tasks like ZSL stems from these weights, which enable you to generalize from the model's training data to new, unseen data with remarkable accuracy.

To better understand where the model weights come from, consider the case of an **artificial neural network (ANN)**, as shown in the following figure:



*Figure 9.4: Example ANN architecture from Chapter 6*

This consists of three main layers: the input layer, the hidden layer, and the output layer. During training, neural networks undergo forward and backward propagation. Forward propagation involves feeding data through a network to generate an output, while backward propagation adjusts weights

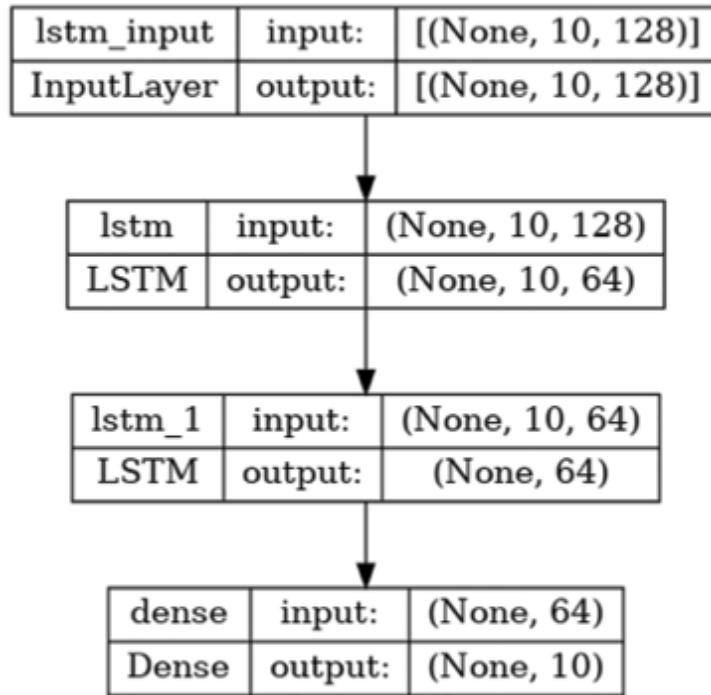
based on the error of the predictions. Through these iterations, the network learns the optimal weights for each neuron, minimizing prediction error and enhancing a model's performance.

## Model architecture

The architecture of a pre-trained model goes hand in hand with its weights. It delineates the structure of how data is processed and transformed across the model's layers and also guides the adaptability of the model to perform novel tasks. For example, deeper language model architectures might be better suited for complex reasoning tasks, while models with specially configured attention mechanisms can offer finer control over the focus of the model during inference. For image recognition, the intermediate representations produced by these models, such as features extracted at various layers by a pre-trained CNN, can also serve as a valuable starting point for classification tasks.

When understanding a machine learning model's architecture, it can be helpful to plot it and its parameters to visualize its key aspects. Here, we illustrate this using Keras's `plot_model()`, as a demonstration of a simple LSTM model composed of two LSTM layers:

```
!conda install pydot
!conda install graphviz
from tensorflow.keras.utils import plot_model
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(10, 128)),
    LSTM(64),
    Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy',
plot_model(model, show_shapes=True, show_layer_names=True)
```



*Figure 9.5: A simple LSTM-based neural network architecture*

This visualization clearly delineates the model's structure, showing the progression from input through two LSTM layers, each with 64 units (or neurons), to a final dense output layer configured with 10 softmax units for class prediction. Based on the labels given in the figure, the structure can be further broken down as follows:

- **Input layer (`lstm_input`)**: Takes in data with a shape of `(10, 128)`, processing sequences of 10 timesteps, each with 128 features
- **First LSTM layer (`lstm`)**: Contains 64 units and returns sequences, processing the input and passing on sequences of the same length to the next LSTM layer
- **Second LSTM layer (`lstm_1`)**: Also has 64 units but does not return sequences, compressing the output from the first LSTM layer into a single vector

- **Dense output layer (dense):** The final layer is a dense layer with 10 units and a softmax activation function, outputting a probability distribution over 10 classes (categories or labels)

## Preprocessing steps

Preprocessing steps are crucial for ensuring that data is compatible with a pre-trained model's training procedure. Here are a couple of examples:

- NLP **tokenization** breaks down text into words or subwords that are consistent with what an LLM model expects
- For images, **normalization** can adjust image pixel values to a common scale to facilitate a model's ability to learn from and generate predictions for data

These preprocessing steps are essential for leveraging pre-trained models effectively, ensuring that input data mirrors the form of the data that the model was trained on.

## Zero-shot learning

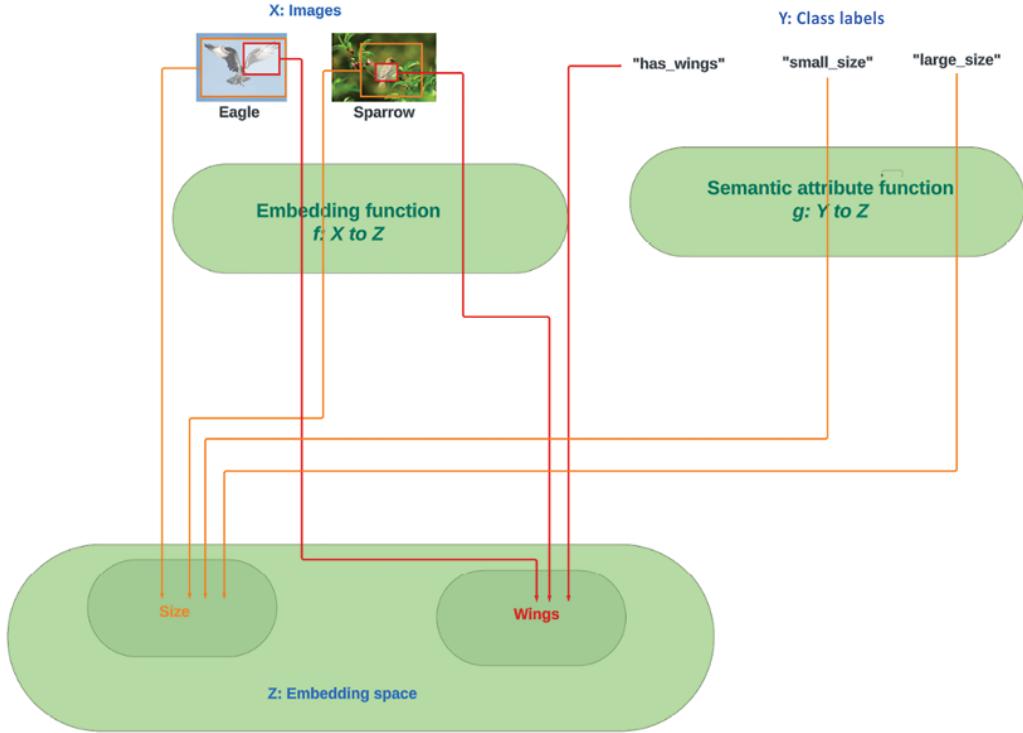
Following our exploration of pre-trained models and their fundamental components, we will now shift our focus to their application in ZSL. As we will discuss later, the fundamentals behind ZSL allow marketers to dynamically generate relevant content and even target their marketing campaigns to individual consumers in near real time. Before we get to those examples, this section will provide some background on ZSL and how it enables models to apply learned knowledge and infer information about tasks or classes that were not explicitly covered during their training. This capability extends the utility of pre-trained models, allowing them to generalize across unseen data and scenarios.

# Mechanics of learning and prediction

At its core, ZSL operates by using transformations and mappings of the input and output in a high-dimensional embedding space, using two primary functions:

- An **embedding function**  $f: X \rightarrow Z$  that transforms input – such as images or text – into feature vectors within the embedding space.
- A **semantic attribute function**  $g: Y \rightarrow Z$  that associates class labels with semantic attributes in the same embedding space. These attributes describe classes in terms of universal, distinguishable features, existing within an attribute space  $A$ .

As a simplified example, take a case where ZSL is applied to distinguish between animals. The embedding function  $f: X \rightarrow Z$  would transform visual images of the animal into high-dimensional feature vectors within the embedding space – for example, taking images of a sparrow and an eagle that are processed to show distinct feature vectors, representing their visual characteristics. Simultaneously, the semantic attribute function  $g: Y \rightarrow Z$  maps class labels, like *sparrow* and *eagle*, to vectors in the same embedding space based on semantic attributes. Attributes for *sparrow* might include `['small_size', 'brown_color', 'has_wings']`, whereas *eagle* could be characterized by `['large_size', 'sharp_beak', 'has_wings']`. These attributes are then quantified, where `small_size` might be encoded as `0.2` on a size scale, and `large_size` as `0.8`.



*Figure 9.6: Example mappings from an input image or class label to an embedding space*

By placing both images and class attributes within the same embedding space, as shown in *Figure 9.6*, a model can match an image's feature vector to the closest class attribute vector. This matching is facilitated even if the model has never seen an image of a sparrow or eagle during training, by recognizing the overlap in their attributes with those of known classes. To find the best class match, the algorithm is tasked with the optimization of a compatibility function, which quantifies the match between an input's feature vector and a class's attribute vector. The ZSL prediction then involves selecting the class that maximizes compatibility for an unseen instance.

## Output parameters

There are a number of key parameters that can be specified to influence the output of ZSL models. These parameters tailor the output by adjusting the

behavior of the model during the sampling process. Three of the most common ones used in the context of text generation with models like GPT include:

- Temperature
- Top P (nucleus sampling)
- Frequency penalty

By influencing how the model's outputs are sampled from the probability distribution generated by the compatibility function, each of these parameters allows adjustments in the creativity, coherence, and diversity of the model's outputs. The following sections provide further detail on the theory underlying each of these parameters.

## Temperature

The temperature parameter plays a crucial role in determining the level of randomness or confidence in the prediction distribution. Mathematically, adjusting the temperature modifies the `softmax` function used to calculate the probabilities of the next word in the following manner:

$$\text{Softmax Temperature Adjusted} = e^{(\text{logit}/T)} / \sum e^{(\text{logit}/T)}$$

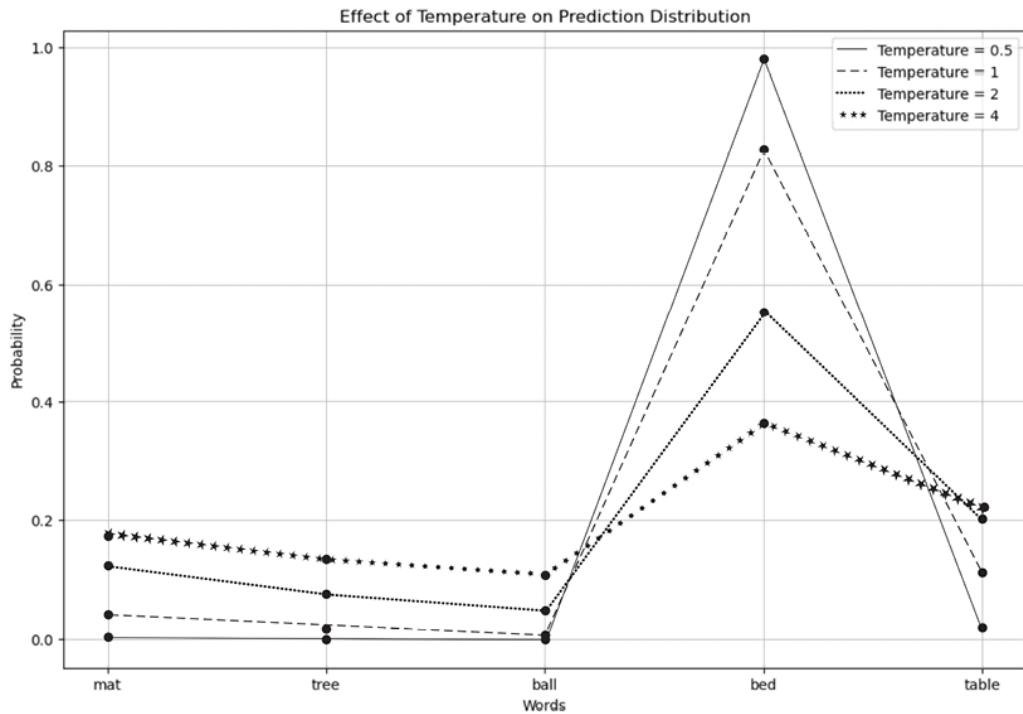
where  $T$  is the temperature and logit represents the raw outputs from the model. A lower temperature sharpens the distribution, making the model's predictions more deterministic and less varied, whereas a higher temperature flattens the distribution, encouraging diversity in the predictions at the cost of potentially introducing less coherence, as the model becomes more likely to sample less probable words.

To illustrate the concept, let's consider a predictive text generation scenario of the next word in the sentence “The cat sat on the \_\_\_\_.” For simplicity,

assume our model considers five possible completions: “mat,” “tree,” “ball,” “bed,” and “table,” with the initial logits reflecting their probabilities (assigned manually in this case for simplicity). We can use the following code to produce a visualization, demonstrating how changing the temperature parameter  $T$  affects the softmax function, altering the probability distribution of these potential next words:

```
logits = np.array([2, 1, 0.1, 5, 3])
labels = ['mat', 'tree', 'ball', 'bed', 'table']
def softmax_temperature_adjusted(logits, temperature):
    exp_logits = np.exp(logits / temperature)
    return exp_logits / np.sum(exp_logits)
temperatures = [0.5, 1, 2, 4]
plt.figure(figsize=(12, 8))
for T in temperatures:
    probabilities = softmax_temperature_adjusted(logits, T)
    plt.plot(labels, probabilities, marker='o', label=f'Temperature {T}')
plt.title('Effect of Temperature on Prediction Distribution')
plt.ylabel('Probability')
plt.xlabel('Words')
plt.legend()
plt.grid(True)
plt.show()
```

This gives us the following output:



*Figure 9.7: Visualization of the impact of the temperature parameter on softmax probability distributions for potential next words in the sentence “The cat sat on the \_\_\_”*

As illustrated in the graph, at lower temperatures the distribution is sharper, concentrating the probability mass on fewer, more likely outcomes, like “bed.” As the temperature increases, the distribution becomes flatter, giving a higher probability to a broader set of potentially less likely word outcomes, such as table or mat.

## Top P

Top P, or **nucleus sampling**, offers a dynamic way to focus the generation process on the most plausible set of outcomes. Instead of considering the entire vocabulary, the model limits its choices to the smallest set of words whose cumulative probability exceeds the threshold P.

This approach can be thought of as dynamically adjusting the breadth of consideration based on the model’s confidence, according to the formula:

$$\text{Cumulative Probability} = \sum_{i=1}^N P(w_i) > P$$

where  $N$  is the number of words considered and  $P(w_i)$  is the probability of the  $i^{\text{th}}$  word. This technique helps maintain a balance between variety and relevance, ensuring that the generated text remains plausible without being overly constrained by the most likely predictions.

## Frequency penalty

The **frequency penalty** addresses the tendency of models to repeat the same words or phrases, enhancing the diversity of the output. It modifies the probability of each word based on its previous occurrences in the generated text:

$$\text{Adjusted Probability} = P(w_i) - \text{Frequency Penalty} \times \text{Occurrence}(w_i)$$

where  $P(w_i)$  is the original probability of the word  $w_i$  and  $\text{Occurrence}(w_i)$  is the number of times  $w_i$  has appeared in the text so far. This adjustment encourages the exploration of new vocabulary and ideas by penalizing words that the model has already used, promoting a richer and more varied output.

## ZSL for marketing copy

We will now discuss the practical application of ZSL through the example of an e-commerce brand that is launching new lines of eco-friendly kitchenware and fashion products. Traditionally, creating compelling product descriptions and promotional content could require significant research and creative effort from writers familiar with a brand's tone and the

intricacies of sustainable design. However, with ZSL, the brand can input a concise description of the product line, emphasizing keywords like “sustainable” and “eco-friendly activewear,” into a pre-trained model, immediately producing an output of brand-appropriate content that is ready for consideration across digital platforms. By automating these initial stages of content generation, the brand can now focus more on higher value efforts like strategy, engagement, and analyzing the effectiveness of its marketing efforts.

In general, to integrate ZSL effectively into your marketing strategy, consider the following iterative process as a template:

1. Define your content goals and the key messages you want to communicate.
2. Create concise prompts that encapsulate these goals and messages, incorporating relevant keywords and themes.
3. Experiment with different parameters (`temperature`, `Top P`, and `frequency penalty`) to adjust the style and diversity of the generated content.
4. Generate multiple content variations to explore different angles and ideas.
5. Review and refine the output, selecting the best options that align with your brand’s tone and objectives.
6. Use the generated content as a starting point for further customization and optimization, based on audience feedback and performance metrics.

In the following subsections, we will focus on steps 1–4 of the preceding workflow, with steps 5 and 6 covered by examples in the chapters that will follow.

# Preparing for ZSL in Python

To demonstrate the Python setup process for ZSL, this section will walk through the basic steps of using both freely available, open-source models, as well as more advanced models requiring paid API-based implementations. We will demonstrate the former with models available in the Hugging Face Transformers library, and we will demonstrate the setup for the latter using OpenAI's API service.

## Staying ahead with Hugging Face updates



Hugging Face's Transformers library frequently updates, making advanced models that were previously behind paid API services freely available. For the latest Hugging Face models available, consult their documentation at

<https://huggingface.co/models>.

The basic setup for ZSL tasks using the gpt2 model available on Hugging Face is:

```
from transformers import pipeline
generator = pipeline('text-generation', model='gpt2')
prompt = "Write a compelling product description for eco-friendly"
completion = generator(prompt, max_length=100, num_return_sequences=1)
print("Generated text using HuggingFace's GPT model:", completion)
```

Running this prompt results in an output such as the following:

```
"Your kitchen will save you time and energy. It will save you money and effort."
```

We can contrast this with the output that is returned from a model with the same parameters but using the more advanced text generation capabilities of OpenAI's GPT-4. Executing this model currently requires the creation of an OpenAI account and then generating an API key that can be substituted in the following code:

```
from openai import OpenAI
client = OpenAI(api_key = 'XXX') #insert your api key here
completion = client.chat.completions.create(
    model="gpt-4",
    messages=[{"role": "user", "content": "Write a compelling p
    max_tokens=100, n=1, temperature=0.7)
print(completion.choices[0].message)
```

This results in a response such as:

```
"Introducing our premium range of eco-friendly kitchenware, desi
```

It's evident from these two responses that advanced models like GPT-4 significantly enhance the relevancy and quality of the generated content. This is because, compared to GPT-2, GPT-4 incorporates major advancements in deep learning architectures, larger training datasets, and more sophisticated fine-tuning techniques. For this reason, we will use results obtained from GPT-4 for the remainder of this chapter.

However, the key to leveraging ZSL effectively lies not just in a model's capabilities but also in effectively choosing the input information that shapes the output. The most critical aspects here include creating an effective prompt, as well as setting other parameters such as temperature and Top P, as discussed earlier in the chapter.



### Use the outputs of LLMs with caution

Relying directly on the output of any LLM for critical campaign content without human review can be risky! Always consider treating the initial outputs of GenAI models as a creative starting point, and then iterate and refine them as needed to ensure alignment with your brand's voice and objectives.

## Creating an effective prompt

Creating an effective prompt is the most crucial step in leveraging ZSL for marketing copy. In ZSL, the prompt effectively becomes the instruction manual for a model, telling it what kind of content to generate, as well as its style, tone, and substance.

The following are some guidelines around how to formulate prompts that will elicit the best possible marketing copy content from the model:

- **Clarity:** Ensure that your prompt is specific about what you want, whether it's a product description, headline, or call to action.
- **Contextual:** Provide sufficient background to guide a model. For eco-friendly products, mention key selling points like sustainability or biodegradability.
- **Creative:** While clarity is crucial, leaving room for creativity can yield surprising and innovative results. Phrases like “Imagine...” or “Create a story where...” can be particularly powerful.
- **Concise:** Lengthy prompts can dilute the focus. Aim for brevity while including essential details, ensuring that a model stays on topic.

In the following sections, we will illustrate the impact of prompt quality through examples, with different types of marketing copy. While good prompts elicit detailed, relevant, and engaging content, poor prompts can lead to vague and uninspiring outputs. To generate these responses, we will define the following function:

```
def generate_response(prompt, model="gpt-4", max_tokens=100, temperature=0.7):
    response = client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": prompt}],
        max_tokens=max_tokens,
        n=n,
        temperature=temperature)
    return response.choices[0].message.content
```

This function will be used with different prompt types in the examples that follow.

## Example 1: Product descriptions

In this example, we will generate product descriptions for our e-commerce brand, which is launching new lines of eco-friendly kitchenware.

The following is an example of a poor prompt:

```
poor_product_description = "Talk about bamboo cutlery."
generate_response(poor_product_description)
```

This produces:

```
Bamboo cutlery is a type of eating utensil made from bamboo, a 1
```

Now, let's look at the following example of a good prompt:

```
good_product_description = "Write a captivating description for  
generate_response(good_product_description)
```

This prompt produces the following output:

```
Discover the perfect harmony of sophistication and sustainabilit
```

From a marketing perspective, this example demonstrates the significance of creating detailed and audience-specific prompts with clear requirements when using ZSL for product descriptions, as well as how this leads to more specificity in the generated response. However, it is worth noting that older consumers may value more straightforward, factual information and, therefore, may favor the more generic prompt's response from an engagement standpoint. Tailoring GenAI outputs at the level of the individual consumer can be crucial as well and is a topic discussed in *Chapter 11*.

## Example 2: Blog post titles

In our next example, we will focus on another type of marketing copy by generating blog post titles for our e-commerce brand.

We'll start by generating a poor prompt:

```
poor_blog_title = "Write a title about kitchenware benefits."  
generate_response(poor_blog_title)
```

This produces the following:

## Exploring the Numerous Benefits of High-Quality Kitchenware

Here's an example of a good prompt:

```
good_blog_title = "Create an intriguing title for a blog post highlighting the benefits of high-quality kitchenware."  
generate_response(good_blog_title)
```

This gives us a more engaging result:

```
Unlocking Sustainable Living: Top 5 Benefits of Biodegradable Kitchenware
```

Comparing these prompts for blog post titles illustrates the impact of specificity and audience targeting on content effectiveness, where a specific prompt highlighting biodegradable kitchenware creates content aligned more with sustainability. In contrast, a vague prompt results in a generic title that would fail to differentiate itself amid a sea of similar content. To tailor the language produced by LLMs even further, we can also use **few-shot learning (FSL)**, the topic of the next chapter. Used effectively, FSL can achieve the same specificity in language but in a way that's aligned with a brand's unique voice, in order to distinguish LLM outputs from what other LLMs might produce, even when given the same prompt.

### Navigating topical content with AI

When generating blog posts, understanding your model's training data recency is crucial. Without current data or web search capabilities, you risk creating content based on



outdated trends that lack the necessary context for relevant outputs.

## Example 3: Social media captions

In this example, we will generate an Instagram caption for a post about our e-commerce brand.

Let's look at a poor prompt first:

```
poor_social_media_caption = "Make an Instagram post for kitchenware"
generate_response(poor_social_media_caption)
```

This yields the following:

```
'Spice up your kitchen game with our premium-quality kitchenware'
```

Next, we will look at an example of a good prompt:

```
good_social_media_caption = "Create an engaging and witty Instagram post for kitchenware"
generate_response(good_social_media_caption)
```

This produces:

```
Kiss plastic goodbye and say hello to our chic, eco-friendly kit
```

The difference between these two prompts for Instagram captions illustrates how a specific prompt generates a caption that not only engages with its witty language but also directly appeals to eco-conscious consumers, likely

increasing likes, shares, and comments – all crucial metrics on social platforms. In contrast, the vague prompt results in a generic and broad caption that, while informative, lacks a focused appeal and may fail to capture the attention of certain potential customers looking for eco-friendly products.

## Impact of parameter tuning

While creating an effective prompt lays the groundwork, fine-tuning the model's parameters is equally essential to align the generated content with the desired marketing style. In this section, we will explore how adjusting parameters like temperature and Top P affect the output of a language model. Transitioning from kitchenware, we will demonstrate this by generating marketing slogans for an eco-friendly and sustainable fashion line launch.

We will do this by defining the following Python function, outputting sets of three variants of the slogan for each alteration to one of these parameters:

```
def generate_slogan(prompt, temperature=1.0, top_p=1.0):
    response = client.chat.completions.create(
        model="gpt-4",
        messages=[{"role": "user", "content": prompt}],
        max_tokens=15,
        temperature=temperature,
        top_p=top_p,          n=3)
    return (response.choices[0].message.content, response.choices[1].message.content, response.choices[2].message.content)
```

Let's use the base case, with both `temperature` and `Top P` set to values of `1.0`:

```
generate_slogan(prompt)
```

This produces:

```
('"Dress with Purpose: Eco-Elegance Never Goes Out of Style."',  
'"Style with Substance, Fashion with Consciousness!"',  
'"Dressing the world, Preserving the planet."')
```

Now, let's tweak each parameter and see the output that we get:

- **Temperature:** Increasing temperature makes responses more diverse and creative but potentially less coherent:

```
generate_slogan(prompt, temperature=1.8)
```

This produces:

```
('"Drape Yourself in Earth Love: Stylishly Sustainable Fashion F  
'"Wear the change, leave no trace."',  
'"Leave no carbon footprints, only stylish one - Wear Eco in St
```

Conversely, lowering the temperature results in more predictable and consistent outputs:

```
generate_slogan(prompt, temperature=0.3)
```

This time, the prompt produces the following output:

```
('"Style with Sustainability: Dress Green, Live Clean!"',  
'"Style with Sustainability: Fashion for a Better Tomorrow"',
```

```
'"Style with Sustainability: Fashion for a Future"'')
```

- **Top P:** Reducing `top_p` from its maximum value of 1.0 narrows the possible variety of generated slogans by making the model more conservative, as it tends to select only the most probable outputs:

```
generate_slogan(prompt, top_p=0.4)
```

This produces:

```
('"Style with Sustainability: Fashion for a Better Future"',
'"Style with Sustainability: Fashion for a Greener Tomorrow"',
'"Style with Sustainability: Fashion for a Greener Tomorrow"')
```

Through these examples, we can observe the significant impact that parameter adjustments can have on the nature of generated content. This demonstrates the importance of parameter tuning in creating marketing slogans that are not only relevant and engaging but also tailored to the style and message of a brand.

## Summary

In this chapter, we went on a journey through GenAI, and ZSL, and their transformative potential in marketing content creation. We introduced foundational GenAI concepts and discussed the mechanisms that allow these models to generate text, images, and more, with a particular focus on text generation. Analyzing contextual embeddings and semantic proximity highlighted the nuances that pre-trained models like GPT and BERT bring to

understanding and generating language with remarkable adaptability and accuracy.

Central to our discussion was the application of ZSL in creating marketing copy, which allows brands to generate compelling content without the need for extensive training data. We outlined a strategic process to integrate ZSL into marketing strategies with the help of examples that emphasize the importance of creating clear, contextual, and creative prompts. This step-by-step approach – defining content goals, experimenting with parameters, and refining outputs – enables marketers to harness the power of LLMs effectively. We also learned how adjusting parameters such as temperature and Top P can help fine-tune the creativity, coherence, and diversity of the generated content. These practical insights will help you optimize marketing copy to align with brand messaging and campaign objectives.

Looking ahead, the next chapter progresses into the more advanced territories of few-shot and transfer learning. Building on the ZSL foundation, we will explore how these techniques can further refine GenAI models for on-target messaging. This involves adapting models to new contexts with minimal examples (FSL) and updating them with brand or customer-specific information (transfer learning), ensuring consistency and relevance in the generated content.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)

# 10

## Enhancing Brand Presence with Few-Shot Learning and Transfer Learning

This chapter explores the capabilities of **few-shot learning (FSL)** and its value in enhancing brand presence through tailored marketing strategies. Building on the insights from **zero-shot learning (ZSL)** covered in the previous chapter, we now focus on how FSL, by utilizing a limited set of examples, enables rapid and effective adaptation of AI models to new tasks. This approach is particularly valuable in marketing, where the ability to swiftly adjust content to align with evolving consumer preferences and market trends is crucial.

Initially, we will introduce some of the fundamental concepts of FSL in the context of meta-learning as an underlying technique that facilitates quick learning from small datasets. We will then explore the synergy between FSL and transfer learning using practical examples; while FSL is effective at quickly adapting to new tasks with few examples, transfer learning complements this by utilizing pre-trained models that are fine-tuned for specific tasks, reducing the need for extensive model retraining.

This chapter highlights how combining FSL with related strategies can optimize marketing efforts, making them more responsive and efficient.

This chapter equips you with the understanding to:

- Grasp the core concepts of FSL and its similarities to and differences from transfer learning
- Recognize the practical benefits of employing FSL to adapt quickly to new marketing conditions with limited data
- Apply FSL and transfer learning techniques to real-world marketing scenarios to enhance brand messaging and consumer engagement

## Navigating FSL

In the marketing domain, the agility to quickly tune content strategies to meet the evolving needs of a brand is invaluable. FSL stands out for its capacity to effectively learn and perform tasks with limited input data. While ZSL is designed to work without any specific examples of the new classes during inference, relying on a generalized, abstract understanding of the task derived from previously learned tasks, FSL uses a small number of examples to adapt to new tasks. This adaptation often relies on a more direct application of learned patterns and can be fine-tuned with data, making it particularly effective when some example data is available. This efficiency enables marketers to rapidly test new strategies, such as personalizing email campaigns for different customer segments or quickly adapting social media content to reflect emerging trends, without the long lead times associated with gathering and training on extensive datasets. For instance, a marketing manager might use FSL to generate tailored marketing copy that aligns with a company's rebranding initiative, as we will discuss in the *Applying FSL to improve brand consistency section*, even when there are limited examples of such rebranded content available.

As we further explore the utility of FSL for enhancing brand presence, it's essential to build on our conceptual understanding of **Generative AI** (**GenAI**) introduced in the previous chapter. FSL hinges on the idea that intelligent systems can learn new concepts or tasks with only a small number of training examples, drawing heavily from prior knowledge and the application of sophisticated meta-learning algorithms. This discussion will extend your foundational knowledge of this concept, bridging the gap between high-level concepts and practical applications that are useful for understanding the hands-on examples given later in this chapter.

At its core, FSL is often facilitated by **meta-learning**, an approach that enables models to quickly adapt to new tasks by using learnings from a variety of prior tasks. However, meta-learning is just one possible approach, and others such as metric-based learning, whereby models are trained to compare new instances against a few labeled examples using a learned metric or distance function, can also be used. Given its broad applicability and effectiveness as an FSL approach, let's explore meta-learning further in the next section.

## **Understanding FSL through meta-learning**

FSL often involves meta-learning, or “learning to learn,” whereby models are designed to quickly adapt to new tasks based on learnings from a wide array of previous tasks.

This is particularly crucial in marketing, where consumer behavior and preferences can be dynamic, requiring models that can pivot quickly without extensive retraining. Meta-learning frameworks in FSL train

models on a variety of tasks, enabling them to develop a generalized understanding or an internal representation that can efficiently map to new tasks with minimal additional input. Some of the key components of meta-learning are:

- **Task variety:** Meta-learning algorithms are exposed to a wide variety of learning tasks during the training phase. This exposure helps the model learn more robust features that are not overly specific to one task but are general enough to be applicable across a spectrum of future tasks. This is analogous to a marketing team working across different campaign types like email or social media and learning to identify core elements that predict success regardless of the specific product or audience.
- **Rapid adaptation:** The primary goal of meta-learning is to enable rapid learning on new tasks. This is achieved through the model's ability to fine-tune itself to new conditions with minimal training data. For instance, if a model trained in a meta-learning framework is faced with a new product launch, it can quickly adjust its parameters to optimize marketing content for the product's target demographic based on its prior knowledge of similar products.
- **Optimization techniques:** Meta-learning involves special training schemes such as episodic training, whereby the model undergoes simulated training episodes. Each episode involves learning from a small set of data and then testing on a new set of data from the same task. This trains the model to generalize well from small data samples.

Now let's look at these fundamentals in action in the following example.

# Implementing model-agnostic meta-learning in marketing

Let's consider an example meta-learning model using a simple adaptation of **model-agnostic meta-learning (MAML)** and how it can be applied to optimize marketing strategies. MAML works by optimizing a model's parameters so that a small number of gradient updates will lead to fast learning on a new task.



### Learn more about MAML

MAML is a widely recognized meta-learning algorithm introduced by Finn et al. in 2017 in the paper *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. For a deeper dive into the topic, including interactive examples, check out the resource:

<https://interactive-maml.github.io/>.

In this simplified implementation, we'll simulate a single marketing task as a training input as a proxy for the multiple tasks that would be considered in a full MAML deployment. We will then train a simple neural network model on the task and then use the result of our meta-training to improve the model's adaptability on a different task. The following diagram provides an illustration of the mechanics of a meta-learning framework, as well as what it might look like when applied to multiple marketing tasks as examples:

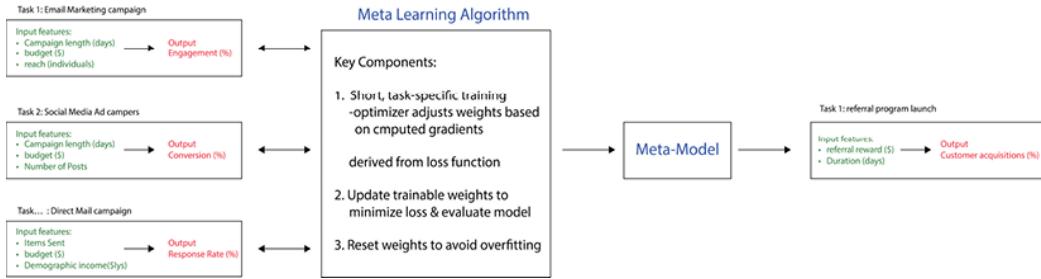


Figure 10.1: Key components of a possible MAML framework in the context of marketing campaigns



### Source code and data:

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/tree/main/ch.10>

Model-agnostic techniques are designed to be broadly applicable across a range of model architectures and tasks. To better understand the mechanics of the MAML approach, we will break down its application into several functions that are simplifications of what would be involved with a full implementation:

1. First, we have a function, `generate_task`, that creates simulated marketing tasks with its own set of coefficients representing campaign data, which, in this case, is represented as a quadratic relationship with noise. The randomly generated coefficients for each task dictate how input data (such as email campaign length, budget, or reach) affect the outcome (such as engagement or conversions). Then the `task` function computes the campaign's success metrics based on the input and the coefficients. Note that we provide seed values so that your output should be consistent with what's shown here, but your results may still vary due to randomness from parallel processing or GPU usage:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
np.random.seed(123)
tf.random.set_seed(123)
def generate_task():
    coefficients = np.random.randn(3)
    def task(x):
        return coefficients[0] * x**2 + coefficients[1] * x + coefficients[2]
    return task, coefficients
```

2. Next, we have a function, `train_model_on_task`, that trains the model on one of the tasks generated by the previous function by performing short bursts of training on the task and using the `Adam` optimizer to update the model's weights. Then TensorFlow `GradientTape` is used for automatic differentiation, calculating the gradients needed to minimize the loss, which measures how well the model's predictions match the actual outcomes of the campaign:

```
def train_model_on_task(model, task, steps=50, learning_rate=0.1):
    optimizer = Adam(learning_rate)
    for _ in range(steps):
        x = np.random.uniform(-3, 3, (50, 1))
        y = task(x)
        with tf.GradientTape() as tape:
            predictions = model(x, training=True)
            loss = tf.reduce_mean((predictions - y) ** 2)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

3. Then we have the `meta_train_model` function, which is at the heart of meta-learning. This handles the meta-training process by optimizing

the model's ability to quickly adapt to new tasks after minimal exposure. In practice, the model is first trained on a task, and then it is evaluated on new data from the same task.

The loss calculated from this evaluation guides adjustment of the model's initial parameters so that just a few updates lead to significant improvements on new tasks. Note that in a full implementation, we would likely use a `reset_weights` argument so that after each task the model's weights are reset to their state before the task-specific training and a `meta_optimizer` that updates the model's initial parameters based on the task performance. This ensures that the model does not overfit to a particular task and retains its ability to generalize across tasks:

```
def meta_train_model(base_model, meta_steps=50,
    for _ in range(meta_steps):
        task, _ = generate_task()
        train_model_on_task(base_model, task, learning_rate=0.1)
```

4. To illustrate this simplified example in action, we can first initialize a very simple neural network with two layers to apply our meta-training function. Here, there is first a dense layer with 10 neurons and a sigmoid activation and then a final dense layer with just one neuron to capture the model's prediction for the success of a marketing campaign based on the input feature:

```
model = Sequential([
    Dense(
        10,
        activation='sigmoid',
        kernel_initializer='random_normal',
        input_shape=(1, )
```

```
    ),
    Dense(1)])
model.summary()
```

This gives us the following summary:

Layer (type)	Output Shape	Param #
dense_40 (Dense)	(None, 10)	20
dense_41 (Dense)	(None, 1)	11

```
Total params: 31 (124.00 B)
Trainable params: 31 (124.00 B)
Non-trainable params: 0 (0.00 B)
```

Figure 10.2: Simple neural network model architecture

5. Let's now generate a proxy for a complex task using your quadratic function and plot the model's performance on this task before training:

```
def plot_predictions(model, task, title):
    x = np.linspace(-3, 3, 100).reshape(-1, 1)
    y_true = task(x)
    y_pred = model.predict(x)
    plt.figure(figsize=(10, 5))
    plt.scatter(x, y_true, color='blue', label='True Values')
    plt.scatter(x, y_pred, color='red', label='Predictions')
    plt.title(title)
    plt.xlabel("Input Feature")
    plt.ylabel("Output Value")
    plt.legend()
    plt.show()
complex_task, _ = generate_task()
plot_predictions(model, complex_task, "Model Performance Be
```

The following plot shows the performance of the model before undergoing meta-training. As we can see, the model is initially unable

to accurately predict the true values, indicating the need for further training and optimization:

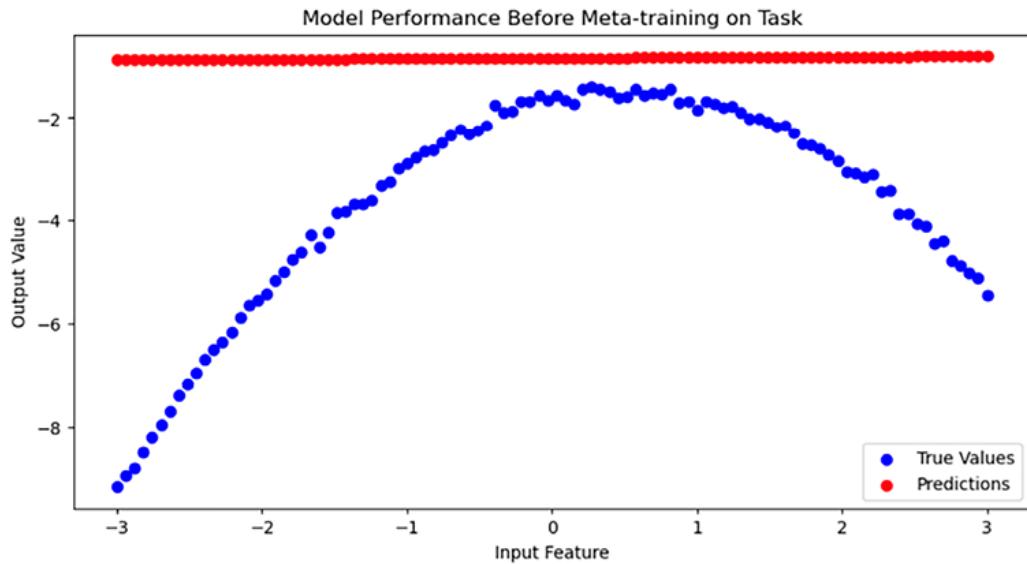
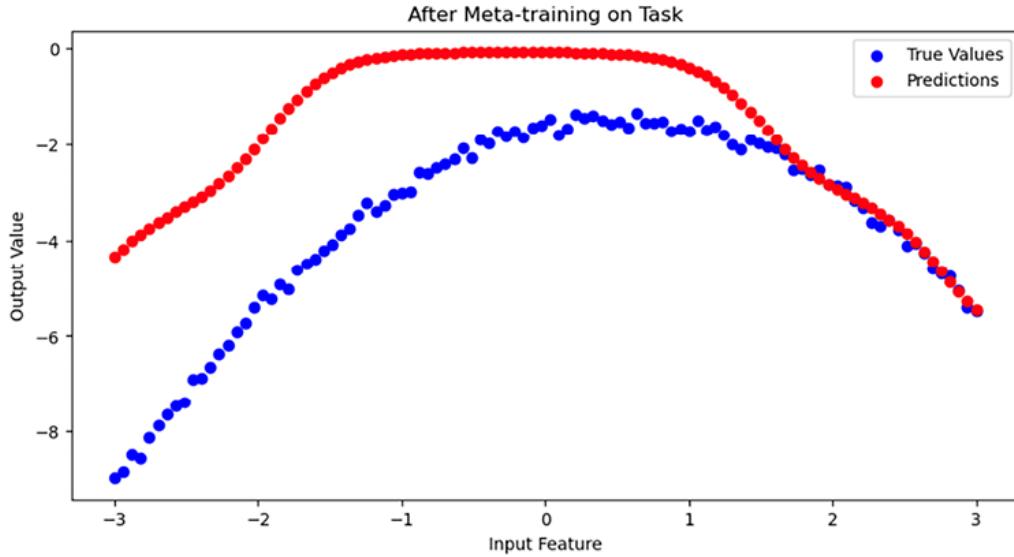


Figure 10.3: Model performance before meta-training on the task

6. Now we can perform meta-training using our algorithm and then plot the model's performance again on the same task after meta-training:

```
meta_train_model(model)
plot_predictions(model, complex_task, "After Meta-training")
```

This yields the following plot:



*Figure 10.4: Model performance after meta-training on the task*

As shown in *Figures 10.3* and *10.4*, the model initially fails to grasp any quadratic relationship in the data. After meta-training, however, the predictions improve. While this is a simplified implementation including only a simple task, this implementation framework gives a baseline for the mechanics of MAML as a tool for preparing your model for FSL.

## Overcoming challenges in FSL

FSL aims to make the most out of minimal data, but this can come with inherent challenges such as overfitting and poor generalization. **Overfitting** occurs when a model, trained on a very small dataset, learns the noise and irrelevant details to the extent that it negatively impacts the performance on new data. To mitigate this, regularization techniques such as L2 regularization and dropout can be used to simplify the model complexity of neural networks, helping prevent the model from learning overly complex patterns that do not generalize well.

## Regularization techniques to prevent overfitting

Two common regularization techniques in neural networks are:

- L2 regularization (weight decay) adds a penalty to the loss function based on the squared magnitude of the model's weights. This discourages large weights, leading to simpler models that generalize better.



Learn more here, including how to compare learning curves to evaluate the impact of regularization:

<https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/l2-regularization>.

- Dropout randomly deactivates (“drops out”) neurons during training to prevent the model from becoming too reliant on specific pathways.

Learn more in the seminal paper *Dropout: a simple way to prevent neural networks from overfitting* by Srivastava et al.

For instance, we could add these types of regularization while building a neural network model suitable for a regression task. As an illustration, the model is designed with a moderately complex architecture and includes several layers with both types of regularization:

```
from tensorflow.keras.layers import Dropout
from tensorflow.keras.regularizers import l2
def build_regularized_model(input_shape):
    model = Sequential([
        Dense(128, activation='relu',
              kernel_regularizer=l2(0.001),
              bias_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(64, activation='relu',
              kernel_regularizer=l2(0.001),
              bias_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(1)
```

```

        Dense(64, activation='relu', input_shape=input_shape,
              kernel_regularizer=l2(0.001)),
        Dropout(0.3),
        Dense(32, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.3),
        Dense(16, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.3),
        Dense(1, activation='linear')
    ])
model.compile(optimizer=Adam(learning_rate=0.001),
               loss='mean_squared_error')
return model
input_shape = (10,)
model = build_regularized_model(input_shape)
model.summary()

```

This yields the following summary:

**Model: "sequential\_1"**

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 64)	704
dropout_1 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2,080
dropout_2 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 16)	528
dropout_3 (Dropout)	(None, 16)	0
dense_6 (Dense)	(None, 1)	17

Total params: 3,329 (13.00 KB)  
Trainable params: 3,329 (13.00 KB)  
Non-trainable params: 0 (0.00 B)

*Figure 10.5. Model architecture summary*

The model embeds L2 regularization within its dense layers to curb the weight size and encourage simpler models. A dropout of 0.3 is also applied after each dense layer, reducing overfitting by deactivating random neurons.

outputs during training. This regularization pattern is applied across the hidden layers to promote the model’s ability to generalize. The following figure provides a visualization of how dropout randomly deactivates neurons during training:

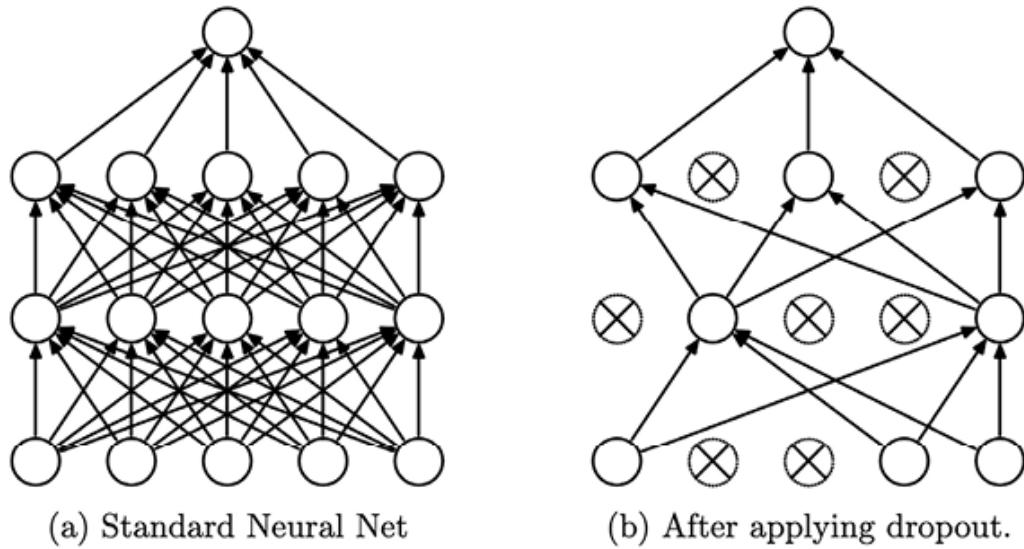


Figure 10.6: Dropout neural net model. Left: A standard neural net with two hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. (source: <https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf>)

As alluded to earlier, **generalization** is another critical challenge. In the field of image processing, data augmentation techniques like image rotation, cropping, or color adjustment are often employed to artificially enhance the dataset’s size and variability, helping the model learn more general features that perform better on new tasks. In NLP applications, data augmentation techniques such as those presented in *Chapter 5* can be employed, along with others such as synonym replacement, in order to increase robustness and aid in better generalization across different contexts.

A significant challenge in FSL is also the difficulty of achieving robust performance with limited training data. Depending on the complexity of the pattern being learned, models may require larger datasets to generalize well and therefore struggle to learn useful patterns without it, leading to poor performance on new tasks. Transfer learning can be a powerful, complementary strategy to address this challenge. By starting with a model pre-trained on a large and diverse dataset, you can leverage the learned features and fine-tune the model on your specific FSL task. This approach enables rapid adaptation to new tasks with minimal data while retaining a broad knowledge base, complementing FSL by providing a rich initial set of features that need only slight adjustments rather than learning from scratch. This powerful approach has its limitations and challenges as well; however, these are topics we will discuss in the following section.

## Optimizing FSL techniques

Key strategies for enhancing the efficacy of FSL models:

- **Overfitting prevention:** Implement regularization techniques to help the model focus on simpler, more general patterns, rather than memorizing specific noise in the training data.
- **Enhancing generalization:** Employ data augmentation to expand the training dataset. For text, approaches like paraphrasing or injecting synonyms help the model learn language variations.
- **Leveraging transfer learning:** Utilize a model pre-trained on a comprehensive dataset and then fine-tune it for your specific FSL task.

# Navigating transfer learning

Transfer learning can enhance how marketing professionals leverage AI by enabling the more effective use of pre-trained models on new tasks with only minor adjustments. While FSL uses a set of examples from the new task for quick adaptation, transfer learning focuses on repurposing an existing model without needing additional examples from the new domain.

This approach capitalizes on the knowledge that models gain from large-scale data in previous tasks, applying it to enhance marketing efforts in completely different areas without the overhead of retraining the model from scratch. Put differently, FSL improves model adaptability using very limited data examples, whereas transfer learning excels in environments where the relationship between past and current tasks is strong but the availability of large enough labeled datasets for training a base model for the new task is difficult or costly to acquire.

An additional advantage of transfer learning over FSL can come from a cost perspective. For example, when using paid API services for state-of-the-art GenAI models, pricing can be related to the input size used to generate the response. In the context of NLP applications, this is often captured by the token count. When fine-tuning via transfer learning, the base model only needs to be adjusted once. This means that the token count paid for during the fine-tuning phase does not recur with every usage of the model, as opposed to FSL, which may require the same relevant examples to be fed in each time.

With transfer learning, the initial cost of fine-tuning the model can be offset by the lower costs for each subsequent inference, as no additional examples need to be fed into the model. Transfer learning therefore becomes cost-

effective when the number of inferences after fine-tuning exceeds a certain threshold, making it cheaper in the long run compared to FSL.

As an example, here is a breakdown to illustrate the potential cost differences between FSL and transfer learning:

- **FSL API costs:**
  - **Base cost:** Cost for tokens used in the prompt (Cost per token × Number of tokens in prompt)
  - **Additional cost per example:** Cost per token × Number of tokens per example
- **Transfer learning API costs:**
  - **Initial fine-tuning cost:** One-time cost for adjusting the pre-trained model using a larger set of example data
  - **Cost per inference:** Cost for each inference using the fine-tuned model

### When to use transfer learning over FSL

Compared to FSL, transfer learning can be more advantageous under the following circumstances:

- **High frequency of usage:** The more frequently the model is used, the quicker the initial fine-tuning cost is amortized.
- **Stable task requirements:** New tasks must be similar enough for the fine-tuned model to perform well without further adjustment.
- **Complex pattern adaption:** While requiring more extensive data, it can adapt deeper model layers to



learn more complex patterns.

# The mechanics of transfer learning

At a high level, transfer learning and its fine-tuning processes are built upon similar foundations established by techniques like ZSL and FSL. Both ZSL and FSL are designed to apply pre-existing knowledge to new tasks.

However, they differ primarily in the approach to adaptation. ZSL hypothesizes about *unseen tasks* based on learned abstract concepts without any specific examples, whereas FSL uses a *handful of examples* to guide the adaptation. Transfer learning extends these concepts by utilizing a base of extensively trained models that can be specifically fine-tuned to a new task, providing a practical balance between the broad generalization of ZSL and the rapid adaptation characteristic of FSL. Transfer learning is often the preferred approach when a robust, pre-trained model exists and there are examples from a highly related domain, but a lack of examples specifically related to the new task.

At a more technical level, fine-tuning in transfer learning involves subtle yet significant adjustments to the model's parameters that are already well-trained on large, diverse datasets. This fine-tuning adapts the learned features to the specifics of a new, related task, often involving adjustments to the deeper, more discriminative layers of the model. Such modifications enable the model to maintain its generalization capabilities while optimizing for performance on specific tasks within the new domain. This is different from FSL, where fine-tuning aims to quickly adapt the model using very few examples by making minimal adjustments, often only to the

model's final layers or via prompt engineering where there can be no adjustments to the trainable parameters.

The following figure illustrates the differences between ZSL, FSL, and transfer learning:

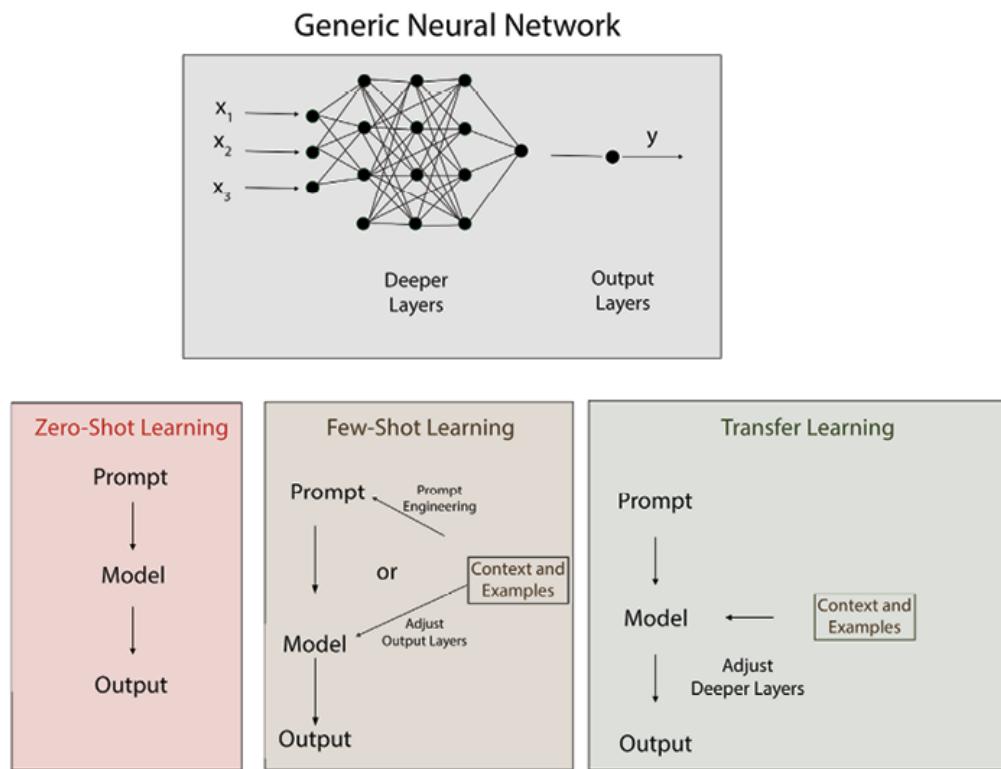


Figure 10.7: Key differences between typical ZSL, FSL, and transfer learning implementations and their mechanics

## Transfer learning using Keras

In this section, in order to provide variety beyond applications just in the field of NLP, we will present a framework for how transfer learning can be implemented in an image classification task. This example will present a framework for how you can adapt a pre-trained image recognition model to

a new marketing-relevant challenge of your choice, such as identifying desired product features.

The first step in the transfer learning process involves finding a pre-trained model that has been extensively trained on a large and diverse dataset. Here, we will use the VGG16 model, known for its robust performance in image recognition. VGG16 is a convolutional neural network that achieves this high performance through training on the ImageNet dataset using over a million images in order to classify them into a thousand image categories, as detailed by Simonyan and Zisserman in their 2014 paper, *Very deep convolutional networks for large-scale image recognition*.

First, we'll summarize the full architecture of this model using `model.summary` to give us a better understanding of its layers and components:

```
from keras.applications.vgg16 import VGG16  
base_model = VGG16()  
base_model.summary()
```

```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 95s 0us/step
Model: "vgg16"

Layer (type)                 Output Shape              Param #
input_layer_2 (InputLayer)   (None, 224, 224, 3)      0
block1_conv1 (Conv2D)        (None, 224, 224, 64)     1,792
block1_conv2 (Conv2D)        (None, 224, 224, 64)     36,928
block1_pool (MaxPooling2D)   (None, 112, 112, 64)     0
block2_conv1 (Conv2D)        (None, 112, 112, 128)    73,856
block2_conv2 (Conv2D)        (None, 112, 112, 128)    147,584
block2_pool (MaxPooling2D)   (None, 56, 56, 128)      0
block3_conv1 (Conv2D)        (None, 56, 56, 256)     295,168
block3_conv2 (Conv2D)        (None, 56, 56, 256)     590,080
block3_conv3 (Conv2D)        (None, 56, 56, 256)     590,080
block3_pool (MaxPooling2D)   (None, 28, 28, 256)      0
block4_conv1 (Conv2D)        (None, 28, 28, 512)     1,180,160
block4_conv2 (Conv2D)        (None, 28, 28, 512)     2,359,808
block4_conv3 (Conv2D)        (None, 28, 28, 512)     2,359,808
block4_pool (MaxPooling2D)   (None, 14, 14, 512)      0
block5_conv1 (Conv2D)        (None, 14, 14, 512)     2,359,808
block5_conv2 (Conv2D)        (None, 14, 14, 512)     2,359,808
block5_conv3 (Conv2D)        (None, 14, 14, 512)     2,359,808
block5_pool (MaxPooling2D)   (None, 7, 7, 512)       0
flatten (Flatten)           (None, 25088)            0
fc1 (Dense)                (None, 4096)             102,764,544
fc2 (Dense)                (None, 4096)             16,781,312
predictions (Dense)         (None, 1000)             4,097,000

Total params: 138,357,544 (527.79 MB)
Trainable params: 138,357,544 (527.79 MB)
Non-trainable params: 0 (0.00 B)

```

*Figure 10.8: Model architecture summary*

As shown here, the VGG16 architecture is a deep neural network consisting of multiple layers designed to process and transform the image into a form that is increasingly abstract and useful for classification tasks. Here's a simple breakdown based on the preceding summary:

- **Input layer:** This accepts images of size 224 x 224 pixels, which is a standard dimension for many image processing tasks. Each image has three color channels (Red, Green, and Blue), which explains the 3 in the shape given in *Figure 10.8*. The None given in each layer at the

start is effectively a placeholder for the batch size, allowing for any number of images to be processed.

- **Convolutional layers (Conv2D):** These perform the core of the work in the network. VGG16 has multiple convolutional layers stacked on top of each other, each with a set number of filters that detect different features in the image at various levels of granularity. For example, the first convolutional layer might detect edges, while deeper layers might identify more complex patterns like textures or specific objects. The progression in the number of filters from 64 to 512 reflects an increase in the complexity of features being detected in the image as it moves through the network.
- **Max pooling layers (MaxPooling2D):** These are interspersed with the convolutional layers and serve to reduce the spatial dimensions (width and height) of the input for the next convolutional layer. For instance, a pooling layer following a 224 x 224 convolutional layer output reduces it to 112 x 112, thus reducing the computation required and helping in the detection of dominant features that are invariant to small shifts and rotations in the input image.
- **Fully connected layers (Dense):** Near the end of the network, the flattened output from the convolutional layers is fed into these densely connected layers. These are used to combine all features from the prior convolutional layers to eventually classify the image into one of the 1,000 categories.
- **Output layer (predictions):** The final layer outputs the probabilities of the image belonging to each of the 1,000 classes trained in the ImageNet dataset. The class with the highest probability is taken as the prediction of the model.

The following is a visualization of this summary taken from a medical research article by Yang et al.

(<https://www.nature.com/articles/s41598-021-99015-3>):

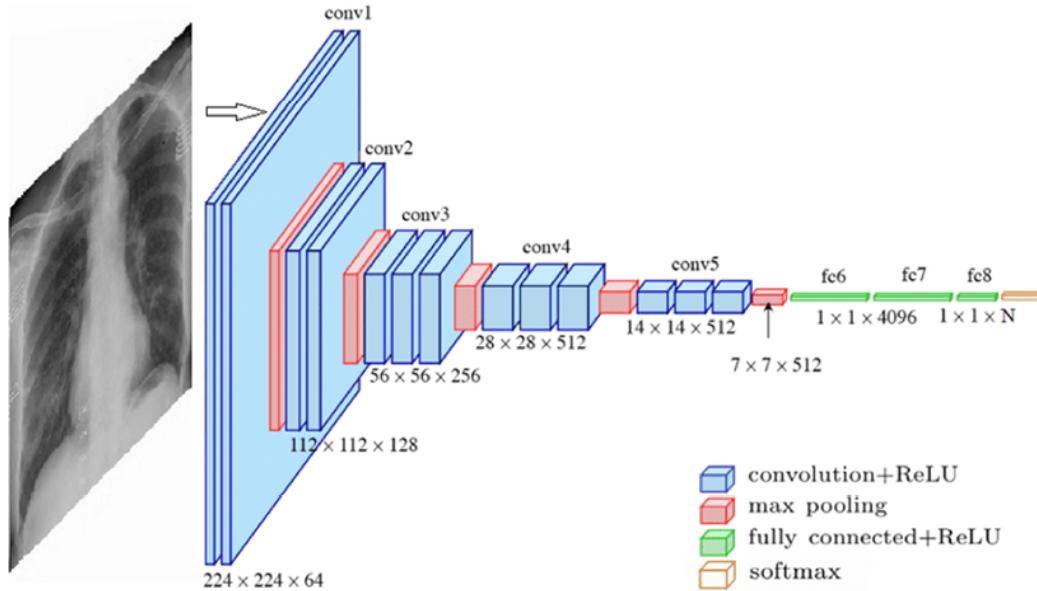


Figure 10.9: Visualization of the architecture of the VGG16 deep neural network

Let us now adapt this model for transfer learning:

1. We will load the model into memory again, but this time with the option `include_top=False` in order to truncate the model after the last convolution layer. This will move the top layers responsible for classification into predefined categories and allow the model to serve as a feature extractor, which means it outputs a complex feature map that represents the input image's key characteristics but doesn't output a specific category prediction:

```
base_model = VGG16(weights='imagenet', include_top=False, j
```

2. We can then adapt this feature-rich model to our new task of product classification by adding our own classification layers that will be trained on top of the feature extractor:

```
from tensorflow.keras import layers, models
model = models.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

3. Finally, the model can be fine-tuned for transfer learning using new, task-specific data. This involves setting the pre-trained base model layers to non-trainable, thereby freezing the weights of the pre-trained layers, which is generally advised to avoid having the initial phase of training forget the majority of the base model's initial feature extraction weights:

```
base_model.trainable = False
model.compile(optimizer='adam', loss='binary_crossentropy',
```

### Self-guided exercise: Fine-tuning VGG16 on your data

To apply transfer learning using the VGG16 model to classify images of product features, follow these steps:

- **Collect images:** Gather images of products with features your company aims to classify. Sources can include online catalogs, social media, or direct captures of physical inventory.



- **Prepare the data:** Resize images to 224 x 224 pixels to match the VGG16 input size and normalize pixel values to range [0, 1].
- **Train the model:** Use the compiled model and fit your images (`new_data`) and their binary labels [0, 1], for instance using:

```
model.fit(new_data, labels, epochs=5,
```

- **Evaluate and fine-tune:** Based on performance, consider adjusting your learning rate or gradually unfreezing layers in the base VGG16 model

## Transfer learning using API services

Transfer learning through API services offers a practical solution for marketing professionals who wish to utilize cutting-edge AI without the complexities of a local implementation. Some of the practical benefits of using an available API service for transfer learning are the following:

- **Access to advanced models:** Continuous updates and optimizations of API-accessible models ensure the use of cutting-edge technology that is more advanced than what may be publicly available.
- **Computational efficiency:** Offloading the computational demands to cloud-based services removes the need for sophisticated hardware that may be inaccessible for individuals or small businesses.

- **Simplified parameter management:** This refers to the automatic handling of learning rates, gradient adjustments, and other complex training parameters that can affect model performance.

To apply transfer learning effectively, it's crucial that the fine-tuning dataset mirrors the unique characteristics of your brand. This could include customer feedback or vetted marketing copy and product descriptions. If a proprietary dataset is unavailable, publicly accessible data such as reviews or social media posts about similar products or services could be used. The key is to compile this data into a single file, typically in CSV format, in a location that is accessible to the API. Each entry then needs to be clearly labeled according to its expected label or output.

The following is an example input in CSV format that gives the model a deeper understanding of what sustainability means in the context of our specific brand, setting the stage for what will be discussed in the hands-on example in the next section:

```
text,label
"Our new line of kitchenware not only uses recycled materials b
"While our products are designed to be eco-friendly, we are als
"Our brand continues to strive for environmental sustainability
"We ensure that all our materials are ethically sourced and pro
"Despite using recycled materials, our company has been critici
```

Now, we can initiate the model fine-tuning using OpenAI's API with the GPT-4 model, replacing `openai.api_key` with your actual key and `path/to/your/dataset.csv` with a path to the dataset's location. Before executing this code, remember to consult the latest OpenAI API documentation (<https://beta.openai.com/docs/>) for any updates or changes in the API usage:

```
import openai
openai.api_key = 'XXX' #your API key here
response = openai.File.create(
    file=open("path/to/your/dataset.csv", "rb"),
    purpose='fine-tune')
file_id = response['id']
fine_tune_response = openai.FineTune.create(training_file=file_
    model="gpt-4",
    n_epochs=5)
```

## Overcoming challenges in transfer learning

While transfer learning can be a powerful approach for fine-tuning pre-trained models to better perform better on new tasks, it is not without its challenges and limitations. The following are a couple of key challenges to consider when applying transfer learning:

- **Model drift:** In the context of transfer learning, model drift refers to the gradual decline in a model's accuracy as the data it was fine-tuned on becomes less representative of the current environment or trends.

For example, imagine using a model pre-trained on marketing data from 2019 to predict consumer preferences for 2024. Initially, the model may perform well after being fine-tuned via transfer learning with more recent 2020 data. However, as consumer behavior shifts due to reasons such as new social media platforms like TikTok gaining in popularity or significant changes in economic conditions, the model may start recommending outdated messaging strategies.

To address model drift, one needs to continuously update the model with recent data from the target domain and monitor its performance

closely. This can be achieved by continuous learning techniques that allow the model to adjust dynamically as new data comes in. Further resources on continuous learning are given in the info box in this section.

- **Domain mismatch:** Domain mismatch in transfer learning occurs when the source domain (where the model was initially trained) and the target domain (where the model is fine-tuned) differ significantly. In transfer learning, this means the pre-trained model's knowledge may not generalize well to the new domain, leading to poor performance.

As an example, consider a model pre-trained on English-language customer reviews from an e-commerce platform, which is then fine-tuned to analyze reviews in Japanese for a local market. Even after fine-tuning, the model may struggle to capture cultural nuances and linguistic differences, leading to incorrect sentiment analysis. This may be because the pre-trained model's reliance on patterns learned from English data makes it less effective when applied to Japanese reviews where the expressions and cultural references are different.

To address this, a more appropriate approach would be to fine-tune the model with a carefully curated dataset of Japanese customer reviews that specifically captures the specific linguistic and cultural nuances of the local market.

### Staying ahead of model drift

To maintain the accuracy of your transfer learning models, especially as consumer behavior evolves, continuous learning is key.



Continuous learning is often implemented within the practices of MLOps. Learn more about MLOps principles and best practices at <https://cloud.google.com/architecture/ml-ops-continuous-delivery-and-automation-pipelines-in-machine-learning>.

Having introduced the fundamentals and techniques behind how FSL and transfer learning can adapt pre-trained models to better address new marketing tasks, we will now demonstrate the importance of FSL through a hands-on marketing example.

## Applying FSL to improve brand consistency

In our previous exploration of ZSL in *Chapter 9*, we demonstrated how a pre-trained model like GPT-4 could generate marketing copy for an e-commerce brand launching eco-friendly products. This ZSL approach, while powerful, primarily relied on the model's ability to infer context and content from generalized pre-training using prompts emphasizing terms like *sustainable* and *eco-friendly*.

As the examples in this section will show, while ZSL provides a solid foundation for generating brand-relevant content, it often lacks the precision required for capturing the deeper, more nuanced aspects of a brand's ethos. This is particularly true for brands whose identity is heavily tied to specific practices or principles that may not be well represented in the generalized training of the **large language model (LLM)**.



## Steps for effective FSL in marketing campaigns

1. **Prompt refinement and execution:** Begin with a basic prompt like one that would be used for ZSL, but now add a few examples of desired outputs or contexts that reflect the brand's unique aspects.
2. **Output analysis:** Evaluate the content to ensure it maintains the necessary brand consistency. Note deviations from these expectations as they provide insights into areas needing further examples.
3. **Iterative refinement:** Based on the feedback and performance of the initial outputs, refine the examples and prompts iteratively, continuing until you achieve your desired marketing KPIs (see *Chapter 2*).

Continuing our scenario from the section on *Zero-shot learning for marketing copy* in *Chapter 9*, we revisit our journey with the sustainability-focused e-commerce brand that we introduced earlier. Now, in response to evolving market trends and corporate directives, the company known for its eco-friendly kitchenware is undergoing a rebranding to align with the growing consumer demand for social responsibility. Recent insights reveal that consumers are increasingly interested in supporting brands that not only offer sustainable products but also demonstrate a tangible commitment to fair labor practices and positive community impact. This shift is particularly pronounced among affluent younger consumers who value ethical production and are willing to pay a premium for products that make a real difference in the lives of workers and communities. Furthermore, this

demographic wants their products not just to have eco-friendly packaging but for them to be deemed “zero waste.”

Let’s demonstrate how GPT-4 can effectively capture our newly defined sustainability campaign goals by enhancing output content through prompt engineering. This includes the subtle clarifications around sustainability mentioned earlier:

- A commitment to fair labor practices
- Zero waste packaging
- Active community engagement

We will exemplify this through a case study aimed at boosting an email marketing campaign. This involves iterative refinement of prompts, closely tied to monitoring key performance indicators previously discussed in *Chapter 2*, to ensure alignment with our rebranded marketing objectives.

## Benchmarking with ZSL and FSL

Before integrating FSL into our email marketing campaign, it’s important to benchmark the effectiveness of ZSL at capturing the refined brand strategy for FSL is introduced. Here, we will compare a baseline response generated using ZSL that’s devoid of specific examples that align with our rebranded sustainability focus. We’ll define a function to execute the API completion and then illustrate how ZSL performs without the nuanced context of our updated campaign goals:

```
from openai import OpenAI
client = OpenAI(api_key = 'XXX' ) #your API key here
base_prompt = "Write a product description for an eco-friendly
def generate_chat_completion(client, prompt, max_tokens=150, n=
    if not advanced_usage:
```

```
        messages = [{"role": "user", "content": prompt}]
    else:
        messages = prompt
completion = client.chat.completions.create(
    model="gpt-4",
    messages=messages,
    max_tokens=max_tokens,
    n=n,
    temperature=temperature)
return completion.choices[0].message.content
zsl_response = generate_chat_completion(client, base_prompt)
print(zsl_response)
```

This gives us the following response:

```
Product Name: EcoPure Bamboo Kitchen Utensil Set
Enrich your cooking experience with our EcoPure Bamboo Kitchen L
Each piece in this set is made from 100% organic bamboo, a renew
```

As you can see, while the response is clear, it is too generic and lacks the specificity and context needed to meet the specific goals of our rebranding initiative.

To see the value FSL can provide, here, we revisit the exact same base prompt as earlier, but also include context specific to our updated campaign goals via FSL:

```
fsl_prompts = [
    {"role": "system", "content": "Describe a product using sus
    {"role": "user", "content": "Our latest kitchenware range n
    {"role": "system", "content": "Write about a kitchenware pr
    {"role": "user", "content": "Each item in our new line of c
    {"role": "system", "content": "Explain how a purchase suppo
    {"role": "user", "content": "With every purchase, a portion
    {"role": "system", "content": base_prompt}]
```

```
fsl_response = generate_chat_completion(client, fsl_prompts, ma
print(fsl_response)
```

This produces the following output:

```
Introducing our new line of eco-friendly kitchenware, designed w
utmost respect for our planet and its inhabitants
. Each piece in this range is crafted from sustainable, biodegra
We believe in the power of ethical production and fair labor
factories that ensure fair wages, safe working conditions, and
, which is why all our products are sourced from .
But our commitment doesn't stop there.
For every purchase, we contribute towards community development
, particularly urban greening initiatives, to make our cities m
```

In the preceding text, the aspects of the LLM-generated content that organically relate to our specific goals—environmental sustainability, fair labor practices, and community engagement—are highlighted for clarity. This comparison with ZSL underscores the capability of FSL to produce content that is more tailored and relevant to the strategic nuances of our rebrand by leveraging the initial contextual examples.

## Developing an email marketing campaign

Now, let's explore the application of FSL in enhancing the personalization and relevance of email marketing campaigns, significantly boosting both customer engagement and conversion rates. The following is a concise guide on how to utilize FSL to optimize email campaigns effectively:

- 1. Initial email creation:** Generate the initial batch of emails using FSL, specifically focusing on your core marketing goals—in this case, sustainability and ethical practices.
- 2. Collect initial metrics and responses:** Analyze the initial reactions from customers to understand what aspects of the content resonate the most. This analysis is crucial in identifying successful elements and areas that may require additional refinement.
- 3. Iterative refinement:** Refine the content of the emails based on the engagement metrics and feedback received from customers.
- 4. Continued feedback integration:** Continuously integrate new insights to keep the email marketing campaign dynamically aligned with evolving customer preferences and market trends.

Now let's examine what these steps entail.

## Step 1: Initial email creation

The first step in employing FSL in an email marketing campaign is to clearly define the campaign's objectives. For our eco-friendly e-commerce brand, the objective might be to launch a new line of eco-friendly kitchenware that aligns with the rebrand goals. Let's consider our objective for this scenario to be the following:

Introducing a new line of sustainable kitchenware that embodies

With the campaign's objective clearly defined, the next step involves creating prompts that instruct the AI on the content that's appropriate for the email message.



## Structuring the initial email for your marketing campaign

The following general structure is helpful in creating an effective initial message for your email marketing campaign that increases engagement and conversion with your audience:

- **Introduction:** Greet the customer and introduce the new product line enthusiastically.
- **Body:** Provide detailed information about the core aspects of the product – in this case, the sustainability and ethical manufacturing aspects.
- **Call to action:** Include a call to action that encourages the recipient to take specific steps such as making a purchase or visiting a website.
- **Closing:** Conclude with a warm closing that reinforces the brand's commitment to the core aspects of the product.

We'll define a starting prompt for the model, which will be used to generate the initial batch of emails. Similar to the earlier product description example, each element of the `fsl_prompts` is designed to progressively build a narrative around the brand's commitment to sustainability, ethical practices, and community involvement, as required by the corporate rebrand, in the following way:

- The first pair of prompts ensures the content appropriately emphasizes the eco-friendly and zero-waste aspect of the kitchenware

- The second set of prompts addresses the context expected when discussing the ethical labor practices involved in the production process
- The final two prompts tie the product to community engagement efforts, linking purchases to initiatives that are important to socially responsible consumers

To ensure that the generated content aligns with our campaign goals, we'll also adjust our base prompt to be more specific to an email campaign and increase the `max_tokens` to reflect the longer content:

```
base_prompt = "Compose a short email to introduce our latest li
fsl_prompts = [
    {"role": "system", "content": "Make sure to highlight the e
    {"role": "user", "content": "Our kitchenware is designed wi
    {"role": "system", "content": "Include information about th
    {"role": "user", "content": "We are committed to ethical pr
    {"role": "system", "content": "Mention how purchasing these
    {"role": "user", "content": "Every purchase supports commun
    {"role": "system", "content": base_prompt}]
fsl_completion = client.chat.completions.create(
    model="gpt-4",
    messages=fsl_prompts,
    max_tokens=300,
    n=1,
    temperature=0.7
)
print(fsl_completion.choices[0].message.content)
```

This produces the following email content:

```
Subject: Discover Our New Eco-Friendly Kitchenware - Upgrade Yo
Hello,
We are thrilled to announce the arrival of our latest line of e
Our new collection is a testament to our commitment to protecti
```

But we don't stop at just the products. We've committed to zero waste. We believe in ethical production and are proud to say that all proceeds from our sales go towards supporting local farmers. Moreover, when you purchase from us, you're also giving back to the environment. We invite you to explore our new line of eco-friendly kitchenware. Click here [insert hyperlink] to begin your sustainable cooking journey. Best regards,

[Your Name]

## Step 2: Collect and analyze initial metrics and responses

Following the deployment of our email campaign with the newly created content, it is crucial to assess its effectiveness using KPIs. For this evaluation, we assume we already have representative data from the first seven days post-launch that will allow us to examine the metrics of open rate, click-through rate, conversion rate, and unsubscribe rate. This analysis helps us identify areas for improvement by visualizing these KPIs over time.

```
import pandas as pd
df_initial = pd.read_csv('initial_launch_data.csv')
def plot_campaign_kpis(df, title='Email Campaign KPIs'):
    plt.figure(figsize=(10, 8))
    plt.plot(df['Date'], df['Open Rate (%)'], label='Open Rate')
    plt.plot(df['Date'], df['Click-Through Rate (%)'], label='CTR')
    plt.plot(df['Date'], df['Conversion Rate (%)'], label='Conv. Rate')
    plt.plot(df['Date'], df['Unsubscribe Rate (%)'], label='Unsubscribe Rate')
    plt.xlabel('Date')
    plt.ylabel('Percentage')
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.show()
plot_campaign_kpis(df_initial, 'Email Campaign KPIs First Week')
```

In the following graph, we can observe the trends in open rates, click-through rates, conversion rates, and unsubscribe rates over the first week following the campaign launch. This allows us to quickly identify patterns, which can be critical for diagnosing issues with the campaign's content or delivery:

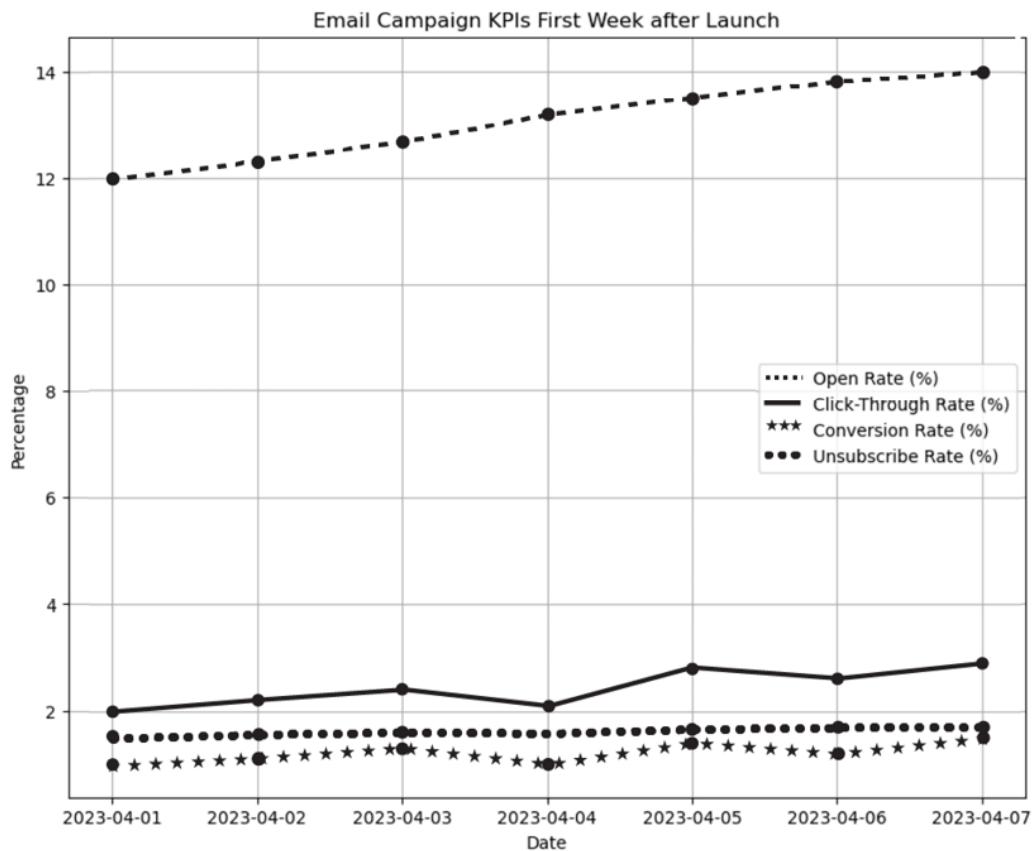


Figure 10.10: Email campaign KPIs during the first week after launch

## Benchmarking your email campaign performance

Evaluating the performance of your marketing campaign metrics can be tricky, especially without industry benchmarks as a reference. To quickly assess whether your



KPIs are on target, start by comparing them to established benchmarks.

Here is a recent comparison of email marketing benchmarks:

<https://www.webfx.com/blog/marketing/email-marketing-benchmarks/>.

We can also calculate and review the average performance across these metrics:

```
print(f"Average Open Rate: {df_initial['Open Rate (%)'].mean():.2f}%")
print(f"Average Click-Through Rate: {df_initial['Click-Through Rate (%)'].mean():.2f}%")
print(f"Average Conversion Rate: {df_initial['Conversion Rate (%)'].mean():.2f}%")
print(f"Average Unsubscribe Rate: {df_initial['Unsubscribe Rate (%)'].mean():.2f}%")
```

This yields the following:

```
Average Open Rate: 13.07%
Average Click-Through Rate: 2.43%
Average Conversion Rate: 1.21%
Average Unsubscribe Rate: 1.61%
```

Based on these KPIs, the following are some observations and possible explanations that we may consider addressing in our next iteration of the email campaign:

- **Low open rates:** This may suggest that the email subject lines are not sufficiently engaging.
- **Click-through rates:** Some recipients view the emails but do not click through at expected rates, possibly due to less compelling content or

offers.

- **Modest conversion rates:** This indicates a need for stronger calls to action or improvements in the effectiveness of landing pages.
- **Rising unsubscribe rate:** This could point to issues with the frequency or relevance of the content.

In addition to these KPIs, assume we've collected feedback from email recipients who responded directly to this email with their questions and concerns. We can also gather information, for example, by employing sentiment analysis techniques discussed in *Chapter 5*, allowing us to determine the core topics and sentiments from product reviews and social media posts from customers who read our emails. Here's an overview of the positive feedback that we will assume to have received:

- Appreciation for **sustainable practices:** "I love that your kitchenware is made from recycled materials."
- Approval of **ethical manufacturing:** "It's reassuring to see a brand commit to fair wages and safe working conditions."
- Enthusiasm for the **product range:** "The stainless steel cookware looks fantastic!"

The following are the highlights of the negative feedback:

- Desire for more **product variety:** "I wish that you advertised more color options for the product lines."
- Questions about **product care:** "I'm concerned about how I can care for these products to ensure they last longer."
- Concerns over **price points:** "The products are great, but they're a bit pricey compared to non-eco-friendly alternatives."

While the positive comments validate our campaign's core messages, the criticisms and questions provide actionable insights for our future email messaging. Using this feedback, in the next section, we will hone in on how the negative critiques can be exploited to refine our email marketing strategies and improve our KPIs.

## Step 3: Iterative refinement

After reviewing the initial metrics and customer feedback from the previous sections, it's clear that while some aspects of the campaign resonate well, there are key areas needing improvement. The next step is to refine our FSL prompts based on the initial metrics, address the specific criticisms from customer feedback, better meet customer expectations, and enhance the overall effectiveness of the campaign. The following are the new elements of the prompt, with commentary included after each element of the prompt, highlighting their value to our task:

```
fsl_prompts = [  
    {"role": "system", "content": "Compose a short email to int  
    {"role": "user", "content": "Discover our new line of eco-f
```

This prompt sets the tone for the email, capturing the key elements from our previous FSL attempt but also emphasizing the importance of an engaging subject line in direct response to the low open rate observed in the initial KPI metrics.

```
{"role": "system", "content": "Mention the product variety  
 {"role": "user", "content": "Now available in a range of vi
```

This response caters to feedback desiring more variety. It emphasizes the range of colors available, aiming to attract customers looking to personalize their purchases.

```
{"role": "system", "content": "Provide care instructions."}  
{"role": "user", "content": "Caring for your eco-friendly k
```

Addressing customer concerns about maintenance, this prompt reassures customers that the products are not only sustainable but also easy to care for, enhancing their appeal by ensuring longevity.

```
{"role": "system", "content": "Address concerns about prici  
{"role": "user", "content": "Invest in quality and sustaina
```

This prompt directly tackles the issue of price sensitivity noted in the feedback. It reframes the cost as an investment in quality and sustainability, arguing for long-term value and environmental impact.

```
fsl_response = generate_chat_completion(client, fsl_prompts, ma  
print(fsl_response)
```

After executing the API recall, we get the following result:

```
Subject: Revolutionize Your Kitchen With Our New Eco-Friendly L  
Dear [Customer],  
We're thrilled to introduce our latest line of eco-friendly kit  
Our kitchenware is crafted using environmentally friendly mater  
Available in a variety of shapes, sizes, and vibrant colors, ou  
Caring for your new eco-friendly kitchenware is a breeze - just
```

While our items may seem initially more expensive, consider the Ready to make the switch to a more sustainable kitchen? Visit o

After deploying these updates emails on April 7, assume we collect the next seven days of KPI data and observe the following updated trends:

```
df_improved = pd.read_csv('improved_data.csv')
plot_campaign_kpis(df_improved, 'Improved Email Campaign KPIs A')
```

This gives us the following plot, where we observe significant changes in the KPIs after FSL refinements:

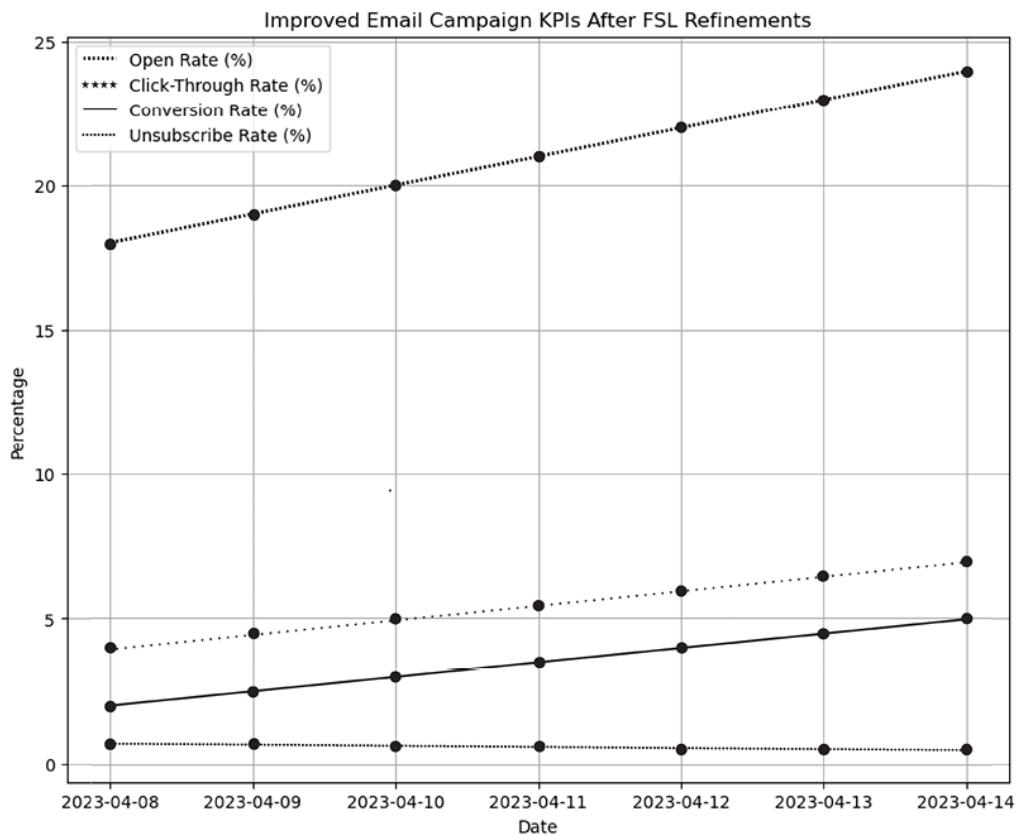


Figure 10.11: Email campaign KPIs after campaign changes and improvements were made in response to customer feedback

Based on this, we have the following average KPIs:

```
print(f"Average Open Rate: {df_improved['Open Rate (%)'].mean()}")
print(f"Average Click-Through Rate: {df_improved['Click-Through Rate (%)'].mean()}")
print(f"Average Conversion Rate: {df_improved['Conversion Rate (%)'].mean()}")
print(f"Average Unsubscribe Rate: {df_improved['Unsubscribe Rate (%)'].mean()}"
```

```
Average Open Rate: 21.00%
Average Click-Through Rate: 5.50%
Average Conversion Rate: 3.50%
Average Unsubscribe Rate: 0.59%
```

By comparing these metrics to the previous ones, we see significant improvements across all areas. While attributing the exact origins of these improvements is not possible without techniques such as A/B testing (*Chapter 6*) or causal inference (*Chapter 3*), one can still speculate what may be behind these improved KPIs:

- **Open rate** improved significantly, which may reflect the more engaging subject lines, including “Revolutionize your Kitchen...”
- **Click-through rate** saw an increase, which may be attributed to the inclusion of requested features like product variety and detailed care instructions, which increased customer engagement
- **Conversion rate** increased, indicating that the email content was not only more engaging but also more effective at convincing customers of the value and relevance of the products, especially after addressing price concerns
- **Unsubscribe rate** decreased, reflecting higher content satisfaction

## Step 4: Continued feedback integration

The final step in utilizing FSL for email marketing campaigns is to establish a robust system for continuous feedback integration. This approach ensures that the campaign remains dynamic and responsive to the evolving needs and preferences of customers, as well as broader market trends. By integrating continuous feedback, the campaign not only maintains relevance but also strengthens customer engagement and promotes brand loyalty over time.

Effective feedback integration requires mechanisms that allow customers to share their thoughts and experiences easily. This includes embedding quick survey links directly in emails, enabling direct email responses, and engaging with customers through social media interactions related to the campaign. These channels facilitate the flow of information from customers back to marketers, providing valuable insights that can be used to refine and improve the campaign continuously.

### Example feedback collection strategies

The following are additional ways to collect direct customer feedback:

- **Live chat feedback:** Implement live chat on your website to allow real-time interactions and immediate feedback.
- **Interactive content:** Use quizzes and polls in your emails or on digital platforms to make feedback collection engaging.



- **Feedback QR codes:** QR codes on products or ads can link to feedback forms so customers can easily respond on their devices.

To effectively analyze and utilize ongoing feedback, setting up a real-time feedback dashboard can be immensely beneficial. This dashboard can serve as a central hub for monitoring and analyzing feedback alongside standard marketing KPIs, providing a comprehensive view of campaign performance. To build such a dashboard, consider using software solutions like Tableau, Microsoft Power BI, or Google Data Studio, which offer powerful tools and intuitive interfaces for creating interactive and real-time data visualizations.

The following are the steps for creating your own feedback dashboard:

1. **Select a dashboard tool:** Choose a platform that best fits your technical capabilities and budget. Tableau, Microsoft Power BI, and Google Data Studio are popular choices due to their robust features and scalability.
2. **Integrate data sources:** Connect the dashboard tool to the data sources that gather your marketing KPIs and feedback. This may include email campaign management tools, social media analytics, and customer feedback systems.
3. **Design the dashboard:** Create visualizations that clearly display the key metrics you need to track. Customize the dashboard to highlight trends, spikes, and dips in campaign performance.
4. **Schedule regular reviews:** Set up regular intervals, bi-weekly or monthly, for example, to review the dashboard insights. Use these

sessions to assess the effectiveness of recent changes and to plan further strategic adjustments based on the data insights.

## Keeping your emails out of the spam folder

To maximize the effectiveness of your email campaigns, it's essential to prevent your messages from landing in the spam folder. Here are three key strategies for achieving this:

- **Set up email authentication:** Implement standards like SPF, DKIM, and DMARC to authenticate your emails. This helps establish trust with email providers to reduce the risk of your emails being marked as spam.
- **Maintain list hygiene:** Regularly clean your email list by removing inactive subscribers and ensuring that all recipients have opted in to receive your emails. This not only boosts engagement but also helps protect your sender reputation.
- **Optimize email content:** Other optimization strategies include personalizing your emails using FSL, limiting the number of links, and maintaining a balanced image-to-text ratio.



# Summary

This chapter explored the potential of FSL and its promise for refining AI-driven marketing strategies to enhance brand presence. Building on the principles introduced through ZSL, we explored how FSL leverages a limited dataset to enable rapid adaptation of AI models to new tasks. This is crucial in the fast-paced marketing domain, where aligning quickly with evolving consumer preferences and market trends can have a significant impact on a brand's relevance and engagement.

While FSL focuses on quick adaptability using minimal examples, transfer learning complements this by applying pre-trained models fine-tuned for specific tasks, thereby minimizing the need for extensive retraining. The chapter emphasized practical strategies combining these methodologies to optimize your marketing efforts. Through approaches like the MAML approach, we demonstrated how you can use meta-learning frameworks for marketing.

As we proceed, the next chapter will introduce the concept of **retrieval-augmented generation (RAG)**. We will explore how RAG can dynamically produce content that reflects the latest available data by integrating generative models with advanced information retrieval techniques. This approach not only increases the relevance of the content generated but also enhances precision in consumer targeting, making your marketing efforts significantly more effective. The upcoming discussion will cover the technical setup of a knowledge retrieval system and practical applications of RAG in marketing, through which we aim to provide you with robust tools for writing precisely targeted marketing content that resonates with your current and potential customers.

# Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)

# 11

## Micro-Targeting with Retrieval-Augmented Generation

This chapter introduces the advanced capabilities of **retrieval-augmented generation (RAG)** and its strategic application in precision marketing, building on the foundations laid by **zero-shot learning (ZSL)** and **few-shot learning (FSL)** discussed in the previous two chapters. Unlike ZSL, which operates without prior examples, and FSL, which relies on a minimal dataset, RAG leverages a real-time retrieval system to enhance generative models, enabling them to access and incorporate the most current and specific information available. This ability allows RAG to surpass the limitations of ZSL and FSL by providing personalized content tailored to individual consumer profiles or current market conditions – capabilities crucial for micro-targeting in marketing.

The chapter will detail the operational framework of RAG, emphasizing its hybrid structure, which merges generative AI with dynamic information retrieval. This synthesis not only ensures the generation of contextually appropriate and accurate content but also introduces a level of personalization that was previously unattainable with either ZSL or FSL alone. We will explore how RAG's real-time data retrieval component plays

a critical role in adapting marketing strategies swiftly to align with consumer behavior and preferences using the consumer interaction data from a multi-product e-commerce platform as an example.

By the conclusion of this chapter, you will be equipped with the knowledge to:

- Understand the integration of RAG with traditional generative models and its superiority in handling real-time, relevant data compared to ZSL and FSL.
- Recognize the enhanced capability of RAG in micro-targeting and personalizing content, which can dramatically improve consumer engagement and conversion.
- Apply RAG concepts to develop cutting-edge marketing strategies that are not only data-driven but also highly adaptable to the nuances of consumers.

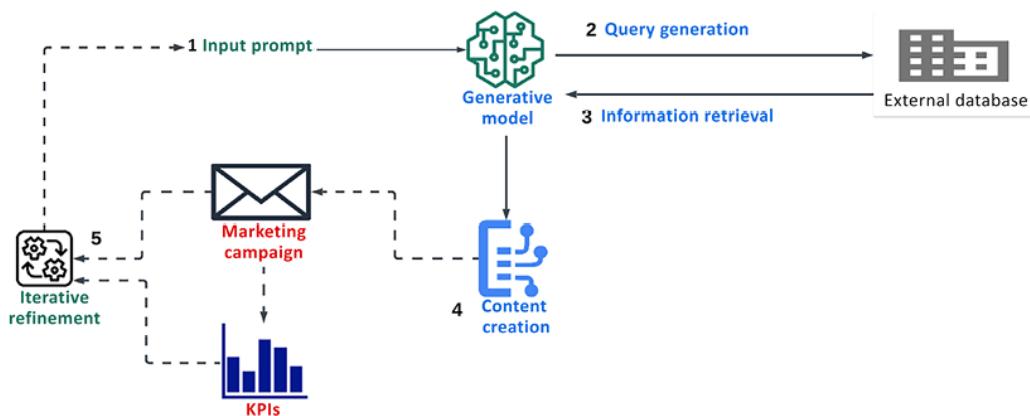
## Introduction to RAG for precision marketing

Generative models, particularly those developed on transformer frameworks like **generative pre-trained transformer (GPT)**, have revolutionized how machines understand and generate human-like text. These models are trained on vast corpora of text data and are capable of learning complex patterns and structures of language that enable them to predict and generate coherent and contextually appropriate text sequences. However, despite their sophistication, pure generative models often lack the ability to incorporate real-time, specific information that isn't explicitly present in their training data.

This is where the “retrieval” component of RAG comes into play. RAG is a fusion of **Generative AI (GenAI)** with information retrieval systems, forming a hybrid model designed to enhance the quality and relevance of generated content. RAG achieves this by incorporating a dynamic retrieval component that pulls relevant information from an external dataset or knowledge base during the generation process. The retrieval system in RAG is designed to query a structured or unstructured database to fetch information that is relevant to the context of the generation task. This approach ensures that the generative model has access to the most current and relevant data to enhance the accuracy and relevance of the content it produces.

## How RAG works

In addition to its role in enhancing content relevance, RAG enables a bidirectional flow of information between generative models and external databases. As illustrated in *Figure 11.1*, the process of RAG can be broken down into several key steps, with each step discussed further:



*Figure 11.1: The key components of the RAG workflow*

Let's explore these steps further:

1. **Input prompt:** In this step, the generative model is fed the user's input prompt or the ongoing (fine-tuned) generation context.
2. **Query generation:** Initially, the generative model produces a query based on the input provided in step 1. This query is written to retrieve the most relevant information from the knowledge base. This bidirectional flow of information between the generative model and the external database fosters a symbiotic relationship, enriching both data utilization and model performance.
3. **Information retrieval:** In step 3, the query is then processed by the retrieval system, which searches through the database to find matches or closely related information. The sophistication of this system can vary, from simple keyword-based searches to more complex neural network models that understand semantic meanings. The retrieved data is then integrated into the content generation process in the next step.
4. **Content generation:** Armed with the retrieved data, in step 4, the generative model then incorporates this information into its ongoing text generation process. This step is crucial as it involves blending the newly retrieved data with the generated content to maintain coherence and flow. This content may serve as the basis for an initiative like a marketing campaign, as illustrated above.
5. **Iterative refinement:** Often, the process is iterative, with the generative model adjusting the queries based on the feedback loop between what has been generated and what needs to be generated next. This is captured in step 5, where the iterative refinement loop ensures the continuous adaptation and optimization of the generative process based on feedback such as marketing KPIs or customer feedback.

Next, we will discuss how the query generation and retrieval steps work from a mathematical perspective.

# Mathematical model of RAG retrieval

The mathematical model of RAG captures the relationship between a document query and its retrieval components (steps 2 and 3 in *Figure 11.1*, respectively), laying the groundwork for its dynamic content generation process. At its core, RAG combines probabilistic frameworks to integrate text generation with information retrieval, which can be expressed through equations that quantify the probabilities involved in generating text given an input and retrieved documents.

The process begins with the generative model creating a query  $q$  based on the input prompt or ongoing context. Simultaneously, the retrieval system processes this query to fetch relevant documents from the database. The retrieval process is governed by a probability model that estimates the relevance of each document in the database given the query  $q$ . This estimation incorporates factors such as semantic similarity and document recency, ensuring that the retrieved information aligns closely with the context of the generation task. Mathematically, the probability of selecting document  $r$  from the database given the query  $q$  can be expressed as:

$$P(r | q, D) = \exp(score(q, r)) / \sum_{r' \in D} \exp(score(q, r'))$$

Where  $D$  represents the set of all possible documents in the database and  $score(q, r)$  is a function that computes the relevance of document  $r$  to the query  $q$ . The score function is typically learned through training and can involve complex embeddings that capture the semantic similarity between the query and the documents. The denominator in the equation ensures that the probability of selecting any document from the database sums to 1.

Now that we know how the model works, let's discuss the critical importance of the external data used in RAG systems.

## The importance of data in RAG

External data acts as the cornerstone of the RAG framework, influencing both the quality of the content generated and the accuracy of the information retrieved. In marketing, where precision and relevance directly correlate with consumer engagement and conversion, the proper application and management of data within a RAG system becomes critical. There are two fundamental components embedded within the retrieval framework that warrant deeper exploration and should be prioritized due to their profound impact on the resulting outcome:

- **Data freshness:** Keeping data up to date allows the model to capitalize on current trends, events, and behaviors, enhancing consumer engagement.
- **Data specificity:** By incorporating detailed consumer data, content can be precisely tailored to individual preferences and behaviors. This specificity not only increases the relevance of marketing messages but also can significantly boost conversion rates.

Now let's look at these concepts in detail.

### Data freshness

In practical terms, data freshness means the system consistently accesses and prioritizes the most recent information available. This is particularly significant in marketing, where information can quickly become outdated due to rapidly changing market conditions or consumer preferences. For example, during a major sales event like Black Friday, having the latest data

allows RAG systems to produce content that highlights current promotions, available stock levels, or last-minute deals, greatly enhancing the effectiveness of marketing campaigns. Unlike previously discussed GenAI approaches such as ZSL and FSL, which primarily rely on pre-existing datasets and may not update their knowledge base in real time, RAG systems integrate a dynamic retrieval mechanism that actively fetches and utilizes the most current data available.

To ensure content remains relevant, RAG systems can adjust their retrieval processes to favor newer documents. Mathematically, this adjustment can be represented by modifying the relevance score to include a term that increases the weight of more recent documents:

$$score(q, r) = relevance(q, r) + \lambda \times recency(r)$$

Where  $\lambda$  is a parameter that determines the impact of a document's recency on its score. By prioritizing recent data, RAG systems can respond more dynamically to the latest trends, ensuring that the content they generate resonates with current consumer behaviors and preferences.

## Data specificity

Data specificity refers to how detailed and relevant the information is in relation to a specific query. In RAG systems, high data specificity ensures that the content retrieved and generated is directly aligned with the user's current needs or questions, therefore enhancing user engagement and satisfaction. While other methods like transfer learning also allow access to detailed datasets, RAG integrates real-time data retrieval with generative capabilities, making it particularly well-suited for applications where up-to-the-minute information is crucial.

In technical terms, RAG systems can use advanced semantic matching algorithms to ensure that the content they retrieve matches the user's query not just by keywords but in overall meaning and context. This approach involves adjusting the scoring mechanism to prioritize documents that are not only relevant but also contextually specific to the user's inquiry:

$$score(q, r) = semantic\_similarity(q, r) \times specificity\_weight(r)$$

For example, consider a user searching for "best skincare for sensitive skin". A RAG system with high data specificity would be able to pull and generate content that not only mentions skincare products but specifically addresses products designed for sensitive skin, potentially including user reviews, expert advice, and the latest product innovations. This level of detail in content generation can significantly improve the effectiveness of personalized marketing campaigns, enhancing customer conversion rates and building brand loyalty.

## **Understanding the retrieval index**

The **retrieval index** is a fundamental component of any RAG system, acting as the bedrock on which the retrieval functionality operates. It ensures that the queries processed by the system are matched with accurate and contextually relevant responses. Managing this index effectively is crucial for the system's performance and can involve several key strategies. Let's look at some management approaches.

## **Indexing strategies**

Effective indexing is vital for the swift and accurate retrieval of information. The backbone of a robust RAG system lies in its ability to quickly sift through vast amounts of data and find information that best matches the user's query. This is achieved through sophisticated indexing strategies that organize data in a way that optimizes search processes, such as:

- **Inverted indices:** These are used to store a mapping from content keywords to their locations in the database. For example, in an e-commerce setting, an inverted index might link terms like “wireless headphones” or “thermal jacket” directly to the relevant product listings. This allows the system to quickly gather all relevant documents during a query, enhancing search efficiency and response speed.
- **Vector space models:** This approach involves converting text into vector forms or embeddings that are easy to compare and analyze. Using algorithms like TF-IDF or neural network-based embeddings, such as those introduced in *Chapters 5, 9, and 10*, these models help in understanding and comparing the semantic similarity between the user's query and available documents, leading to more nuanced and contextually appropriate responses.
- **Graph-based indexing:** This approach is useful for applications like social media analytics where understanding complex user relationships enhances content targeting. More generally, this approach is valuable when relationships between data points are as important as the data itself.
- **Multi-dimensional indexing:** This technique is beneficial in cases such as geographic data applications such as real estate analysis where

efficient searches across multiple attributes like location and time are needed. This method is particularly valuable for queries that involve ranges or require simultaneous consideration of several dimensions.

Choosing the appropriate indexing strategy depends on the application's data needs and query complexity. Inverted indices are optimized for quick keyword lookups, ideal for environments where speed is crucial. Vector space models offer richer semantic analysis and are better suited for contexts requiring a deeper understanding of content. For handling more complex data relationships or multiple query dimensions, graph-based and multi-dimensional indexing provide valuable solutions.

## **Data curation and updating**

To maintain its efficacy, the retrieval index must be regularly updated and curated. The following are a couple of concepts to consider:

- **Dynamic updating:** As new information becomes available or old information becomes obsolete, the index needs to be updated to reflect these changes. This ensures that the RAG system remains effective and relevant over time. For instance, in a marketing campaign, promotional content might need frequent updates to reflect the most current offers by the company so that the RAG system can adjust its responses.
- **Automated monitoring:** Implementing automated systems to continuously monitor and update indices can greatly enhance a RAG system's responsiveness to changes. In digital marketing, this could include adjusting strategies based on new consumer trend analyses to make sure the marketing content is aligned with the economic environment.

Proper data updating and curation are key for keeping RAG systems useful, especially in marketing, given that these systems currently do not include advanced data management features and often need separate practices to manage data curation and updates.

## RAG implementation challenges

There are several challenges in RAG implementation that are worth highlighting. As discussed in the previous section, ensuring the quality and relevance of the underlying external database is crucial, and often separate data engineering practices are needed to achieve this. The following are a couple of other major issues that RAG systems can face:

- **Handling long documents:** RAG systems need to break retrieval documents into smaller chunks to avoid memory issues and slow response times. However, breaking longer documents into smaller chunks can lead to a loss of context. To address this, solutions such as summarization algorithms and context-aware chunking methods can be used.
- **Retrieving relevant data from multiple sources:** Combining data from various sources can be challenging due to inconsistencies and differences in data formats. For instance, if customer data comes from both chat history and tabular purchase data, the system might face issues linking the two sources and generate marketing messages based on inconsistent information. Techniques such as multi-modal AI models (see *Chapter 12*) and knowledge graphs can help address this.

Despite these challenges, major organizations such as Meta AI Research, Google AI, and Microsoft have successfully implemented RAG systems. For instance, Google has integrated RAG to enhance the relevance of its

search results and Microsoft integrates RAG within Azure AI services to improve virtual assistants' capabilities. RAG is also becoming increasingly accessible to smaller organizations through services that offer to curate their existing enterprise data into formats that are compatible with RAG applications.

## **Applications of RAG in marketing**

RAG systems that uniquely combine real-time data retrieval with advanced content generation offer a number of transformative applications in marketing.

Given that these systems enable marketers to craft personalized and contextually aware content by integrating the most relevant data into the content creation process, there are some pivotal areas where RAG systems can significantly enhance marketing strategies:

- **Dynamic personalized content creation:** RAG systems excel in generating advertising content that adapts in real time to changes in user interactions, browsing behaviors, or purchase histories. By accessing data specific to an individual's recent activities, RAG can tailor advertisements to match personal preferences and interests.

**Example:** Imagine a user recently explored camping gear online. A RAG system could use this data to dynamically create ads for related items like hiking boots or travel guides. This targeted advertising not only increases the relevance and appeal of the ads but also boosts engagement rates and potential conversions. We will discuss other examples of this in the hands-on exercise presented later in this chapter.

- **Segment-specific content generation:** RAG enables marketers to produce content finely targeted to specific demographic segments, enriched with the latest data relevant to these groups. This strategy ensures the content deeply resonates with its intended audience, therefore enhancing reader engagement and brand loyalty.

**Example:** A RAG system might generate blog posts for first-time home buyers that include up-to-date mortgage rates and real-time housing market trends. This not only provides valuable information but also establishes the brand as a credible resource in the home-buying journey.

- **Enhanced customer engagement:** By harnessing the power of real-time data retrieval and advanced content generation, these systems enable the production of highly personalized content that resonates deeply with individual customers. Whether through personalized emails, targeted social media posts, or bespoke newsletters, RAG systems ensure that all communications are timely, relevant, and tailored to each customer's unique profile.

**Example:** For a customer who recently celebrated a significant event like an anniversary, RAG can generate personalized greetings or offers that reflect the customer's preferences and past interactions with the brand. This could include curated content, special promotions, or personalized messages from brand ambassadors, creating a highly personalized and memorable customer experience.

- **Automated response generation for customers:** RAG enhances customer support by generating informative and contextually relevant responses to inquiries. By pulling information from FAQs, product manuals, or customer databases, RAG systems deliver precise, customized answers.

**Example:** When a customer inquires about the return policy for an online purchase, RAG can generate a response that includes details tailored to the item's category or the date of purchase. This application of RAG in customer support not only improves response times but also boosts overall customer satisfaction.

Now that you have the theoretical knowledge in place, let's move on to the application and get familiar with the process of building a knowledge retrieval system.



If you would like to explore RAG further, here are a couple of valuable resources:

- *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* (<https://arxiv.org/pdf/2005.11401.pdf>) is a seminal paper by researchers from Meta AI that explores the integration of retrieval and generation to enhance language models for knowledge-intensive tasks.
- *Searching for Best Practices in Retrieval-Augmented Generation* (<https://arxiv.org/abs/2407.01219>) is a more recent review/survey of the literature covering key advancements and current challenges.

# Building a knowledge retrieval system for

# marketing with LangChain

This section will explore the practical aspects of setting up a knowledge retrieval system specifically tailored for marketing purposes using LangChain, a library designed to facilitate the integration of language models with retrieval systems. We will do this via an example that readers can follow to construct their own retrieval systems for enhancing the capabilities of generative AI in their own marketing strategies.

## Introduction to LangChain

LangChain is a framework that significantly enhances the capabilities of language models by integrating them with retrieval systems. Its flexible design allows it to support a variety of backend databases and customizable retrieval strategies using modules like `langchain-retrieval` and `langchain-storage`.



### LangChain essentials

LangChain facilitates the combination of language models with robust retrieval systems. This integration captures the principles of RAG, enabling applications to generate content that is not only human-like but also highly relevant and informed by the latest data.

Further documentation and resources can be found on LangChain's official documentation page:  
[https://python.langchain.com/docs/get\\_started/introduction/](https://python.langchain.com/docs/get_started/introduction/)

LangChain stands out as a top choice for Python developers looking to integrate advanced AI capabilities into their marketing strategies. While other tools such as LlamaIndex specialize in data indexing and retrieval for quick access to large datasets, LangChain offers a more versatile platform for creating complex NLP applications. Some of the key features making LangChain ideal for Python developers are the following:

- **Ease of integration:** LangChain simplifies the incorporation of complex AI models with retrieval databases through high-level abstraction, allowing you to focus more on creating unique application logic rather than wrestling with backend complexities, speeding up development and deployment processes.
- **Modularity:** The framework's high modularity supports a diverse range of language models and retrieval databases, alongside custom logic for specific interactions.
- **Scalability:** LangChain is designed to scale effortlessly from handling small tasks to managing large-scale applications, such as real-time personalized advertising and extensive email marketing campaigns. This scalability ensures that as marketing strategies grow and evolve, their technological infrastructure can grow without needing a complete overhaul.

In the following sections, we will dive deeper into setting up LangChain, designing a retrieval model, and integrating this system with generative models to demonstrate its application in real-world marketing scenarios.

## **Understanding the external dataset**

The dataset that we'll be using as the source of external data for our retrieval model comes from a large multi-category e-commerce platform and captures user interactions over the course of one month (December 2019). It includes different types of events related to product interactions by users, such as viewing, adding to the cart, removing from the cart, and purchasing.

We will use this as a basis for our setup process of a knowledge retrieval system with LangChain as it will set the stage for the hands-on example we will walk through in the final part of the chapter, where we will be using the user interactions from this same database for the purpose of RAG micro-targeting of consumers.



### Source code and data:

<https://github.com/PacktPublishing/Machine-Learning-and-Generative-AI-for-Marketing/tree/main/ch.11>

### Data source:

<https://www.kaggle.com/code/annettecatherinepaul/ecommerce-analysis-and-recommendations/input>

Before incorporating this dataset into our retrieval model, it's important to thoroughly understand its characteristics to ensure effective data ingestion and utilization in the retrieval system. A preliminary examination of the data can tailor the retrieval architecture and refine the indexing strategy, facilitating optimized query responses. To start, let's examine the first few entries to understand the type of data each column contains:

```
import pandas as pd
df = pd.read_csv("2019-Dec.csv")
df.head(3)
```

This yields the following:

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
2019-12-01 00:00:00 UTC	remove_from_cart	5712790	1487580005268456287	NaN	f.o.x	6.27	576802932	51d85cb0-897f-48d2-918b-ad63965c12dc
2019-12-01 00:00:00 UTC	view	5764655	1487580005411062629	NaN	ond	29.05	412120092	8adff31e-2051-4894-9758-224bfa8aec18
2019-12-01 00:00:02 UTC	cart	4958	1487580009471148064	NaN	runail	1.19	494077766	c99a50e8-2fac-4c4d-89ec-41c05f114554

Figure 11.2: First three rows from the Kaggle e-commerce user interactions dataset

The dataset features columns such as `event_time`, which marks when an event occurred, `event_type`, indicating the nature of the interaction (viewing, purchasing, or adding or removing from the cart), and other identifiers such as `product_id`, `category_id`, `brand`, and `price`. Taken together, this will help us understand the user's journey through the e-commerce platform, including their actions, the products involved, and the pricing dynamics during their sessions.

To deepen our understanding, we can perform basic statistical analyses that highlight the distribution and range of key data points by running the `describe` function:

```
df.describe()
```

The output shows that the dataset comprises over 3.5 million events:

	<b>product_id</b>	<b>category_id</b>	<b>price</b>	<b>user_id</b>
<b>count</b>	3.533286e+06	3.533286e+06	3.533286e+06	3.533286e+06
<b>mean</b>	5.473054e+06	1.555023e+18	8.871856e+00	5.223318e+08
<b>std</b>	1.331331e+06	1.689262e+17	1.986474e+01	8.494819e+07
<b>min</b>	3.752000e+03	1.487580e+18	-7.937000e+01	1.180452e+06
<b>25%</b>	5.726191e+06	1.487580e+18	2.060000e+00	4.866830e+08
<b>50%</b>	5.811429e+06	1.487580e+18	4.210000e+00	5.566496e+08
<b>75%</b>	5.859462e+06	1.487580e+18	7.140000e+00	5.828019e+08
<b>max</b>	5.917178e+06	2.235524e+18	3.277800e+02	5.954145e+08

*Figure 11.3: Key statistics for numeric columns in the dataset*

The product prices range significantly from as low as -\$79.37 (possibly indicating returns or pricing errors) to a high of \$327.80, with a mean of around \$8.87. Such variability suggests diverse consumer behavior and purchasing power, which are crucial for segmenting and targeting marketing campaigns.

We can also obtain a deeper understanding of two more key columns that are not included in the above summary, `event_type` and `brand`, which help us identify consumer preferences and buying patterns:

```
df['event_type'].value_counts()
```

This gives us the following output:

```
event_type
view           1728331
cart            927124
remove_from_cart  664655
purchase        213176
Name: count, dtype: int64
```

*Figure 11.4: Counts of event\_type values in the dataset*

This first output shows that views are the most common `event_type` (approximately 1.73 million instances), followed by additions to the cart, removals from the cart, and purchases. The substantial drop from cart interactions to actual purchases (over 700,000 fewer purchases than cart additions) is a point of interest for improving conversion rates. More generally, the reduction of counts from views to cart interactions to overall purchases highlights a typical e-commerce sales funnel.

By running our second command of `value_counts` on the `brand` column, we can also see which brands appear most commonly throughout the dataset:

```
df['brand'].value_counts()
```

The following is the output:

```
brand
runail      256217
grattol     155920
irisk        152518
masura       125185
bpw.style    81394
...
voesh        3
shifei       1
macadamia   1
ibd          1
dessata     1
Name: count, Length: 252, dtype: int64
```

*Figure 11.5: Counts of brand values in the dataset*

It is also important to understand the null values in the data. This gives us a way to understand both the data quality and key information around missing values that will need to be accounted for when indexing data for our retrieval index:

```
df.isnull().sum()
```

The following result indicates significant gaps in `category_code` (almost 3.5 million missing values) and `brand` (over 1.5 million missing values) data:

```
event_time          0
event_type          0
product_id          0
category_id         0
category_code      3474821
brand               1510289
price               0
user_id              0
user_session        779
dtype: int64
```

*Figure 11.6: Counts of missing values across columns in the dataset*

This could hinder our ability to perform detailed product category analysis and brand-specific marketing using this dataset. The minor missing count in `user_session` (779) suggests nearly complete tracking of user sessions, which is most vital for our goal of analyzing user behavior throughout their sessions.

## Designing the retrieval model with LangChain

As we move forward in constructing our knowledge retrieval system using LangChain, the next step involves the effective indexing of our e-commerce dataset. Indexing is a foundational process that enables the efficient retrieval of data. This section outlines the setup of an indexing system to support the queries that will allow us to achieve our goal of micro-targeting based on their shopping behaviors. To implement our retrieval system, we will utilize Elasticsearch for our backend system due to its robust full-text search capabilities and suitability for handling large datasets.

# Install and connect to Elasticsearch

Before creating the index, we need to ensure that Elasticsearch is installed and running on our system. Follow these steps based on your operating system:

Here are the steps to be followed by Windows users:

1. **Download:** Visit the Elasticsearch official download page (<https://www.elastic.co/downloads/elasticsearch>) and download the latest version for Windows.
2. **Install:** Extract the downloaded ZIP file to your desired location and navigate to the extracted folder.
3. **Start:** Open Command Prompt as `Administrator`, change directory to where Elasticsearch is installed (e.g., `cd C:\path\to\elasticsearch-<version>\bin`), and execute `elasticsearch.bat` to start Elasticsearch. Substitute `<version>` with the specific version of Elasticsearch that was downloaded from their official download page.

macOS/Linux users can follow these steps:

1. **Download:** Use the below command in a terminal window, replacing `<version>` with the latest version:

```
curl -L -o https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-<version>.tar.gz
```

2. **Install:** Extract the file with the below command and navigate to the newly created directory:

```
tar -xzf elasticsearch-<version>-darwin-x86_64.tar.gz
```

3. **Start:** Open a terminal window and change to the `bin` directory:

```
cd elasticsearch-<version>/bin
```

Start Elasticsearch by running:

```
./elasticsearch
```

If you have an ARM-powered MacBook, you may need to install via Homebrew instead. Instructions for this, as well as alternative ways of installing Elasticsearch, can be found on their website:

<https://www.elastic.co/guide/en/elasticsearch/reference/7.17/install-elasticsearch.html>.

After starting Elasticsearch, verify that it is running by using the following `curl` command in your terminal or command prompt:

```
curl http://localhost:9200
```

You should see a JSON response from Elasticsearch indicating that it is running correctly, such as the following:

```
{
  "name" : "Device-name.local",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "2QND1H4pxAyi75_21C6rhw",
  "version" : {
    "number" : "7.17.4",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "79878662c54c126ae89206c685d9f1051a9d6411",
    "build_date" : "2022-05-18T18:04:20.964345128Z",
    "build_snapshot" : false,
```

```
        "lucene_version" : "8.11.1",
        "minimum_wire_compatibility_version" : "6.8.0",
        "minimum_index_compatibility_version" : "6.0.0-beta1"
    },
    "tagline" : "You Know, for Search"
}
```

With Elasticsearch running, you can now connect from Python using the Elasticsearch client. Here's how to set it up:

```
!conda install elasticsearch
from elasticsearch import Elasticsearch
es = Elasticsearch(["http://localhost:9200"])
if not es.ping():
    raise ValueError("Connection failed")
else:
    print("Connected to Elasticsearch!")
```

## Indexing data in Elasticsearch

Next, we can define the data schema and index data from the dataset. Given the diversity of data types and volume of data captured within the dataset, it is important to configure our indexing mappings to efficiently handle the range and type of data present. This mapping for our index is informed by our earlier exploratory analysis that we performed on this dataset, where we discovered that our dataset consists of over 3.5 million entries with data types ranging from integers and floating-point numbers to categorical and textual data. This diversity necessitates a robust indexing strategy to ensure efficient data retrieval and query performance:

```
mapping = {
    "mappings": {
        "properties": {
```

```

    "event_time": {"type": "date"},
    "event_type": {"type": "keyword"},
    "product_id": {"type": "integer"},
    "category_id": {"type": "long"},
    "category_code": {
        "type": "text",
        "fields": {
            "keyword": {
                "type": "keyword",
                "ignore_above": 256
            }
        }
    },
    "brand": {
        "type": "text",
        "fields": {
            "keyword": {
                "type": "keyword",
                "ignore_above": 256
            }
        }
    },
    "price": {"type": "float"},
    "user_id": {"type": "long"},
    "user_session": {"type": "keyword"}
}
}
}

```

Some key aspects of the decisions that were made in defining the above mapping types are the following:

- **Numeric and identifier fields:** `product_id`, `category_id`, and `user_id` are all stored as integers with significant ranges. Given their use as identifiers, they are mapped to Elasticsearch's integer and long data types, which are optimized for numeric operations and comparisons. `price`, a floating-point field, shows a wide range of

values, from negative (likely indicating refunds) to over 300. This field is mapped as a float to accurately represent and query price data.

- **Categorical and textual fields:** `event_type` and `user_session` are categorical fields with high cardinality. These are mapped as keyword types, which support the exact matching necessary for filtering and aggregations. `brand` and `category_code`, however, contain textual data that might be used for full-text search as well as for exact matches. These fields are defined with the text type and a keyword multi-field to allow both full-text searching and aggregations.
- **Date fields:** `event_time` represents timestamps and is mapped as a date. This allows for time-based queries, which are essential for analyzing trends over time.

Next, we will create the index based on the mapping defined above using:

```
es.indices.create(index='ecommerce_data', body=mapping, ignore=
```

Upon execution, this should result in a response such as the following:

```
ObjectApiResponse({'acknowledged': True, 'shards_acknowledged':
```

With these configurations, your Elasticsearch environment is now fully equipped for indexing and querying our dataset.

## Data ingestion into Elasticsearch

Ingesting the dataset into the Elasticsearch index is the step that involves transferring data from our DataFrame into Elasticsearch. This process uses

an efficient bulk indexing method, which is essential for handling large volumes of data effectively.

Starting from the dataset load step, we include handling of the inconsistent and missing values that we identified earlier and replace these with `None` types to avoid errors in the indexing process. Depending on your device hardware, the indexing process may take some time, ranging from minutes to hours, given the dataset size. You can track this progress using the `tqdm` progress bar.

To further optimize the indexing performance, we include the `chunk_size` parameter in the bulk indexing method. This parameter controls the number of documents processed in each bulk request and adjusting its value can impact the speed of indexing and memory usage. In addition to `chunk_size`, we also suggest truncating the data before indexing if you still experience memory issues. To truncate the data, you can use a `pandas` function such as `df = df.head(n)` to limit the DataFrame to the first `n` rows. Let's take a look at the code:

```
from elasticsearch import helpers
from tqdm import tqdm
def generate_data(df):
    for index, row in tqdm(df.iterrows(), total=len(df), desc=""):
        doc = {
            "_index": "ecommerce_data",
            "_source": {
                "event_time": pd.to_datetime(row['event_time']),
                "event_type": row['event_type'],
                "product_id": int(row['product_id']) if pd.notna(row['product_id']) else None,
                "category_id": int(row['category_id']) if pd.notna(row['category_id']) else None,
                "category_code": row['category_code'] if pd.notna(row['category_code']) else None,
                "brand": row['brand'] if pd.notna(row['brand']) else None,
                "price": float(row['price']) if pd.notna(row['price']) else None,
                "user_id": int(row['user_id']) if pd.notna(row['user_id']) else None
            }
        }
        yield doc
```

```
        "user_session": row['user_session'] if pd.notna(row['user_session'])  
    }  
    yield doc  
success, _ = helpers.bulk(es, generate_data(df), chunk_size=500)  
print(f"Indexed {success} documents successfully.")
```

Upon completion of the indexing process, you should see an output similar to the following:

```
Indexing documents: 100%|██████████| 353  
Indexed 3533286 documents successfully.
```

## Integrating with LangChain using GPT

Integrating LangChain with your Elasticsearch backend involves setting up the environment so that LangChain can use an LLM to dynamically generate content based on the data retrieved from Elasticsearch. We will demonstrate this using the gpt-3.5-turbo LLM, but you should refer to the latest LangChain documentation for the latest models available

([https://api.python.langchain.com/en/latest/chat\\_models/langchain\\_openai.chat\\_models.base.ChatOpenAI.html](https://api.python.langchain.com/en/latest/chat_models/langchain_openai.chat_models.base.ChatOpenAI.html)):

1. The first step involves initializing the GPT model as a LangChain model. You may also need to generate an OpenAI API key for the model if you haven't done this step already, a process that currently includes the creation of an OpenAI account. Further details on the instructions for this can be found on the OpenAI website (<https://platform.openai.com/docs/introduction>):

```
from langchain_openai import ChatOpenAI
# Replace 'XXX' with your OpenAI API key
llm = ChatOpenAI(openai_api_key='XXX', model= "gpt-3.5-turk")
```

- Once the model is set up, the next step involves constructing queries to fetch relevant data from Elasticsearch. This data serves as the input for GPT, allowing it to generate contextually relevant content based on user behavior.

For example, here is how you can define a function to retrieve data based on any matching query from Elasticsearch:

```
def retrieve_data_from_es(query):
    response = es.search(index="ecommerce_data", body={"query": query})
    return response['hits']['hits']
```

This function `retrieve_data_from_es` takes a dictionary representing the Elasticsearch query and returns a list of documents that match this query. The example provided fetches records associated with a specific user ID, allowing for the generation of personalized content based on the user's previous interactions, such as products viewed or added to the cart. For example, to grab content related to `user_id` 576802932, we can execute the following:

```
query = {"user_id": "576802932"}
data = retrieve_data_from_es(query)
```

- The `data` response retrieved from Elasticsearch includes detailed records of the user's activities, each of which is tagged with precise timestamps, product identifiers, category details, and session

information. We can see examples of these types of interactions via the following:

```
removal_example = next(item for item in data if item['_source']['event_type'] == 'remove_from_cart')
view_example = next(item for item in data if item['_source']['event_type'] == 'view')
print("Removal Example:\n", removal_example)
print("\nView Example:\n", view_example)
```

The following screenshot shows the output:

```
Removal Example:
{'_index': 'ecommerce_data', '_type': '_doc', '_id': '54uE0o8B7yecbdra0pil', '_score': 1.0, '_source': {'event_time': '2019-12-01T00:00:00+00:00', 'event_type': 'remove_from_cart', 'product_id': 5712790, 'category_id': 148758005268456287, 'category_code': None, 'brand': 'f.o.x', 'price': 6.27, 'user_id': 576802932, 'user_session': '51d85cb0-897f-48d2-918b-ad63965c12dc'}}

View Example:
{'_index': 'ecommerce_data', '_type': '_doc', '_id': '-4uE0o8B7yecbdra0pim', '_score': 1.0, '_source': {'event_time': '2019-12-01T00:00:58+00:00', 'event_type': 'view', 'product_id': 5817779, 'category_id': 1487580010872045658, 'category_code': None, 'brand': 'irisk', 'price': 0.79, 'user_id': 576802932, 'user_session': 'f7257a5b-5449-4136-aa56-e373f484ffa2'}}
```

Figure 11.7: example entries for remove\_from\_cart and view event types from the dataset

In these examples, one entry shows a user removing an item from the cart around midnight, suggesting reconsideration or a preference change. The second entry captures a view event, where a user is browsing products but has not yet made a purchase decision. Understanding these interactions can help in designing personalized re-engagement strategies to encourage users to complete their purchases and enhance conversion rates.

Now that we have designed, implemented, and tested the framework for our knowledge retrieval model, let's explore the application of this model via an example.

## Implementing RAG for micro-targeting based on customer

# **data**

Having thoroughly analyzed the dataset and constructed a robust retrieval system, we now transition from theoretical frameworks to practical implementation. In this section, we will learn how to apply RAG to dynamically address common challenges in digital marketing, such as outdated information in trained models and capturing recent user interactions. Traditional **zero-shot learning (ZSL)** and **few-shot learning (FSL)** models, while powerful, often lag in real-time responsiveness and rely on pre-existing data, limiting their effectiveness in such a fast-paced marketing scenario.

To overcome these limitations, we will utilize RAG to generate marketing content that is not only up to date but also deeply relevant to current consumer behaviors. By integrating our retrieval system with GPT, we can pull the latest user interaction data directly from our database. With RAG, we can also generate real-time content tailored to individual user scenarios, effectively re-engaging customers who have abandoned their carts. This approach allows us to generate marketing strategies that are informed by the most recent and relevant user activities, something that static models cannot achieve.

## **Determining the campaign strategy**

As we transition from the theoretical frameworks discussed earlier, it's important to understand the strategic underpinnings of the upcoming example where we demonstrate how RAG can be applied. Before diving into the real-world examples, we will set the stage for the basis of our

micro-targeting strategies by performing a couple more crucial elements of exploratory analysis on the dataset in order to determine our optimal campaign strategy and its goals.

## Message timing

Efficiency in digital marketing not only reduces costs but also enhances the effectiveness of campaigns. One strategic component for our campaign will be the optimization of the timing for our marketing messages. As discussed in *Chapter 4*, understanding the time-series trends in user interactions can be particularly insightful for this. Rather than performing a date-based analysis, since we only have access to one month of data, we will instead examine how interactions vary by time of day so that we can use this information to give us insight into the optimal time of day for user engagement for the following months. We can extract the insights needed to make this decision via the following:

```
import matplotlib.pyplot as plt
df['event_time'] = pd.to_datetime(df['event_time'])
df['time_of_day'] = df['event_time'].dt.hour
time_of_day_data = df.groupby(['time_of_day', 'event_type']).size()
fig, ax = plt.subplots(figsize=(12, 6))
time_of_day_data.plot(ax=ax, title='User Interactions by Time of Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Events')
plt.xticks(range(0, 24))
plt.grid(True)
plt.show()
```

This gives us the following graph:

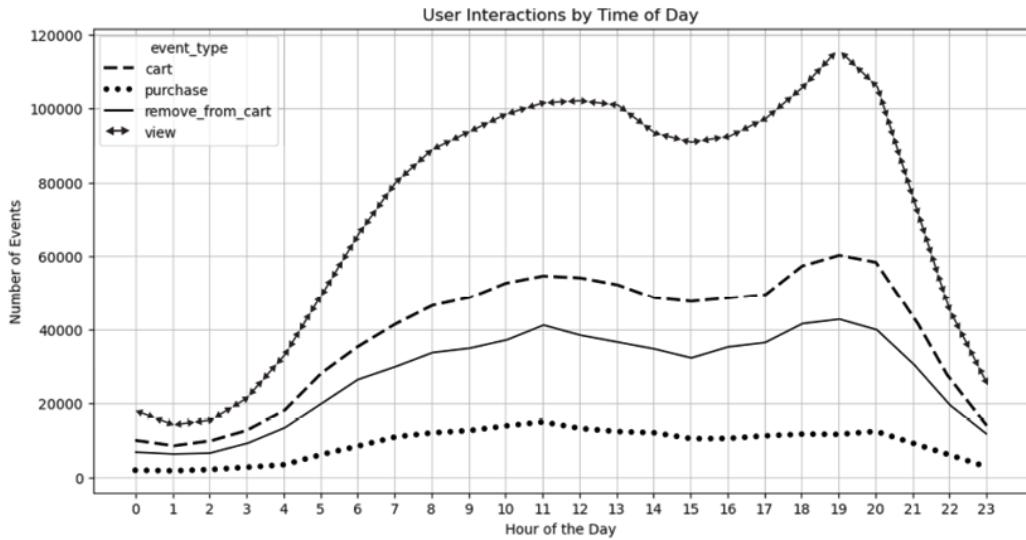


Figure 11.8: Number of user interactions by event type across different hours of the day

From the plotted data, we observe a few distinct user behavior patterns throughout the day that have important implications for the timing of our micro-targeting strategy:

- 1. Peak viewing hours:** The highest user activity, specifically views, occurs between 17:00 and 20:00. This peak period is optimal for pushing advertisements and personalized content to maximize exposure and engagement.
- 2. High transaction activities:** Cart additions and purchases are also generally high from 17:00 to 20:00. This indicates not only active browsing but also a higher propensity to finalize purchases, making it an ideal time for promotional offers.
- 3. Cart removal insights:** Cart removals closely mirror the two preceding trends and peak between 18:00 and 20:00, suggesting a reconsideration phase among users. This period could be strategically targeted with reminders or incentives such as discount offers or free shipping to convert hesitations into sales.

**4. Early morning and late night trends:** While there's a gradual increase in activity from the morning, it sharply declines after 21:00, indicating late night hours might not be as effective for targeted campaigns.

Given these patterns, a targeted strategy can be specifically implemented during the 19:00 to 21:00 window to mitigate cart abandonment rates. During this time, deploying personalized re-engagement campaigns such as sending reminder emails, offering limited-time discount codes, or displaying pop-up messages with special offers can be particularly effective.

For your own analysis, it is important to consider that behavior patterns will differ when analyzing user interactions across different time zones. It is therefore important to confirm that user time zones are accurately recorded and stored in your database, or that time zone differences are accounted for after the fact based on the region of the user. Failing to do so could result in analysis that involves a convolution of user behaviors across regions, leading to poor message timing.

## Choosing the brand

As we continue to refine our marketing strategy, it's crucial to identify which brands within our dataset present the highest potential for increasing conversion rates. In previous analyses, we explored the frequency of brand appearances to gauge their prevalence. Let's now go deeper by examining the top five brands to understand how different types of interactions vary across these brands:

```
top_brands = df['brand'].value_counts().nlargest(5).index  
brand_event_type_counts = df[df['brand'].isin(top_brands)].grou
```

```
Brand_event_type_counts
```

This yields the following output:

event_type	cart	purchase	remove_from_cart	view
brand				
<b>bpw.style</b>	21995	7014	18014	34371
<b>grattol</b>	37841	8171	26651	83257
<b>irisk</b>	45841	10583	29565	66529
<b>masura</b>	33788	6985	34049	50363
<b>runail</b>	80411	18199	49510	108097

Figure 11.9: Number of user interactions by event type for the five most popular brands

Analyzing the brand interactions, `bpw.style` stands out as a candidate for focused improvement, especially in terms of cart abandonment metrics. While `bpw.style` shows a substantial presence in the dataset, there's a noticeable discrepancy between the number of items added to carts (21995) and those removed (18014). This pattern suggests a significant gap between initial interest and final purchasing decisions.

To quantify the opportunity for improvement, let's calculate the cart abandonment rate for the `bpw.style` brand:

```
abandon_rate_bpw = brand_event_type_counts.loc['bpw.style', 're  
print(f"Cart Abandonment Rate for bpw.style: {abandon_rate_bpw:
```

```
Cart Abandonment Rate for bpw.style: 0.82
```

Addressing the gap in cart abandonment could significantly boost `bpw.style`'s performance by converting potential sales into actual purchases.

## **Using LangChain for micro-targeting**

Earlier, we established a retrieval system integrated with GPT to dynamically generate content based on the data from Elasticsearch. We will now utilize the marketing campaign goals that we just defined to leverage the insights gained from our analysis of user behavior by brand and timing to create highly personalized content aimed at enhancing customer engagement. We will start with two examples that are user-specific, before transitioning to a third example that is targeted specifically toward consumers of the `bpw.style` brand.

### **Case study 1: Targeted product discounts**

Given that our data indicates that user interactions, especially cart additions and removals, peak roughly between `17:00` and `21:00`, we will use this timeframe as the optimal window for sending out marketing messages, capturing when users are most active and likely reconsidering their purchase decisions. The preferred medium for this outreach will be email, which allows for rich, personalized content and can effectively re-engage users by reminding them of items left in their carts or providing time-sensitive offers.

To generate tailored marketing content, we deploy a function that constructs prompts for GPT based on user interactions, transforming these data points into actionable marketing strategies:

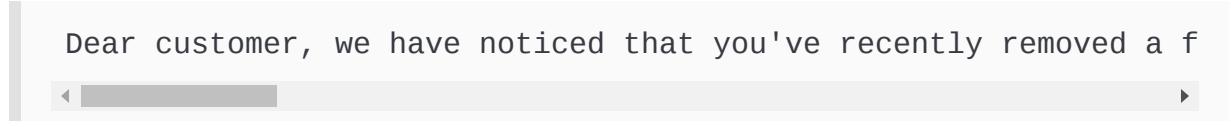
```
def generate_content(data):
    if not data:
        return "No data available to generate content."
    messages = [
        ("system", "You are an assistant that generates marketi")
    ]
    for item in data:
        source = item['_source']
        product_description = f"{source['event_type']} the prod"
        messages.append(("human", product_description))
    messages.append(("human", "Based on these interactions, sug"))
    try:
        response = llm.invoke(messages)
        return response.content
    except Exception as e:
        return f"Error generating content: {str(e)}"
```

This function constructs a series of messages to feed into the LLM, each representing a significant user interaction, such as adding to a cart or removing from it. These descriptions culminate in a request for a marketing strategy tailored to these activities, prompting the LLM to generate a custom message.

Going back to our earlier analysis of the data stored in our retrieval index for “`user_id`”: “`576802932`”, we can run an analysis on this customer and obtain a response such as the following:

```
query = {"user_id": "576802932"}
data = retrieve_data_from_es(query)
generate_content(data)
```

This yields the following:



Dear customer, we have noticed that you've recently removed a f

Our marketing campaign could time this prompt generation to take place based on the most recent customer interaction data for this consumer as of roughly 17:00 and then send the email shortly thereafter. This not only provides a direct incentive to reduce abandonment but also creates a sense of urgency that the customer is more likely to act upon.

## Case study 2: Product upselling

Building on the insights from the first case study, which focused on reducing cart abandonment rates through targeted discounts during peak interaction hours, this second case study explores a complementary strategy. Here, we aim to enhance product upselling by offering personalized product recommendations based on user behavior and preferences during similar peak hours.

This function is designed to harness user data to not only remind them of their abandoned carts but also to upsell related or complementary products based on their browsing and purchasing history. By doing this, we not only aim to recover abandoned carts but also increase the average order value:

```
def generate_upsell_content(data):
    if not data:
        return "No data available to generate content."
    messages = [("system", "You are an assistant that generates
for item in data:
    source = item['_source']
    messages.append(("human", f"Identify complementary prod
messages.append(("human", "Suggest an upselling strategy th
```

```
try:  
    response = llm.invoke(messages)  
    return response.content  
except Exception as e:  
    return f"Error generating content: {str(e)}"
```

This function is designed to construct a narrative that not only addresses the immediate need to reduce cart abandonment but also strategically suggests additional products that enhance the user's initial choice, potentially increasing the transaction value.

We can now apply this function to the same consumer as last time and see the result, this time prompting the model to return a full email message in its entirety:

```
query = {"user_id": "576802932"}  
data = retrieve_data_from_es(query)  
output = generate_upsell_content(data)
```

This produces the following email content:

Subject: You've left something behind!  
Dear [Customer's Name],  
We noticed that you recently viewed a Runail product for just \$0.40 on our website, but didn't go through with the purchase. We thought we'd let you know that this item is still available and waiting for you!  
But wait, there's more! Why not complete your self-care routine with these complementary products that you might like:  
1. Runail Professional Nail Polish - Adds a pop of color to your nails after using the Runail product you viewed.  
2. Runail Hand Cream - Nourishes your hands and nails, ensuring they stay healthy and beautiful.  
3. Runail Nail File - Helps shape your nails perfectly to complement the Runail product you viewed.  
And the best part? If you purchase the Runail product along with any of these complementary items, we'll offer you a 15% discount on your total purchase.  
So why wait? Come back and complete your purchase, and upgrade your self-care routine today!  
Best,  
[Your Name]  
[Your Position]

*Figure 11.10: Tailored upselling content strategy generated for user 576802932, based on their interaction data retrieved from the Elasticsearch database*

## Case study 3: Real-time content customization for bpw.style

Building on our strategy to utilize RAG for micro-targeting, this case study focuses specifically on enhancing engagement and reducing cart abandonment rates for the `bpw.style` brand. Recognizing the disparity between cart additions and completions, as previously analyzed, we target users who might benefit from a gentle reminder or an incentive to complete their transactions. This query will specifically look for cart addition and removal events:

1. Here is a function with the Elasticsearch query to fetch this specific type of brand interaction data:

```
def retrieve_bpw_style_data(es_client):
    query = {
        "bool": {
            "must": [
                {"match": {"brand": "bpw.style"}},
                {"terms": {"event_type": ["cart", "remove_f"]}}
            ]
        }
    }
    response = es_client.search(index="ecommerce_data", body=query)
    return response['hits']['hits']
```

2. With the targeted data retrieved, we can now integrate this into our `generate_reengagement_content` function. The modified function will use the data fetched by our new query function to generate personalized re-engagement strategies:

```
def generate_reengagement_content(es_client):
    data = retrieve_bpw_style_data(es_client)
    if not data:
        return "No data available to generate content."
    messages = [
        ("system", "You are an assistant that creates re-engagement content."),
    ]
    for item in data:
        source = item['_source']
        interaction_desc = f"User showed interest {item['interaction']}"
        messages.append(("human", interaction_desc))
    messages.append(("human", "Generate a short personalized message"))

    try:
        response = llm.invoke(messages)
        return response.content
    except Exception as e:
        return f"Error generating content: {str(e)}"
```

3. We can run our new query and content generation commands via the following:

```
marketing_message = generate_reengagement_content(es)
print(marketing_message)
```

This yields the following:

Subject: Your bpw.style Favorites are Waiting for You!

Dear [Customer's Name],

We noticed that you've been browsing our collection and have added some of our stunning bpw.style products to your cart. However, it seems like you didn't complete your purchase.

We understand, life gets busy and sometimes it's easy to forget where we left off. That's why we've saved your cart for you! You can view your selections anytime and pick up where you left off.

We'd also like to offer you an exclusive 10% off coupon to use on your next purchase. Use the code BPW10 at checkout.

Remember, our items are in high demand and we can only reserve your cart for the next 48 hours. So don't miss out on the items you love!

Click [\[here\]](http://www.bpw.style/cart) to view your cart and complete your purchase.

If you have any questions or need further assistance, feel free to reply to this email or contact our customer support.

Happy shopping!

Best,  
The bpw.style Team

*Figure 11.11: Personalized re-engagement strategy created for users interested in bpw.style products who have abandoned their carts*

As shown by the output, by honing in on specific brands that exhibit high cart abandonment rates, we can tailor marketing strategies that address unique challenges related to brand perception and customer engagement. This approach allows for the creation of highly specific content that resonates with the brand's audience, potentially transforming browsing behaviors into completed transactions. While this strategy emphasizes brand-specific data, it can seamlessly integrate with user-specific insights illustrated earlier, including demographic information, if available, to enhance the precision and relevance of the marketing messages.

## Summary

This chapter has provided a detailed exploration of RAG and its transformative impact on precision marketing. By integrating generative AI with dynamic retrieval systems, RAG overcomes the limitations inherent in previous models like ZSL and FSL by incorporating real-time, context-specific data into content generation. This enables an unprecedented level

of personalization in marketing strategies, enhancing the relevance and efficacy of marketing content tailored to individual consumer preferences and current market conditions.

We've used practical examples and mathematical models to demonstrate how RAG effectively combines data freshness and specificity, thereby elevating consumer engagement and optimizing conversion rates.

The discussion also covered the technical mechanisms that underpin RAG, from query generation and information retrieval to the iterative refinement of generated content. These elements ensure that the content not only resonates with the audience but also stays aligned with the latest trends and data insights, a crucial advantage in today's rapidly evolving digital marketing landscape.

As we look to the future, in the next chapter, we will move on to the broader landscape of AI and ML in marketing, consolidating the knowledge acquired throughout this book while exploring emerging technologies. We will combine current methods and predicted advancements in AI that will revolutionize marketing even further. Additionally, we will examine how AI is becoming integral to emerging digital platforms that offer novel ways to engage customers and personalize marketing efforts. This forward-looking perspective will give you the insights and skills that you will need to navigate the evolving AI landscape, preparing you for the future of digital marketing.

## **Join our book's Discord space**

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)

# 12

## The Future Landscape of AI and ML in Marketing

We are now entering the final stretch of our exploration into the integration of **machine learning (ML)** and **Generative AI (GenAI)** in marketing. The landscape of AI and ML is dynamic and rapidly evolving, however, setting the stage for even more exciting changes. So far, we've explored how GenAI and data-driven insights have revolutionized marketing strategies, guiding us in everything from decoding marketing KPIs and obtaining detailed customer insights to crafting compelling content, enhancing brand presence, and micro-targeting consumers. This chapter aims to consolidate our understanding of key AI and ML concepts and project some of their future applications in marketing.

In particular, we will start by reconciling the GenAI fundamentals introduced in previous chapters before exploring emerging technologies such as multi-modal GenAI and advanced model architectures and tools that synthesize diverse data types for creating personalized content tailored to consumer behaviors and preferences. While *Chapter 7* discussed personalized recommendations using techniques such as market basket analysis, collaborative filtering, and other traditional recommendation methods, this chapter introduces how this can be achieved using multiple data types, such as text, images, and videos, using multi-modal GenAI. We

also explore more advanced model architectures, such as ReAct, which combine reasoning and action to adapt to user behavior in real time, to further push the boundaries of user personalization. Lastly, we will examine how AI is being integrated with novel platforms like augmented and virtual reality (AR/VR) to transform traditional marketing strategies into more immersive consumer experiences.

By the end of this chapter, you will gain:

- A consolidated view of fundamental GenAI concepts from past chapters
- A more comprehensive understanding of current AI and ML technologies shaping marketing and how they're expected to evolve
- Insight into the integration of AI with new digital platforms and its implications for future marketing practices

## Consolidating key AI and ML concepts

In this section, we will briefly revisit the core GenAI concepts discussed in previous chapters to prepare us for our exploration of emerging technologies and future trends. In each case, you can refer to the original chapter for a more in-depth discussion of the content and its theory.

In *Chapter 9*, we first laid the foundation for understanding the origins of GenAI models through a discussion of probabilistic models and contextual embeddings. Key concepts to remember from that chapter include:

- **Bayesian inference and probabilistic models:** Bayes' Theorem guides the model to adjust its understanding by integrating prior

knowledge with new evidence. This adaptive process is crucial for scenarios where the model encounters entirely new contexts—enabling it to refine its predictions without the need for extensive retraining.

- **GenAI models:** Foundational models such as **generative adversarial networks (GANs)**, **variational autoencoders (VAEs)**, **long short-term memory (LSTM)** networks, and **Generative Pre-Trained Transformers (GPTs)** drive innovative content creation in text, image, and sequence generation.
- **Contextual embeddings and pre-trained models:** Contextual embeddings allow AI to dynamically adjust understanding based on the surrounding text, while pre-trained models, trained on large datasets, significantly reduce the cost and time-to-deployment for AI-driven solutions. This paves the way for **zero-shot learning (ZSL)** where models can perform tasks not covered in their initial training.

In *Chapter 10*, we introduced how pre-trained models can be adapted to perform better on new tasks using **few-shot learning (FSL)** and transfer learning. The core topics discussed in that chapter include:

- **FSL:** FSL trains ML models to adapt to new tasks with only a few labeled examples. This technique involves meta-learning or “learning to learn,” allowing models to develop representations that generalize effectively to new scenarios.
- **Transfer learning:** Transfer learning applies knowledge from previous tasks to improve performance on new tasks, reducing the need for extensive training data. This method is particularly useful when there is a strong relationship between past and current tasks.
- **Cost-benefit analysis of FSL and transfer learning:** This evaluates when to use each method based on specific requirements and

constraints, considering factors such as cost, frequency of usage, and the need to capture complex patterns.

The following figure illustrates some of the differences between these techniques:

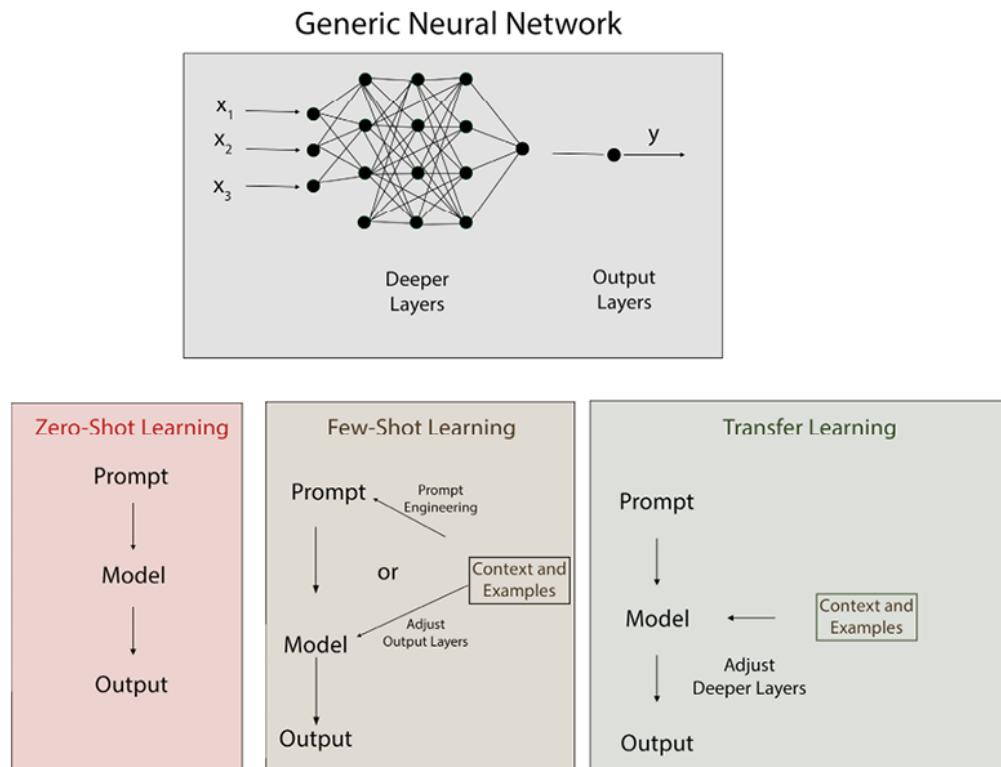


Figure 12.1: Key differences between typical ZSL, FSL, and transfer learning implementations and their mechanics

Lastly, in *Chapter 11*, we covered how micro-targeting individual consumers was made possible with **retrieval-augmented generation (RAG)**. The discussion in this chapter included:

- **RAG:** RAG enhances traditional generative models by integrating real-time data retrieval, producing personalized and timely content. This hybrid approach relies on external databases to provide contextually relevant information.

- **Applications of RAG in marketing:** RAG is used for personalized advertising, dynamic content creation, targeted marketing by demographics, and enhanced customer engagement. It enables marketers to create highly personalized content by incorporating the most current and specific information.
- **Building a knowledge-retrieval system with LangChain:** LangChain facilitates the combination of language models with robust retrieval systems. This integration supports generating content informed by the latest available data.

Now that we've recapped these key concepts, let's look ahead at some next-gen technologies that you can use.

## **Next-generation AI technologies in marketing**

This section highlights some of the significant advancements and novel model architectures that are redefining the boundaries of what's possible in AI for marketing. Each of these technologies helps us push the envelope in creating more dynamic, responsive, and personalized marketing solutions. The current maturity of these technologies will also be discussed and, in the case of ReAct and multi-modal GenAI, these technologies are already available in products and can be used by early adopters for their marketing strategies.

### **From RAG to ReAct**

ReAct, short for “Reasoning and Acting,” represents a significant evolution in GenAI systems by integrating sophisticated reasoning and external tool

interactions. ReAct builds upon the capabilities of RAG, ZSL, FSL, and prompt engineering, by incorporating chain-of-thought processes, which involve the AI system breaking down complex tasks into a sequence of smaller, manageable steps. Chain-of-thought prompting is crucial in GenAI as it guides the AI through a logical sequence of steps to reach a conclusion, improving the accuracy and relevance of its responses. This mimics human-like reasoning and also facilitates interactive tool use, making AI not only a responder but also a proactive agent in real-world applications.

## How ReAct works

ReAct integrates AI capabilities with operational actions, embodying a model where the AI's output can initiate real-world tasks and transactions. At its core, ReAct can be viewed through the framework of enhanced decision-making, where outputs are not just informational but actionable.

ReAct's real power lies in its capacity to interact seamlessly with external tools – not only databases for real-time data retrieval but also calculators for on-the-fly computations and even the internet for gathering broader contextual insights. This interaction enables the system to perform complex tasks, from calculating optimal marketing spend to finding a competitor's product information based on the most recent update of their website. The following diagram provides an overview of the key components of a ReAct system:

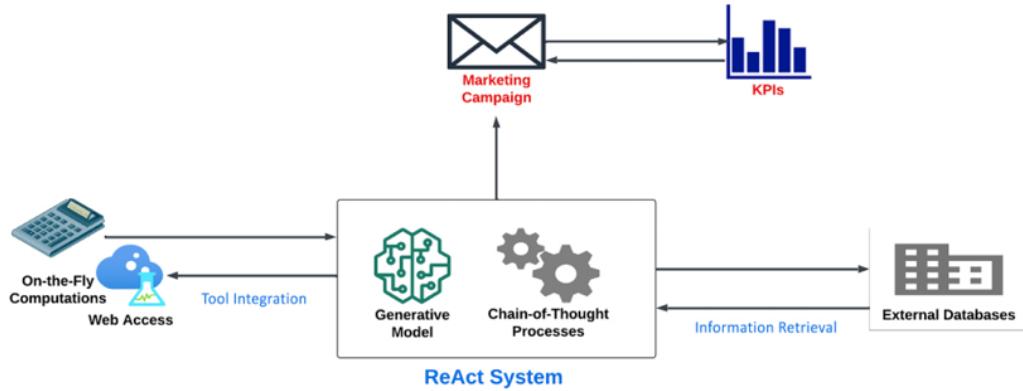


Figure 12.2: A ReAct system can integrate various tools to support a marketing campaign

## Applications in marketing

ReAct can propel marketing campaigns beyond the realm of RAG by enabling not only data retrieval but also direct, real-time interactions and executions within various external systems. This advanced functionality allows marketing strategies to automate complex processes and personalize customer interactions in ways that data retrieval systems like RAG cannot. Applications of these capabilities can include the following:

- **Real-time inventory management:** ReAct can directly interact with inventory systems to adjust marketing strategies based on stock levels. For instance, if a product is selling faster than expected, ReAct can increase advertising spend and intensify promotional activities to capitalize on the trend, while also notifying supply chain management systems to adjust inventory levels.
- **Dynamic pricing adjustments:** Leveraging ReAct, companies can implement dynamic pricing models where prices adjust in real time based on demand, competitor pricing, and inventory status. This could involve ReAct monitoring competitor prices online and automatically

adjusting its own pricing to stay competitive during high-traffic events like Black Friday.

- **Interactive customer service:** ReAct can enhance customer service by not only generating responses based on FAQs and customer data but also by executing actions like booking appointments, processing returns, or issuing refunds directly through customer service chatbots.

## Model architecture advances

As the field of GenAI continues to evolve, several notable advances have emerged that push the boundaries beyond foundational models such as GANs, VAEs, LSTMs, and transformers. These developments not only enhance the capabilities of existing models but also introduce new paradigms that are shaping the future of AI in creative and impactful ways. Here, we explore three such innovations:

- **Diffusion models:** These are revolutionizing the quality and diversity of generated images. High-quality images that are realistic to the consumer are central to optimizing customer engagement with marketing content.
- **Neural radiance fields (NeRFs):** These are transforming the way we create and interact with 3D content, enabling the marketing of VR and AR applications, which are discussed later in the chapter.
- **Capsule networks:** These offer a new approach to understanding complex data structures and spatial hierarchies that are also central to VR and AR applications.

Let's look at these in greater detail.

# Diffusion models

One of the most significant recent advancements in generative models is diffusion development. These innovative models operate through a sophisticated process that gradually learns to reverse the addition of noise to data, effectively “denoising” it step by step to recover the original data. This approach has proven to be remarkably effective for generating high-quality, high-resolution images, often surpassing the performance of traditional GANs in various scenarios. By employing a series of transitions that progressively refine the data quality, diffusion models excel at producing detailed and varied outputs. This capability makes them particularly useful for applications that demand high fidelity and diversity, such as digital art creation and photorealistic rendering for marketing materials.

The process utilized by diffusion models can be visualized through a sequence of transformations where each step incrementally denoises the input data, moving closer to the original structure. The following diagram illustrates this transformative sequence, showcasing how the model iteratively refines from a noisy state back to clear, structured data over several steps:

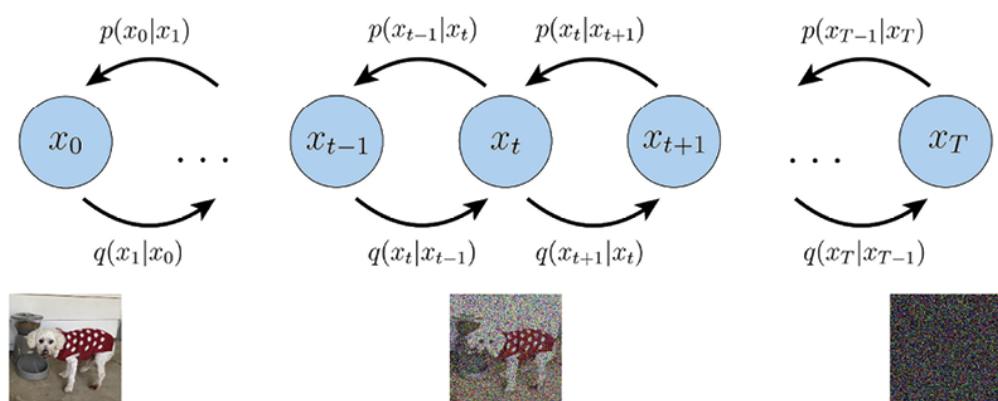


Figure 12.3: The iterative denoising process in a diffusion model (source: <https://arxiv.org/pdf/2208.11970>)

In this diagram,  $x_0$  represents true data observations such as natural images,  $x_T$  represents pure Gaussian noise, and  $x_t$  is an intermediate noisy version of  $x_0$ . Each  $q(x_t|x_{(t-1)})$  is modeled as a Gaussian distribution that uses the output of the previous state as its mean.

Diffusion models are currently used in state-of-the-art text-to-image image models such as DALL-E 3, Google's Imagen, and Midjourney. OpenAI also has a diffusion model called Sora, which generates video, although at present this model has not been released to the public.

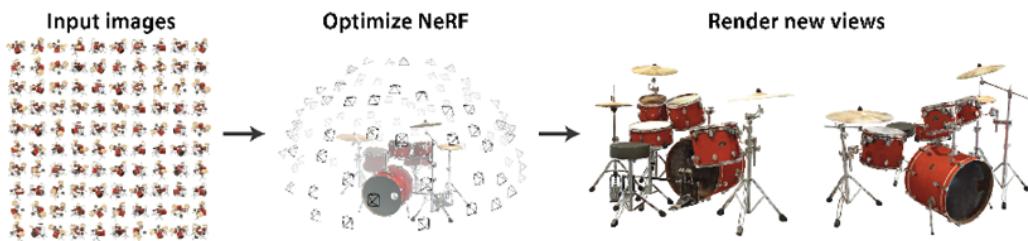
## Neural radiance fields

In the domain of 3D content generation, particularly for VR and AR applications, NeRFs have emerged as a groundbreaking technology. NeRFs use a fully connected neural network to model a scene's volumetric density and color from a set of sparse 2D images. This model can then be used to render photorealistic 3D scenes from any viewpoint, providing an extremely powerful tool for creating immersive virtual environments. The implications for marketing are vast, allowing brands to create more realistic and interactive 3D advertisements or virtual product showcases. Outside of marketing, NeRFs are also used in other applications including video games and autonomous vehicle navigation.

From a technical perspective, NeRFs use a set of input views and train a network using a positional encoding technique that transforms these views into a continuous 5D scene representation. Here, 5D refers to the 5D continuous function that NeRF models use to represent a scene, with 3 dimensions given by the spatial location  $x$ ,  $y$ ,  $z$ , and a viewing direction defined by two additional dimensions. This 5D space thus encompasses

both the position and the direction from which the position is viewed, which are crucial for rendering scenes accurately from any viewpoint.

The rendering process involves casting rays through the scene and computing the color and opacity using volume rendering techniques, which are defined by learned neural network parameters. This process is computationally intensive and often requires optimization such as hierarchical sampling to efficiently generate images. The following figure is a visual depiction of the process:



*Figure 12.4: Rendering new views via NeRF using a set of input views that are randomly captured from different viewpoints (source: <https://arxiv.org/pdf/2003.08934>)*

## Capsule networks

Capsule networks represent an intriguing development in neural networks, offering a promising alternative to traditional convolutional networks by modeling hierarchical relationships in data better. This capability can be particularly useful in tasks where spatial relationships are key, such as parsing complex scenes in AR applications or understanding the layout of web pages for automated design tasks. To date, reports of capsule network architectures being used in commercial products and technologies are limited, although various open source projects and implementations, such as CapsNet-Keras and CapsNet-TensorFlow, are available for experimentation and research.

Capsule networks structure their architecture around groups of neurons, known as capsules, which are designed to capture and maintain the probabilistic existence and instantiation parameters of different types of entities in the data. Unlike traditional convolutional networks that lose spatial hierarchies between features due to pooling layers, capsules retain this information through dynamic routing – a process that allows capsules in one layer to send their outputs to appropriate parent capsules in the next layer based on the learned “agreement” between the capsules. This mechanism requires implementing routing algorithms, such as dynamic routing by agreement, which typically involves complex iterative procedures adjusting the coupling coefficients between capsules. As a simplified illustration of what a capsule network is, we can compare a traditional artificial neuron found in a standard neural network with the capsule:

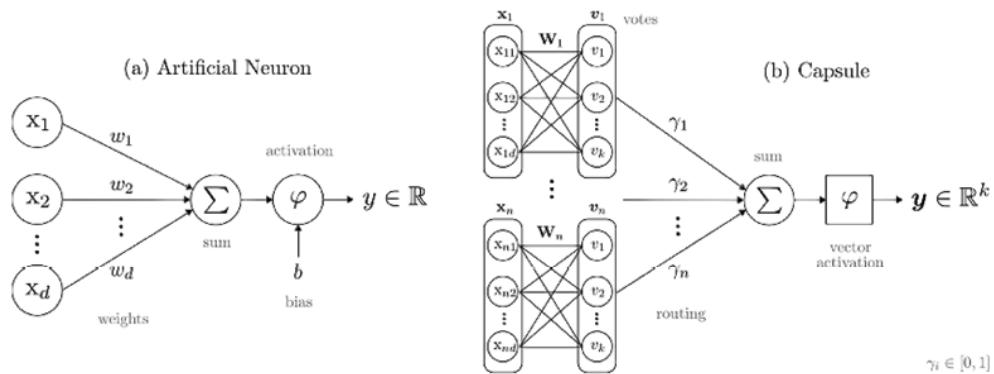


Figure 12.5: (a) The artificial neuron as found in standard neural networks and (b) depiction of a capsule as found in capsule networks (source: <https://arxiv.org/pdf/2206.02664>)

## Multi-modal GenAI

Building on the transformer-based architectures introduced in *Chapter 9*, multi-modal GenAI systems represent an advanced evolution of these

systems. These architectures can interpret and generate content by processing multiple data modalities, such as text, images, and videos, and go beyond single-mode models by integrating and cross-referencing data from diverse sources. This integration facilitates a deeper understanding of complex content and contexts, enabling the AI to perform sophisticated “cross-modal” operations.

## Technical foundations

Multi-modal models modify the traditional transformer architecture to encode diverse data forms into a shared latent space, allowing for operations across modalities. For example, these systems can generate descriptive text from a video clip or synthesize a video from a textual description, demonstrating their ability to bridge different types of information seamlessly. The core of multi-modal AI’s effectiveness lies in its sophisticated neural network architectures that feature cross-modal attention mechanisms. These mechanisms are crucial for allowing the model to focus and integrate features from different modalities effectively.

The following diagram simplifies how this process works at the conceptual level:

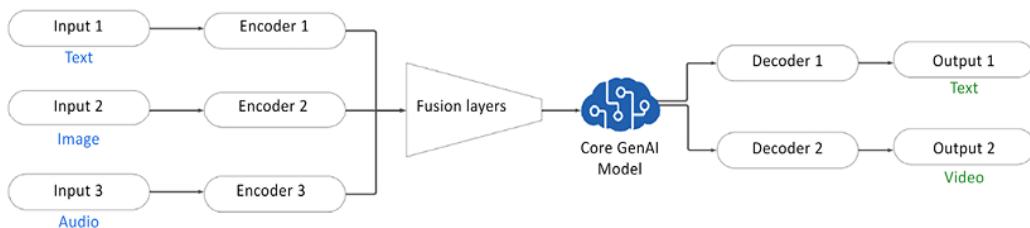


Figure 12.6: Simplified architecture of a multi-modal GenAI system

In this example, different input modalities for text, image, and audio data are fed into encoders for processing by the fusion layer. This fusion layer

combines the modalities using methods like cross-modal attention before feeding them into a core generative model. This core model is designed to process the fused multi-modal data to generate new output predictions. These outputs are then decoded to their desired modality format – in this case, text and video – for the end user.

## **Multi-modal applications**

The sophisticated approach of multi-modal GenAI allows marketers to more deeply understand and interact with consumers by incorporating diverse data sources. As multi-modal AI processes this wide array of inputs, it enables more personalized, relevant, and dynamically adaptable marketing strategies that resonate with consumers on multiple levels. The following are a few specific applications of this technology that demonstrate its potential to create more effective marketing campaigns. In each application, marketing KPIs to consider in the implementation of these strategies are given. These should be tracked to evaluate the effectiveness of the multi-modal GenAI approach and determine the ROI that it brings relative to its cost via techniques such as A/B testing (*Chapter 6*).

## **Enhanced consumer engagement**

Consider the enhanced storytelling that can come from synthesizing various data forms to allow multi-modal AI to generate rich storytelling experiences. For example, a travel agency might create a personalized travel video based on a customer's previous searches, destinations explored on Instagram, and positive reviews from similar trips. This targeted storytelling can foster a stronger emotional connection with consumers, leading to higher engagement. In implementing this strategy, it is worth tracking KPIs such as the click rate to see how much more likely users are

to interact with the multi-modal GenAI content than with traditional content.

## **Real-time content adaptation**

Multi-modal AI is crucial for real-time content adaptation in marketing, leveraging its ability to analyze and synthesize data from multiple sources effectively. For instance, consider an apparel brand that uses multi-modal AI to optimize its marketing strategies. This AI system doesn't just analyze textual data like weather forecasts, it also evaluates visual data from social media trends and video content from recent fashion shows. By assessing this combination of text, images, and video, the AI can detect subtle shifts in consumer preferences or sudden changes in weather patterns. In this case, monitoring the KPI of conversion rate is one way to see how the potentially more relevant content from GenAI leads to greater sales as opposed to traditional content that does not take into account recent trends.

## **Interactive customer support**

Retailers can implement multi-modal AI to provide interactive customer support through chatbots that understand and process both text and image inputs. For example, a customer could upload a picture of a damaged product, and the AI system could analyze the image, identify the issue, and provide appropriate solutions or initiate a return process. This capability enhances customer satisfaction by providing quick and accurate support, reducing the need for human intervention and improving overall customer satisfaction and brand loyalty. In this case, we can monitor the KPIs of first response time, issue resolution, and customer retention when users interact with the GenAI system. While the first response time is very likely to

improve, maintaining the quality of the issue resolution is an important factor to consider upon deployment.

## AR and VR as emerging digital platforms

In the dynamic world of digital marketing, AR and VR will become increasingly pivotal. They provide immersive and engaging experiences that help brands stand out in the ever-evolving digital marketplace. AR superimposes digital elements onto the physical world via devices like smartphones, while VR transports users into entirely digital environments through headsets, enabling brands to craft unique customer journeys.



### The difference between VR and AR

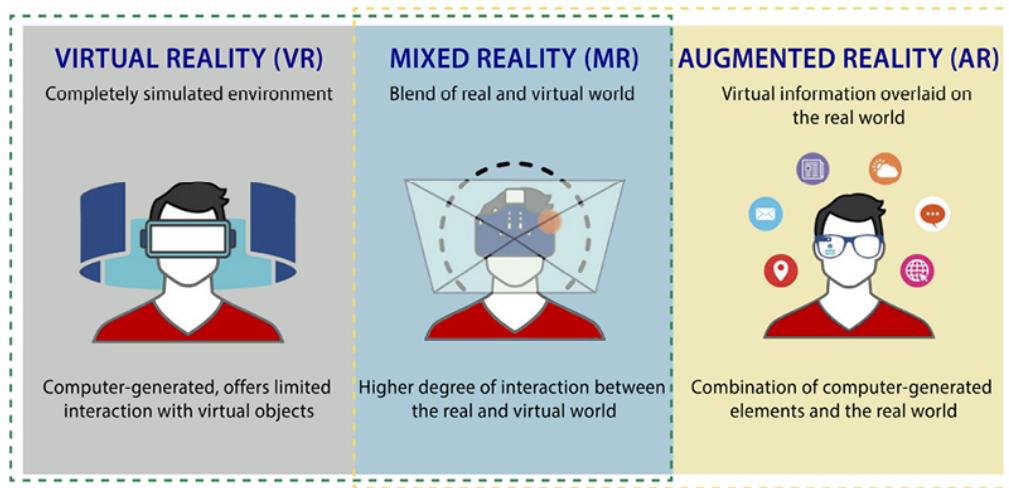
VR involves a complete immersion experience that shuts out the physical world. With VR, users experience a digital environment where they interact with 3D worlds.

AR enhances the real world by overlaying digital information onto it. AR users can see the world around them while interacting with virtual objects placed in it.

**Mixed reality (MR)** is an emergent trend that blends these experiences, enabling users to interact with both physical and virtual elements within the same session.

These advancements are empowered by the predictive and adaptive capabilities of AI and have opened new dimensions for marketing professionals to engage their audiences. With projections indicating a

significant increase in the adoption of these technologies, marketers should understand their potential to reshape customer interactions.



*Figure 12.7: Schematic illustration of VR, AR, and MR (source: <https://onlinelibrary.wiley.com/doi/10.1002/adfm.202007952>)*

## The role of ML in AR for marketing

AR is not just about overlaying digital content onto the real world, it's about doing so in a way that is personalized, relevant, and seamlessly integrated into the user's environment. The transformation of AR into a powerful marketing tool is significantly driven by the integration of AI and ML. Let's discuss some examples of how this can be achieved. You can already see examples of this in many apps that are available for your smartphone.

## Personalized AR experiences

AI algorithms are at the heart of personalized AR experiences. By analyzing data from user interactions, preferences, and past behavior, these algorithms

can tailor AR content to fit individual needs and interests.

For example, when a user points their smartphone camera at a shelf in a store, AI can display AR overlays showing products they are likely to be interested in, based on their shopping history. This not only enhances the user experience but also increases the likelihood of purchase.

The use of facial recognition technology can further personalize experiences, such as applying virtual makeup in real time, allowing customers to see how products would look on their faces before making a purchase. This level of customization makes the shopping experience more engaging and can lead to higher conversion rates.

## **Geolocation-based AR campaigns**

Geolocation technology combined with AI enables marketers to deliver highly targeted AR content that is not only based on a user's profile but also on their physical location. This is particularly effective in location-based marketing where promotions can be tailored to local events or the proximity of stores.

For instance, users near a concert venue might see AR promotions for related merchandise or nearby restaurant deals. AI enhances these experiences by considering not just the location but also the context of the user's situation, such as time of day or weather conditions, offering more relevant and timely content that improves engagement rates.

## **Seamless product integration**

ML is critical in achieving seamless integration of virtual products into real-world environments. Advanced image recognition and spatial awareness

capabilities allow AR systems to place virtual items realistically within the user's environment.

For instance, an AR furniture app powered by the NeRF model architecture described earlier in this chapter can analyze the space available in a room and suggest furniture items that would fit both physically and aesthetically.

## **The role of ML in VR for marketing**

VR provides immersive brand storytelling, transporting users into a fully branded, interactive universe that takes customer engagement to the next level by fully immersing users in a branded virtual environment. With the aid of AI and ML, these virtual worlds can be personalized, dynamic, and responsive, creating experiences that strengthen the relationship between brands and their customers. While these applications are currently less common than AR at present, the following sections outline some examples of these principles in action.

### **Immersive storytelling and brand experiences**

AI enables brands to deliver compelling narratives in VR by personalizing content based on the user's interests and behavior. Instead of generic experiences, users can explore dynamic storylines that adapt to their preferences.

For instance, in a VR showroom for an automobile brand, customers might explore different virtual environments tailored to their interests, such as adventure or family travel. AI analyzes their past choices and behavior to

dynamically suggest features or experiences that are relevant, enhancing engagement and making the content more memorable.

## **Behavioral data insights**

In a VR environment, every user action can be tracked and analyzed. AI and ML algorithms convert these interactions into valuable behavioral data insights.

Some examples could be how long users spend exploring certain products, which paths they follow, or where their gaze lingers, which can provide critical data points. These insights can then be used to refine future campaigns, personalized recommendations, and adjust content in real time. This behavioral data can also inform product development by helping brands understand what features and attributes resonate most with their target audiences.

## **Virtual storefronts and events**

AI enables the creation of highly personalized virtual storefronts and events that offer a unique shopping experience. These storefronts can be designed to replicate or even enhance a physical shopping experience. Users are guided through personalized recommendations based on their previous purchases, browsing history, and demographic data.

In virtual events, AI-driven chatbots and avatars can also provide immediate customer service and answer questions, making the interaction feel natural and informative. Brands can also host live events within these virtual spaces, allowing customers to engage directly with brand ambassadors and influencers.

# Summary

As we conclude our exploration of the future landscape of ML and GenAI in marketing, we have reflected on the advancements that have revolutionized marketing strategies. Throughout this book, we've covered how AI and data-driven insights can guide every aspect of marketing – from decoding KPIs and gathering deep customer insights to crafting compelling content and micro-targeting audiences. The landscape of AI in marketing is not only dynamic but rapidly evolving, setting the stage for further transformative changes to come.

In the next frontier of developments, we've highlighted emerging technologies such as multi-modal GenAI and advanced model architectures that synthesize diverse data for creating consumer-tailored content. We've also examined the integration of AI with novel platforms like AR and VR, which can be used to transform traditional marketing approaches into more immersive consumer experiences.

Considering the long-term implications of these technologies, a key development to watch out for in the future is multi-modal AI, which will enable hyper-personalized marketing. By integrating data from various sources (text, images, video, and audio), marketers can create highly customized content that resonates personally with each consumer. We're also moving toward semi-autonomous marketing systems capable of overseeing entire campaigns with minimal human intervention. While fully autonomous systems will be further away, these systems will be a useful guide for humans to more efficiently handle everything from strategy development and content creation to deployment and real-time optimization. Advanced AI will provide deeper and more accurate consumer insights by analyzing large amounts of data across multiple

channels, leading to better prediction models for consumer behavior. The integration of AI with AR and VR will create more immersive consumer experiences, blurring the lines between digital and physical interactions.

Looking forward, the final chapter will address the crucial themes of ethics and governance in AI-enabled marketing. This discussion will focus on navigating the complex landscape of ethical concerns and regulatory frameworks essential for deploying AI responsibly. By tackling issues such as data privacy, algorithmic bias, and the necessity for transparency, we will underscore the importance of ethical practices that safeguard consumer trust and brand integrity. Through this focus, you will gain the insights needed to understand the elements of robust governance structures and compliance strategies, ensuring your AI applications are both effective and ethically sound.

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)

# 13

## Ethics and Governance in AI-Enabled Marketing

The pervasive use of AI and ML in marketing raises various ethical concerns, including data privacy, algorithmic bias, and the need for transparency, all of which can directly impact consumer trust and brand integrity. In this final chapter, we address the crucial topic of the ethical considerations and governance challenges associated with the AI technologies introduced in previous chapters.

This chapter will also explore major regulatory frameworks such as the **General Data Protection Regulation (GDPR)** and **California Consumer Privacy Act (CCPA)**, which play a critical role in addressing some of these ethical concerns. These regulations help shape the deployment and data management policies around AI technologies in marketing. It's important to understand these ethical and regulatory aspects for leveraging AI responsibly and effectively in marketing strategies. This chapter covers how marketers can navigate these challenges through suggested ML best practices, governance structures, and compliance considerations.

By the end of this chapter, you will gain:

- An ethical understanding of responsible AI deployment in marketing

- Insight into regulatory frameworks governing AI and data use in marketing
- Strategies for model transparency, governance policies, and compliance to promote responsible AI use

## **Ethical considerations in AI/ML for marketing**

In the evolving landscape of ML for marketing, ethical considerations are crucial in maintaining consumer trust and adhering to responsible business practices. As AI becomes increasingly integral to customer engagement, personalization, and targeting, marketers must remain aware of the ethical implications of their data-driven strategies. Concerns such as transparency, bias, and fairness need careful consideration to ensure that AI/ML applications are both effective and aligned with ethical standards.

To address these concerns, this section will discuss strategies for making your ML predictions as explainable as possible, mitigating bias, and grounding your generative AI outputs in truth. The handling of sensitive consumer data is another topic that requires careful consideration. Failing to appropriately handle consumer data can have not only legal ramifications but also devastating public relations impacts on the reputation and trust that consumers hold for a brand.

## **Model transparency and explainability**

In the realm of marketing, **transparency** in AI deployment is paramount. Consumers are often skeptical of how their data is used and how AI systems make decisions that impact their interactions with brands. Transparency requires clear communication about how AI models are trained, how they function, and how data is processed, focusing on understanding the decision-making process of AI models.

One example highlighting the importance of transparency is the 2018 incident involving Amazon's AI recruitment tool. The tool, which was intended to help automate their recruitment process, was found to be biased against women. It was discovered to favor male candidates over female ones because it was trained on resumes submitted over a ten-year period, predominantly from men. Amazon had to scrap the tool after the bias was uncovered, leading to reputational damage.

In order to achieve transparency, marketers should prioritize **explainability** – which is the ability to articulate, in simple terms, how their AI models reach specific decisions. Transparency is facilitated by model explainability tools, as we will discuss in the next section. The following are some reasons why explainability is so important for AI models used for marketing:

- **Consumer trust:** Explaining AI decisions builds consumer trust, particularly when the model's recommendations impact sensitive matters like personalized advertising.
- **Regulatory compliance:** Some data privacy regulations, such as GDPR, require organizations to potentially explain automated decision-making.
- **Bias detection:** Interpretability aids in identifying potential biases within LLMs, ensuring fair treatment of different groups.



## The challenge of explainability in large language models (LLMs)

Explainability in LLMs and other large AI models can be challenging due to their complex architectures and large number of parameters. As these models become more accurate, their decisions can become more complex and harder to interpret. This underscores the importance of model explainability techniques.

Next, we will discuss some model explainability tools that are helpful in arriving at greater transparency in modern AI models.

# Model explainability tools

ML models such as logistic regression or decision trees present highly interpretable parameters that offer direct explainability to their predictions. For instance, with decision trees, as presented in *Chapter 3*, we can visualize the exact branches in logic that led to the model's decision being made. However, the pursuit of transparency in complex, modern AI models presents greater challenges, particularly in deep learning and LLMs. These models operate through intricate, multilayered networks that are not easily interpretable. However, several tools and techniques have emerged to tackle this complexity.

One such tool is the concept extraction framework introduced by OpenAI for GPT-4. This framework enhances transparency by extracting key concepts that the model uses to make predictions. It works by analyzing the internal representations and activations within the model to identify patterns and concepts that influence its decisions. By isolating these key concepts,

the tool can provide a clearer understanding of how the model interprets and processes data.

The Patchscopes framework, developed by Google Research, is another valuable tool that provides a comprehensive and unifying approach for inspecting hidden representations in LLMs. Patchscopes allows for a broader exploration of the internal dynamics of LLMs. It works by visualizing and analyzing the hidden states and attention patterns within the model. It allows users to track how information flows through the network, identifying which parts of the input text are being focused on at different layers and heads of the transformer architecture, providing a detailed understanding of how the model processes and generates language. For further discussion on hidden states and attention patterns, you can refer to *Chapter 9*.



## LLM explainability tools

**Patchscopes:** For more details on Patchscopes and its array of applications, visit the Google Research blog (<https://research.google/blog/patchscopes-a-unifying-framework-for-inspecting-hidden-representations-of-language-models/>) on Patchscopes. You can also explore the `pathscopes` Python package to explore its implementation.

**Concept extraction:** See OpenAI's post for more details (<https://openai.com/index/extracting-concepts-from-gpt-4/>), including links to the research paper, code, and example feature visualizations.

For instance, Patchscopes can be utilized to investigate how different linguistic patterns are represented within the model and how these patterns influence the model's predictions. This is particularly useful for understanding which parts of a marketing message are prioritized by the AI model, enabling marketers to generate more effective personalized recommendation messages that avoid potential bias.

The following figure shows Patchscopes decoding what is encoded in the representation of `It` in the source prompt (left), by using a predefined target prompt (right):

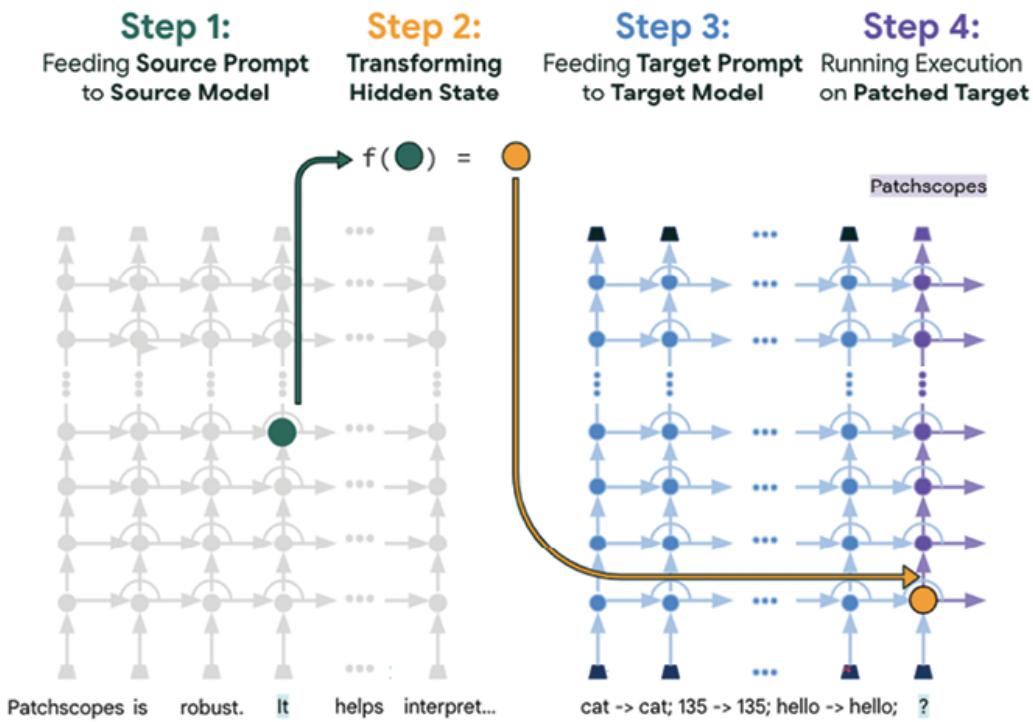


Figure 13.1: The Patchscopes workflow (source: <https://research.google/blog/patchscopes-a-unifying-framework-for-inspecting-hidden-representations-of-language-models/>)

As an example, let's see how we can visualize one fundamental component of an LLM's decision-making process: attention. To illustrate this, we will consider what we can learn about word importance from one of the earlier

prompts introduced in *Chapter 10* for our environmentally friendly kitchenware e-commerce brand. If you recall, the prompt we first introduced using zero-shot learning (ZSL) for our company's rebranded sustainability marketing campaign was:

```
Write a product description for an eco-friendly kitchenware product  
focusing on brand ethics.
```

We will use the BERT LLM for this example as opposed to GPT-4 to enable us to analyze the attention heads. The choice of BERT for this example is because the GPT-4 API currently does not expose their internal attention weights to users and this approach requires full access to these states for analysis and visualization. We can first extract the model attention values using the following code:

```
import numpy as np  
import tensorflow as tf  
from transformers import TFAutoModel, AutoTokenizer  
model_name = "bert-base-uncased"  
tokenizer = AutoTokenizer.from_pretrained(model_name)  
model = TFAutoModel.from_pretrained(model_name, output_atten  
text = "Write a product description for an eco-friendly kitchen  
inputs = tokenizer(text, return_tensors='tf')  
outputs = model(inputs)  
attention = outputs[-1][-1].numpy()
```

The array of `attention` weights in the last line of the code represents how the model distributes its focus across the different tokens in the input text. Each value indicates the importance given to a specific token by the model at various layers and heads of the transformer architecture. High values suggest that the model considers those words more significant in processing the text and generating the output.

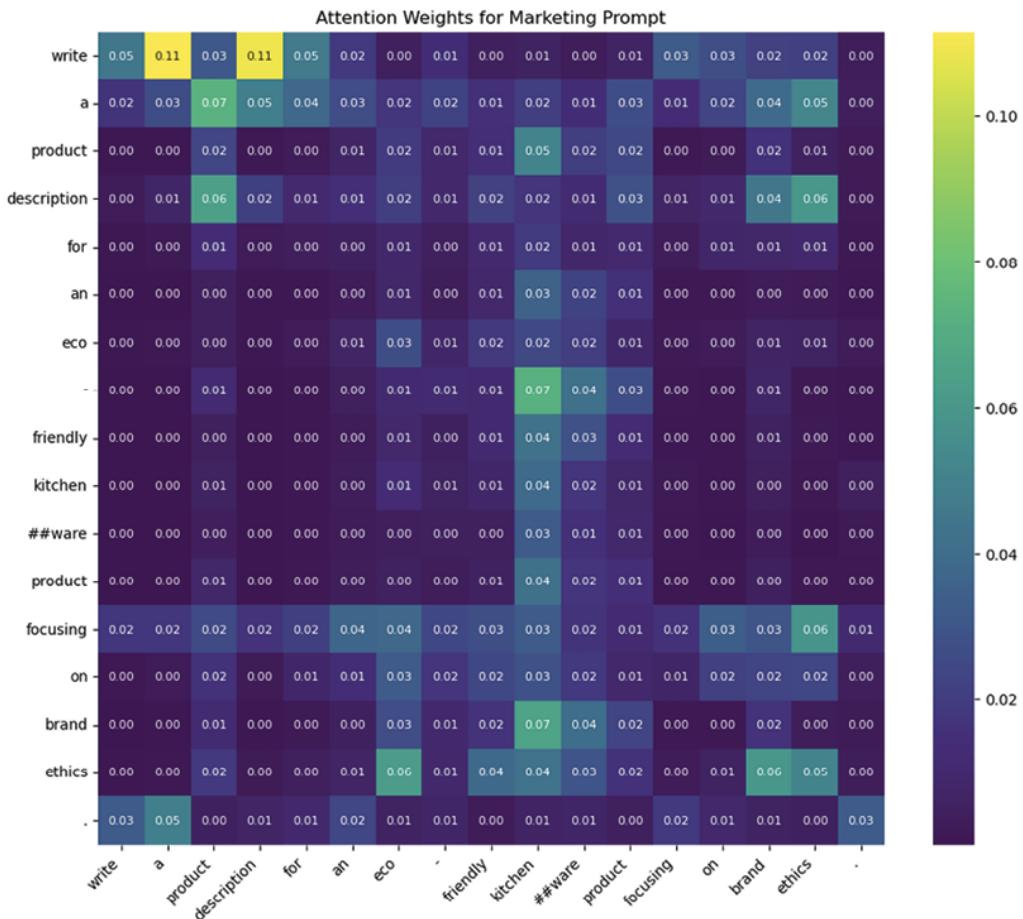
We can visualize these attention weights using a heatmap to gain a more intuitive understanding of how the model processes our text prompt. In simple terms, the code will perform the following three major steps:

1. **Process attention weights** to remove extra dimensions that aren't needed for the plot.
2. **Convert tokens** from their numerical IDs back to actual words, excluding special tokens that are used for the model's internal processing but do not represent actual words, such as separators to indicate sentence boundaries.
3. **Create a heatmap plot** to visualize the attention weight and see how much focus the model gives to each word in the text prompt.

Here is the complete code:

```
import matplotlib.pyplot as plt
import seaborn as sns
attention = attention.squeeze(axis=0)
tokens = tokenizer.convert_ids_to_tokens(inputs["input_ids"]る
tokens = tokens[1:-1]
attention = attention[:, 1:-1, 1:-1]
fig, ax = plt.subplots(figsize=(12, 10))
sns.heatmap(attention[0], annot=True, ax=ax, cmap="viridis", xt
ax.set_title('Attention Weights for Marketing Prompt')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```

This gives us the following plot:



*Figure 13.2: Attention weights for the marketing prompt “Write a product description for an eco-friendly kitchenware product focusing on brand ethics”*

From this visualization, we can see how the model allocates its attention across the various tokens. The following are some key points to consider in interpreting attention values in this visual:

- **Diagonal patterns:** Attention along the diagonal often indicates that the model is focusing on individual words and their immediate contexts. This can show how the model considers the word itself and its close neighbors in the sequence.
- **Cross-attention:** Off-diagonal patterns reveal how the model connects different words in the sentence. For instance, in the preceding heatmap, the connection between the word `ethics` (y axis) and terms

like `eco` (x axis) shows higher attention weights, indicating that the model understands the relevance of these terms in describing the product.

- **Individual weights:** Examining the individual weights, we see that words such as eco, kitchen, and ethics have higher attention values, suggesting that the model prioritizes these terms when generating a relevant product description.

The model then uses these attention weights to determine which words are most important in the context of the prompt and how they relate to each other, allowing it to determine the most meaningful text output. From a marketer's perspective, these weights can be used to ensure that the key values of your marketing message are being emphasized by the model – and if not, you can use this as a tool to iterate until you've found a prompt that does.

## Bias mitigation

Mitigating bias and ensuring fairness are central to ethical and accurate AI in marketing as they directly influence both the trustworthiness and effectiveness of marketing campaigns. Unchecked biases can lead to unfair treatment of certain demographic groups, eroding consumer trust and potentially resulting in legal and reputational repercussions. Bias can also lead to poor segmentation, as discussed in *Chapter 8*, which can be highly detrimental to the effectiveness of marketing campaigns. Unlike transparency and explainability, which focus on understanding and communicating how models make decisions, bias mitigation and fairness are about ensuring that these decisions do not disproportionately harm or favor specific groups.

In the following sections, we will discuss strategies that address various aspects of bias in AI, along with marketing examples illustrating their importance.

### Strategies for mitigating bias in AI marketing models

- **Training data bias:** Audit and diversify training data to ensure proper demographic representation.
- **Algorithmic fairness during training:** Use methods like adversarial debiasing to minimize predictions based on protected data.
- **Diverse evaluation data:** Test models across varied demographic groups to identify and address disparities.
- **Ground LLMs:** Use external sources to verify and refine generative AI outputs for factual accuracy.
- **Chain-of-thought reasoning:** Break down generative AI decisions into logical steps to promote fairness.

## Addressing training data bias

Training data can reflect societal biases, which can become embedded in AI model predictions. This is particularly problematic in marketing, where biased recommendations or advertising can alienate segments of the consumer base. To address training data bias, it is valuable to conduct thorough audits of the data for appropriate demographic representation. Techniques discussed in past chapters, such as oversampling underrepresented groups or permuting existing training samples, can also help reduce such biases by making the training data more balanced.

As an example, consider a marketing model trained predominantly on data from urban, affluent consumers. Properly segmenting your customer base is key, and this model might underperform when targeting rural or lower-income segments due to their lack of representation in the training data. By incorporating data from a variety of socioeconomic backgrounds, the model can better understand and predict the behaviors of a more diverse audience based on their demographic.

## **Algorithmic fairness techniques**

Ensuring algorithmic fairness in AI models can be achieved using more specialized techniques that take place during model training or fine-tuning. Some of these techniques include equalized odds, which ensures that different groups have similar error rates, and disparate impact remover, which preprocesses data to reduce bias and create more equitable outcomes. Additionally, fairness constraints in optimization integrate fairness metrics directly into the training process, allowing the model's optimization to account for fairness criteria.

As an example, adversarial debiasing introduces fairness constraints by training a model to minimize prediction error while reducing another classifier's ability to predict protected attributes. The implementation of this involves creating an adversarial model that attempts to predict the protected attribute, such as race, from the main model's outputs or internal representations. During training, the main model is penalized not only for prediction errors but also for the adversarial model's success in predicting the protected attribute. This way, the main model learns to make predictions that are less correlated with protected attributes, promoting fairness.

In a marketing application, a model predicting a consumer's interest in financial products should not overly favor one racial group unless it is specifically designed to address existing financial services disparities as an explicit goal. Failing to account for this can lead to the model making predictions based on race, whereas it would be more ethical to use factors such as income, credit score, or geographic location that are directly relevant to financial product interest.

## Diversity in model evaluation data

Evaluating models using data containing diverse demographic groups ensures that predictions do not unfairly favor specific segments. This involves incorporating demographic attributes during model validation to identify and address any disparities. Tools and frameworks like Fairness Indicators by TensorFlow and AI Fairness 360 by IBM can help in evaluating model fairness by providing metrics and visualizations that highlight biases in model predictions.



### Tools and frameworks for model evaluation

**Fairness Indicators by TensorFlow:** Enables computation of fairness metrics, allowing comparison across subgroups and highlighting disparities. Learn more at [https://www.tensorflow.org/tfx/guide/fairness\\_indicators](https://www.tensorflow.org/tfx/guide/fairness_indicators).

**AI Fairness 360 by IBM:** Offers an open source toolkit of algorithms and metrics to detect and mitigate bias. Learn more at <https://aif360.res.ibm.com/>.

Take, for example, a model used to recommend job advertisements. Such a model should be evaluated to ensure it performs well across different demographic groups, such as gender and ethnicity. If the model shows a bias toward recommending high-paying jobs predominantly to white male users, adjustments can be made to balance recommendations, ensuring that qualified female users and those of different ethnicities receive equally relevant and high-quality job suggestions.

## Grounding for LLMs

LLMs can generate biased or factually incorrect information due to biases in their training data. Grounding involves using external databases or knowledge sources to verify and refine model outputs, ensuring more factual accuracy. By grounding LLMs with authoritative sources, a brand can ensure that the content generated for marketing campaigns is accurate and less biased. Additionally, grounding helps mitigate hallucination – or cases where the model generates information that is not based on reality.

You may remember that in *Chapter 11*, we discussed micro-targeting with **retrieval-augmented generation (RAG)**. Grounding is also useful for effective micro-targeting, as it allows LLMs to pull in reliable information from previous customer interactions to generate accurate and personalized marketing messages. For example, directing the model to access a verified product database ensures that descriptions and specifications are more accurate, preventing the model from fabricating features or benefits during the content creation process. While reducing the LLM temperature parameter (as explored in *Chapter 9*) of your model can reduce hallucinations, it does not guarantee that hallucinations will be eliminated.

## Chain-of-thought reasoning

Lastly, we have chain-of-thought reasoning, introduced in the context of *ReAct* in *Chapter 12*. Chain-of-thought reasoning breaks down complex tasks into simpler, sequential steps, allowing the AI to follow a logical process to arrive at a solution. Chain-of-thought reasoning can be used in marketing to promote more accurate and fair decision-making by explicitly accounting for the considerations around fairness at each decision step.

Consider a marketing model for personalized product recommendations, a topic that was explored in *Chapter 7*. Without chain-of-thought reasoning, the model might disproportionately suggest a financial product such as subprime credit cards to minority applicants based on biased data patterns related to race or neighborhood, leading to discriminatory practices. By implementing chain-of-thought reasoning, we can prompt the model to systematically evaluate relevant factors like individual credit history, transaction patterns, and specific financial needs in order to generate its decisions. With this logical breakdown stored in the model's memory, it can now follow a fairer process to arrive at a recommendation based on individuals' financial behaviors.

## Balancing privacy with personalization

The increasing reliance on consumer data to power ML marketing models can pose significant privacy concerns. While consumers desire personalized experiences, they also value their privacy, making it crucial for marketers to find a delicate balance between these two priorities. **Ethical AI** deployment requires safeguarding sensitive personal data while delivering

individualized recommendations and targeted marketing campaigns. Achieving this balance is essential not only for compliance with legal standards but also for maintaining consumer trust and brand reputation. The following are strategies to consider to help your marketing campaign have the precision and personalization you need without compromising on the need for customer privacy.

## Federated learning

**Federated learning** is a technique that can significantly enhance privacy by decentralizing data processing. Instead of sending raw customer data to a central server for analysis, federated learning processes the data locally on consumer devices and only transmits encrypted model updates to a central system. The central server then aggregates these updates to improve the shared global model without directly accessing the participant's sensitive data. This minimizes data leakage risks and ensures that data remains private – allowing organizations to provide personalized services based on sensitive financial or healthcare data while keeping personal information secure.

In practice, federated learning works as follows:

1. A baseline model is stored on a central server and then copies of this model are shared with client devices.
2. As users interact with the model, local data generated on the individual devices is used to train and improve the model locally.
3. Periodically, the locally trained model parameters are sent back to the central server, where they are aggregated to enhance the overall model.
4. The updated central model is then redistributed to user devices, which continue to refine it based on new local data.

This iterative process ensures continuous improvement while maintaining data privacy. The workflow is illustrated in the following diagram:

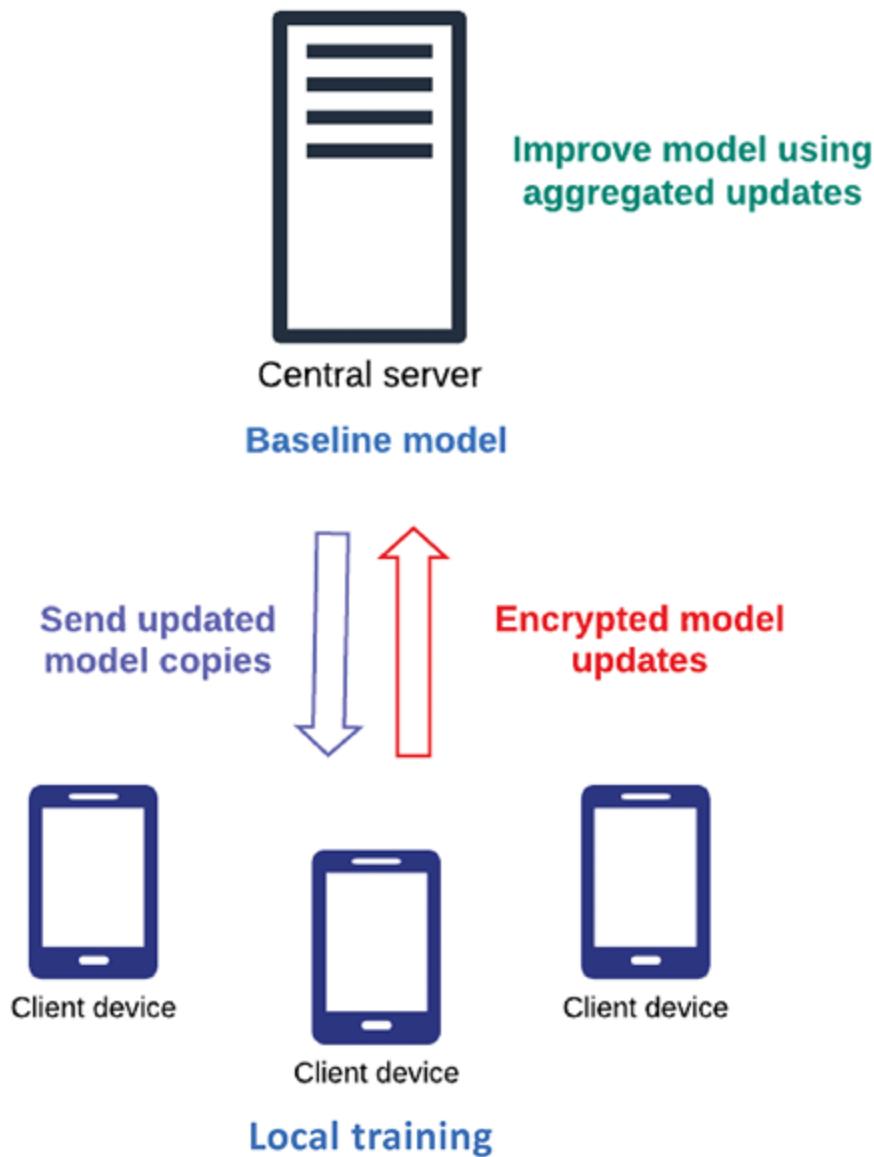


Figure 13.3: Federated learning workflow

One of the most ubiquitous applications of federated learning can be found in smartphones, enhancing functionalities such as facial recognition, word prediction, and voice recognition. In a marketing context, federated learning can be utilized for personalized medical records management while

complying with HIPAA regulations. For example, a healthcare provider could use federated learning to analyze patient data locally on devices, improving diagnostic models without ever transferring sensitive health information to a central server

## **Differential privacy and anonymization**

**Differential privacy** involves adding noise to data to obscure the identifying characteristics of the original data without losing meaningful trends or signals that can be learned from the data characteristics. In practice, differential privacy is applied to conserve the privacy of collected data before analysis. **Anonymization** is a complementary concept that involves transforming **personally identifiable information (PII)** into a format that cannot be traced back to an individual.

For example, if the company wants to understand the purchase behavior of customers in a particular region, the following steps can be taken:

1. **Data collection:** Collect purchase data from customers in the target region.
2. **Data anonymization:** Apply anonymization techniques, such as pseudonymization, to replace identifiers with unique but non-identifiable keys.
3. **Noise addition:** Apply differential privacy techniques to add noise to the data, ensuring that individual purchase behaviors or demographics are obscured.
4. **Aggregate analysis:** Perform analysis on the anonymized data to identify trends and patterns that can inform ad targeting strategies.

5. **Ad targeting:** Use the insights gained from the analysis to optimize ad targeting, ensuring that the ads are relevant to the audience without exposing individual data.

We will next discuss how you can implement *Step 2* (data anonymization) and *Step 3* (noise addition) of this process. For the remaining steps, you can refer to the content in previous chapters where the fundamentals of these steps are covered:

- **Step 1:** *Data collection* in *Chapter 5*
- **Step 4:** *Aggregate analysis via segmentation* in *Chapter 8*
- **Step 5:** *Ad targeting via personalized recommendations* in *Chapters 7, 10, and 11*

## Data anonymization

Data anonymization involves transforming PII into a format that cannot be traced back to an individual. One effective technique is pseudonymization, where identifiers are replaced with unique but non-identifiable keys. An example of how this can be implemented on customer purchase and demographic records to remove their first and last names, replacing these fields with a unique but non-identifiable `customer_id`, is as follows:

```
import pandas as pd
import hashlib
customer_data = pd.DataFrame({
    'first_name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank'],
    'last_name': ['Smith', 'Jones', 'Brown', 'Johnson', 'Davis', 'Wilson'],
    'age': [25, 30, 22, 40, 35, 28, 26, 33, 29, 37],
    'income': [20000, 35000, 27000, 50000, 45000, 30000, 32000],
    'purchase_amount': [100, 150, 200, 250, 220, 140, 180, 160]
})
def pseudonymize_id(first_name, last_name):
    return hashlib.sha256((first_name + last_name).encode()).hexdigest()
```

```
customer_data['customer_id'] = customer_data.apply(lambda row:  
anonymized_data = customer_data.drop(columns=['first_name', 'la  
display(anonymized_data)
```

This produces the following output:

	age	income	purchase_amount	customer_id
0	25	20000	100	f44922c82b2b398132e11e37f886b3feb2013e3dd15803...
1	30	35000	150	d7609f6bce0523c1d94bb3acefccd104ad5369a4fc827c...
2	22	27000	200	a8a680e8d5b91c178b95d4c8895cf791bf558eee10ce84...
3	40	50000	250	7cb5eb19bd248ccbec0a3c38e18059fb42a88d3616307f...
4	35	45000	220	317b261ea8234ff1e8e38625b1612ca2b8459a0ac36e81...
5	28	30000	140	f5a380b98877052126820d124909a00b6f19e86a9d5228...
6	26	32000	180	c3b77a1f2da3bdfa0cd07d1cd3f74187ece2eb55e0da16...
7	33	38000	160	d2aeada25e977a2886a48e32700644eef6e58b1f6b4d34...
8	29	31000	190	1a1037aeae35e7eab6408438d1d279fd9733d8519a043c...
9	37	47000	230	e9fbef9a35e46192a05b3fc06652a98ef0f92500f1ff...

Figure 13.4: Customers' purchase data after anonymization

In this code, we use the `hashlib.sha256` function to hash the combination of `first_name` and `last_name`. **SHA-256** is a cryptographic hash function that produces a unique, fixed-size hash value. After generating the `customer_id`, we drop the original PII columns of `first_name` and `last_name` to prevent re-identification.

## Noise addition

After anonymizing the data, we apply differential privacy techniques to add noise and obscure individual purchase behaviors to further protect consumer privacy. We will perform this via the following steps:

1. Scale the numeric features (age, income, and purchase amount) using `MinMaxScaler` to ensure the noise addition is proportionate across

different data scales.

2. Set  $\epsilon$ , a parameter that controls the trade-off between privacy and data utility, to a value such as `5.0`. A smaller  $\epsilon$  value such as `1.0` offers higher privacy but introduces more noise and reduces data utility.
3. Use the Laplace mechanism (`np.random.laplace`) to add noise drawn from a Laplace distribution, with the amount of noise determined by the sensitivity (range of the data) divided by epsilon.
4. Finally, inverse transform the data to bring it back to its original scale, which is important for interpreting the results in their original context.

### Adding noise with the Laplace mechanism

To protect privacy, we use the Laplace mechanism to add noise drawn from a Laplace distribution. The Laplace distribution is a continuous probability distribution and is widely used in differential privacy to add noise to data to ensure that individual data points cannot be easily identified.



Learn more at

<https://numpy.org/doc/stable/reference/random/generated/numpy.random.laplace.html>.

The preceding steps can be performed and the result visualized using the following code:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
anonymized_data[['age', 'income', 'purchase_amount']] = scaler.
```

```

epsilon = 5.0
def add_noise(data, epsilon):
    sensitivity = np.max(data) - np.min(data)
    noise = np.random.laplace(0, sensitivity / epsilon, data.shape[0])
    return data + noise
noisy_data = anonymized_data.copy()
noisy_data[['age', 'income', 'purchase_amount']] = add_noise(anonymized_data[['age', 'income', 'purchase_amount']])
noisy_data[['age', 'income', 'purchase_amount']] = scaler.inverse_transform(noisy_data)
print("Noisy Data: \n", noisy_data)
average_purchase = noisy_data['purchase_amount'].mean()

```

We can see the result of these data transformations using the following plotting function, comparing the original, anonymized, and noisy data to highlight how differential privacy can obscure individual data points while still maintaining the general data trends:

```

def plot_data_with_trend_lines(original_data, anonymized_data, noisy_data):
    fig, ax = plt.subplots(1, 3, figsize=(18, 6))
    ax[0].scatter(original_data['age'], original_data['purchase_amount'])
    z = np.polyfit(original_data['age'], original_data['purchase_amount'], 1)
    p = np.poly1d(z)
    ax[0].plot(original_data['age'], p(original_data['age']), "r--")
    ax[0].set_title('Original Data')
    ax[0].set_xlabel('Age')
    ax[0].set_ylabel('Purchase Amount')
    ax[1].scatter(anonymized_data['age'], anonymized_data['purchase_amount'])
    z = np.polyfit(anonymized_data['age'], anonymized_data['purchase_amount'], 1)
    p = np.poly1d(z)
    ax[1].plot(anonymized_data['age'], p(anonymized_data['age']), "r--")
    ax[1].set_title('Anonymized Data')
    ax[1].set_xlabel('Age')
    ax[1].set_ylabel('Purchase Amount')
    ax[2].scatter(noisy_data['age'], noisy_data['purchase_amount'])
    z = np.polyfit(noisy_data['age'], noisy_data['purchase_amount'], 1)
    p = np.poly1d(z)
    ax[2].plot(noisy_data['age'], p(noisy_data['age']), "r--")
    ax[2].set_title('Noisy Data with Differential Privacy')
    ax[2].set_xlabel('Age')

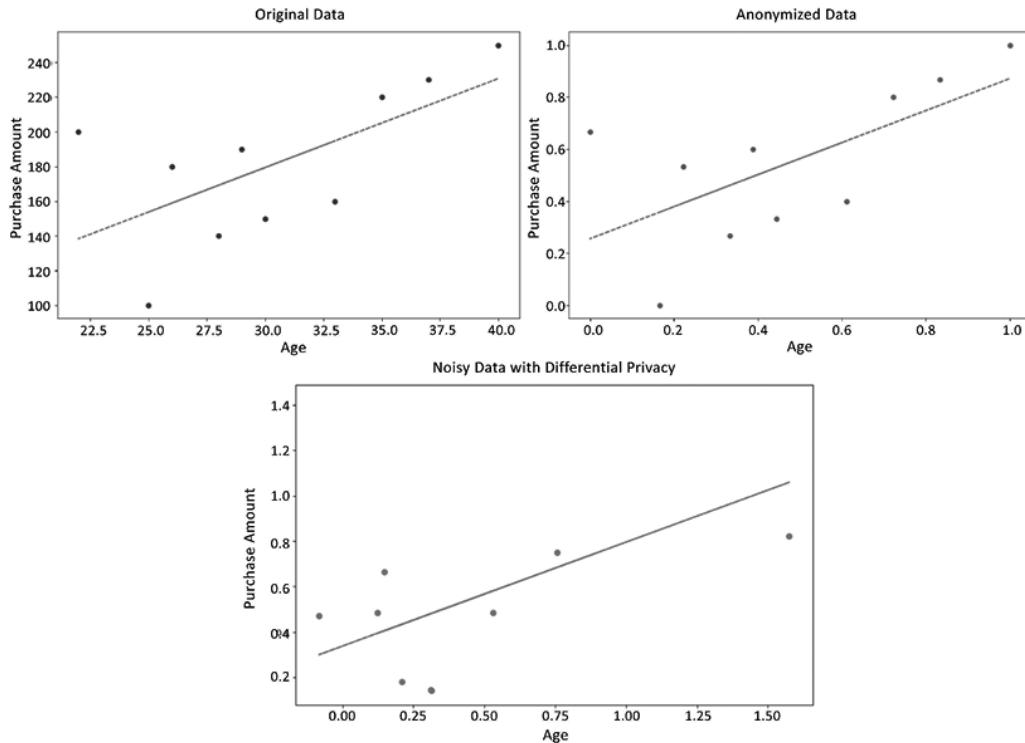
```

```

    ax[2].set_ylabel('Purchase Amount')
    plt.tight_layout()
    plt.show()
plot_data_with_trend_lines(customer_data, anonymized_data, noise)

```

We then get the following graphs:



*Figure 13.5: Illustration of the effects of differential privacy on customer purchase data*

As shown in the third plot with differential privacy applied, there is a substantial distortion in the data points but, despite the noise, the overall trend is still discernible.

It's important to note the trade-off between privacy and utility. In general, as privacy increases, data utility decreases. However, this differential privacy technique effectively masks individual data points and enhances privacy while still retaining the overall trend. This balance is crucial for

marketing applications where protecting consumer personal data while maintaining meaningful insights is essential.

## **Governance and regulatory compliance**

Adhering to appropriate governance practices while maintaining regulatory compliance is paramount. In this section, we will explore some of the many aspects related to these topics, including intellectual property protection and key components of building an ethical governance framework.

We will then look at some key regulatory compliance considerations around data use and processing, highlighting key frameworks like GDPR and CCPA, as well as touch upon other global regulations and industry-specific guidelines.

## **Intellectual property protection**

AI models rely on vast amounts of data, which often include proprietary or even copyrighted content. Ensuring the protection of intellectual property and copyright rights, both of your organization and others, is essential to avoid legal issues in the development and deployment of AI models. In the following subsections, we discuss key aspects of data and model licensing and attribution, internal data ownership and security, and data management and collection practices. These topics collectively address the need to ethically source, manage, and protect the data used in AI applications to maintain compliance with intellectual property laws.

# **Data and model licensing and attribution**

When sourcing data from third parties for AI and ML applications, it is crucial to establish clear licensing agreements that outline the permitted usage. This is especially important for training datasets that may contain copyrighted materials such as images, videos, and text. Not following these guidelines can lead to lawsuits, such as when stock photo provider Getty Images sued Stability AI in 2023 for using over 12 million of its copyrighted images to train its image-generation systems. According to Getty, not only were some images highly similar but some outputs even included modified versions of their Getty Images watermark. Ethical sourcing of openly available data can also play a significant role in addressing this. However, while open datasets are cost-effective and accessible, they may still contain copyrighted or sensitive material. For instance, a dataset sourced from an open platform like GitHub or a public database may include copyrighted code snippets or sensitive personal information. Due diligence is required to verify that open datasets comply with IP laws and the terms of their open licenses, such as use that is limited only to research applications.

# **Internal data ownership and security**

Organizations often have access to proprietary consumer data, such as purchase history or behavioral insights, which are important for developing effective marketing strategies. However, recent data breaches highlight the importance of protecting this information. To ensure this data is protected from misuse or unauthorized access, it is essential to clearly define ownership and access rights within the organization. For instance, a recent

example highlighting the importance of robust data security is the 2023 T-Mobile data breach.

This incident involved two separate security lapses: the exposure of employee data, including email addresses and partial social security numbers, and a system error that exposed customer payment data. Such breaches highlight the need for stringent permissions and advanced security measures like encryption and multi-factor authentication to protect sensitive information. For example, a marketing team might use encryption to secure customer data both in transit and at rest, ensuring that even if the data is intercepted or accessed without authorization, it remains unreadable.

## **Data management and collection practices**

As a general rule, marketers should adopt data minimization practices to collect only the data required for a specific marketing goal. This often involves leveraging sampling techniques to collect a representative subset of the entire dataset. By analyzing this smaller, yet statistically significant, sample, marketers can gain accurate insights without the need to collect data from the entire population. This minimizes the risks of widespread data breaches and promotes compliance with privacy regulations such as GDPR and CCPA, which will be discussed later in this chapter.

By limiting data collection to a clearly defined purpose, organizations can also reduce their exposure to potential data misuse. Any collected data should be used solely for the stated purpose, and if additional uses are identified later, explicit consent should be obtained from the consumer. For example, if a marketing team initially collects data for a targeted email campaign, they should seek further consent if they later wish to use this

data for a different, unrelated marketing initiative. Consumers should have a clear understanding of what data is collected, how it will be used, and their rights regarding data control. One way of implementing this is through privacy policies, which clearly communicate what data is collected, how it is used, and the benefits of sharing information for personalized marketing. Privacy policies should be user-friendly and comprehensible, offering clear opt-in and opt-out mechanisms. For instance, a company's privacy policy might explain that data collected from website interactions will be used to personalize product recommendations, but users can opt out if they prefer not to have their data used in this way.

## **Ethical governance frameworks**

Establishing an ethical governance framework can be valuable for organizations utilizing AI in marketing. This framework involves creating internal structures and policies that guide the responsible use of AI technologies. To be effective, this framework must be communicated clearly to all employees working with AI technology and embraced at every level of the organization, including the company's senior leadership team.

By forming internal AI ethics committees, developing comprehensive AI policies and guidelines, and fostering continuous training and awareness among staff, organizations can proactively address ethical dilemmas and promote transparency and fairness in their AI applications before it becomes an issue.

## **Internal AI ethics committees**

Forming internal AI ethics committees with diverse membership allows organizations to evaluate AI marketing strategies from multiple

perspectives. These committees may include members such as ethicists, lawyers, engineers, and business strategists. For example, ethicists can provide insights into complex moral issues while engineers can clarify technical nuances. Business strategists help assess operational risks.

However, the effectiveness of these committees in guaranteeing transparency and fairness is not guaranteed, and it's crucial for these committees to have clear mandates and the authority to influence decisions to ensure they are truly able to impact ethical risks. While smaller companies may lack the resources to establish formal committees, it remains crucial for them to prioritize ethical considerations in their AI practices, possibly by designating a responsible individual or small team to oversee these issues.

## **AI policy development and reporting**

Developing clear AI policies and guidelines provides a foundational framework for responsible company-wide marketing practices. These guidelines should encompass data usage, algorithmic fairness, model auditing, and the ethical implications of targeting specific consumer segments.

For instance, a marketing team using an AI model to personalize advertisements should document the datasets used for training, any preprocessing techniques applied to the data, and the algorithms selected for building the model. This documentation should also outline known limitations, biases, and potential risks, such as the model's tendency to favor certain demographics.

From a marketer's perspective, once these limitations and biases are identified, they can be addressed by tweaking the marketing strategies. This

might involve reassessing target audiences to promote more diverse representation, creating more inclusive messaging that resonates with broader demographics, and continuously monitoring campaign performance to identify and correct any emerging biases. Regular compliance audits and transparent reporting are crucial for tracking adherence to regulations and identifying areas for improvement. For example, an audit might review how consumer data is stored and accessed, ensuring that encryption and access control measures are in place.

## **Continuous training and awareness**

Educating marketing teams about AI ethics, data governance, and emerging regulations ensures that all stakeholders understand their responsibilities. Continuous training programs empower teams to recognize ethical dilemmas and make informed decisions, promoting a culture of compliance and ethical behavior. Regularly updated training sessions can also keep employees informed about the latest regulatory changes and best practices, ensuring that ethical considerations are integrated into everyday marketing activities.

## **Regulatory compliance**

Compliance with international, national, and industry-specific regulations is crucial for ensuring data privacy and ethical standards are maintained, fostering consumer trust, and avoiding legal repercussions. In this section, we will explore key regulatory frameworks, such as GDPR and CCPA, as well as select global regulations and industry-specific guidelines, to provide a better understanding of the regulatory landscape impacting AI-enabled marketing.

# **General Data Protection Regulation (GDPR)**

GDPR is a comprehensive data privacy law that primarily affects EU citizens, enforcing strict rules on how personal data is collected, stored, and processed. Here are some of its key features:

- Marketers must ensure transparent data collection practices, providing clear and concise information about what data is being collected and for what purposes.
- Consumers must be given explicit control rights over their data, including the ability to access, correct, and delete their information.
- GDPR mandates secure data handling practices to protect personal data from breaches and unauthorized access. This includes implementing appropriate technical and organizational measures, such as encryption and regular security audits.
- GDPR also regulates automated decision-making processes. If an AI system makes decisions that significantly affect individuals, such as through profiling or personalized marketing, provisions must be in place for human intervention and for providing meaningful explanations about how these decisions are made.



**Becoming familiar with GDPR for marketing**

For more detailed information on GDPR guidelines and its implications for marketing practices, visit the official GDPR website (<https://gdpr-info.eu/>).

Of particular interest to marketing professionals is Chapter 3, Articles 12–23, of GDPR describing the rights of the data subject. The following screenshot from the GDPR website highlights the key topics covered within this chapter:

## Chapter 3

# Rights of the data subject

<b>Section 1</b>	– Transparency and modalities
Article 12	– Transparent information, communication and modalities for the exercise of the rights of the data subject
<b>Section 2</b>	– Information and access to personal data
Article 13	– Information to be provided where personal data are collected from the data subject
Article 14	– Information to be provided where personal data have not been obtained from the data subject
Article 15	– Right of access by the data subject
<b>Section 3</b>	– Rectification and erasure
Article 16	– Right to rectification
Article 17	– Right to erasure ('right to be forgotten')
Article 18	– Right to restriction of processing
Article 19	– Notification obligation regarding rectification or erasure of personal data or restriction of processing
Article 20	– Right to data portability
<b>Section 4</b>	– Right to object and automated individual decision-making
Article 21	– Right to object
Article 22	– Automated individual decision-making, including profiling
<b>Section 5</b>	– Restrictions
Article 23	– Restrictions

GDPR  
Table of contents

Figure 13.6: Chapter 3 of GDPR (<https://gdpr-info.eu/chapter-3/>) covering the rights of the data subject

If governed by GDPR, it is essential that your company is aware of the legal implications of these regulations and is performing activities in a way that allows full compliance.

## **California Consumer Privacy Act (CCPA)**

Following the discussion on GDPR, it is essential to understand CCPA, which provides a robust framework for data privacy in California. While CCPA is specific to California, given the lack of a comprehensive federal mandate in the U.S., California often acts as a trendsetter. California was the first state to enact comprehensive data privacy legislation and over a dozen other states have enacted comprehensive data privacy laws since then. The following are some of the key CCPA mandates:

- CCPA grants California residents specific rights regarding their personal data, including the right to know what personal data is being collected about them, the purposes for which it is used, and to whom it is disclosed. It also imposes obligations on businesses that collect and process such data, and they must provide clear and accessible privacy notices detailing these aspects.
- Consumers have the right to access their personal data, request its deletion, and opt out of the sale of their personal information. For example, a marketing firm must include an easily accessible “Do not sell my personal information” link on its website, allowing consumers to opt out of data sales.
- In addition to these rights, CCPA requires businesses to implement reasonable security measures to protect consumer data from breaches and unauthorized access. Companies must ensure that their data-

handling practices comply with these requirements to avoid substantial fines and legal actions.

- CCPA also mandates that businesses respond to verified consumer requests within specific timeframes, ensuring that individuals can exercise their rights effectively.



### Complying with CCPA

For detailed information on CCPA compliance and how it affects marketing practices, visit the official CCPA website (<https://oag.ca.gov/privacy/ccpa>).

The following screenshot provides a detailed overview of the rights granted to California consumers under CCPA:

## 1. What rights do I have under the CCPA?

If you are a California resident, you may ask businesses to disclose what personal information they have about you and what they do with that information, to delete your personal information, to direct businesses not to sell or share your personal information, to correct inaccurate information that they have about you, and to limit businesses' use and disclosure of your sensitive personal information:

- **Right to know:** You can request that a business disclose to you: (1) the categories and/or specific pieces of personal information they have collected about you, (2) the categories of sources for that personal information, (3) the purposes for which the business uses that information, (4) the categories of third parties with whom the business discloses the information, and (5) the categories of information that the business sells or discloses to third parties. You can make a request to know up to twice a year, free of charge.
- **Right to delete:** You can request that businesses delete personal information they collected from you and tell their service providers to do the same, subject to certain exceptions (such as if the business is legally required to keep the information).
- **Right to opt-out of sale or sharing:** You may request that businesses stop selling or sharing your personal information (“opt-out”), including via a user-enabled global privacy control. Businesses cannot sell or share your personal information after they receive your opt-out request unless you later authorize them to do so again.
- **Right to correct:** You may ask businesses to correct inaccurate information that they have about you.
- **Right to limit use and disclosure of sensitive personal information:** You can direct businesses to only use your sensitive personal information (for example, your social security number, financial account information, your precise geolocation data, or your genetic data) for limited purposes, such as providing you with the services you requested.

You also have the right to be notified, before or at the point businesses collect your personal information, of the types of personal information they are collecting and what they may do with that information. Generally, businesses cannot discriminate against you for exercising your rights under the CCPA. Businesses cannot make you waive these rights, and any contract provision that says you waive these rights is unenforceable.

*Figure 13.7: Overview of privacy rights for California consumers under CCPA  
(<https://oag.ca.gov/privacy/ccpa#sectiona>)*

# Global regulations and standards

As data privacy concerns grow worldwide, various countries have developed their own regulations to protect personal data. Marketers operating internationally must be aware of these regional differences and implement global data strategies that comply with diverse regulations. The following are a few major countries and their respective data regulations:

- **Canada:** The Personal Information Protection and Electronic Documents Act governs how private sector organizations collect, use, and disclose personal information in the course of commercial business. For more details, visit the Office of the Privacy Commissioner of Canada website (<https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipEDA/>).
- **Brazil:** The Lei Geral de Proteção de Dados Pessoais establishes comprehensive data protection rules similar to GDPR, focusing on transparency, consumer rights, and security measures. More information can be found on the National Data Protection Authority website (<https://www.gov.br/anpd/en>).
- **Japan:** The Act on the Protection of Personal Information regulates the handling of personal data in Japan, emphasizing the protection of individual rights and the responsibilities of data controllers. For further information, see the **Personal Information Protection Commission (PPC)** website (<https://www.ppc.go.jp/en/>).
- **South Africa:** The Protection of Personal Information Act sets conditions for the lawful processing of personal information to protect individuals from harm and to ensure their privacy. More details can be

found on the Information Regulator website (<https://www.justice.gov.za/inforeg>).

## Industry-specific guidelines

In addition to government regulations, various industries have developed specific ethical standards and guidelines for data use to address their unique challenges and responsibilities. Depending on their industry, marketers should stay updated on these requirements to ensure compliance with industry norms:

- **Healthcare:** The **Health Insurance Portability and Accountability Act (HIPAA)** in the United States sets strict standards for protecting patient health information. Marketers in the healthcare industry must ensure that any use of patient data complies with HIPAA's privacy and security rules. More information can be found on the U.S. Department of Health and Human Services website (<https://www.hhs.gov/hipaa/index.html>).
- **Finance:** The **Financial Industry Regulatory Authority (FINRA)** provides guidelines for the use of customer data in the financial sector, focusing on privacy and the protection of sensitive financial information. Compliance with these guidelines helps prevent data breaches and maintains consumer trust. For more details, visit the FINRA website (<https://www.finra.org/rules-guidance/key-topics/customer-information-protection>).
- **Education:** The **Family Educational Rights and Privacy Act (FERPA)** protects the privacy of student education records. Educational institutions and marketers working with these institutions

must ensure that student data is used in compliance with FERPA regulations. More information can be found on the U.S. Department of Education website (<https://www2.ed.gov/policy/gen/guid/fpcos/ferpa/index.html>).

- **Retail:** The Payment Card Industry Data Security Standard sets guidelines for handling and securing credit card information. Retailers and marketers should adhere to these standards to prevent fraud and data breaches. For more details, visit the PCI Security Standards Council website (<https://www.pcisecuritystandards.org/standards/pci-dss/>).

## Summary

In this final chapter on ethics and governance in AI-enabled marketing, we have discussed the critical considerations and challenges associated with the use of AI technologies. We explored the ethical implications of data privacy, algorithmic bias, and the necessity for model transparency, emphasizing how these factors directly impact consumer trust and brand integrity. We also examined key regulatory frameworks, such as GDPR and CCPA, and discussed the importance of compliance in mitigating legal risks and fostering responsible AI practices.

We addressed practical strategies for ensuring ethical AI deployment in marketing, covering model explainability, bias mitigation, and balancing privacy with personalization. By establishing robust governance frameworks, including the formation of internal AI ethics committees, developing clear AI policies, and fostering continuous training and

awareness, organizations can navigate the complex ethical landscape effectively. Understanding and adhering to industry-specific guidelines and global data regulations are essential for maintaining compliance and protecting consumer rights across diverse regions.

As we conclude our exploration of AI and ML in marketing, it is clear that the future holds immense potential for innovation and growth. However, with this potential comes the responsibility to implement these technologies ethically and transparently. By integrating the insights and strategies discussed throughout this chapter, you can ensure that your AI applications are not only effective but also uphold the highest ethical standards.

Embrace these challenges with confidence and creativity, and as you apply the concepts that we've covered throughout the book, remember that continuous learning and adaptation are key to staying ahead. Good luck on your journey going forward!

## Join our book's Discord space

Join our Discord community to meet like-minded people and learn alongside more than 5000 members at:

<https://packt.link/genai>



[OceanofPDF.com](http://OceanofPDF.com)



[packt.com](http://packt.com)

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

## Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

At [www.packt.com](http://www.packt.com), you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

# Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



## Generative AI with LangChain

Ben Auffarth

ISBN: 978-1-83508-346-8

- Understand LLMs, their strengths and limitations
- Grasp generative AI fundamentals and industry trends
- Create LLM apps with LangChain like question-answering systems and chatbots
- Understand transformer models and attention mechanisms

- Automate data analysis and visualization using pandas and Python
- Grasp prompt engineering to improve performance
- Fine-tune LLMs and get to know the tools to unleash their power
- Deploy LLMs as a service with LangChain and apply evaluation strategies
- Privately interact with documents using open-source LLMs to prevent data leaks

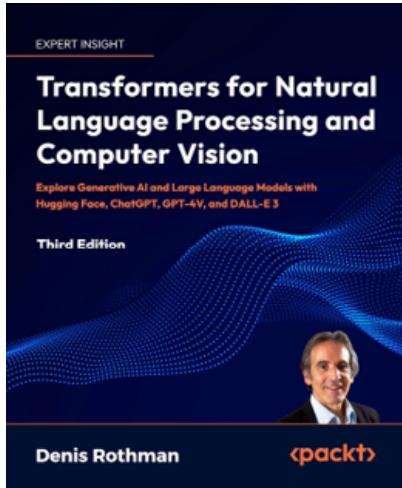


## Building LLM Powered Applications

Valentina Alto

ISBN: 978-1-83546-231-7

- Explore the core components of LLM architecture, including encoder-decoder blocks and embeddings
- Understand the unique features of LLMs like GPT-3.5/4, Llama 2, and Falcon LLM
- Use AI orchestrators like LangChain, with Streamlit for the frontend
- Get familiar with LLM components such as memory, prompts, and tools
- Learn how to use non-parametric knowledge and vector databases
- Understand the implications of LLMs for AI research and industry applications
- Customize your LLMs with fine tuning
- Learn about the ethical implications of LLM-powered applications



## **Transformers for Natural Language Processing and Computer Vision – Third Edition**

Denis Rothman

ISBN: 978-1-80512-872-4

- Breakdown and understand the architectures of the Original Transformer, BERT, GPT models, T5, PaLM, ViT, CLIP, and DALL-E
- Fine-tune BERT, GPT, and PaLM 2 models
- Learn about different tokenizers and the best practices for preprocessing language data
- Pretrain a RoBERTa model from scratch
- Implement retrieval augmented generation and rules bases to mitigate hallucinations
- Visualize transformer model activity for deeper insights using BertViz, LIME, and SHAP
- Go in-depth into vision transformers with CLIP, DALL-E 2, DALL-E 3, and GPT-4V



# Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit [authors.packtpub.com](https://authors.packtpub.com) and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

# Share your thoughts

Now you've finished *Machine Learning and Generative AI for Marketing*, we'd love to hear your thoughts! If you purchased the book from Amazon, please [click here to go straight to the Amazon review page](#) for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

[OceanofPDF.com](#)

# Index

## A

### **A/B testing** [11](#), [213](#)

conducting, for optimal model choice [213](#), [214](#)

simulating [214-218](#)

two-tailed t-test [219](#), [220](#)

### **Additive time-series decomposition method** [120-124](#)

### **AI advancements** [145](#)

#### **AI marketing models**

bias, mitigating strategies [417](#)

#### **AI/ML, for marketing** [396](#)

bias, mitigating [416-419](#)

ethical considerations [410](#)

ethics and governance [168](#)

explainability [410](#), [411](#)

model architecture advances [398](#)

model explainability tools [411-416](#)

multi-modal GenAI [401](#)

privacy with personalization, balancing [420-426](#)

Python environment, setting up for [13](#)

Python libraries, installing for [14](#)

ReAct [396](#)

transparency [410](#)

**Akaike Information Criterion (AIC)** [131](#)

**Anaconda distribution**

installing [13](#)

**anomaly attribution** [102-105](#)

**anonymization** [421](#)

**API services**

using, for transfer learning [340](#), [341](#)

**applications in marketing, ReAct**

dynamic pricing adjustments [397](#)

interactive customer service [398](#)

real-time inventory management [397](#)

**Apriori algorithm** [226-228](#)

**area charts** [65](#)

**area under the curve (AUC)** [193](#)

**ARIMA model** [129](#)

components [129](#)

components in diagnostic plot [132](#), [133](#)

diagnostics [131](#), [132](#)

forecasting with [133-135](#)

training [129-131](#)

**artificial neural network (ANN) model** [201](#), [305](#)

architecture [202](#)

backward propagation [202](#)

forward propagation [202](#)

**association\_rules function [228-233](#)**

parameters [229](#)

**AUC - ROC Curve [192, 193](#)**

**augmented reality (AR) [403](#)**

**autocorrelation [115-117](#)**

**autoregression [129](#)**

**autoregressive integrated moving average (ARIMA) model [116, 128](#)**

## B

**bar charts [65](#)**

**Bayesian inference and probabilistic models [394](#)**

**Bayes Information Criterion (BIC) [131](#)**

**Bayes' theorem [287](#)**

**BERT [279](#)**

**BIA Advisory report**

reference link [254](#)

**brand narrative**

tracking [176-179](#)

## C

**California Consumer Privacy Act (CCPA)** [168](#), [409](#), [432](#), [433](#)

URL [432](#)

**campaign strategy, micro-targeting**

brand, selecting [385](#), [386](#)

determining [383](#)

message timing [383-385](#)

**capsule networks** [400](#)

**categorical variable** [74](#)

categorical function [75](#)

dummy variables [75](#)

factorize function [74](#)

regression analysis [77](#)

versus continuous variable [82-84](#)

**causal analysis** [88](#), [91-93](#)

**causal effect estimation** [90](#), [91](#)

causal model [91-93](#)

estimation [93](#), [94](#)

refutation [95](#), [96](#)

**causal inference** [88](#), [89](#)

causal effect estimation [90](#), [91](#)

graphical causal model [96](#)

**causal influence** [97-101](#)

**chain-of-thought reasoning** [419](#)

**class imbalance** [151](#)

addressing [153](#)

evaluating [151-153](#)

GenAI, for data augmentation [154, 155](#)

traditional strategies [153](#)

## **collaborative filtering [223, 234](#)**

item-based collaborative filtering [245](#)

recommendation methods, usage [251, 252](#)

recommendation systems [234-236](#)

user-based collaborative filtering [236-238](#)

## **confusion matrix [160, 161, 194, 195](#)**

## **contact frequency modeling [10](#)**

## **contextual embeddings [164, 299, 300, 394](#)**

significance [300](#)

## **continuous bag of words (CBOW) [164](#)**

## **continuous variable [72, 73](#)**

versus categorical variable [82-84](#)

## **conversion rate [29, 35](#)**

calculating [35-37](#)

demographics [37, 38](#)

sales channel [38-41](#)

## **convolutional neural networks (CNNs) [290](#)**

## **cost-benefit analysis**

of FSL and transfer learning [395](#)

## **cost per acquisition (CPA) [29, 32, 33, 48, 49](#)**

marketing promotions [52-55](#)

sales channel [50, 51](#)

**cost per lead (CPL)** [31](#)

**customer churn prediction** [10](#)

**customer churn rate (CCR)** [32](#)

**customer conversion**

predicting, with deep learning algorithms [201, 202](#)

predicting, with tree-based algorithms [184](#)

**customer data** [10](#)

demographic details [10](#)

online behavior patterns [10](#)

purchasing history [10](#)

social media interactions [10](#)

**customer life cycle** [184](#)

**customer lifetime value (CLV)** [12, 29, 32, 33, 41, 43](#)

box plot [43](#)

distribution [41, 42](#)

geolocation [44-46](#)

histogram chart [43](#)

products [46, 47](#)

**customer retention**

benefits [254, 255](#)

impact, analyzing [255-263](#)

**customer retention rate (CRR)** [32](#)

## **customer segmentation [253](#)**

with product interests [278-282](#)

with purchase behaviors [263](#)

# **D**

## **dashboard [66](#)**

### **data**

preparing, for sentiment analysis [147](#)

### **data and model**

licensing and attribution [427](#), [428](#)

### **data anonymization [422](#), [423](#)**

### **data, for sentiment analysis**

class Imbalance [151](#)

traditional NLP techniques, for data preparation [148](#)

### **data freshness [365](#)**

### **data specificity [366](#)**

### **data splitting [24](#)**

### **data updating and curation**

automated monitoring [368](#)

dynamic updating [367](#)

### **decision tree analysis [85-88](#)**

### **decision tree interpretation [80](#), [81](#)**

continuous variable, versus categorical variable [82-84](#)

decision tree analysis [85-88](#)

target variable [82](#)

**decision trees** [80](#)

**deep learning (DL) models** [141](#), [201](#)

building, for time-series data [141](#)

deep neural network models, building [210](#)

train and test sets, using [203](#)

wide neural network models, building [203](#)

**Deep Neural Network models** [202](#)

building [210](#)

evaluating [212](#), [213](#)

predicting [211](#)

training [210](#), [211](#)

**developer APIs**

accessing [170](#)

**differential privacy** [421](#)

**diffusion models** [398](#)

**digital marketing** [4](#), [5](#)

## E

**email opening likelihood** [10](#)

**entropy information gain** [81](#)

**estimation** [93](#)

**ethical AI deployment** [420](#)

**ethical governance framework** [428](#)

continuous training and awareness [430](#)

internal AI ethics committees [429](#)

**explainability** [410](#)

**explainability, in LLMs**

challenges [411](#)

## F

**Family Educational Rights and Privacy Act (FERPA)** [435](#)

**feature engineering** [156](#)

exploring, with TF-IDF [157](#)

**federated learning** [420](#), [421](#)

**few-shot learning (FSL)** [13](#), [323](#), [361](#), [394](#)

applying, to improve brand consistency [343](#), [344](#)

benchmarking with [344-346](#)

challenges, overcoming [330-333](#)

learning, through meta-learning [324](#)

MAML, implementing in marketing [325-330](#)

navigating [324](#)

techniques, optimizing [333](#)

versus transfer learning [334](#)

**Financial Industry Regulatory Authority (FINRA)** [435](#)

**frequency penalty** [312](#)

**FSL, in email marketing campaign**

developing [346](#)

developing, steps [347-359](#)

performance, benchmarking [351](#)

**funnel charts** [65](#)

## G

**GenAI and ML, in marketing** [393](#)

**General Data Protection Regulation (GDPR)** [168](#), [409](#), [430-432](#)

**generalization** [333](#)

**Generalized AutoRegressive Conditional Heteroskedasticity (GARCH)** [140](#)

**generative adversarial networks (GANs)** [289-291](#), [394](#)

**generative AI (GenAI)** [7](#), [8](#), [145](#), [286](#), [324](#)

for data augmentation [154](#), [155](#)

foundational models [288](#)

models [394](#)

probabilistic approach [286-288](#)

usage, considerations [298](#)

**generative AI (GenAI), foundational models**

GANs [289-291](#)

LSTMs [293-296](#)

transformer-based models [296-298](#)

VAEs [292](#), [293](#)

**Generative Pre-Trained Transformers (GPTs)** [279](#), [296-298](#), [394](#)

components [297](#)

## **geospatial analysis**

used, for mapping sentiment analysis [179, 180](#)

## **Gini Impurity [81](#)**

## **global regulations and standards [434](#)**

## **Gradient-Boosted Decision Tree (GBDT) model [185, 186, 196](#)**

evaluating [198-200](#)

predicting [197](#)

training [196, 197](#)

## **graphical causal model [96](#)**

anomaly attribution [102-105](#)

causal influence [97-101](#)

## **H**

## **Health Insurance Portability and Accountability Act (HIPAA) [434](#)**

## **heat maps [63-65](#)**

## **I**

## **indexing strategies**

graph-based indexing [367](#)

inverted indices [367](#)

multi-dimensional indexing [367](#)

vector space models [367](#)

## **intellectual property protection [427](#)**

**interaction variable** [78, 79](#)

**item-based collaborative filtering model** [245](#)

recommendations, based on item similarities [246-248](#)

recommendations, based on purchase history [248-251](#)

## J

**JupyterLab** [14](#)

launching [14](#)

**Jupyter notebook** [16](#)

cells, creating [16](#)

cells, running [16](#)

cell types, modifying [16](#)

keyboard shortcuts [17](#)

saving [17](#)

sharing [17](#)

## K

**Keras**

using, for transfer learning [336-340](#)

**key performance indicators (KPIs)** [8, 29, 30, 69](#)

computing, from data with Python [34](#)

correlations [63-65](#)

ongoing KPI, tracking with dashboards [66, 67](#)

marketing performance, tracking [65](#)

visualization, selecting [65](#)

visualizing, from data with Python [34](#)

## **K-means clustering [264-267](#)**

randomness [268](#)

with log transformation [271-274](#)

without log transformation [267-271](#)

## **knowledge retrieval system**

building for marketing, with LangChain [370](#)

external dataset, using [371-375](#)

retrieval model, designing with LangChain [375](#)

## **Kullback-Leibler (KL) [292](#)**

# **L**

## **LangChain [370, 396](#)**

features [371](#)

knowledge-retrieval system, building with [396](#)

reference link [371](#)

## **LangChain, for micro-targeting**

product upselling case study [388](#)

real-time content customization for bpw.style, case study [389-391](#)

targeted product discounts case study [386, 387](#)

using [386](#)

## **Laplace mechanism**

noise, adding with [424](#)

**large language models (LLMs)** [145](#), [253](#), [299](#), [343](#)

grounding [419](#)

**latent Dirichlet allocation (LDA)**

latent topics, discovering with [174-176](#)

**lemmatization** [150](#), [151](#)

**linear regression** [70](#)

**Local Interpretable Model-agnostic Explanations (LIME)** [167](#)

**logistic regression** [70](#)

**long short-term memory (LSTM) networks** [287](#), [293-296](#), [394](#)

components [294](#)

## M

**Markdown**

using, for documentation [16](#)

**market basket analysis** [223](#)

product analytics with [224-226](#)

**marketing, history**

integration, of AI/ML [5-8](#)

pre-AI era [3](#), [4](#)

with AI/ML [2](#)

**marketing, core data science techniques** [8](#)

A/B testing [11](#)

AI, integrating for enhanced insights [12](#), [13](#)

customer data [10](#)

customer lifetime value (CLV) [12](#)

experimentation [11](#)

predictive analytics [9](#), [10](#)

segmentation [12](#)

targeting [12](#)

## **marketing KPIs [30](#)**

Awareness stage [31](#)

Conversion stage [31](#)

Engagement stage [31](#)

Loyalty stage [33](#), [34](#)

Retention stage [32](#), [33](#)

## **marketing performance**

tracking, with key performance indicators (KPIs) [65](#)

## **meta-learning [324](#)**

components [325](#)

used, for learning FSL [324](#)

## **misclassifications [161-163](#)**

## **mixed reality (MR) [404](#)**

### **ML in AR, for marketing [404](#)**

geolocation-based AR campaigns [405](#)

personalized AR experiences [405](#)

seamless product integration [405](#)

### **ML in VR, for marketing [406](#)**

behavioral data insights [406](#)

immersive storytelling and brand experiences [406](#)

virtual storefronts and events [406](#)

## **ML model [140](#), [141](#)**

building [156](#)

classification report [159](#), [160](#)

evaluating [159](#)

feature engineering [156](#)

training [157](#), [158](#)

## **ML model, classification report**

confusion matrix [160](#), [161](#)

misclassifications [161-163](#)

## **ML models, with Google Colab notebooks**

exploring [289](#)

## **ML model, training steps [17](#)**

data, loading [18](#)

data preparation, for ML [23](#), [24](#)

exploratory data analysis (EDA) [19-22](#)

model, evaluating [26](#), [27](#)

model, training [25](#), [26](#)

required libraries, importing [17](#)

## **mlxtend package**

association\_rules function [229](#)

## **model accuracy [27](#)**

## **model-agnostic meta-learning (MAML)**

implementing, in marketing [325-330](#)

## **model architecture advances**

capsule networks [398-400](#)

diffusion models [398](#)

neural radiance fields (NeRFs) [398, 399](#)

## **model evaluation data**

diversity [418](#)

## **moving average (MA) [112-115, 129](#)**

## **multi-layer perceptron (MLP) models [141](#)**

## **multi-modal GenAI**

technical foundations [401](#)

## **multi-modal GenAI applications [402](#)**

enhanced consumer engagement [402](#)

interactive customer support [403](#)

real-time content adaptation [403](#)

## **multiplicative time-series decomposition method [124-128](#)**

# **N**

## **named entity recognition (NER) [170](#)**

enhancing, with custom training [171](#)

performing, on dataset for fictional retailer [170](#)

## **natural language processing (NLP) [13, 143](#)**

## **net promoter score (NPS) [33](#)**

**neural radiance fields (NeRFs)** [399](#)

**nucleus sampling** [311](#)

## O

**one-time customer**

versus repeat customers [254](#)

**overfitting** [330](#)

versus underfitting [191](#)

## P

**partial autocorrelation** [116](#)

**Personal Information Protection Commission (PPC)** [434](#)

**personally identifiable information (PII)** [421](#)

**precision** [192](#)

**predictive analytics** [9](#), [10](#)

customer conversion, with deep learning algorithms [201](#), [202](#)

customer conversion, with tree-based algorithms [184](#)

**pre-trained LLMs**

implementing [164](#), [165](#)

performance, evaluating [165-167](#)

using [163](#), [164](#)

**pre-trained models** [299](#), [304](#), [394](#)

architecture [306](#), [307](#)

components [304](#)

preprocessing steps [307](#)

weights [305](#), [306](#)

## **principal component analysis (PCA) [279](#)**

### **product analytics**

with market basket analysis [224-226](#)

### **product analytics, with market basket analysis**

Apriori algorithm [226-228](#)

association\_rules function [228-233](#)

### **Prophet model [135](#)**

components [135](#)

forecasting with [136-139](#)

references [135](#)

training [136](#)

## **purchase/conversion likelihood [10](#)**

### **Python**

used, for computing KPIs from data [34](#)

used, for visualizing KPIs from data [34](#)

ZSL, preparing [313](#), [314](#)

### **Python environment**

setting up, for AI/ML projects [13](#)

setup, verifying [15](#)

### **Python libraries**

installing, for AI/ML [14](#)

# R

## **RAG, for precision marketing [362](#)**

applications [368-370](#)  
components [363](#)  
data curation and updating [367](#)  
data freshness [365](#)  
data specificity [366](#)  
functioning [362-364](#)  
implementation challenges [368](#)  
indexing strategies [366, 367](#)  
mathematical model [364](#)  
retrieval index [366](#)  
significance of data [365](#)

## **RAG implementation, for micro-targeting [382](#)**

campaign strategy, determining [383](#)  
LangChain, using [386](#)

## **Random Forest model [185](#)**

building [187](#)  
evaluating [191-194](#)  
predicting [191](#)  
target and feature variables [187-190](#)  
training [190, 191](#)

## **ReAct [396](#)**

applications in marketing [397, 398](#)

working [396](#), [397](#)

**rectified linear unit (ReLU) activation function** [204](#)

**recurrent neural network (RNN) models** [141](#), [287](#)

**refutation** [95](#), [96](#)

**regression analysis** [70](#), [71](#), [140](#)

    categorical variable [74](#)

    continuous variable [71](#)-[74](#)

    interaction variable [78](#), [79](#)

    target variable [71](#)

**regulatory compliance** [430](#)

    California Consumer Privacy Act (CCPA) [432](#), [433](#)

    General Data Protection Regulation (GDPR) [430](#)-[432](#)

    global regulations and standards [434](#)

    industry-specific guidelines [434](#), [435](#)

**repeat customers**

    versus one-time customers [254](#)

**retrieval-augmented generation (RAG)** [13](#), [298](#), [361](#), [395](#), [419](#)

    applications in marketing [396](#)

    references [370](#)

**retrieval index** [366](#)

**retrieval model, designing with LangChain** [375](#)

    data indexing, in Elasticsearch [377](#)-[379](#)

    data ingesting, into Elasticsearch [379](#), [380](#)

    Elasticsearch, connecting to [377](#)

Elasticsearch installation [375](#), [376](#)

LangChain integration, GPT used [380-382](#)

**return on investment (ROI)** [29](#), [56-58](#)

per promotion [61](#), [62](#)

per sales channel [59](#), [60](#)

**ROC Curve** [193](#)

plotting [193](#), [194](#)

**root mean squared error (RMSE)** [123](#)

## S

**scatter plots** [65](#)

**seasonal ARIMA (SARIMA)** [140](#)

**segmentation** [12](#)

**semantic proximity** [300-303](#)

**sentiment analysis**

advancements [145](#)

data, preparing [147](#)

ML model, building [156](#)

performing [156](#)

pre-trained LLMs, using [163](#), [164](#)

translating, into actionable insights [168](#)

**sentiment analysis, in marketing**

application [144](#), [145](#)

significance [144](#)

## **sentiment analysis, insights**

brand narrative, tracking [176-179](#)  
data, collecting from other platforms [170](#)  
dataset, creating [168](#)  
geospatial analysis, used for mapping [179, 180](#)  
latent topics with LDA, discovering [174-176](#)  
NER, performing on dataset for fictional retailer [170](#)  
topics and themes [171](#)  
Twitter data, collecting [168-170](#)  
word clouds, using [172-174](#)

**Shapley value** [101](#)

**sigmoid function** [204](#)

**silhouette score** [275- 278](#)

**Stanford Sentiment Treebank (SST-2)** [164](#)

**static embeddings** [164](#)

**statistical models** [140](#)

**support vector machine (SVM)** [141](#)

**Synthetic Minority Over-sampling TEchnique (SMOTE)** [153](#)

## **T**

**temperature parameter** [310](#)

**Temporal Fusion Transformer (TFT)** [141](#)

**term frequency-inverse document frequency (TF-IDF)** [156](#)

feature engineering, exploring [157](#)

## **time series analysis** [108](#)

autocorrelation [115-117](#)

moving averages [113-115](#)

Moving Averages [112](#)

product trends [118, 119](#)

trends [108-112](#)

## **time-series data**

DL model, building for [141](#)

## **time series forecasting models** [128](#)

ARIMA model [129](#)

Prophet time-series modeling [135](#)

## **time-series model**

forecasting with [140](#)

types, of approaches [140, 141](#)

## **traditional marketing** [4](#)

## **traditional NLP techniques, for data preparation**

lemmatization [150, 151](#)

text data, cleaning [148, 149](#)

tokenization and stop word removal [150](#)

## **traditional sentiment models**

performance, enhancing [163](#)

## **transfer learning** [395](#)

challenges, overcoming [342, 343](#)

mechanics [335, 336](#)

navigating [333](#), [334](#)

versus FSL [334](#)

with API services [340](#), [341](#)

with Keras [336](#)-[340](#)

**transparency** [410](#)

**tree-based machine learning algorithms** [185](#), [186](#)

**tree-based models** [140](#)

**T-SNE**

reference link [280](#)

**t-value** [213](#)

**Twitter**

data, collecting [168](#)-[170](#)

**two-sample t-test** [213](#)

**two-tailed t-test** [219](#), [220](#)

## U

**UMAP**

reference link [280](#)

**underfitting**

versus overfitting [191](#)

**user-based collaborative filtering model** [236](#)-[238](#)

recommending, by most similar customer [238](#)-[241](#)

recommending, by top products bought by similar customers  
[241](#)-[245](#)

## V

**variational autoencoders (VAEs)** [292](#), [293](#), [394](#)

versatile applications [292](#)

**virtual reality (VR)** [403](#)

**visual EDA** [23](#)

**visualization**

selecting, for KPIs [65](#)

## W

**waterfall charts** [58](#), [66](#)

**website login frequency modeling** [10](#)

**wide neural network models** [202](#)

building [203](#)

evaluating [208-210](#)

predicting [207](#)

training [203-207](#)

**word clouds**

using [172-174](#)

## Z

**zero-shot learning (ZSL)** [13](#), [285](#), [299](#), [308](#), [323](#), [361](#), [394](#)

benchmarking with [344-346](#)

effective prompt, creating [315](#), [316](#)

effective prompt, creating examples [316-318](#)

for marketing copy [312](#)  
learning and prediction, mechanics [308](#), [309](#)  
output parameters [309](#), [310](#)  
parameter tuning, impact [319](#), [320](#)  
preparing, in Python [313](#), [314](#)

## ZSL, output parameters

frequency penalty [312](#)  
nucleus sampling [311](#)  
temperature parameter [310](#), [311](#)

# Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily.

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below:



<https://packt.link/free-ebook/9781835889404>

2. Submit your proof of purchase.
3. That's it! We'll send your free PDF and other benefits to your email directly.

[OceanofPDF.com](http://OceanofPDF.com)