

Certified Data Science Practitioner (CDSP) (Exam DSP-110)

Certified Data Science Practitioner (CDSP) (Exam DSP-110)

Part Number: CNX0011

Course Edition: 1.0

Acknowledgements

PROJECT TEAM

Authors	Media Designer	Content Editor
Jason Nufryk	Brian Sullivan	Geoff Graser
Sarah Haq		

CertNexus wishes to thank Ed Griebel, Stacey McBrine, and Semih Kumlu for their instructional and technical expertise during the creation of this course.

Notices

DISCLAIMER

While CertNexus, Inc. takes care to ensure the accuracy and quality of these materials, we cannot guarantee their accuracy, and all materials are provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. The name used in the data files for this course is that of a fictitious company. Any resemblance to current or future companies is purely coincidental. We do not believe we have used anyone's name in creating this course, but if we have, please notify us and we will change the name in the next revision of the course. CertNexus is an independent provider of integrated training solutions for individuals, businesses, educational institutions, and government agencies. The use of screenshots, photographs of another entity's products, or another entity's product name or service in this book is for editorial purposes only. No such use should be construed to imply sponsorship or endorsement of the book by nor any affiliation of such entity with CertNexus. This courseware may contain links to sites on the Internet that are owned and operated by third parties (the "External Sites"). CertNexus is not responsible for the availability of, or the content located on or through, any External Site. Please contact CertNexus if you have any concerns regarding such links or External Sites.

TRADEMARK NOTICES

CertNexus and the CertNexus logo are trademarks of CertNexus, Inc. and its affiliates.

All other product and service names used may be common law or registered trademarks of their respective proprietors.

Copyright © 2021 CertNexus, Inc. All rights reserved. Screenshots used for illustrative purposes are the property of the software proprietor. This publication, or any part thereof, may not be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, storage in an information retrieval system, or otherwise, without express written permission of CertNexus, 3535 Winton Place, Rochester, NY 14623, 1-800-326-8724 in the United States and Canada, 1-585-350-7000 in all other countries. CertNexus' World Wide Web site is located at www.certnexus.com.

This book conveys no rights in the software or other products about which it was written; all use or licensing of such software or other products is the responsibility of the user according to terms and conditions of the owner. Do not make illegal copies of books or software. If you believe that this book, related materials, or any other CertNexus materials are being reproduced or transmitted without permission, please call 1-800-326-8724 in the United States and Canada, 1-585-350-7000 in all other countries.

Certified Data Science Practitioner (CDSP) (Exam DSP-110)

Lesson 1: Addressing Business Issues with Data Science.....	1
Topic A: Initiate a Data Science Project.....	2
Topic B: Formulate a Data Science Problem.....	14
Lesson 2: Extracting, Transforming, and Loading Data..	27
Topic A: Extract Data.....	28
Topic B: Transform Data.....	57
Topic C: Load Data.....	92
Lesson 3: Analyzing Data.....	107
Topic A: Examine Data.....	108
Topic B: Explore the Underlying Distribution of Data.....	120
Topic C: Use Visualizations to Analyze Data.....	140
Topic D: Preprocess Data.....	174
Lesson 4: Designing a Machine Learning Approach....	225
Topic A: Identify Machine Learning Concepts.....	226

Topic B: Test a Hypothesis.....	239
Lesson 5: Developing Classification Models.....	251
Topic A: Train and Tune Classification Models.....	252
Topic B: Evaluate Classification Models.....	304
Lesson 6: Developing Regression Models.....	329
Topic A: Train and Tune Regression Models.....	330
Topic B: Evaluate Regression Models.....	367
Lesson 7: Developing Clustering Models.....	377
Topic A: Train and Tune Clustering Models.....	378
Topic B: Evaluate Clustering Models.....	400
Lesson 8: Finalizing a Data Science Project.....	415
Topic A: Communicate Results to Stakeholders.....	416
Topic B: Demonstrate Models in a Web App.....	427
Topic C: Implement and Test Production Pipelines.....	440
Appendix A: Mapping Course Content to CertNexus® Certified Data Science Practitioner (CDSP) (Exam DSP-110).....	459
Solutions.....	461
Glossary.....	471
Index.....	485

About This Course

For a business to thrive in our data-driven world, it must treat data as one of its most important assets. Data is crucial for understanding where the business is and where it's headed. Not only can data reveal insights, it can also inform—by guiding decisions and influencing day-to-day operations. This calls for a robust workforce of professionals who can analyze, understand, manipulate, and present data within an effective and repeatable process framework. In other words, the business world needs data science practitioners. This course will enable you to bring value to the business by putting data science concepts into practice.

Course Description

Target Student

This course is designed for business professionals who leverage data to address business issues. The typical student in this course will have several years of experience with computing technology, including some aptitude in computer programming.

However, there is not necessarily a single organizational role that this course targets. A prospective student might be a programmer looking to expand their knowledge of how to guide business decisions by collecting, wrangling, analyzing, and manipulating data through code; or a data analyst with a background in applied math and statistics who wants to take their skills to the next level; or any number of other data-driven situations.

Ultimately, the target student is someone who wants to learn how to more effectively extract insights from their work and leverage that insight in addressing business issues, thereby bringing greater value to the business.

This course is also designed to assist students in preparing for the CertNexus® Certified Data Science Practitioner (CDSP) (Exam DSP-110) certification.

Course Prerequisites

To ensure your success in this course, you should have at least a high-level understanding of fundamental data science concepts, including, but not limited to: types of data, data science roles, the overall data science lifecycle, and the benefits and challenges of data science. You can obtain this level of knowledge by taking the CertNexus DSBIZ™ (Exam DSZ-110) course.

You should have also have experience with high-level programming languages like Python. Being comfortable using fundamental Python data science libraries like NumPy and pandas is highly recommended. You can obtain this level of skills and knowledge by taking the Logical Operations course *Using Data Science Tools in Python*®.

In addition to programming, you should also have experience working with databases, including querying languages like SQL. Several Logical Operations courses can help you attain this experience:

- *Database Design: A Modern Approach*
- *SQL Querying: Fundamentals (Second Edition)*
- *SQL Querying: Advanced (Second Edition)*

Course Objectives

In this course, you will implement data science techniques in order to address business issues.

You will:

- Use data science principles to address business issues.
- Apply the extract, transform, and load (ETL) process to prepare datasets.
- Use multiple techniques to analyze data and extract valuable insights.
- Design a machine learning approach to address business issues.
- Train, tune, and evaluate classification models.
- Train, tune, and evaluate regression and forecasting models.
- Train, tune, and evaluate clustering models.
- Finalize a data science project by presenting models to an audience, putting models into production, and monitoring model performance.

The CHOICE Home Screen

Logon and access information for your CHOICE environment will be provided with your class experience. The CHOICE platform is your entry point to the CHOICE learning experience, of which this course manual is only one part.

On the CHOICE Home screen, you can access the CHOICE Course screens for your specific courses. Visit the CHOICE Course screen both during and after class to make use of the world of support and instructional resources that make up the CHOICE experience.

Each CHOICE Course screen will give you access to the following resources:

- **Classroom:** A link to your training provider's classroom environment.
- **eBook:** An interactive electronic version of the printed book for your course.
- **Files:** Any course files available to download.
- **Checklists:** Step-by-step procedures and general guidelines you can use as a reference during and after class.
- **Spotlights:** Brief animated videos that enhance and extend the classroom learning experience.
- **Assessment:** A course assessment for your self-assessment of the course content.
- Social media resources that enable you to collaborate with others in the learning community using professional communications sites such as LinkedIn or microblogging tools such as Twitter.

Depending on the nature of your course and the components chosen by your learning provider, the CHOICE Course screen may also include access to elements such as:

- LogicalLABs, a virtual technical environment for your course.
- Various partner resources related to the courseware.
- Related certifications or credentials.
- A link to your training provider's website.
- Notices from the CHOICE administrator.
- Newsletters and other communications from your learning provider.
- Mentoring services.

Visit your CHOICE Home screen often to connect, communicate, and extend your learning experience!

How To Use This Book

As You Learn

This book is divided into lessons and topics, covering a subject or a set of related subjects. In most cases, lessons are arranged in order of increasing proficiency.

The results-oriented topics include relevant and supporting information you need to master the content. Each topic has various types of activities designed to enable you to solidify your understanding of the informational material presented in the course. Information is provided for reference and reflection to facilitate understanding and practice.

Data files for various activities as well as other supporting files for the course are available by download from the CHOICE Course screen. In addition to sample data for the course exercises, the course files may contain media components to enhance your learning and additional reference materials for use both during and after the course.

Checklists of procedures and guidelines can be used during class and as after-class references when you're back on the job and need to refresh your understanding.

At the back of the book, you will find a glossary of the definitions of the terms and concepts used throughout the course. You will also find an index to assist in locating information within the instructional components of the book. In many electronic versions of the book, you can click links on key words in the content to move to the associated glossary definition, and on page references in the index to move to that term in the content. To return to the previous location in the document after clicking a link, use the appropriate functionality in your PDF viewing software.

As You Review

Any method of instruction is only as effective as the time and effort you, the student, are willing to invest in it. In addition, some of the information that you learn in class may not be important to you immediately, but it may become important later. For this reason, we encourage you to spend some time reviewing the content of the course after your time in the classroom.

As a Reference

The organization and layout of this book make it an easy-to-use resource for future reference. Taking advantage of the glossary, index, and table of contents, you can use this book as a first source of definitions, background information, and summaries.

Course Icons

Watch throughout the material for the following visual cues.

Icon	Description
	A Note provides additional information, guidance, or hints about a topic or task.
	A Caution note makes you aware of places where you need to be particularly careful with your actions, settings, or decisions so that you can be sure to get the desired results of an activity or task.
	Spotlight notes show you where an associated Spotlight is particularly relevant to the content. Access Spotlights from your CHOICE Course screen.
	Checklists provide job aids you can use after class as a reference to perform skills back on the job. Access checklists from your CHOICE Course screen.
	Social notes remind you to check your CHOICE Course screen for opportunities to interact with the CHOICE community using social media.

1

Addressing Business Issues with Data Science

Lesson Time: 2 hours, 10 minutes

Lesson Introduction

Before you dive into the technical details of data science, you need to understand how data science fits within a larger business context. On the job, you'll be applying your skills to achieve one or more business goals, so it's important to keep those goals in mind all throughout the project. In this lesson, you'll identify the nature of a data science project and your role within it, as well as identify overall strategies for leveraging data science as a solution to business issues.

Lesson Objectives

In this lesson, you will:

- Kick off a data science project by defining the project's scope and working closely with the project's stakeholders.
- Leverage a data science approach in addressing business issues.

TOPIC A

Initiate a Data Science Project

A lot goes into designing a project before the actual work can begin. Even though you may not be making the big decisions, your input will still be valuable at the preliminary stages. It's also important for practitioners to understand the project's design principles so that they can stay focused on bringing value to the business.

Data Science

Data science is a discipline that involves accumulating data, analyzing the data, extracting value from the data, and presenting the value of the data in a meaningful way. It is not a single event, but an entire process that capitalizes on the idea that data is the primary driver of business success in the modern world. Such success can come in many forms, including immediate financial benefits, long-term improvements to business operations, greater public acceptance, brand recognition, and many more.

Data science is a somewhat flexible term and can refer to different concepts and practices depending on who you ask, or when you ask them. Still, there are some common tasks that either make up the whole of an organization's data science practices, or are one component in a larger ecosystem. Examples include:

- Data collection
- Data analysis
- Data mining
- Data wrangling/munging
- Data presentation
- Machine learning
- Deep learning

As a data science practitioner, you may do one, some, or even all of these tasks. Regardless of what you're responsible for on any given project, it's important to understand how they come together to bring benefit to the business. Ultimately, data science is like any other business process—it exists to serve the needs of the business and accomplish its goals.

Design Thinking

Before you begin establishing a project that employs data science, you should engage in some design thinking. **Design thinking** is an approach to generating business ideas that focuses on human needs and innovation. Traditional design approaches may have identified a superficial need for a product or service by looking at current trends, or perhaps the business would just go through the motions and design a "new" product that everyone expects of them. But design thinking tries to break free of the status quo and instead take a deeper dive into what people really need, why they need it, and how that need can best be fulfilled.

A design thinking process encourages empathy, as without it, it's not really possible to truly understand someone's needs and desires. Empathy requires that you set aside preconceived notions or biases and try to see things from someone else's point of view. This can also lead to innovation, as designers are no longer stuck in one perspective, and instead learn from new experiences.

Design thinking also encourages asking questions to help generate ideas. Even if there are no immediate answers to those questions, asking them at all is a good first step in determining what the business should pay closer attention to as the project develops. For example, a question might be, "Are there needs or wants that people have that we've been neglecting due to a perceived risk?"

Design thinking is not necessarily about coming up with a perfect solution right away. It's more akin to a process of brainstorming; as people generate ideas and then discuss those ideas, they can begin to imagine how data science might be a major factor, or perhaps how it might not. There may be alternative solutions that don't incorporate data science, or incorporate it in different ways than what you've done in the past. Having a set of options can reduce risk and make the path forward more clear. It can also increase buy-in from employees and customers alike.

So, to summarize, design thinking is a process of:

- Empathizing with your target audience.
- Asking questions to outline any issues that people have.
- Brainstorming ideas that can address these issues.



Note: Some definitions of design thinking also incorporate prototyping, testing, and implementation of ideas.

Project Scope

As a data science practitioner, you may not be directly involved in the high-level management aspect of the project. However, you'll still likely need to participate in or at least have an understanding of key project components. One of those components is the project scope. **Scope** refers to an outline of all aspects of the project, including any constraints set on the project that it's not meant to exceed.

A project's scope is defined early in the project management lifecycle. In a data science context, this can include:

- **Software and hardware resources.** Data science projects will need to allocate both hardware and software resources to the team. Software resources can include programming languages like Python and R, third-party libraries, development environments, testing environments, and so forth. Hardware resources can include central processing units (CPUs), graphics processing units (GPUs), data storage devices like solid-state drives (SSDs), network devices like routers, and more. How those resources are allocated is also part of the scope; i.e., are they on premises or provisioned in the cloud?
- **Practitioners and other stakeholders.** The scope must also determine who will be working on the project, and in what capacity. Some practitioners are better equipped to do one task, whereas other practitioners should be set on a different task. Even non-practitioners may have an impact on the project, such as IT employees who maintain the network that data science practitioners use to communicate or share data. You may even include external customers in the scope of the project if they need to provide input.
- **Processes and procedures.** The scope must clearly define processes and procedures for accomplishing data science tasks, otherwise personnel won't be able to work consistently.
- **Measures of success.** Like any project, a data science project must have a way to measure success so that you can be confident that its objectives are fulfilled.
- **Timelines and deliverables.** Project members must be aware of what end product they are expected to produce, and when. They may also need to produce incremental deliverables at certain stages to ensure the project is going smoothly.

Scope Creep

Placing constraints on a project is important because, without them, the project may continue to grow, taking on more and more, until the project is no longer sustainable or fails to meet expectations—a concept called **scope creep**. You need to think about what is necessary in achieving your business goals versus what is not. Investing in multiple GPUs might cut down on model creation time, but the expense might outweigh the benefit. You might want to create multiple complex and sophisticated deliverables, but this might not be feasible in your timeline or with your budget. Or, maybe you don't have enough data (or enough quality data) to move forward on an idea.

You could also run into technical roadblocks, like not having access to ready-made tools that can address your specific needs. There might also be risks involved in implementing an idea, such as those that expose cybersecurity vulnerabilities or have significant ethical implications. You must account for these limitations when defining the scope of the project.

Project Specifications and Objectives

During the design of a data science project, team members need to identify the project's specifications. Specifications are a clear set of instructions and explanations for how the project's ultimate deliverable(s) will achieve business objectives. This means a specification document is not an abstract or philosophical exercise like a mission statement or design thinking, but a collection of detailed information that is required for development and implementation. For a data science project, components of the specification document can address the following topics.

Topic	Description
Project description	This is a short explanation of what the product or service is intended to do, who it's intended to serve, and what form(s) it will take.
Objectives	These can include high-level objectives that map directly to the ultimate product, like helping users more easily find products they are interested in. You can also have lower-level objectives that map to individual deliverables, like producing a model that can classify product images, or finishing development of a user interface for an e-commerce app, or successfully testing a product recommendation engine before deploying it to users, and so on.
Key performance indicators (KPIs)	KPIs are the primary way of measuring success in most projects. In the data science world, a KPI might be the score that a machine learning model receives when it is validated against test data. The organization might have a target score it's expecting to meet, or they might use the score as a means of comparing multiple attempts at modeling the data. KPIs can also include: the time it takes to build a model or related solution; the degree to which processes can be automated as part of a data science pipeline; user satisfaction to a test solution; the direct value a solution is bringing to the business (financial or otherwise); the performance of the project itself in terms of timeliness, resource usage, and communication; user engagement metrics like page view count and bounce rate; and much more. Any KPIs included in the project specifications must be assessed on a continuing basis so that any problems can be corrected early.
Available data	No data science project can get off the ground without the data itself. The project specifications should clearly outline where data will (or already has) come from, what state it's in, how it will be used for data science specifically, and how it will be managed in a typical IT context (e.g., access, backups, cybersecurity protections, etc.). There may also be data that you don't yet have access to, or that may be created during the process. The specifications should make it clear what data you need and how you plan to acquire it.
Analysis methods	Most team members aren't going to receive that data in a perfect state, or know everything about that data up front. There needs to be various analyses performed on the data to optimize it for use all the way through the project lifecycle. The specifications should outline what kinds of analysis will be performed, why it's necessary, and what benefit it will provide.

Topic	Description
Testing methods	At some point, the solutions will need to be tested so they don't go out to the public with critical issues. The specification document should outline the types of tests that apply to the project and when they should be conducted. For example, you might conduct a private beta test of your product recommendation system among a few employees so they can determine if the system does what it's supposed to and doesn't introduce errors or out-of-scope features.
Deployment and maintenance	Most data science projects will need to be implemented into a wider ecosystem after testing is done. The product recommendation system needs to go out to all of the users of the company's app, so it needs to go through a deployment phase. Likewise, most solutions aren't just deployed and then ignored; they will need to undergo maintenance so they can gradually improve and stay useful. The specification document should include planned methods for achieving both a smooth deployment and steady maintenance.

Specifications are meant to be handed off to developers and other practitioners, which probably means someone like you. You need to be able to internalize these specifications so that you don't do something out of scope or fail to account for an important factor. You may not be involved in drafting the specifications, but you will be responsible for following them.

Project Timeline

Like any other project, a data science project must begin and end within a certain time frame. The project timeline, or schedule, represents the team's plan for starting and finishing activities on specific dates and in a certain sequence. By planning out the project's activities, the team will be able to optimize how their time is spent, and better guarantee success. Also, tracking activities on a timeline can ensure that the project leaders and other stakeholders are kept informed of the team's progress.

The timeline also specifies planned dates for meeting project **milestones**, or events during a project that trigger a reporting requirement or that require approval from stakeholders before proceeding with the project. In a data science project, an early milestone might be a report on the state of data that was collected. The report might indicate that the data needs to be cleaned, perhaps in one way more than others, before it can be useful. This will flow nicely into the next phase of the project where the cleaning process is performed. The team has a plan, rather than just improvising at each step along the way.

How to determine a good timeline can be difficult in projects that deal in complex technical activities like data science. Obviously, if you have a hard start date and end date, you'll know the overall time you have to work on the project. A list of milestones can also help, since you'll know what you need to do, and in what order. Still, this usually isn't enough to create a useful timeline. You need to determine the expected duration of each activity, when it needs to be performed in relation to the other activities, and if it has any dependencies on other activities. As a practitioner, you may not create the timeline itself, but you'll probably need to provide input on these issues. For example, your project leader might ask for help determining where to place the creation of the first testable machine learning model in the timeline. So, you can consider factors like: the volume of data used in training, the algorithm used, the power of available hardware the model will be trained on, etc. This will better equip you to answer the question.

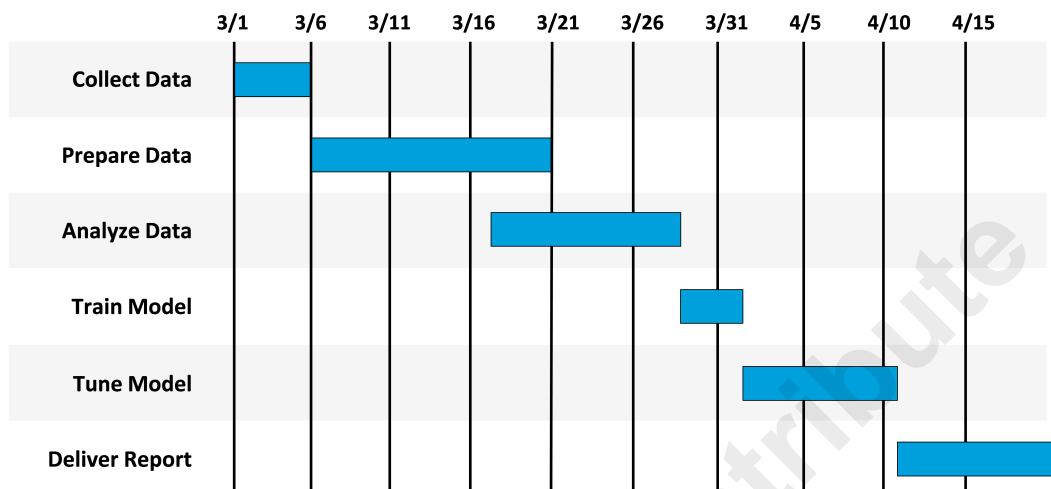


Figure 1–1: Representing a project timeline in a Gantt chart.

Project Deliverables

During the project timeline, one or more deliverables will need to be produced. A **deliverable** is any tangible, measurable result or outcome required to complete a project or portion of a project. A deliverable might be the ultimate product that the project was intended to produce. Or, it could be a building block that eventually becomes the ultimate product when many such deliverables are combined. In some cases, a deliverable may just be a report or other supporting material that doesn't necessarily make its way into the final product.

You can also divide deliverables into mandatory and optional. Mandatory deliverables, as the name implies, cannot be ignored and must be completed for the project to be considered a success. For example, if your business wants to develop a system that can intelligently recommend products to customers browsing an online store, the machine learning model that takes customer data as input and outputs a list of recommended product ID numbers is a mandatory deliverable. The website or app that delivers these recommendations in a user-friendly environment, if it doesn't already exist, would also be a mandatory deliverable. The overall solution cannot function without both of these major components in place.

Optional deliverables are less common, as the success of the project is not dependent on them being completed. They might be features that are nice to have, but lower priority. Remember, all of these deliverables must be completed within a timeline, so if finishing a low-priority feature takes away too much time spent on more important features, the project will be in jeopardy. So, the low-priority feature might be marked as an optional deliverable that the team will only focus on if they have time. In a data science project, an example of an optional deliverable might be the creation of a website that demonstrates the project's results to stakeholders. Some type of reporting at the end of a project is necessary, but building a website will probably take much more time than a typical document or slide show presentation. The website would be more dynamic and aesthetically pleasing, but it's not crucial for obtaining stakeholder buy-in.

Project Stakeholders

A project **stakeholder** is a person who has a vested interest in the outcome of a project or who is actively involved in its work. Stakeholders take on various roles and responsibilities; their participation in the project will have an effect on its outcome and its chance of success. There are many different types of stakeholders, both internal to the organization and external.

Common stakeholders include:

- **Customers/end users.** These are the people who will use the product or service generated by the project. They typically provide payment for the project output, and can also provide feedback from a usability perspective.
- **Sponsors/champions.** These are individuals or groups that provide finances, management support, and overall control of the project. They are integral to ensuring that the project outcomes are on track and within budget.
- **Program/project managers.** These are the people who manage the overall project or an aspect of a project. They typically provide guidance as far as timelines and resource allocation.
- **Team members.** These people are the practitioners who work directly to develop the project. They can provide insight into what tools, technologies, and other resources are needed in order for the project to be successful.
- **Business partners.** These organizations might sell the product or service, or they might provide resources during development. Since they're external, they tend to bring a fresh "outsider" perspective to the project.
- **Governments.** Various government organizations might take interest in projects that are large in scope or have legal ramifications. They provide rules and frameworks for legal issues.
- **Societies.** Some projects can have a profound impact on society at large, so people throughout a society will take notice. As a collective, people might express concerns about a project and its impact on everyday life.

Some or all of these stakeholders might be relevant to the specific project you're working on. You should consult with your project team leaders to get an accurate picture of who is involved.

Stakeholder Requirements

Stakeholders have competing interests, needs, priorities, and opinions. They may have conflicting visions for the project's successful outcome. Project leaders must identify internal and external stakeholders as early as possible, learn what their needs are, and secure their participation in defining the project's parameters and success criteria. As a practitioner, you need to at least understand the requirements that stakeholders have for the project so that you don't just work in a vacuum. After all, the whole purpose of the project is to bring value to the business, and stakeholder requirements define the form that value takes.

Part of stakeholder requirements is access to the project's milestones, as explained earlier. These milestones will ensure stakeholders remain confident about the direction the project is taking. Or, if necessary, they may choose to intervene in order to keep the project on track or fix issues with personnel, resource allocation, budgeting, timelines, and so on.

Proof of Concept (POC)

Another requirement that stakeholders may have is a **proof of concept (POC)**. A POC is evidence that supports the feasibility of the product or service that the project is meant to create. If successful, it signals to the stakeholders that the project can indeed fulfill its stated purpose, and that work can proceed. Therefore, a POC comes in the beginning of a project's lifecycle, when the design of the product or service is being evaluated. It is not an in-depth prototype that is meant to show *how* a product or service is developed, but merely that it *can* be developed.

In a data science context, a POC is usually a condensed version of the overall data science process: collecting a small amount of data, preparing that data in minimal ways, analyzing the data using just a few methods, building a simple model from the data, and so on. The POC lacks polish and is much smaller in scale than the full project will be, but it should be enough to convince stakeholders that you have a workable plan.

Minimum Viable Product (MVP)

Stakeholders can also expect a **minimum viable product (MVP)**. An MVP is a version of the product or service that is, at the bare minimum, usable by early adopters. The point is to solicit feedback from these early adopters so that the team can improve the product or service iteratively, until it meets some threshold for being a truly "finished" product. That way, the team doesn't waste time developing a finished output that has little market acceptance.

Since MVPs are customer facing, they also differ from prototypes and POCs, which are meant for internal review only. MVPs also differ in that the organization has a higher level of commitment to the direction they take, so you're less likely to scrap an MVP and start over.

In a data science project, an MVP might be a version of a product recommendation system that the organization can push to a select number of website accounts or app users. The system works (i.e., it recommends products), but it remains to be seen what the early adopters think of the experience.

Data Privacy and Security

Data privacy and security are huge concerns for all stakeholders in a project. Although it may not seem like you're responsible for keeping data private and secure, everyone who handles data is. The data you work with on a daily basis may contain **personally identifiable information (PII)** that could be used against someone, or the data you work with could contain proprietary information that would cause significant harm to the organization if it were to fall into the wrong hands. Whatever the case may be, protecting data is a top priority for a data science practitioner since you likely have access to volumes and volumes of sensitive information.

There are many security techniques that organizations employ to keep data safe—enough to fill entire courses. You don't have to become an expert in all of them, but you should at least be aware of the different policies that set expectations for security and privacy. Some of those policies are enforced by the organization, whereas some are enforced by law. You need to identify which policies are applicable to your organization and to your project, as your responsibilities may change based on the circumstances.

Example Security Policies

There are some policies that are relevant to many organizations that conduct data science. The following table provides some major examples.

Policy, Law, Regulation, or Standard	Description
Organizational policy	Most organizations have at least some type of policy that deals with cybersecurity topics, including the handling of data. Such policies might focus on the internal handling of data among employees, external handling by contractors, external handling by customers and the public, or perhaps all of these. Each organization's policy is different, so make sure you're up to speed with your organization's expectations.

Policy, Law, Regulation, or Standard	Description
GDPR	The General Data Protection Regulation (GDPR), which went into full effect in the European Union in 2018, is intended to protect individual privacy by holding data collection and data processing entities accountable for the information of EU citizens. The regulation applies to all entities that collect or process the personal data of EU citizens, even if the entity is not based in the EU, and includes provisions concerning the export of personal data outside of the EU. The GDPR ultimately upholds the privacy rights of individuals (e.g., the right to correct inaccurate personal data), enforces restrictions and security obligations for organizations (e.g., report data breaches within 72 hours), and issues penalties for non-compliance (e.g., fines up to €20 million or 4 percent of global turnover).
HIPAA	The Health Insurance Portability and Accountability Act (HIPAA) was enacted in 1996 to establish several rules and regulations regarding healthcare in the United States. With the rise of electronic medical records, HIPAA standards have been implemented to protect the privacy of patient medical information through restricted access to medical records and regulations for sharing medical records.
CCPA	California Consumer Privacy Act (CCPA) enhances the data privacy and protection rights of California residents, specifically an individual's right to know what data about them is collected and how, if at all, it is being sold or disclosed to other entities. The CCPA also grants individuals the right to access their personal data and deny other entities the ability to sell their data. Organizations that do business in California and meet one or more of the thresholds defined in the law are responsible for complying with CCPA. If these organizations suffer a breach in which personal data is disclosed, they may be in violation of the law and therefore subject to fines and other legal action.
PCI DSS	The Payment Card Industry Data Security Standard (PCI DSS) is a proprietary standard created by the credit card industry that specifies how organizations should handle information security for major card brands, including Visa, MasterCard, American Express, Discover, and JCB—all of which provide a mandate for the standard. The standard is intended to increase controls on cardholder data to reduce fraudulent use of accounts. Organizations or merchants that accept, transmit, or store cardholder data (regardless of size or number of transactions) must comply with this standard.

Data Access

Data in most organizations is access controlled so that only authorized users are allowed to view, modify, and/or delete that data. To uphold company security policy and any legal requirements, you'll likely be subjected to that access control program while working on a data science project. This is particularly true in the early stages of the project when you need to actually collect that data, but it can also apply to how you use data during development and how data is used during deployment.

You'll need to work with your IT team to obtain access permissions to any data that is currently under their purview. For example, your project may require you to work with customer data—perhaps you're trying to segment customers to enhance your targeted marketing capabilities. This

data already exists in databases maintained by the IT team, so you can't just expect to connect to the databases and have free rein. There's almost certainly one or more access control barriers that have been put in place. The nature of these barriers will differ from company to company, and from database to database, but it's common for administrators to dole out access credentials when necessary. Those credentials might come in the form of a user name and password, or they could require more complex forms of authentication, like biometrics.

When working with database administrators or personnel who have been designated data owners, make sure you provide clear explanations as to why you need access to the data, how the data will be used, and who will need access. This will help the IT team determine how to go about provisioning access so as to minimize risk. They might hook you in directly to the master database if you need up-to-date records throughout the project, or they might give you access to an offline copy of the master database so that there won't be any risk of disruption. Also, to prevent any kind of data leakage, IT personnel might want to limit the flow of data outside the network, so you should also communicate any remote access requirements for when project members are not in the office.

In some cases, the data you're looking for will be controlled by an external party, not necessarily your own organization. If the dataset is released under an open source license, you don't need permission to access or use it. If it's not, you'll need to contact whatever third party is supplying the data and ask for permission. Like with internal data, you'll probably need to communicate the why, how, and who of data access before you're given permission. It's a good idea to get this permission in writing so that there's a clear path of accountability.

Data Governance

Data governance is a concept in which stakeholders ensure that those who govern data in an organization are performing their duties in a way that fulfills the enterprise's strategies and objectives and creates value for the business. Other than evaluating data management performance, data governance seeks to mitigate the risks that are associated with the access and use of data.

As a data science practitioner, you may not be directly in charge of guaranteeing good governance practices in your organization; however, an important element of good governance is proper risk management. The stakeholders that oversee data governance will expect there to be a risk management framework in place that can both keep risk low and mitigate any growing risks. These principles will directly align with business objectives that mandate keeping the enterprise safe from threats and on good legal ground. To assuage the concerns of stakeholders, you should be prepared to communicate how your data science project team members mitigate risks to data.

To do this effectively, you need to be aware of any organizational security policies and legal frameworks that impact how you use data during the project, and make sure that you abide by best practices for security and privacy when handling the data. You also need to ensure that team members are accessing data according to the access control guidelines set forth during the project specification phase, and aren't deviating from them. Your ultimate role in data governance is to ensure that your actions during the project are in compliance with stakeholder requirements.

Guidelines for Initiating a Data Science Project



Note: All Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Follow these guidelines when initiating a data science project.

Initiate a Data Science Project

When initiating a data science project:

- Identify and meet with all stakeholders early and continually in the process.
- Unless you're working on a pure research and exploratory project, identify as soon as possible the questions that need to be answered by the project.

- Determine all sources where data will be coming from and all receivers of the data.
- Keep in mind that communication is just as important as the raw results and analyses.
- Be ready to adapt a project when data changes, becomes unavailable, or is unsuitable to answer the questions asked.
- Understand how long the conclusions will be needed for and how often they will need to be performed. A project that needs to answer a question one time will have different operational and maturity requirements than one that will be performing analyses monthly or weekly, or one that will be generating conclusions continuously based on realtime data.
- Watch out for scope creep in data science projects.
- Be sure that the analyses being done are useful. The data dictates the conclusions that can be made, but ensure that real business questions are being answered.
- Consider how the project will be automated if the analysis is not a one-time event, especially if analyses are required frequently or for a long period of time.
- Consider having a mix of scientists, engineers, and operational talent on long-running projects.

ACTIVITY 1–1

Initiating a Data Science Project

Scenario

You work for Rudison Technologies, an organization that provides consultation services for a wide variety of technologies and disciplines, including data science. Rudison is headquartered near Greene City National Bank (GCNB), which provides various financial services to local residents. GCNB wants to replicate the success that other financial institutions have had with a data-driven approach to business operations. Because this kind of work is new to GCNB, they've turned to Rudison for help. You'll work with members of the project team and other stakeholders to generate ideas and raise concerns that could influence the design of the project.

1. So GCNB is clear about what they're asking for, what is data science?
2. What stakeholders might be involved in the project, and how might each type of stakeholder contribute?
3. GCNB wants to improve its ability to market its banking services to customers. This will hopefully keep customers better informed about the bank's services, leading to higher adoption rates for those services, thereby increasing revenue in various ways. This is a good start, but Rudison wants to encourage a design thinking approach to the project so that it has a clearer picture of GCNB's needs and the needs of their customers.

What are some additional questions that you could ask of GCNB to help generate ideas for the project?

4. Now it's time to discuss the project's scope. GCNB has agreed to play a supervisory role during the project, whereas Rudison will provide the personnel, the resources, and the actual technical work.

What else do both organizations need to discuss with regards to the project's scope before proceeding?

5. The business partners are starting to discuss the project's specifications and objectives. The main objective, as stated before, is to increase customer adoption of the bank's services. There are also several sub-objectives, like producing a system that can determine which customers the bank should target in their marketing campaigns, and which customers it shouldn't bother with.

What might be some key performance indicators (KPIs) for either of these objectives?

6. **What are some potential security concerns that could impact the project?**

TOPIC B

Formulate a Data Science Problem

Now that you have a greater understanding of the project's goals and design principles, you can start thinking about how to approach the project using the power of data science. In this topic, you'll contextualize your work as a potential solution to a data-driven problem.

The Data Science Process

Data science is not just a single task, or even a loose collection of tasks, but a repeatable process. Following this process is crucial for developing efficient and effective data science projects, and will keep you focused and goal oriented while you work. Like any other process, there are several tasks involved, and the tasks flow into each other so that there is a natural progression.

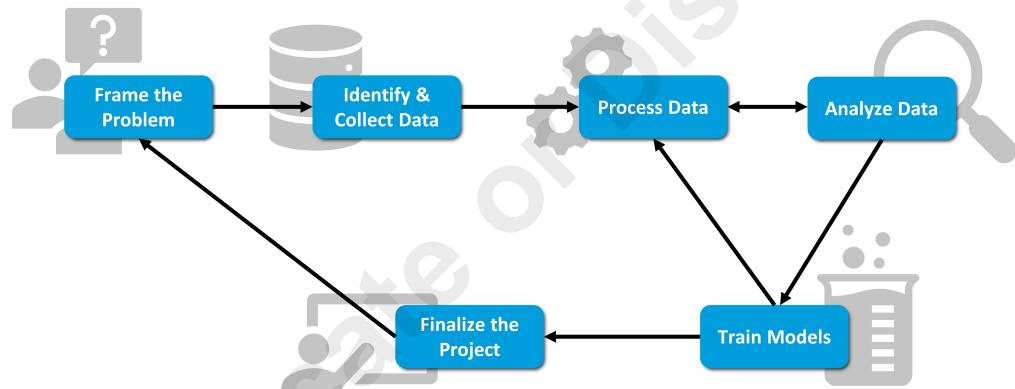


Figure 1–2: The data science process.

The general steps of the data science process are as follows:

- **Frame the problem.** You start a project by identifying some kind of business issue, concern, or question that needs to be answered. You then frame this business need as a problem to be solved. If you determine that practicing data science is the best way to solve the problem, you can continue on.
- **Identify and collect data.** Your next step is to actually discover the data that will be most relevant to solving the data science problem. You'll also need access to that data if you don't already have it.
- **Process the data.** Once you have access to the data, you need to mold the data into a useful shape. This will make the data easier to work with and more conducive to further analysis and modeling. The extract, transform, and load (ETL) process is of primary importance in this phase.
- **Analyze the data.** Now that the data is in better shape, you can perform exploratory analysis on the data. The goal here is to reveal insights that might drive business decisions, as well as influence later model development. The results from this phase will also likely prompt you to perform additional processing to make the data even more useful for your purposes.
- **Train models.** In many cases, a major goal of a data science project is to create a model based on data. This model can make predictions or draw other conclusions that will hopefully improve one or more aspects of the business.

- **Finalize the project.** Lastly, you'll need to present your model and other relevant results to stakeholders so that they can make informed decisions. If the stakeholders accept your results and agree to your proposed solution(s), you'll need to push those solutions into production.

Keep in mind that this process is flexible depending on your business needs. In your own projects, you may find that you need to return to certain tasks multiple times to make meaningful progress. Or, you might need to skip certain tasks that are unnecessary or have already been performed. Also, the process can be cyclical and doesn't necessarily end. After you put a model into production, you might encounter a new business need or collect a new set of data from which to start the process over again.

Ultimately, you need to determine what overall process is best for you and your organization.

The Data Science Skillset

Certain aspects of data science projects tend to make them different from traditional IT or software development projects. In business, they tend to depend deeply on the quantity, quality, and type of data that is available to inform a particular business issue. Often, the motivation for a data science project is based on the idea that data already coming in to an organization might be useful in addressing specific business issues or generally improving results. So, someone charged with turning that data into actionable results—a data science practitioner—needs to have a deep understanding of their own data environment, as well as external data environments that might also play a role in the project.

The data science practitioner also needs to possess several key skills in order to be effective. After all, data science is not just a matter of handling data, but of using that data in beneficial ways.

Although data science practitioners can have a variety of roles, responsibilities, and experience levels, there are some skills that are common to most in the industry:

- **Statistics and probability.** It's very important that you have at least a basic understanding of statistical concepts and probability, as these underpin the core ideas behind analyzing data and leveraging machine learning techniques.
- **Programming.** Data science practitioners typically don't program software to the same depth as a traditional developer, but it's almost impossible to avoid writing code at all. Most robust data science tools are leveraged through programming languages like Python and R. You don't need to be a master at programming, but you need to be proficient.
- **Data wrangling.** This is the skill of transforming data into more usable forms, and it's one of the largest parts of a data science practitioner's day-to-day job.
- **Database querying.** Much of a business's data will be stored in one or more databases. You need to be able to retrieve and update data in these databases using technologies like SQL.
- **Data visualization and communication.** You need to be able to communicate your findings to other people, not just understand them yourself. Telling stories through data visualization is especially important in getting your work accepted by those that make the decisions for the business.
- **Machine learning.** Machine learning is not always a requirement of a data science project, but it's often a major goal. You need to understand the basic theory behind machine learning algorithms, as well as the process of training, tuning, and evaluating machine learning models.
- **Critical thinking.** Gaining useful insights from your data is easier said than done in a lot of cases. You need to be able to spot patterns while also avoiding making assumptions that lead you astray. Intuition also plays a big role in knowing what to look for.
- **Teamwork.** Data science practitioners usually don't work in isolation. You're likely part of a larger team, so you need to be able to work well with your fellow team members. Information should be shared openly so all team members are on the same page.

Shifting Skillsets

In many organizations, the lines between traditional job roles (such as the merging of development and operations in a **DevOps** approach) have blurred. In some cases, there may be proficient software developers who also happen to be skilled data science practitioners. But even in organizations where people wear many different hats, it is important to recognize the shift in skillsets that occurs at different points in a data science project, and make sure that people with the right skillsets are involved at the right time.

Some problems may require hardware you don't have, and you may have to set up new systems based on hardware configurations that are new to you, such as GPUs or computer clusters. Setting up, configuring, and managing these systems requires knowledge and skills beyond those of the traditional IT skillset. In some cases, you may find it makes more sense to provision the systems you need through cloud services, which introduces yet another skillset into the mix. So, the data science process does not always mandate the same skills every single time. The requirements of the project, as well as many other factors (e.g., available resources), will have an effect on what skills are necessary for accomplishing the objectives.

All of this is to say that you (or other team members) may be called on to learn new skills that benefit the project. This suggests another set of skills you need to possess: being able to adapt to new circumstances, and a willingness to expand your abilities.

Problem Formulation

Problem formulation is the process of identifying an issue that should be addressed, and putting that issue in terms that are understandable and actionable. Problem formulation is often a key component of the early project management phases, as it can dictate the actual impetus for a project as well as its fundamental requirements. However, it can also involve a level of depth that is only really appropriate until after the project is underway, when more detailed information is available.

To make a problem "understandable" is to ensure it can be easily defined in terms of business needs. To make a problem "actionable" is to give high-level direction as to how the project members can approach solving the problem. Most data science projects will benefit greatly from some kind of problem formulation. The following outlines a general approach to problem formulation that you might use.

Frame the problem.

Write down a description of the problem you intend to solve, written clearly as though you planned to hand it off to someone else for further development. The process of having to put it into words will help you think through and clarify your intentions.

For example, you might frame a problem as: "The organization has collected demographic data about our customers, but there are no current processes in place to leverage that data in ways that could increase revenue or brand recognition."

Identify why the problem must be solved.

Aspects of this include:

- **Rationale**—Identifying why you need to solve this problem will help you evaluate the result, make appropriate compromises as you produce the solution, and determine when you have accomplished what you set out to do.
- **Benefits**—Identify how the organization will benefit from the solution. This information will be useful if you need to get buy-in from stakeholders, or sell others on the need for additional resources or time to get the job done. It will also be useful to refer to when you try to determine how you will evaluate your success.
- **Lifetime and use**—Identify who will use the solution and how long it will be used. This will help you determine how "polished" the solution must be, how you will need to support it, and help you consider deployment requirements if there will be other users.

Being clear about these things will help to ensure you solve the right problem, in a way that will actually produce the benefits you were seeking in the first place. For example, you might say: "The organization will likely lose its competitive edge in the marketplace if it does not leverage its customer data. By leveraging this data, the organization will be better equipped to target its marketing efforts at certain groups and individuals, as well as be more informed about what products and services are worth developing and which are not. Both of these can lead to improved sales and greater business efficiency. The solution may be continually used and updated by internal team members as necessary, so that the customer information and any insights gleaned from it will be current."

Provide background information that will help solve the problem.

As a reminder to yourself if you're developing the solution, or to provide guidance to others who will be working with you or for you, as you think through the problem, document such things as:

- **Assumptions**—A list of assumptions about the problem, such as:
 - Sources of data that are acceptable or unacceptable to use.
 - Special operating requirements, such as software environments you must use.
 - Business context, such as project timelines, who will use the solution, and so forth.
- **Reference problems**—A list of similar problems you have solved through applied data science. This will inform the approach you take toward implementing a solution.

Determine whether the problem is appropriate for data science.

Once you have clearly defined the problem, you can explore whether it is more appropriate to be solved as a data science problem or as a traditional programming problem. While data science may always seem like the more attractive option, it is not without its risk. It can be difficult to go through the entire process effectively, especially if the quality or quantity of data is insufficient. The process can also become expensive and time consuming, potentially to a degree that it outweighs its benefits.

So, you should always ask yourself and your colleagues if the solution to the problem can be developed using simpler methods and strategies. Data science in the business world must serve the business first and foremost, and if it cannot be justified for a particular scenario, then it shouldn't be chosen.

Common Issues Addressed by Data Science

The following table lists some common business scenarios that data science is used to address. Despite the term "problem" formulation, these aren't necessarily problems in the sense that the business will suffer greatly without a solution. They might just be circumstances that the business can take advantage of to improve its profitability, market viability, impact on communities and society at large, and so on.

Category	Business Issue
Staff and facility	<ul style="list-style-type: none"> • Predict where and when office and call center staff, crews, fleet vehicles, and equipment are most likely to be needed to maximize use and productivity. • Identify product inventory that can be reduced to lower inventory carry costs. • Process and interpret incoming help desk calls and accurately route them to the right person for problem resolution.

Category	Business Issue
Sales and customer experience	<ul style="list-style-type: none"> • Make the most effective use of marketing and sales by focusing contact based on a customer's interests, browsing, and buying history. • Recommend additional/other products based on those the customer has already shown interest in. • Provide "smart" software tools to guide staff on calls with customers to enable them to provide better service and support.
Finance	<ul style="list-style-type: none"> • Process loan applications, contracts, and other documents, providing recommendations such as loan approval, contract terms that should be reviewed, and so forth. • Price financial portfolios in overly aggressive markets by discovering latent features from the data in portfolios. • Identify risk within a portfolio based on current market trends.
Security	<ul style="list-style-type: none"> • Detect and prevent intrusions on computer networks. • Sift through transactional data (call detail records, point of sale transactions, trading activity, and so forth) to identify specific activities that may be fraudulent. • Use realtime security camera analysis to track the movement of people within a building to perform surveillance, identifying unusual traffic patterns or behavior, such as violent behavior, vandalism, and unaccompanied suspicious packages.
Visual interaction	<ul style="list-style-type: none"> • Support augmented reality apps that enable users to point their mobile device camera at an object or location and view overlays with product information, pricing, coupons, guided tour narration, and so forth. • Provide a virtual "try it out" feature for online shopping, where the customer can see what they will look like when wearing the product, see what the product will look like in their home, and so forth. • Deliver visual presentations to customers, tracking their eye movement, facial expressions, and other behavior to adjust the presentation accordingly. Frequent customers can be identified using facial recognition to refer to their past shopping history and preferences.
Process guidance and navigation	<ul style="list-style-type: none"> • Provide visual overlays in a surgeon's field of view during a medical procedure, highlighting anatomy, the location of underlying bones, nerves, blood vessels, organs, and so forth. • Use robotic delivery carts to transport mail, packages, and materials throughout a lab facility, using computer vision to avoid obstacles. • Guide farm equipment through the optimal route when harvesting crops.

Category	Business Issue
Automated inspection	<ul style="list-style-type: none"> Scan cast metal parts automatically as they emerge from an injection mold, and eject flawed parts into a rework bin. Inspect glass bottles in a bottling plant for cracks, and missing, misplaced, or incorrect labels, as they pass down an assembly line. Use flying drones to scan properties for forest fires, missing persons, wildlife counts, and so forth.

Modeling Data

For the issue of identifying glass bottles with defects on an assembly line, the ultimate solution might be something like a physical camera that takes pictures of each bottle as it moves down the assembly line, with an accompanying piece of software that scans the image, looks for defects, and raises an alert if it detects any. As a data science practitioner, you may or may not have a hand in building the final application of your work. What you *are* likely to work on directly is the mechanism that actually makes the decision, "Does this bottle have defects or does it not?" This mechanism is called a model.

A **statistical model** is a mathematical system that generates assumptions about data. It uses statistical methods and probabilities to map the relationships between variables in the data. A statistical model is often used to reach conclusions about a wider population of data based on an existing sample of data that is provided to the model. Consider the example of a dataset that includes the ages and heights of a few hundred people. You could create a statistical model to observe the general pattern of age as it relates to height in your dataset (the sample), and then extrapolate that information to an entire country (the population). The model can answer questions like, "At what age do Norwegian citizens usually stop growing in height?" or, "What is the typical height of a U.S. citizen between the ages of 18 and 30?" The amount and types of questions that can be answered by a model are dependent on the data and your own curiosity.

It is common to think of statistical models as making predictions, but not all conclusions a model can draw are predictive in the strictest sense. It's more precise to refer to the output of a model as an *estimation*. For example, the model might predict what height someone will be when they are older. The model could also make a non-predictive estimation, like what height someone is right now if only given their age and nationality. The model could also make proactive decisions, like suggesting a diet and exercise regimen to someone based on their height and age.

Statistical models are driven by data. This is why there needs to be an entire data science process to begin with—without the right data in the right form, the model will not be able to do its part in addressing your unique business issues.

Data Science Outcomes

The models produced by the data science process can fit into one of several categories. Each category describes a particular task that the model performs. You can also think of these tasks as outcomes, since the model is applied to a real-world issue in order to return some kind of result. Dividing models into these outcomes is not just an academic exercise—it will have a significant impact on the approach you use to develop the model, especially when it comes to machine learning.

Outcome	Description
Classification	<p><i>Given a new instance of data, identify the class it belongs in.</i></p> <p>The goal of a classification model may be to simply place the data instance into a pre-defined class or category. For example, given various attributes of a plant, return a value of 1 (poisonous) or 0 (not poisonous). Or the goal may be to estimate the probability that the observation belongs to various classes—75% likelihood of being poisonous, 25% likelihood of not being poisonous.</p>
Regression	<p><i>Given a new instance of data, estimate the value it has for a numeric variable.</i></p> <p>Regression models are used to determine the relationships between variables, generating an outcome that is numeric (rather than categorical, like in classification). For example, based on patterns inherent in already measured data, a regression model might estimate the market value for a collectible item, like a trading card. Or a regression model might be used to make a prediction. For example, given various demographic factors, return a value that predicts the population of a city 10 years from now.</p>
Clustering	<p><i>Without any prior knowledge of a dataset's structure, identify components that belong grouped together.</i></p> <p>Clustering models are similar to classification models in that they can categorize data. But, clustering is used when the input data does not already have pre-defined classes. For example, you may want to segment your customers into groups for targeted marketing purposes, but you don't necessarily know what those groups are. Clustering enables you to discover those groups, as well as place a new data instance into one of those groups.</p>

Learning Modes

In the realm of machine learning, the outcomes described in the previous table are actually subsets of broader *learning modes*. These learning modes describe *how* a model learns from the data, and they have an effect on the content and form of the data itself. There are two primary learning modes:

- **Supervised learning.** In this mode, you create the model by providing a dataset for which you already have a set of correct answers, called **labels**. Labels are also referred to as the "ground truth." The model learns by analyzing patterns within each record, and comparing them to the correct answers within the label set. In this way, the model learns by example what patterns result in a particular answer. This approach is typically used where a labeled dataset is available, and the model must make estimations based on patterns in the data or must correctly place items into an appropriate category. Regression and classification are supervised learning methods.
- **Unsupervised learning.** In this mode, a set of answers (labels) is not provided. Instead, the model learns by looking for distinct patterns within the data and making observations. The model might figure out how to organize related or similar items into groups. Clustering, therefore, is an unsupervised learning method.

There are also less prominent learning modes like *reinforcement learning*, which is used to train robots to interact with a physical environment, and *semi-supervised learning*, which leverages both labeled and unlabeled datasets at the same time.

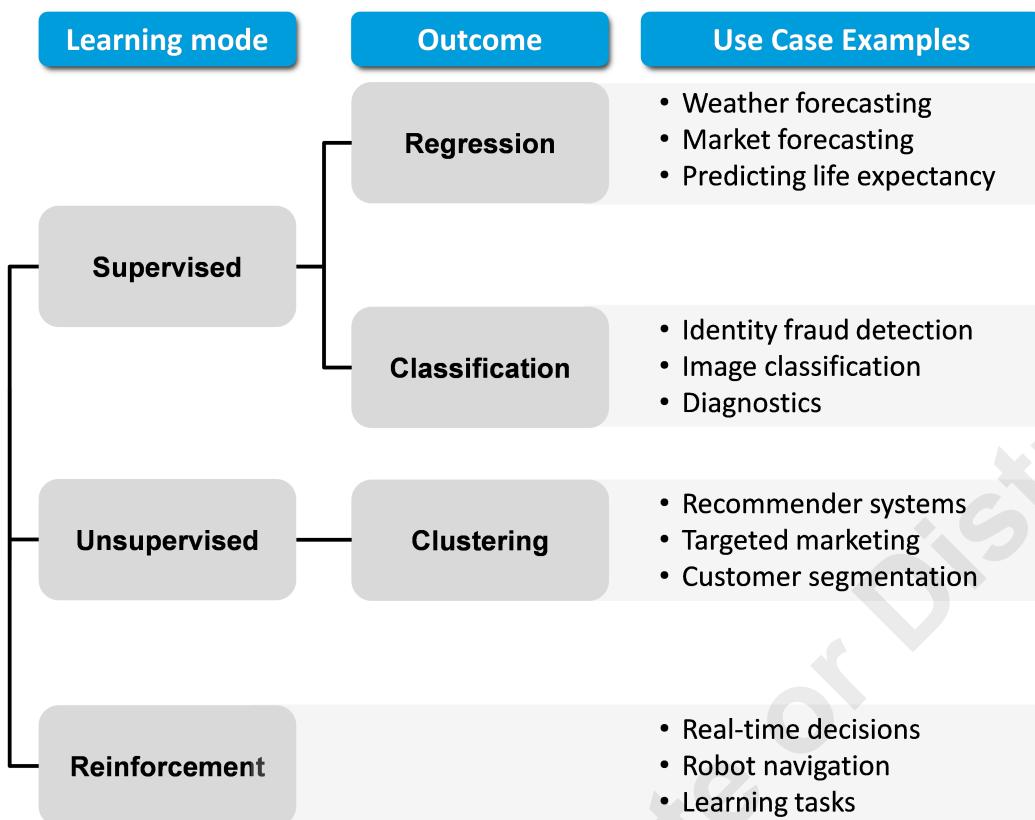


Figure 1–3: Learning modes mapped to outcomes and use cases.

Randomness and Uncertainty

Remember that a statistical model makes *estimations*. That word implies a sense of uncertainty. That's because, by their very nature, statistical models are non-deterministic. You don't know every single person's exact height and age, whether past, present, or future. The model therefore can't answer your questions with complete confidence. It can't say, "Person X is definitely 75 inches tall" if it doesn't actually have that data. Instead, statistical models can be said to model *beliefs* or *assumptions*. If the model were a person, it might say, "I believe with a high degree of confidence that Person X is above 70 inches tall." This is because statistical models are based on probability rather than definite truth.

Probability is related to the concept of randomness. Randomness suggests a lack of identifiable pattern. When it comes to data, there are typically various aspects of randomness to that data—which data points are sampled, the order in which they are sampled, the samples that are used for creating and testing the model, and so on. There is also randomness in the algorithms and other mathematical formulas that create statistical models. Two different algorithms may produce similar outcomes, but they may also produce slightly different results simply because they take different steps to obtain those results.

So, statistical models are often described using the term *stochastic*. With stochastic modeling, individual data samples are inherently random and can't be perfectly determined. But taken together, the entire set of data can be shown to follow a general pattern. By analyzing the general patterns established by the entire set, the model can make reasonably good decisions about individual data samples. This is why, despite the element of randomness in statistical models, they are still useful for many tasks and can even exhibit a degree of artificial intelligence (AI).

There are some techniques that can help you mitigate the effects of randomness in statistical models. At the moment, you just need to be aware that data science does not offer guarantees or promises when it comes to solving problems—only estimations. Still, if properly devised, these estimations can be extremely powerful.

Guidelines for Formulating a Data Science Problem

Follow these guidelines when formulating a data science problem.

Formulate a Data Science Problem

When formulating a data science problem:

- **Describe the problem in plain language.** Having to actually write down the problem in simple, direct words requires you to think through what you really need to accomplish.
- **Identify the ideal outcome.** Regardless of the technical solution you come up with, identify what *business* outcome you are trying to achieve.
- **Identify where the data will come from.** You must be able to obtain historic data representing the outputs you want to be able to estimate. Although some manipulation of existing data is possible, if you simply can't obtain the right data, you may need to reformulate your data science problem to use data that you *can* obtain.
- **Determine when and how the inputs and outputs will be used.** Practical considerations of when input data is available and when output data must be produced may have an impact on the design of your solution. For example, input data that your solution depends on may be released in batches, once a month. If a solution based on months-old data will not suffice, then you'll need to reconsider the solution.
- **Identify ways the problem might be solved without data science.** Brainstorming ways to solve the problem without using data science will help you focus on the business logic and may lead you to simpler, more elegant solutions, even if you do end up using a data science solution.
- **Frame the business issue as a data science problem.** From a business perspective, you have already identified the type of outcome you need to produce. Now consider how to express that outcome in terms of various types of outcomes a statistical model might produce (regression, classification, clustering, and so forth). While you may eventually end up using an advanced model, simpler models are easier to think through and implement, so initially try to see if you can find a simple way to cast the problem, such as "predict the ideal sale price for a new product given various factors" (regression) or "recommend new products that a customer might be interested in purchasing" (classification or clustering).

ACTIVITY 1–2

Formulating a Data Science Problem

Scenario

Both organizations believe they have a good plan for moving forward. Rudison will develop multiple systems that internal employees can use to make decisions about how best to market their banking services to existing and prospective customers. These systems will be incorporated into a larger software platform that can provide realtime insights into the bank's business operations. The partners have settled on a six-month timeline for the project, where Rudison will provide deliverables to GCNB at the end of each month. These deliverables will be reports of the team's progress at that point in time, as well as its outlook for the following month. The final deliverable will be the overall business insights platform. Before you can begin doing hands-on work, however, you need to formulate the problem(s) your systems will try to solve.

1. The first step of the problem formulation process is to frame the problem.

How might you frame the problem that the project is trying to solve?

2. The next step is to identify why the problem must be solved.

What are the components that go into identifying the "why" of the problem, and how might you address them for this project?

3. The third step in problem formulation is to obtain background information that will help solve the problem.

What background information does GCNB need to provide, or that you need to research yourself, to help you develop the solution?

4. The last step is to determine whether the problem is actually appropriately solved using data science. The team has agreed that data science is necessary to achieve GCNB's business goals.

Why not just leverage data science anyway, even if the problem can be solved without it?

5. The team starts thinking about the data science outcomes it wants to achieve through the development of statistical models. You know that the data you're likely to work with includes information about customer demographics like age, level of education, career, etc., as well as information specific to their banking activity, including their contact history with GCNB's marketing campaigns, their loan status, what devices they use to do their banking, how many transactions they've made, how much money is in their account, etc.

Given what you know so far, what are some potential ways that this data might be used to achieve outcomes in classification?

6. What are some potential ways this data might be used to achieve outcomes in regression?
 7. What are some potential ways this data might be used to achieve outcomes in clustering?
 8. Why is it important to recognize the role of randomness and uncertainty in data science projects?
-

Summary

In this lesson, you kicked off a data science project by identifying key phases and stakeholders involved in the project. You also reviewed several major factors that go into a data science project, like stakeholder requirements, data accessibility, and security concerns. You also formulated a data science problem in order to begin addressing business needs through a data-centric approach. By undertaking these preliminary tasks, you are much better prepared to conduct the project from start to finish.

What stakeholder requirements are part of the projects you've worked on, or might occur in your organization?

What data science outcomes (classification, regression, clustering, etc.) might be most applicable to your projects?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

2

Extracting, Transforming, and Loading Data

Lesson Time: 4 hours, 45 minutes

Lesson Introduction

The first truly hands-on technical phase of the data science process is actually a combination of related tasks known as extract, transform, and load (ETL). This is where you, the data science practitioner, start to mold and shape the data so that it can be as useful as possible for the later steps in the data science process. In this lesson, you'll go through each ETL task in order.

Lesson Objectives

In this lesson, you will:

- Collect data from multiple sources so it is available to be transformed and cleaned.
- Prepare and clean data so that it can later be loaded into its ultimate destination.
- Load data into its ultimate destination so that it can be analyzed and modeled.

TOPIC A

Extract Data

The first step in the ETL process is extraction. You'll begin putting your data science skills into practice by retrieving and consolidating data.

Datasets

In a data science project, a **dataset** is a collection of data that will be directly used to accomplish the business goals set forth in the project specifications. The "collection" aspect is important, because data must be packaged as a discrete object before it can be applied to most data science tasks. Data does not always start out this way, however; it may come to you in looser form, with individual pieces spanning multiple repositories, or it may be mixed in with other data that isn't relevant to the problem at hand. In either case, it'll be your job to ensure that the data is placed in one or more sets where each set contains data points that are similar to each other.

For example, you might have access to a data repository that includes information about internal salespeople, like how much they've sold, how much commission they've made, and so on. You also have access to multiple repositories that each include information about customer transactions. Some customer data is stored in Database A, some in Database B, and so on—and the way in which those customers are described may be different as well. Perhaps Database A includes demographics, like a customer's age, whereas Database B includes the actual transaction amount. The salesperson data can more readily be called a dataset because data describing the same concept is already in one place. However, the customer transaction data is spread out among many different sources, and takes many different forms, even though it all relates to the same idea. If you were to work with the data as is, you'd have a tough time trying to apply that idea consistently. If the data were placed into one or more sets, however, it'd be much more usable.

There's really no limit to what a dataset can include. Datasets can encompass many different concepts from many different fields. While prototypical datasets like a customer transaction set or a salesperson set are useful as examples, you shouldn't feel constrained by them. Your dataset may take the form of something that is completely unique to your industry or even your organization. What's important is that you've collected data that can help achieve the project's goals.

Structure of Data

Datasets may be structured or unstructured. **Structured data** is in a format that facilitates searching, filtering, or extracting data—such as a spreadsheet or database, in which data categories are separated and/or labeled. Specific chunks of data (such as height, age, first name, last name) can be easily retrieved for any record using programming code or a querying language such as Structured Query Language (SQL).

Unstructured data, on the other hand, is not as easy to query. Examples include such things as images, video, or audio files, data posted on social media sites, the content of email documents, and so forth. Information in such formats is not necessarily recorded in neat, pre-defined containers like a spreadsheet or database. Nonetheless, unstructured data is often a very important source of information in data science, and often represents a much larger portion of a business's data than structured data.

Some data may be considered **semi-structured data**. For example, while the content of email data might be unstructured, the email documents themselves do contain some structure. Fields associated with the sender, recipient, send date, and so forth provide structured data that can be directly searched, filtered, and extracted.

Some data formats (such as XML and JSON documents) may be treated as either structured or unstructured, depending on the consistency of their structure and content. In some situations, such as logging output from a server or application, these documents might have a very consistent structure, while in other cases, such as human-authored documents, they may be very inconsistent.

Terms Describing Portions of Data

Data sources such as databases, spreadsheets, and file formats such as CSV (comma-separated values) typically organize data within columns and rows. Columns may be called fields, and rows may be called records. The data stored within the intersection of a column and row may be called a value. In data science, there are additional names for these entities, and how they are used depends on the context.

There are many alternative names for each row/record. Common ones include data example, data instance, data observation, and data point (especially when graphed). A statistical model considers these "examples" of some aspect(s) of an environment when it takes them as input.

Those aspects of an environment are the columns, which are called **attributes** or **features**. Features contain the variables the model evaluates to make its estimations. For example, in a dataset you're using to determine whether or not someone should be accepted for a bank loan, features the model uses to make this decision might include whether or not the applicant has ever defaulted on a loan (`default`), and how long the applicant has been a customer at the bank (`date_joined`). These columns are said to be among the features the model uses. The total number of different features it uses are counted to identify the model's **dimensions**.

Sometimes you may refer to the individual data value held in a feature. For example, `age = 5` describes the age feature as having a particular value (5) for this data example.

Occasionally, the word *feature* is used to refer to this specific combination of variable plus value. In most cases, however, practitioners use the word *feature* to describe the column/variable itself, and just use the term *value* for the specific measurement.

The diagram shows a screenshot of a spreadsheet application with a table titled "users". The table has columns: user_id, age, job, marital, education, default, housing, loan, and con. The "age" column is highlighted with a blue box labeled "Feature". The value "58" in the first row of the "age" column is highlighted with a blue box labeled "Value". A blue box labeled "Data example" points to the first row of the table. The table contains 16 rows of data.

	user_id	age	job	marital	education	default	housing	loan	con
1	9231c446-cb16-4b2b...	58	management	married	tertiary	no	yes	no	NULL
2	bb92765a-08de-496...	44	technician	single	secondary	no	yes	no	NULL
3	573de577-49ef-42b9...	33	entrepreneur	married	secondary	no	yes	yes	NULL
4	d6b66b9d-7c8f-4257...	47	blue-collar	married	NULL	no	yes	no	NULL
5	fade0b20-7594-4d9a...	33	NULL	single	NULL	no	no	no	NULL
6	c6aee0d4-2a86-4bac...	35	management	married	tertiary	no	yes	no	NULL
7	1fa7d4fb-3e4a-463a...	28	management	single	tertiary	no	yes	yes	NULL
8	d20059f3-84b7-4ec5...	42	entrepreneur	divorced	tertiary	yes	yes	no	NULL
9	0cedabc3-6141-43c6...	58	retired	married	primary	no	yes	NULL	NULL
10	bc3d8e25-4619-4395...	43	technician	single	secondary	no	yes	no	NULL
11	59b30e4f-753c-47e8...	41	admin.	divorced	secondary	no	yes	no	NULL
12	cd92181a-0e26-4a72...	29	admin.	single	secondary	no	yes	no	NULL
13	f6f04cfb...	53	technician	married	secondary	no	yes	no	NULL
14	8c406417-19a9-4c6c...	58	technician	married	NULL	no	yes	no	NULL
15	139c01b6...	57	services	married	secondary	no	yes	no	NULL
16	cf24cb61...	51	retired	married	primary	no	yes	no	NULL

Figure 2-1: The different portions of data.

Data Quantity

There are several factors that affect how a dataset is both understood and used. One of those factors is the quantity of data. "Quantity" in this case can refer to either the number of data examples, the number of features, or both. So a high-quantity structured dataset can have many rows, many columns, or many rows and columns. A high-quantity unstructured dataset, like a collection of written documents, may include many total words and many different types of words. It's important to make these distinctions because different types of quantity are relevant to different types of problems. For example, some machine learning algorithms perform better with one type over another.

Generally, the more informative data you have (i.e., the more meaningful features), the better the model it will produce. If a dataset has only 2 features for each customer, it might be difficult for the model to understand how those variables relate, and it may fail to find meaningful differences between each of the customers. On the flip side, you could have 80 columns describing your customers in a meaningful way, but if you only had 10 customers on record, your model may not have enough examples to make a proper estimation. Having a large number of examples can help to minimize the influence of a few bad data points.

It's not always feasible for you to have a high-quantity dataset, so you may need to compromise. One rule of thumb calls for having at least 10 times as many records as the number of features that the model is using. Of course, more than that would be even better.

Big Data

No discussion of data quantity is complete without talking about big data. **Big data** refers to massive quantities of data that cannot easily be translated into actionable intelligence using traditional methods and technologies. New sources of business data, such as the Internet of Things (IoT), web and mobile apps, and social media platforms, may dwarf the amounts of data coming from traditional business data sources, such as databases, business transactions, and so forth.

This data scales to become *big* due to at least three major factors, which are sometimes described by three Vs—volume, variety, and velocity.

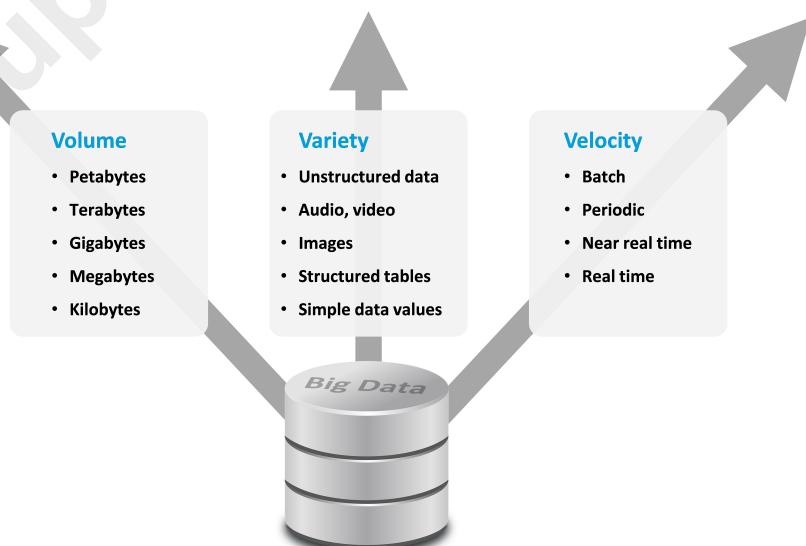


Figure 2-2: The three Vs of big data.

Factors of Big Data	Description
Volume	The sheer number of bytes of data being captured is one of the dimensions of big data. Organizations have taken advantage of the scalability offered by cloud storage, and have developed applications that accumulate massive amounts of data storage.
Variety	In the past, much of the data captured for use in data analysis came from structured data sources, like databases and transaction records. More commonly, complex and unstructured types of data are captured from sources such as e-commerce websites, social media sites, and IoT sensors. A wide variety of data types are captured, such as images, voice recordings and other audio, video, web logs, and social media posts.
Velocity	The speed at which data processing must occur. With data sources such as IoT sensors, data must often be collected and processed in real time or near real time.



Note: Veracity (the accuracy or reliability of data) is often included as a fourth V of big data's challenges. This refers to the fact that many sources of big data may not be completely reliable.

Data Sources

Various terms are used to refer to the various types of repositories where data is collected. While these terms are intended to represent different concepts, they are sometimes used interchangeably.

Data Repository	Description
Data lake	<ul style="list-style-type: none"> Purpose: Machine learning, big data analytics, predictive analytics, and data discovery. Data may be used at any time or never at all. A specific purpose for keeping the data may not yet exist, but it is kept for possible future needs. Source: Structured and unstructured data from a variety of sources such as sensors, websites, business apps, mobile apps, server logs, and so forth. Structure: Widely varying. Data is typically kept in its original forms, which may include non-traditional data types such as web server logs, sensor data, social network activity, text, and images. Consuming and storing such data can be expensive and difficult.
Operational data store	<ul style="list-style-type: none"> Purpose: Collects, aggregates, and prepares data for use in operations. May feed into a data warehouse. Source: Transactional data captured from multiple enterprise applications and other sources. Structure: Data has been structured for fast and easy access, but may require additional preparation before it can be transferred to a data warehouse.
Data warehouse	<ul style="list-style-type: none"> Purpose: Business intelligence, batch reporting, data visualization. Source: Relational data captured from multiple, relational sources including applications, transactional systems, operations databases. Structure: Data has been structured for fast and easy access.

Data Repository	Description
Data mart	<ul style="list-style-type: none"> Purpose: Data used by a particular department or business function to support analytics. Source: A subsection of the data warehouse, housing data specifically intended to support a particular department or business function. Structure: Data has been structured for fast and easy access.

Note that most of these sources incorporate relational databases. Relational databases are not so much a separate source as they are a technique of storing data, typically in an organized, consistent, and normalized format. Such databases usually support applications directly, like applications that process transactions.

Third-Party Data

Even if most of your data comes from internal sources, you may find that your datasets aren't as complete as you'd like them to be. For example, you may have plenty of information about customers in terms of how they interact with the business—what they buy, how they spend, and so on. But these aren't the only factors that could be useful to you. For instance, if you had more information about customer demographics, like what regions they're from, their ages, genders, etc., you might have a more well-rounded dataset from which to build a model. If you can't get this data yourself, you may need to look elsewhere. Enter third-party data.

Some people or organizations are in the business of providing data collection services to their clients. Those clients may rely entirely on the third party for all of the data used in a project, or just a portion of their overall data, like in the previous example. Even if you're at the point where you're not sure which of these situations you're in—or you don't think you'll need to rely on external data at all—it's still a good idea to research third-party data providers. After all, you may discover a type of data that you never thought of. Or, you might discover greater volumes of data that you already have, which you could leverage to improve your project.

Data providers can provide just about any type of data, but some types are more commonly sought after than others. This can be due to the difficulty of collecting such data, or due to the fact that some providers have a positive reputation in their chosen field. Demographic data, particularly among large populations like cities, countries, or even the world, can be hard to come by due to the sheer amount of effort involved in surveying such a large population. Government census bureaus, like the U.S. Census Bureau, are perhaps the most prominent providers of demographic information. Another example of data that organizations look externally for is market data. For example, financial company Bloomberg offers various services that can help an organization collect and track market data in specific sectors. News publications like the Wall Street Journal and Reuters also collect market data.

Open Datasets

Not all external data carries a price. Some data is provided to the public for free. That's free as in price, and free as in freedom. These are called *open datasets*, and they are based on the principle of free and open source software (FOSS). Researchers make open datasets available to anyone to use and republish as they wish—without restrictions for copyright, patents, or other control mechanisms. They typically share these datasets through open source data sharing sites. Hundreds of such sites are accessible over the web. The following table lists some prominent examples.

Item	Description
University of California Irvine Machine Learning Repository	The University of California Irvine School of Information and Computer Science hosts this repository that includes hundreds of datasets, many of which are cleaned and ready to be used. Datasets are classified by the type of machine learning problem they are meant to address. https://archive.ics.uci.edu/ml/index.php
Kaggle	Google hosts this online repository and community of data science practitioners. Each dataset functions as a community site where you can discuss data, find and share public code, create your own projects in notebooks, and read and post blog articles. In Kaggle competitions, which regularly attract more than a thousand teams and individual competitors, companies post problems and practitioners compete to build the best model. Researchers have published papers in peer-reviewed journals based on their performance in Kaggle competitions. https://www.kaggle.com/datasets
Registry of Open Data on AWS	Amazon Web Services (AWS) hosts a registry of open datasets covering many different fields such as satellite imagery, web crawler data, public transportation, bird migrations, and so forth. A search feature is provided to help you find the dataset you are looking for. The directory provides detailed dataset descriptions and project examples. This site is particularly convenient if you are using AWS; data transfer will be very quick since the datasets are hosted on AWS storage services. https://registry.opendata.aws/
Microsoft Research Open Data	Microsoft Research, the research subsidiary of Microsoft, hosts this repository that provides a collection of free datasets covering fields such as natural language processing, computer vision, and domain-specific sciences. This site is particularly convenient if you are using Microsoft Azure cloud services. You can download or copy datasets directly to an Azure cloud-based Data Science Virtual Machine. https://msropendata.com/
Open Media Library (OpenML)	OpenML is an online experiment database for data science and machine learning, which hosts tens of thousands of open source datasets. Resources are categorized as datasets (rows of data in table form), tasks (a dataset, together with an example machine learning task to perform, such as classification or clustering and an evaluation method), flows (a particular machine learning algorithm from a particular library or framework), and runs (a flow applied to a particular task). https://www.openml.org/
Government repositories	Numerous governments share their datasets through sites such as the EU Open Data Portal (https://data.europa.eu/euodp/data/dataset), Data.gov (United States), Data.gov.uk (United Kingdom), Data.gov.in (India), and Open.canada.ca (Canada).

Public APIs

Some open datasets are available for direct download from a website. Some are available through public **application programming interfaces (APIs)**, which facilitate interaction between multiple

software environments. In practical terms this means an API provides a developer with programmatic access to data and functions related to that data through a programming language. Working with data dynamically is easier with an API as opposed to downloading discrete files, or accessing a database directly.

For example, the U.S. government provides APIs that pertain to several of its agencies. The Department of Agriculture has several APIs, including Food Data Central, which supplies nutritional information; the Organic INTEGRITY Database API, which provides data about organic food operations; the PubAg Database, which includes text articles about the agricultural industry; and more. There are many more APIs across various industries, both governmental and non-governmental. The "Big Tech" companies like Google, Amazon, and Microsoft tend to have the most popular APIs for accessing their various services.

Most public APIs are web based, meaning you connect to a service *endpoint* using a web protocol (usually HTTPS). An API might have multiple endpoints as a way of categorizing different types of data or services. After connecting to an endpoint, you issue various requests to pull down whatever specific data you're looking for. The nature and format of these requests differ from API to API, and are usually documented on the provider's official website. Note that, although most public APIs are free of charge to use, they do often come with limits. Some APIs allow you to connect using any number of devices and issue any number of requests, but it's more common for the provider to issue API keys to anyone interested in using their services. These keys essentially identify and authenticate you to the service so that your usage of the service can be tracked. Most services will therefore limit the number of connections and/or requests you can issue over a certain time period. Signing up for a public key is almost always free, but you should still consult the API's documentation so that you're aware of any restrictions.



Note: Web-based APIs are also called RESTful APIs, referring to the dominant standard for interacting with web services: representational state transfer (REST).

Public API Example

Here's an example of both connecting to the Food Data Central API and issuing a request using Python:

```
import requests
foods = requests.get('https://api.nal.usda.gov/fdc/v1/foods/list? \
&pageSize=1&api_key=DEMO_KEY')
print(foods.json())
```

This API call issues a GET request to the /foods/list endpoint. That endpoint specifically returns a paged list of foods and their attributes in an abridged form. The pageSize parameter specifies the number of pages to return, and api_key is where you'd supply your own unique key. This example uses a demo key, which is much more limited.

The API's output format is JavaScript Object Notation (JSON). So, each entry on a "page" will appear something like this (when formatted):

```
[
  {
    'dataType': 'Survey (FNDDS)',
    'description': '100 GRAND Bar',
    'fdcId': 1104067,
    'foodNutrients': [
      {
        'number': '203',
        'name': 'Protein',
        'amount': 2.5,
        'unitName': 'G',
      }
    ],
  },
]
```

```

    'publicationDate': '2020-10-30',
}
]

```

Web Scraping

A lot of useful data on the public Internet isn't provided through APIs, but exists solely in a web-based markup language like HTML or XML. For example, a web page might have a <table> element with various entries, similar to a relational table in a spreadsheet. To access this content, you'll need to use a web scraping library to extract that raw markup from a website. Then, using a web parser, you can extract data from the appropriate fields to put it in a more manageable format.

Extract, Transform, and Load (ETL)

Extract, transform, and load (ETL) is the process of combining data from multiple sources, preparing the data, and loading the result into a destination. It is the first major phase of the data science process that involves hands-on practice with the data. The overall objective of ETL is to essentially take a mess and clean it up. You don't want to go into the modeling phase with data from multiple, disparate sources; nor do you want to model data that is inconsistent and has numerous issues. This will significantly impair your model's performance, slow the project down, and lead to unfulfilled business goals. Even if you think your data is clean, or if it just comes from a single source, you still need to put it through ETL to make sure it's in the best state it can be.

ETL is actually one of the most time-consuming tasks in the overall data science process when it comes to direct human effort. This is especially true if you're working with large volumes of data, including big data. ETL requires a careful and methodical approach to handling data since every dataset and every application of a dataset to a problem is different. There is no one sure-fire way to get your data into the perfect state every time, but there are general best practices that you'll start learning shortly.

From a high level, each step of the ETL process can be described as follows:

- **Extract**—Pulls all or some of the data from various sources, which may have similar or dissimilar data structures.
- **Transform**—Converts data into a proper and uniform storage format or structure.
- **Load**—Inserts data into the destination where it will be stored for later analysis and modeling.

Python and Jupyter Notebook

There are many programming languages and tools that can perform ETL. One of the most commonly used programming languages is Python, and one of the most useful Python-based tools is Jupyter Notebook. Jupyter Notebook is an execution environment that provides the user with a web-based interface for entering, running, outputting, and analyzing code in a configurable document format. Jupyter Notebook can be used in a production environment, but it is more often used for executing quick data science tasks, as well as for teaching data science concepts.

Data Validation

Before you begin extracting data, you should consider doing some upfront data validation to make your job easier when you get to the transform phase. At this point, you don't need to spend time correcting individual errors or small groups of errors—that can come later. However, you can still focus on invalid data from a holistic sense. For example, if you're pulling data from multiple files, one of those files might be mostly or entirely corrupt. All of the data values might be scrambled or just incomprehensible. That's a problem you can easily identify, whether manually or automatically. You would then be able to drop this data source from consideration during the extraction process.

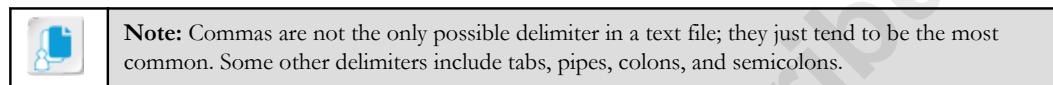
Some issues are a little more subtle, like data that is very out of date. Maybe you're trying to project product sales in the immediate future. If one of your data sources has sales data from 20 years ago,

it's probably not worth adding to the dataset. So if you're able to validate the age of the data, you'll have an easier time culling what you don't want to include.

Quick validation at the front end of the ETL process can save you time and effort down the line.

Comma-Separated Values (CSV) Files

A comma-separated values (CSV) file is a text-based file that holds data, where each value is separated by a comma (,) character. This separating character is also called a *delimiter*. The delimiter helps programs that take the file as input parse the data inside the file, often as a table with rows and columns. By looking at the delimiter, the program can determine what text is a row, what text is a column label, and what text is a value for a specific column.



Since CSV files are text based, they contain just that—text. They are not compiled as an executable or other binary file, so it's possible to open them in any text editor to see their contents. In the following figure, you can see the contents of a CSV file as it appears in a simple text editor.

Time	Humidity	Dew Point	Wind Speed	Temperature
2021-01-27 00:00:00	92	27	12	29
2021-01-27 01:00:00	92	26	10	28
2021-01-27 02:00:00	88	25	10	28
2021-01-27 03:00:00	89	24	9	27
2021-01-27 04:00:00	89	24	8	27
2021-01-27 05:00:00	84	22	9	26
2021-01-27 06:00:00	84	22	12	26
2021-01-27 07:00:00	84	22	19	26
2021-01-27 08:00:00	85	21	10	25
2021-01-27 09:00:00	81	21	13	26
2021-01-27 10:00:00	81	21	10	26
2021-01-27 11:00:00	75	20	12	27
2021-01-27 12:00:00	81	22	13	27
2021-01-27 13:00:00	78	21	9	27
2021-01-27 14:00:00	78	21	8	27
2021-01-27 15:00:00	81	21	12	26
2021-01-27 16:00:00	69	18	12	27
2021-01-27 17:00:00	66	15	12	25
2021-01-27 18:00:00	60	15	8	24

Figure 2-3: A CSV file containing weather data.

Data Read from CSV Files

In most data science environments, you'll want to read a CSV file so that it appears as a table, making it easier for a human to understand and work with. Most data science libraries have functions that can read a CSV file and put it into a programming object—for example, a Python `DataFrame` or an R `data.frame`. How the CSV file is parsed will differ between function and language, but most give you options for specifying things like:

- What data type each column should be cast as.
- What row should be used to determine the column headers.
- What column should be used to determine the row indices.

- What character indicates the start of a comment and should be ignored by the parser.
- What character to actually use as the delimiter, if not a comma.

In some cases, these options are inferred based on the nature of the data. For example, the pandas `read_csv()` function in Python will automatically convert the text '1.32' to a string data type since it is enclosed in quotation marks. That same function will use the first row of the text file as the column headers. When you read CSV files into a programming environment, it's important to verify that everything was read in a way that you were expecting. You don't want the column headers to become actual data values, for example.

CSV files are some of the simplest ways of storing data and therefore a common format for sharing open datasets. However, they are less common in the business world, especially when it comes to proprietary data or big data that needs to be managed closely. Still, businesses sometimes export their data into CSV files for use in other systems.

Guidelines for Reading Data from CSV Files



Note: All Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Follow these guidelines when reading data from CSV files.

Read Data from CSV Files

When reading data from CSV files:

- Ensure the records in CSV files are consistently formatted.
- Ensure the delimiters used in CSV files are consistent (e.g., don't mix commas and tabs in the same file).
- Ensure that any column headers and row indices are correctly parsed by the CSV reading function.
- Ensure any comments or unwanted rows/columns are not included in the dataset when reading CSV files.

ACTIVITY 2-1

Reading Data from CSV Files

Data Files

/home/student/CDSP/ETL/Extracting, Transforming, and Loading Data.ipynb
 /home/student/CDSP/ETL/data/consumer_loan_complaints.csv

Scenario

Greene City National Bank (GCNB) has given you access to the sources of data that you've agreed could be useful for achieving its business goals. The data is spread out among different files and databases, and is stored in different formats. From what the data owners at GCNB have told you, there are four main tables of data:

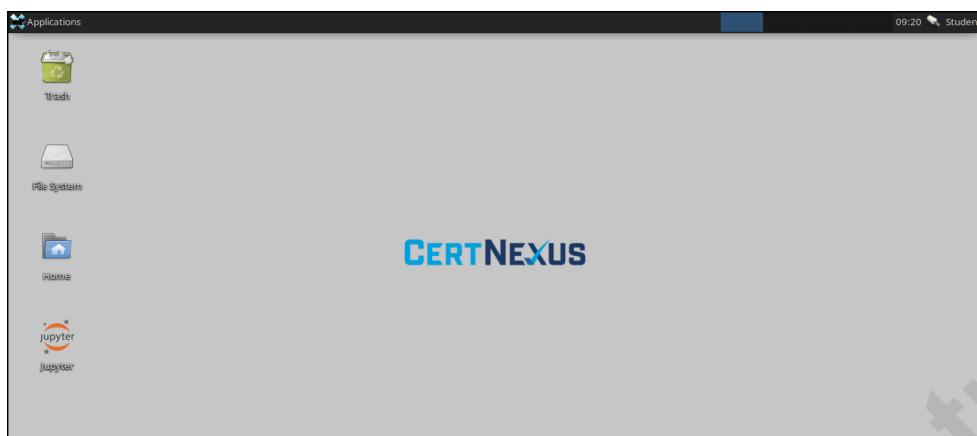
- **Users:** This table includes rows of users and various columns that describe each user's demographics and banking information. This is the largest table that GCNB provided in terms of the sheer volume of data.
- **User devices:** This table tracks what kind of device each user primarily uses when banking electronically with GCNB.
- **User transaction history:** This table includes records of individual financial transactions and the monetary value of each transaction.
- **Consumer loan complaints:** This table includes information about complaints that users have sent to GCNB when going through the loan process.

Your job is to start pulling all of this data together so that it can eventually be cleaned and consolidated into a single working environment. The consumer loan complaint data is contained within its own comma-separated values (CSV) file, so you'll start by reading that data first.

	Note: The system on the VM is configured to log the user in automatically. If you are prompted at any time to log in, the account is named student and the password is Pa22w0rd .
	Note: Activities may vary slightly if the software vendor has issued digital updates. Your instructor will notify you of any changes.

1. Start the VM.
 - a) Start the **Oracle VM VirtualBox** application.
 - b) In the **Oracle VM VirtualBox Manager**, from the list on the left, select **CDSP** and select **Machine→Start→Normal Start**.

- c) Wait as the virtual machine finishes loading and the operating system starts.



2. Start Jupyter Notebook.

- a) On the desktop, double-click the **Jupyter** icon to launch the Jupyter Notebook server and open a web client.



- The server launches first in a terminal window. Then the web browser is launched to provide the user interface for Jupyter Notebook.
 - The web client shows a listing of directories on the computer.
 - You can use this listing to navigate to folders that contain notebooks you want to open.
- b) Select **CDSP**.



Note: Select the text "CDSP"—not the folder icon next to it.

The `/home/student/CDSP` directory contains various subdirectories for different notebooks and datasets.

c) Select ETL.

The ETL subdirectory contains a subdirectory named **data** and a notebook file named **Extracting, Transforming, and Loading Data.ipynb**.



- d) Select **Extracting, Transforming, and Loading Data.ipynb** to open it.
- e) If line numbers aren't visible, select **View→Toggle Line Numbers**.

3. Import the relevant software libraries.

- a) View the cell titled **Import software libraries**, and examine the code listing below it.

```

1 import sys      # Read system parameters.
2 import pandas as pd # Manipulate and analyze data.
3 import sqlite3    # Manage SQL databases.
4
5 # Summarize software libraries used.
6 print('Libraries used in this project:')
7 print('- Python {}'.format(sys.version))
8 print('- pandas {}'.format(pd.__version__))
9 print('- sqlite3 {}'.format(sqlite3.sqlite_version))

```

- b) Select the cell that contains the code listing, then select **Run**.
- c) Verify that the version of Python is displayed, as are the versions of pandas and sqlite3 that were imported.

```

Libraries used in this project:
- Python 3.8.5 (default, Sep 4 2020, 07:30:14)
[GCC 7.3.0]
- pandas 1.1.3
- sqlite3 3.33.0

```

4. Load a CSV file as a DataFrame.

- a) Scroll down and view the cell titled **Load a CSV file as a DataFrame**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 complaints_data = pd.read_csv('data/consumer_loan_complaints.csv')
```

- c) Run the code cell.

This code uses the pandas `read_csv()` function to read the comma-separated values (CSV) file that holds the consumer loan complaints.



Note: Some of the code blocks don't have an output. You can tell that a code block was executed by the presence of a number inside the brackets to the left of the block, like [2].

5. Preview the first three rows of the data.

- Scroll down and view the cell titled **Preview the first three rows of the data**, then select the code cell below it.
- In the code cell, type the following:

```
1 complaints_data.head(n = 3)
```

- Run the code cell.
- Examine the output.

	<code>user_id</code>	<code>Date received</code>	<code>Product</code>	<code>Issue</code>	<code>Consumer complaint narrative</code>	<code>State</code>	<code>ZIP code</code>	<code>Submitted via</code>	<code>Date sent to company</code>	<code>Company response to consumer</code>	<code>Timely response?</code>	<code>Consumer disputed?</code>	<code>Complaint ID</code>
0	44fefdad-7045-4be5-890e-12e84ae6fdc9	01/27/2016	Consumer Loan	Account terms and changes	NaN	AL	35180	Phone	01/27/2016	Closed with explanation	Yes	No	1760486
1	c49d5d60-909f-406b-b7ff-51143fc6b650b	08/26/2014	Consumer Loan	Account terms and changes	NaN	NC	278XX	Phone	08/29/2014	Closed with non-monetary relief	Yes	No	1001740
2	9b2cd5d2-900e-4052-831f-6489fd6568af	08/22/2012	Consumer Loan	Account terms and changes	NaN	TN	37205	Referral	08/23/2012	Closed with non-monetary relief	Yes	No	140039

Each row in the table is a user that submitted a complaint. The columns are as follows:

- `user_id`**: An arbitrary hexadecimal string that uniquely identifies the user.
- `Date received`**: The date the complaint was received by the organization.
- `Product`**: The product the complaint is about.
- `Issue`**: The type of issue the complaint is about.
- `Consumer complaint narrative`**: The text of any written complaints. This field is optional for the complainant to fill out.
- `State`**: The U.S. state the user resides in.
- `ZIP code`**: The ZIP code the user resides in.
- `Submitted via`**: The method the user took to file the complaint.
- `Date sent to company`**: The date the complaint was sent by the user.
- `Company response to consumer`**: How the organization handled the complaint.
- `Timely response?`**: Whether or not the organization's response was, according to some metric, given within an acceptable time period.
- `Consumer disputed?`**: Whether or not the user disputed the action the organization took in response to the complaint.
- `Complaint ID`**: A unique identifier for the complaint itself.

6. Keep this notebook open.

SQL Queries

Structured Query Language (SQL) is a programmatic approach to managing data in a relational database. SQL statements, called queries, are used to retrieve data, create data, modify data, and delete data. These actions are issued using a specific syntax. SQL is highly versatile and works with

many relational database management systems like MySQL, Oracle Database, Microsoft SQL Server, and more.

A thorough exploration of every SQL query that might be useful to a data science practitioner is beyond the scope of this course. However, when it comes to pulling data from a database during the extraction phase, there are some common approaches. The most relevant keyword for retrieving data stored in a database is `SELECT`. When paired with the `FROM` keyword, you can write a query that returns the data you're looking for. For example, to retrieve a column called `name` from a table called `employees`, you could write:

```
SELECT name FROM employees
```

This will return all rows in the table, but only values for the `name` column.

<code>id</code>	<code>name</code>	<code>department</code>	<code>yrs_service</code>	<code>name</code>
1	John	HR	5	John
2	Roberta	HR	2	Roberta
3	Ted	Finance	8	Ted
4	Alyssa	Finance	9	Alyssa
5	Genevieve	IT	10	Genevieve
6	Andrew	IT	3	Andrew

You can also return all columns using the wildcard character (*). This can be useful if you plan on doing your data preparation using a tool other than SQL, and you just want to get all of the data into your working environment. However, keep in mind that doing this with very large databases could take a very long time or lead to performance issues. You may want to break up the data into portions, whether by rows or columns, to make it more manageable.

It's also worth considering how you can filter data that meets certain conditions. This is a good way to extract data on a granular level without having to pull in every value. Retrieving data based on a condition is done using the `WHERE` keyword. For example, to get all columns of the database, but *only* the rows for employees who work in Human Resources:

```
SELECT * FROM employees
WHERE department = 'HR'
```

<code>id</code>	<code>name</code>	<code>department</code>	<code>yrs_service</code>	<code>id</code>	<code>name</code>	<code>department</code>	<code>yrs_service</code>
1	John	HR	5	1	John	HR	5
2	Roberta	HR	2	2	Roberta	HR	2

It's also common during the extraction phase to aggregate table data as it comes in. That way, you can make sure the data is in a format that's easier for you to interpret, and/or a format that's more

consistent with other data sources that you're using. You can do this using the `GROUP BY` keyword, paired with an aggregation function like `COUNT`, `SUM`, `AVG`, etc. For example, let's say you have a column called `yrs_service` in the `employees` database, and you want to return a table that lists each department and the average years of service among all employees in that department:

```
SELECT department, AVG(yrs_service) AS avg_yos
FROM employees
GROUP BY department
```

The diagram illustrates the process of aggregating data. On the left, there is a flat table with six rows. The columns are `id`, `name`, `department`, and `yrs_service`. The data is as follows:

<code>id</code>	<code>name</code>	<code>department</code>	<code>yrs_service</code>
1	John	HR	5
2	Roberta	HR	2
3	Ted	Finance	8
4	Alyssa	Finance	9
5	Genevieve	IT	10
6	Andrew	IT	3

A large blue arrow points from this table to the right, indicating the transformation process. On the right, there is an aggregated table with three rows. The columns are `department` and `avg_yos`. The data is as follows:

<code>department</code>	<code>avg_yos</code>
HR	3.5
Finance	8.5
IT	6.5

Of course, there are many more ways you can use SQL queries to extract data. It all depends on what data you have, how it's structured, and what you want to do with it. Be sure to experiment with different queries so that you can learn what works best for you.

UNION Statements

Another useful keyword when extracting SQL data is `UNION`. This enables you to combine `SELECT` statements to retrieve *distinct* values from multiple sources. For example, let's say you have two tables that list employee information instead of just one aggregated table, and you want to retrieve the name of every employee. You could issue a query like so:

```
SELECT name FROM emp_table_1
UNION
SELECT name FROM emp_table_2
ORDER BY name
```

This will retrieve every unique name from both tables. However, some people may have the same name, so you can make sure you're retrieving every name, regardless of duplication, by using `UNION ALL` instead.

NoSQL Queries

NoSQL refers to any non-relational database technology. It is not necessarily a specific language, protocol, or technology, but merely a way of describing any database that doesn't store data as relational tables. NoSQL is popular in the world of big data since it tends to be faster and more efficient than relational databases at handling large volumes of unstructured or semi-structured data.

There are several approaches to NoSQL databases, including:

- **Key-value databases.** These are similar to dictionaries or hash tables in programming languages. A *key* is a unique identifier that has a specific *value* associated with it. Each key can have a different number and/or type of values. When you query this kind of database, you can only query the key itself, as the values are not directly known to the database. Key-value databases are therefore considered *opaque*.
- **Document databases.** This is similar to a key-value database, except that the data is stored within documents. A document is an object with a structured format (often JSON or XML) that is *transparent* to the database. So, unlike with a traditional key-value database, you can actually

- query by both key and value(s). Despite this structured approach, they are still not relational because each document can have different fields.
- **Wide-column databases.** Superficially, these databases look like relational tables since the data is stored in rows and columns. However, these tables have no defined structure, so the columns can vary from row to row.
 - **Graph databases.** In this type of database, data is represented as a collection of *nodes*, where each node is connected to one or more nodes. The connections are also called *edges*, which describe how the connected nodes are related.

Because NoSQL has multiple distinct approaches, there is no universal querying standard like what SQL is to relational databases. Each implementation is different, though some solutions do offer RESTful querying APIs to help streamline the process. Also, keep in mind that the approach your NoSQL solution uses will dictate the form and capabilities of your queries. In a key–value database, you won't be able to retrieve a specific key by querying a value, for instance. Ultimately, you should consult your solution's documentation for help with querying.

NoSQL Solutions

Some examples of popular NoSQL solutions include Redis, MongoDB, Couchbase, Amazon DynamoDB, Azure Cosmos DB, and Google Cloud Datastore.

Guidelines for Extracting Data with Database Queries

Follow these guidelines when extracting data with database queries.

Extract Data with Database Queries

When extracting data with database queries:

- Ask yourself the following questions about the data:
 - How frequently will new data be available?
 - How frequently will data be refreshed from the data source?
 - How available and accessible will the data be?
- When dealing with access to data:
 - Ensure there are permissions in place for databases and tables.
 - Prefer to set read-only access on the data.
 - Coordinate when data will be accessed with maintenance, backups, and other downtime.
- When extracting data:
 - Use queries that are the most efficient for whatever task you're trying to perform.
 - Use indices where possible for query criteria.
- When dealing with large volumes of data:
 - Keep in mind that queries involving larger datasets take longer to run and download.
 - Limit the number of columns in a query for large datasets.
 - Consider breaking up large datasets into multiple subsets, partitioning columns that are indexed.
 - Track data that has already been collected so that only incremental data is retrieved.

ACTIVITY 2-2

Extracting Data with Database Queries

Data File

/home/student/CDSP/ETL/data/user_data.db

Before You Begin

Extracting, Transforming, and Loading Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Create a connection to the SQLite database**. Select **Cell→Run All Above**.

Scenario

The remaining three tables GCNB provided to you are all contained within an SQL database. You have been given access to this database, but in order to actually retrieve the data, you'll need to execute some SQL queries. You also want to begin shaping the data so that it's in a more workable form. For example, the transaction history database currently records each transaction separately, even transactions made by the same user at different times. Rather than keep it like this, you want to aggregate the transactions for each user so that it's easier to integrate this data with the other tables when the time comes.

1. Create a connection to the SQLite database.

- Scroll down and view the cell titled **Create a connection to the SQLite database**, then select the code cell below it.
- In the code cell, type the following:

```
1 conn = sqlite3.connect('data/user_data.db')
2 conn
```

- Run the code cell.

You are connecting to an SQLite database, which is just a file on the local file system. In a production environment, you'd likely connect to an SQL server using credentials.

- Examine the output.

```
<sqlite3.Connection object at 0x7f2ba83c6c60>
```

The `conn` object established a connection to the database.

2. Read the users data.

- Scroll down and view the cell titled **Read the users data**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 # Write a query that selects everything from the users table.
2
3 query = 'SELECT * FROM users'
```

- c) Run the code cell.

You're defining the query that you'll pass to the database in the next code block.

- d) Select the next code cell, then type the following:

```
1 # Read the query into a DataFrame.
2
3 users = pd.read_sql(query, conn)
4
5 # Preview the data.
6
7 users.head()
```

- e) Run the code cell.

- f) Examine the output.

	user_id	age	job	marital	education	default	housing	loan	contact	duration	campaign	pdays	previous	poutcome	term_deposit
0	9231c446-cb16-4b2b-a7f7-dfc8b25aaaf6	58	management	married	tertiary	no	yes	no	None	261	1	-1	0	None	no
1	bb92765a-08de-4963-b432-496524b39157	44	technician	single	secondary	no	yes	no	None	151	1	-1	0	None	no
2	573de577-49ef-42b9-83da-d3cfb817b5c1	33	entrepreneur	married	secondary	no	yes	yes	None	76	1	-1	0	None	no
3	d6b66b9d-7c8f-4257-a682-e136f640b7e3	47	blue-collar	married	None	no	yes	no	None	92	1	-1	0	None	no
4	fade0b20-7594-4d9a-84cd-c02f79b1b526	33	None	single	None	no	no	no	None	198	1	-1	0	None	no

The database file includes a `users` table, the first few rows of which are printed. Like with the consumer complaints file, the users are identified using the same hexadecimal ID numbers. You can consider this the primary key for most of these tables. There are also various columns:

- `age`: The age of the user.
- `job`: The user's job title.
- `marital`: The user's marital status.
- `education`: The user's level of education.
- `default`: Whether the user has defaulted on a loan.
- `housing`: Whether or not the user has a housing loan.
- `loan`: Whether or not the user has a personal loan.
- `contact`: The method the user and organization use to communicate.
- `duration`: The duration of the last contact session with the user, in seconds.
- `campaign`: Number of times the user was contacted for the current marketing campaign.
- `pdays`: Number of days that passed after the user was contacted from a previous campaign.
- `previous`: Number of times the user was contacted prior to the current campaign.
- `poutcome`: The result of the previous campaign.
- `term_deposit`: Whether or not the client subscribed to a term deposit.
- `date_joined`: The date the user signed up for an account.

- g) Select the next code cell, then type the following:

```
1 # Check the shape of the data.
2
3 users.shape
```

- h) Run the code cell.
i) Examine the output.

```
(45216, 16)
```

This table has 45,216 rows and 16 columns.

3. Read the device data.

- a) Scroll down and view the cell titled **Read the device data**, then select the code cell below it.
b) In the code cell, type the following:

```
1 query = 'SELECT * FROM device'
2
3 device = pd.read_sql(query, conn)
4
5 device.head()
```

- c) Run the code cell.
d) Examine the output.

	user_id	device
0	9231c446-cb16-4b2b-a7f7-ddfc8b25aaaf6	mobile
1	bb92765a-08de-4963-b432-496524b39157	desktop
2	573de577-49ef-42b9-83da-d3cfb817b5c1	mobile
3	d6b66b9d-7c8f-4257-a682-e136f640b7e3	tablet
4	fade0b20-7594-4d9a-84cd-c02f79b1b526	mobile

The database file also includes a `device` table. Once again, the `user_id` field is present. There's also a `device` column that lists the user's preferred type of device to use when banking electronically.

- e) Select the next code cell, then type the following:

```
1 device.shape
```

- f) Run the code cell.
g) Examine the output.

```
(45117, 2)
```

This table has 45,117 rows and 2 columns.

4. Read the transactions data.

- a) Scroll down and view the cell titled **Read the transactions data**, then select the code cell below it.

- b) In the code cell, type the following:

```

1 # Read the user transactions in the last 30 days.
2
3 query = 'SELECT * FROM transactions'
4
5 transactions = pd.read_sql(query, conn)
6
7 transactions.head()

```

- c) Run the code cell.
d) Examine the output.

	user_id	transaction_id	amount_usd
0	9231c446-cb16-4b2b-a7f7-ddfc8b25aaf6	transaction_5180	1332
1	9231c446-cb16-4b2b-a7f7-ddfc8b25aaf6	transaction_5607	726
2	9231c446-cb16-4b2b-a7f7-ddfc8b25aaf6	transaction_6765	85
3	573de577-49ef-42b9-83da-d3cfb817b5c1	transaction_6170	1
4	573de577-49ef-42b9-83da-d3cfb817b5c1	transaction_6090	1

The final table in the database lists transaction information for users. Aside from the familiar `user_id`, there are two other columns:

- `transaction_id`: A unique identifier for each transaction.
- `amount_usd`: The total amount of the transaction in U.S. dollars. Positive transactions indicate a deposit, whereas negative transactions indicate a withdrawal.

- e) Select the next code cell, then type the following:

```
1 transactions.shape
```

- f) Run the code cell.
g) Examine the output.

```
(140034, 3)
```

This table has 140,034 rows and 3 columns. There are so many more rows in this table than the others because of its transactional nature; for example, the first three rows are all transactions by the same user.

5. Aggregate the transactions data.

- a) Scroll down and view the cell titled **Aggregate the transactions data**, then select the code cell below it.

- b) In the code cell, type the following:

```

1 # Aggregate data on the number of transactions and the total amount.
2
3 query = '''SELECT user_id,
4                 COUNT(*) AS number_transactions,
5                 SUM(amount_usd) AS total_amount_usd
6             FROM transactions
7             GROUP BY user_id'''
8
9 transactions_agg = pd.read_sql(query, conn)
10
11 transactions_agg.head()

```

The query you'll run can be expressed as follows: From the `transactions` table, for each user ID, count each individual transaction and make it a new column. Also add up the USD of each transaction for a user ID and make it a new column. And, group the rows by each unique user ID.

- c) Run the code cell.
d) Examine the output.

	user_id	number_transactions	total_amount_usd
0	0001570d-8aed-465e-b547-8981651084ed	3	792
1	000548ed-aa18-4eef-b8ed-68a9126e33ab	2	1044
2	00069959-4d55-460e-bb76-ae13ddbd80a6	5	0
3	000bab00-aec4-4ee2-81a6-1f897c38726b	19	0
4	000cbcac8-212f-46fb-b58f-861dada34284	2	399

Now, you have a table where each user has all of their transactions consolidated into a single row. The number of those transactions and the total amount of those transactions appear as new columns.

- e) Select the next code cell, then type the following:

```
1 transactions_agg.shape
```

- f) Run the code cell.
g) Examine the output.

```
(35211, 3)
```

There are now 35,211 rows and 3 columns in this aggregate dataset of user transactions.

6. Keep this notebook open.

Data Read from Cloud Storage

Some or all of the data you incorporate in your project may be stored in the cloud. This is a popular option for organizations that need to store big data, but don't want to take on the burden of storing it on premises. However, the process of extracting data from cloud storage is different for every service. You'll need to consult the cloud service's documentation to learn the correct protocol for accessing data. Some of the major cloud storage solutions you might encounter include:

- **Amazon Simple Storage Service (S3).** This is part of the Amazon Web Services (AWS) suite of cloud services. S3 in particular provides object storage, which means data is contained as

discrete objects or "blobs" as opposed to files in a hierarchy or a block storage device. This approach is simple and therefore highly scalable. The object containers in S3 are called "buckets" and they can be accessed using keys that identify and authenticate any person or device who requests data. Amazon provides web-based APIs for accessing the data, in much the same way as the public APIs discussed previously.

- **Google Cloud Storage Buckets.** This is the Google Cloud equivalent to Amazon S3. It provides object-based storage in buckets, and can also be accessed using web-based APIs.
- **Azure Blob Storage.** This is Microsoft's object-based cloud storage service. As you'd expect, it also has web-based APIs for accessing data.

In addition to object storage, each of these organizations also provides data lake services. As explained previously, a data lake stores data in its raw form, regardless of its structure. Because of this, accessing data from cloud-based data lakes can be more challenging, but these providers do attempt to streamline the process and provide tools to make data consumption easier. In the case of AWS and Google Cloud, data lakes are offered as a solution that integrates existing storage and management services, rather than being a distinct service. Azure Data Lake, however, is a distinct service.

Data Consolidation

Other than acquiring data and bringing it into a data science environment, the other major part of the extracting phase of ETL is to consolidate data. In other words, to combine similar data into a more streamlined form that is more conducive to being transformed and loaded.

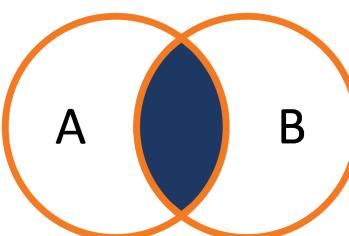
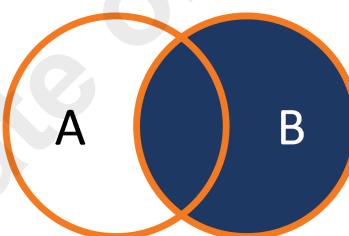
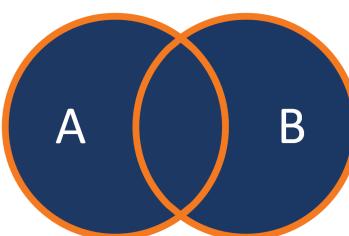
Consolidation can refer to a couple different things depending on the context. One way to look at consolidation is to consider data from a granular perspective; i.e., down to the data values themselves. For example, one data source that tracks kitchen appliances might have a column called `weight_lbs`, and another data source that tracks cutlery has a column called `weight_ounces`. Both values refer to the same feature—the weight of the product. If you want to combine these sources into a single dataset, you may be fine with having two different weight columns. But you may also want to streamline the data by consolidating the weights into one column, whether it's weight in pounds, weight in ounces, or something else. You can therefore convert the data into your chosen unit of measurement. Ultimately, it'll be easier to manage the dataset.

Another way of performing consolidation is on the structure of the data as a whole. Let's say you have two data sources that both track kitchen appliances. The first data source has many different columns that describe each appliance, including `weight`, `price`, `units_sold`, and more. The second source lists some of the same exact products, but with one column that isn't in the first source—`color`. The second source also includes some product rows that aren't even in the first source. You need to figure out a way to consolidate these two sources into a single dataset that contains information about kitchen appliances. You don't want to duplicate rows, you don't want to fail to include all products, and you don't want to drop any columns. What you need to do is figure out how best to merge the data—a process called a *join*.

Data Joins

There are multiple ways to join data, but first, you need to identify a common key that exists in all of the sources you're trying to merge into one. A *primary* key is a column that uniquely identifies row entries in a relational database. It's usually something like a name, product ID, email address—anything that is unique to each entry in the table. In some cases, two tables might each have a primary key that refers to the same identifier. The tables can be joined using this identifier. However, in other cases, the two tables have different primary keys but share some other common column. Those tables can be joined using that common column.

There are actually several types of joins, and it can be easier to understand how they work by looking at Venn diagrams that show the intersection of data after the join. In these examples, the first data table (also called the "left" table) is Table A, and the second data table (also called the "right" table) is Table B.

Join Type	Returns	Visual
Inner join	Only rows from Table A and Table B that match according to a key column.	
Left outer join	All rows from Table A, along with column data from Table B for any matching rows.	
Right outer join	All rows from Table B, along with column data from Table A for any matching rows.	
Full outer join	All rows from a combination of both tables.	

 **Note:** There are other types of joins, but these are the most common.

Inner Join Example

Since inner joins are the most common type of join, here's an example. Let's say Table A (books) includes data for a book store:

book_id	book_name	book_format
1010	<i>The Odyssey</i>	Epic poetry
1020	<i>Othello</i>	Play
1030	<i>Crime and Punishment</i>	Novel
1040	<i>Bartleby, the Scrivener</i>	Short story
1050	<i>Animal Farm</i>	Novella

And Table B (*orders*) includes order information:

order_id	book_id	order_date
1	1040	2021-02-18
2	1050	2021-02-18
3	1010	2021-02-19
4	1080	2021-02-19

You want to return the name of the book for each order that was put in, so you need to do an inner join. You can do this using various data management tools, but here's how to perform the join in SQL:

```
SELECT books.book_id, books.book_name, orders.order_date
FROM books
INNER JOIN orders
ON orders.book_id = books.book_id
ORDER BY books.book_id
```

The result is:

book_id	book_name	order_date
1010	<i>The Odyssey</i>	2021-02-19
1040	<i>Bartleby, the Scrivener</i>	2021-02-18
1050	<i>Animal Farm</i>	2021-02-18



Note: Notice how the order for book_id 1080 is missing; this is because no such book exists in the books table.

Guidelines for Consolidating Data from Multiple Sources

Follow these guidelines when consolidating data from multiple sources.

Consolidate Data from Multiple Sources

When consolidating data from multiple sources:

- Ask yourself the following questions when combining data:
 - What common fields are there between the datasets, and how can you match them?
 - Will you combine the datasets into an existing set or into a new set?
 - How will you handle joins on missing data?
 - How will you handle duplicated data?
- Consider how data can be consolidated.

- Identify same or similar data formats that can be combined.
 - String "123.45" vs. float 123.45.
 - £100 to \$135 via currency conversion.
 - Imperial to metric conversions (e.g., gallons to litres).
 - Languages conversions (e.g., "实验第一" to "Experiment one").
 - Occupation code conversions (e.g., "Purchasing Manager" can be converted to code 11-3061 in the Standard Occupational Classification System of the U.S. government).
 - Date conversions (e.g., 03/01/2021 to 2021-03-01).
- Identify character set conversions (e.g., EBCDIC, ASCII, ISO Latin-8, and Unicode).
- Identify scaling factors between data points (e.g., weights in pounds vs. tons).
- Consider how you'll get data from multiple locations into one location.
- Identify data stored on local file systems, corporate intranet, and cloud providers.

ACTIVITY 2–3

Consolidating Data from Multiple Sources

Before You Begin

Extracting, Transforming, and Loading Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Merge the device table with the users table**. Select **Cell→Run All Above**.

Scenario

You've extracted all of the relevant data from each source. The data tables currently live in separate objects, but they share a common key: `user_id`. Instead of keeping them all separate, you'll begin consolidating them using this primary key so they end up as a single table. In particular, you'll create a master table that includes user demographics, banking information, device usage, and transaction history. Having this master table will make many forthcoming data science tasks easier.

1. Merge the `device` table with the `users` table.

- Scroll down and view the cell titled **Merge the device table with the users table**, then select the code cell below it.
- In the code cell, type the following:

```

1 # Do a left join, as all users in the users table are of interest.
2
3 query = '''SELECT left_table.*,
4                  right_table.device
5          FROM users AS left_table
6          LEFT JOIN device AS right_table
7              ON left_table.user_id = right_table.user_id'''
8
9 users_w_device = pd.read_sql(query, conn)

```

You're going to be doing a left join on this data, as all of the users in the `users` table (left) are of interest, and you want to merge any records that match the `user_id` field from the `device` table (right).

- Run the code cell.
- Select the next code cell, then type the following:

```
1 users_w_device.head(n = 3)
```

- Run the code cell.

- f) Examine the output.

d	age	job	marital	education	default	housing	loan	contact	duration	campaign	pdays	previous	poutcome	term_deposit	date_joined	device
5-	58	management	married	tertiary	no	yes	no	None	261	1	-1	0	None	no	1998-08-23	mobile
a-	44	technician	single	secondary	no	yes	no	None	151	1	-1	0	None	no	2008-07-15	desktop
f-	33	entrepreneur	married	secondary	no	yes	yes	None	76	1	-1	0	None	no	2002-06-04	mobile

If you scroll the table all the way to the right, you'll see the `device` column has been merged for each user.

- g) Select the next code cell, then type the following:

```
1 users_w_device.shape
```

- h) Run the code cell.
i) Examine the output.

```
(45216, 17)
```

There are 45,216 rows and 17 columns in this new table. The number of rows is equal to the original `users` table, indicating that the join worked as expected.

2. Close the database connection.

- a) Scroll down and view the cell titled **Close the database connection**, then select the code cell below it.
b) In the code cell, type the following:

```
1 conn.close()
```

- c) Run the code cell.
It's always a good idea to close the connection to a database when you're no longer using it. The next merge you'll perform will use pandas instead of SQL.

3. Merge `users_w_device` with `transactions_agg`.

- a) Scroll down and view the cell titled **Merge `users_w_device` with `transactions_agg`**, then select the code cell below it.
b) In the code cell, type the following:

```
1 # Do a right join so users won't be lost.
2
3 users_w_devices_and_transactions = \
4 transactions_agg.merge(users_w_device,
5                         on = 'user_id', how = 'right')
6
7 users_w_devices_and_transactions.head()
```

You're creating a new `DataFrame` that will merge the aggregated transactions data with the combined users and devices table. You're doing a right join so that `users_w_device` (right) won't lose any users that aren't also in `transactions_agg` (left).

- c) Run the code cell.

- d) Examine the output.

	user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	campaign	p
0	9231c446-cb16-4b2b-a7f7-ddfc8b25aa6	3.0	2143.0	58	management	married	tertiary	no	yes	no	None	261	1	
1	bb92765a-08de-4963-b432-496524b39157	NaN	NaN	44	technician	single	secondary	no	yes	no	None	151	1	
2	573de577-49ef-42b9-83da-d3cfb817b5c1	2.0	2.0	33	entrepreneur	married	secondary	no	yes	yes	None	76	1	
3	d6b66949-7c8f-4257-a682-e136f640b7e3	NaN	NaN	47	blue-collar	married	None	no	yes	no	None	92	1	
4	fade0b20-7594-4d9a-84cd-c02f79b1b526	1.0	1.0	33	None	single	None	no	no	no	None	198	1	

This new table not only has the main user information including what device they use, but it also includes the number of transactions and the total amount spent on those transactions. Even where transaction data is missing (e.g., the second row), the row is preserved.

- e) Select the next code cell, then type the following:

```
1 # Make sure number of rows is equal to users_w_devices table.
2
3 users_w_devices_and_transactions.shape
```

- f) Run the code cell.
g) Examine the output.

```
(45216, 19)
```

As expected, the new table has the same number of rows as the `users_w_devices` table (45,216). The table has also grown to include 19 total columns.

4. Keep this notebook open.

TOPIC B

Transform Data

The next step in the ETL process is transformation. You'll spend this topic adjusting your data so that it's in a more useful state.

Preliminary Data Transformation

The *transform* phase of the ETL process can occur after you've extracted data from the necessary sources. As the name implies, transformation involves some type of change to the data. Quite a bit of these changes can occur up front, because you'll know what to look for. And, the tools you're working with can make it much more obvious when something in your data looks "wrong" and needs to be corrected. However, in some cases, you won't be able to make certain changes until after you've analyzed the data to know what to transform and how. That's why you should think of your changes at this point as being preliminary, rather than final. Still, you can get quite a bit done at this phase.

Data Preparation and Cleaning

There are several reasons why you might want to transform data, but the two most common have to do with preparing the data and cleaning it.

Data preparation is the process of altering data so that it more effectively supports other data science tasks, particularly analysis and model development. Since these tasks are so vital to meeting business goals, data preparation is a necessary component of achieving success with any data science project. There are many individual tasks that can go into preparing data, several of which you'll learn. Overall, the purpose of preparing data is to correct any issues that you can identify before loading data into its final destination. These issues can be at a macro level (e.g., unstructured data from one source that is poorly fit into a structured format), or they can be at a micro level (e.g., individual data values are incorrect).

Data cleaning is really a subset of preparation, and just refers to addressing inaccuracies and other problems with data. This can include duplicated data, data with the wrong data type or formatting, corrupted data, missing data, and so on. To actually "clean" the data could mean to change the offending data, or it could mean to simply remove it. Each has its benefits, and one may be more desirable or practical than the other in certain situations. For example, if many records have the same mistaken value in the same column, it might be easy to correct that value. Removing too many records could impair the dataset. On the other hand, if only a few records have problematic values, but those values don't follow any identifiable pattern, it may be difficult to correct them. So, you might choose to drop those records instead. The choice comes down to what action you believe will be the most feasible for you to take, while minimizing any negative effects that might appear later on.

The entire process of data preparation can be tedious and may take a significant amount of time on a data science project. Reflecting the challenge of the task, it is sometimes called **data wrangling** or **data munging**, particularly when it is performed manually or outside of formal, repeatable processes. However, several software libraries provide functions that enable you to automate the process of data preparation. This is especially valuable when you must repeat the cleanup process on other datasets or when new data is added over time.



Note: Whenever you perform operations on data, consider creating a backup copy so you can revert back to the original dataset should anything go wrong.

Types of Data

When discussing the types of data, you can really be referring to one of two things: the high-level feature types, or the machine-readable data types. The former refers to the overall way that data values represent one or more observations. This has a significant impact on how you can work with that data, including how it's prepared, analyzed, and modeled. There are three major types of features:

- **Quantitative** (or numerical) data holds number values that express magnitude. For example, a `Price` feature has a magnitude since it is measuring how much of something there is.
- **Qualitative** (or categorical) data holds a value from a set of values that are typically limited. For example, the `Color` feature for a car might hold values such as red, blue, and black. Note that there is no inherent ranking in categorical data—no color is "better" than the others in this example.
- **Ordinal** data is not entirely categorical since it can be ordered. But it's also not entirely quantitative since it doesn't measure magnitude. For example, a `Grade` feature might hold a student's grade, which can be A, B, C, D, or F. An inherent rank/order is represented (for example, an A is a better grade than a C).

On the other hand, data types, also called data formats, refer to the ways that programming languages represent data for execution by computer hardware. These data types have a direct impact on how a program interprets data. Example data types you're probably familiar with include:

- **Integers**, such as 25.
- **FLOATS**, such as 12.78.
- **STRINGS**, such as 'Red'.
- **BOOLEANS**, such as `True` or `False`.
- **DATETIMES**, such as 2021-03-01.

It's important to understand these distinctions, as features and data types do not always map one to one. For example, not all categorical data is represented by a string. The feature `PreferredCustomer` might contain the integer 1 instead of the string 'Yes' and the integer 0 instead of the string 'No'.

Operations You Can Perform on Different Types of Features

Statistical computations can't be indiscriminately applied to just any type of feature. For example, it is not particularly apparent how or why you would calculate the *mean* value of a set of colors because there is no inherent order or sequence. It would be much more straightforward to calculate the most common (*mode*) color.

The following table describes the statistical measures that make sense to be performed on each type of feature.

Feature Type	Description
Quantitative	<p>Various center and spread measures can logically be made on a set of quantitative data values.</p> <p>Valid measures for the quantitative dataset (13, 13, 31, 38, 72) are shown here.</p> <ul style="list-style-type: none"> • Mean: $(13 + 13 + 31 + 38 + 72) / 5 = 33.4$ • Median: 31 • Mode: 13 • Range: $72 - 13 = 59$

Feature Type	Description
Qualitative	<p>Most center and spread measures are inappropriate for qualitative data, although mode can still be determined.</p> <p>Valid measures for the qualitative dataset ('blue', 'red', 'black', 'black', 'black') are shown here.</p> <ul style="list-style-type: none"> • Mode: 'black'
Ordinal	<p>Valid measures for the class dataset ('A', 'A', 'C', 'F', 'D') are shown here.</p> <ul style="list-style-type: none"> • Median: 'C' • Mode: 'A'



Note: Not all numerical data is quantitative. For example, a user ID doesn't have a magnitude, and it would not make sense to perform mathematical operations on such a number.

Continuous vs. Discrete Variables

A **continuous variable** is a quantitative variable whose values are uncountable and can extend infinitely within a certain range. Consider the time it takes a long-distance runner to complete a marathon. An omniscient being could say that someone finished the race in exactly 3 hours, 10 minutes, 37 seconds, 845 milliseconds; and so on, continuing indefinitely for perfect precision. Obviously, the human judges will have to settle for a stopwatch with an imperfect amount of precision. The stopwatch might report that the race took 3 hours, 10 minutes, and 37 seconds. Even though the variable wasn't measured with perfect precision, it is still considered a continuous variable because it can take on all real values between a set of values. Another way of thinking about continuous variables is that they are typically represented as fractions, like $3 \frac{1}{2}$ or 1.5. If you were to convert the stopwatch's measurement into a decimal fraction, you would get 3.1769444... hours.

Certain analysis methods and machine learning algorithms cannot effectively work with continuous variables. In contrast, a **discrete variable** is one whose values are countable and limited, because there is a definite gap between each value in a range of values. A feature like `Time in Minutes` that uses an integer data type is a discrete variable because it is either 120, 121, 122, 123, etc., and cannot be divided into more precise values. As opposed to continuous variables, discrete variables are typically represented as whole numbers.

The difference between continuous and discrete variables will influence how data is treated in multiple phases of the data science process, including data transformation. For example, continuous variables are typically formatted as floats, whereas discrete variables are typically formatted as integers.

Data Parsing

Parsing data involves taking data as input and then representing it in a certain structure or syntax. You can parse raw data and turn it into well-formatted data; for example, you might have a CSV file where each line of text becomes a record in a relational table. Or, you can parse data that already has a definite form, but you want to change that form into something else. For example, data on a web page can be converted from HTML format to something like a `DataFrame`. There are many ways data can be represented in HTML, so the parser must be capable of identifying specific elements, separating them into distinct objects, and reforming them in the destination format.

Most data science tools have automated parsing capabilities with common data formats. They can determine the data type and general structure that is required for the data. You saw this with pandas being able to read a CSV file into a `DataFrame` without you having to specify exactly how to read the file. Still, some types of data will require more manual work to parse. Markup languages like the

aforementioned HTML tend to require greater manual input since they are rather flexible when it comes to representing data. You may need to provide instructions to a parser or configure it in some way so that it is more successful at converting the data.

Whenever necessary, you should ensure that all of your data is parsed into your working format(s) before you continue with the rest of your transformation tasks.

Guidelines for Parsing Data

Follow these guidelines when parsing data.

Parse Data

When parsing data:

- Whenever possible, prefer to parse data from a well-structured format, rather than more permissible formats like HTML.
- Consider that binary formats are faster to parse and will take up less storage space, but they are much harder if not impossible for a human to read.
- Consider that binary formats may only be readable by the system or library that originally created it.

Data Irregularities

Lower-level cleaning tasks involve addressing "irregular" data, or any data that doesn't conform to expectations. There are many types of problems that can indicate irregular data, including:

- Corrupted or unusable data.
- Incorrectly formatted data.
- Duplicated data.
- Placeholder data.
- Null or missing data.

You must identify as many instances of these problems as you can and address them as part of your overall data preparation efforts.

Identification of Corrupted or Unusable Data

Corrupted data can take many forms, so there is not necessarily one catch-all method for identifying it. Some corrupted data is more conspicuous and can be identified by the value itself. For example, if you have a feature called `color` and one of the values for a record is '`◆◆◆◆`', you can be reasonably sure that some sort of encoding error is happening since it includes replacement characters instead of an actual color name. Identifying this would be easy, as you can just filter your dataset to only show non-valid color names, and this will appear. You might then choose to remove the record entirely, or estimate what color is most likely based on several factors.

Another relatively obvious indication of corrupted data is when the data type for a column is something it very clearly shouldn't be. For example, in pandas, each column of a `DataFrame` must contain one data type. A column cannot include both integers and floats; only one or the other. When the data is pulled into the `DataFrame`, the parsing engine determines the data type based on the way the data appears. If all values in a column are numbers that include decimal points, the column will be cast as a float. However, if just one value deviates, the entire column could be recast. For example, if the string '`blue`' appears just once in the `Price` column, the entire column will be strings. So, the regular data will show up as '`1.23`', '`6.34`', '`8.91`', and so on. In this case, you can identify irregular data by the effect it has on data within the same context.

Some data doesn't have such overt signs of corruption, but is still relatively easy to identify. This usually happens when your knowledge of the domain helps you find anomalies. For example, if you had a dataset of major cities and a feature called `Population`, a population of 5 would almost

certainly indicate a mistake. Perhaps the data source you pulled this info from recorded the population in millions rather than individually. Whatever the reason, you could filter the dataset to only show values that lie outside an acceptable range; in this case, something like 100,000 to 40,000,000.

Unfortunately, some corrupted or unusable data can't be easily identified. Maybe one of the cities has a Population value of 3,567,100—a perfectly reasonable figure, but one that was just recorded incorrectly. You can try comparing that data to other data values, like checking to see if the city population is greater than the population of the country it's in, but you may need to do some more external research to be sure. This process of verifying individual data values isn't something you can always automate, and can quickly become tedious. If you detect a pattern of incorrect values in certain portions of the dataset (e.g., all values recorded at a certain time), then you might need to consider that entire portion unusable and drop it.

Guidelines for Handling Irregular and Unusable Data

Follow these guidelines when handling irregular and unusable data.

Handle Irregular and Unusable Data

When handling irregular and unusable data:

- Keep in mind that irregular data is data that has inconsistencies between records; for example, varying records (record lengths, improperly ordered columns) or data formats (date formats, numeric representations, character sets).
- Correct data with inconsistent columns between records before input. Automated systems can often correctly interpret when columns are omitted at the end of a record, but it becomes more difficult when cells are omitted from the middle of a record.
- Consider the different ways you could handle irregular data:
 - By utilities or applications within or outside conventional data science tools. These can perform simple transformations like processing strings.
 - On the data source itself before it is exported to a data science environment. This can make it easier to correct data since the context of the original data is readily available.
 - By manual processes; for example, manually reviewing data sourced from image scans or optical character recognition (OCR).
 - There has been some progress in improving the accuracy of techniques like OCR using machine learning, so manual review may become easier.
 - There are online or cloud-based services that distribute the correcting of data, such as creating small human intelligence tasks on Amazon's Mechanical Turk (called "HITs"). There is some effort required to split up the corrections into individual jobs, which makes this approach useful only when there is a large volume of data to correct.
- Consider that, often, the only way to handle unusable data is to recreate it from source data. This is because unusable data, by definition, cannot be transformed by conventional methods into something that can be read or interpreted by a data science workbench, or data which is not suitable for analysis and modeling.

ACTIVITY 2-4

Handling Irregular and Unusable Data

Before You Begin

Extracting, Transforming, and Loading Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Identify data where age is greater than 150**. Select **Cell→Run All Above**.

Scenario

The tables that GCNB sent over have thousands of records, and it's unlikely that they are in a pristine state. There's a chance they include at least some corrupt or faulty data, whether as a result of data entry errors or something else. In any case, you need to find any unusable data and deal with it so that it doesn't cause issues later on. You have a suspicion that some of the user ages may have been recorded incorrectly, so you'll look for evidence of that and take the appropriate action, if necessary. There may also be other circumstances that could indicate faulty data, so you'll do some more investigation.

1. Identify data where age is greater than 150.

- Scroll down and view the cell titled **Identify data where age is greater than 150**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_w_devices_and_transactions[users_w_devices_and_transactions.age > 150]
```

- Run the code cell.
- Examine the output.

	user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	campaign
7228	44fefdad-7045-4be5-890e-12e84aeafdf9		NaN	NaN	178	blue-collar	married	primary	no	yes	no	None	691
10318	9b2cd5d2-900e-4052-831f-648916d568af		2.0	3165.0	891	management	married	tertiary	no	yes	no	None	278

There are two users whose age is greater than 150, which suggests the data is corrupted. So, you will remove these rows from the dataset.

2. Drop incorrect data.

- Scroll down and view the cell titled **Drop incorrect data**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 users_cleaned = \
2 users_w_devices_and_transactions[users_w_devices_and_transactions.age < 150]
3
4 users_cleaned.shape
```

You're creating a new DataFrame with all of the same data, except for the two corrupt rows. Dropping entire rows with faulty data isn't the only approach you could take, especially if you believe the values in the other columns are accurate. However, since there are only two affected records out of tens of thousands, it's safe to just remove them.

- c) Run the code cell.
d) Examine the output.

```
(45214, 19)
```

There are now 45,214 rows—two fewer than the original.

3. Identify more potentially erroneous data.

- a) Scroll down and view the cell titled **Identify more potentially erroneous data**, then select the code cell below it.
b) In the code cell, type the following:

```
1 # Compare age to device.
2
3 pd.crosstab(users_cleaned['age'], users_cleaned['device'])
```

There may be other instances of anomalous data that aren't as obvious, and are only identifiable when placed in a specific context. Here you're using `crosstab()` to generate a frequency table where the number of devices used by each age will be shown.

- c) Run the code cell.

- d) Examine the output.

device	desktop	mobile	tablet
age			
18	5	6	1
19	10	22	3
20	11	33	6
21	16	44	19
22	30	87	11
...
90	1	1	0
92	1	1	0
93	0	2	0
94	0	1	0
95	0	1	1

77 rows × 3 columns

The frequency table shows that younger users are much more likely to use electronic devices to do their banking than people who are 90+ years old. Since the number of devices used by these elderly users is so sparse (one to two total per age), it may suggest anomalous data. However, this isn't necessarily the case, so you'll leave the data alone for now. Still, it's worth the effort to at least investigate potential anomalies.

4. Keep this notebook open.

Correction of Data Formats

Data items may be represented differently in the various sources from which you obtain data. Different database systems and data stores support different data types and might store values with different levels of precision. For example, a seemingly straightforward feature such as `color` might be stored as a string, an integer, or even a float. When you combine like values from multiple sources, they must use a consistent data type, one that is compatible with the working environment you're using as well as the database you plan to load that data into. These values also need to be in a format that will support both analysis and modeling of data later on.

So, as part of the data cleaning process, you'll need to inspect your dataset to identify any features whose data types need to change. As mentioned before, data objects like a `DataFrame` will attempt to determine the optimal data type to use for a column, and may be forced to use an unexpected or undesired data type if just one value deviates from the norm. Even if all of the values follow the same pattern, the data type may still not be what you want. For example, it's common practice to use a decimal data type instead of a float when it comes to money. Decimals have the highest level of precision and won't introduce rounding errors. However, many tools will automatically parse any numbers with a decimal point as a float. So, you'll need to convert these values to decimal.

You could also run into issues where numbers with decimal points should have been cast as floats, but were mistakenly made integers. This is a significant loss of precision and will likely have a negative impact on the data's viability. The opposite is a little more common, however—whole numbers being parsed as floats. For example, with the feature `Units Sold`, you can't have sold half of a unit, so you probably want that column to be an integer. But many tools will automatically make any number a float, so a quantity of 1 will become `1.0`. This isn't always a big deal since no precision is lost, but it can make the data harder to interpret.

Thankfully, most data science environments provide functions for easily converting data from one type to another. However, keep in mind that not all conversions are going to work. You can convert the string '1.34' to a float 1.34 because the string contains a number and only a number. But, if you try to convert the string 'two' to an integer, it's probably not going to work. However, you can convert just about any value to a string.

Date Conversion

Most data type conversion is relatively straightforward. However, dealing with dates and times deserves a special mention. In its most basic form, a date or time can be represented as a string, e.g., '2021-03-01 00:03:50' refers to March 1st, 2021 at 3:50 A.M. The problem is that there's a lot of distinct information being conveyed, and no easy way to extract individual portions. What if you wanted to retrieve just the month? You could try to parse the string by extracting it from a specific position or between certain characters, but this can get tedious. It's also prone to error. After all, there are many ways to format dates and times. The method you use for parsing the prior string isn't going to work if the value is formatted as 'March 1st, 2021'.

To address these problems, most programming languages implement a special type of data type called a datetime. Each language implements it differently, but they all tend to fulfill the same purpose: to store a date and/or time in a consistent format that is easy to extract individual components from. So, if you have a datetime variable `dt` whose value is 2021-03-01 00:03:50, you could call `dt.month()` and it'll retrieve the month. Not only that, but datetimes are flexible, meaning you could retrieve the integer 3 directly, or you could retrieve the string 'March'.

In many cases, if you extract a raw date or time from a text file or database type that your data science environment doesn't understand, it will simply pull the value in as a string. However, you can also configure the environment to explicitly parse the values as a datetime. Most environments are smart enough to detect the many different ways to write a date and time, so if some sources use YYYY-MM-DD and other sources spell the whole thing out, you don't necessarily have to worry about making them consistent. As long as they're both recognized as dates/times and are parsed as datetimes, you should be good to go. If the environment you're pulling the data into can't automatically convert values to datetime, you may need to use other libraries to do so manually.

Guidelines for Correcting Data Formats

Follow these guidelines when correcting data formats.

Correct Date Values

When correcting date values:

- Consider that dates come in many formats. With some care and foreknowledge, they can be transformed into the required date format.
- Consider that there are many formal and informal date formatting standards.
 - ISO 8601 uses YYYY-MM-DD or YYYYMMDD as the calendar date.
 - Most data science tools and programming languages will read this date natively.
 - Some platforms may not read ISO 8601-formatted dates. However, they will often understand dates in YYYY-MM-DD and YYYY-MON-DD and times in HH:MM:SS with and without an AM/PM period indicator.
- Keep in mind that dates formatted with the year at the end of the string can be ambiguous. In the U.S., the date would be interpreted as month followed by day, whereas European convention is to interpret it as day followed by month.
- Consider your knowledge of the source data when deciding how to transform dates.
- For easiest processing, normalize dates in all data sources. This will simplify import and interpretation of data by a data science workbench.

Correct Numeric Values

When correcting numeric values:

- Consider that numeric values can take many forms in data files. This includes both data type (float, integer) and also the representation of these values. Values may be represented by human-readable characters (ASCII 0–9 and digit group separators like commas and periods, and occasionally spaces or underscores) or raw binary values.
- Keep in mind that some data science tools will ignore embedded digit group separators. It's still important to know the format of the numbers to know if . or , is the floating point indicator.
- Keep in mind that numeric values can have a leading + or - to indicate sign.
- Keep in mind that negative numbers can be surrounded by parentheses.
- Keep in mind that binary values are not human readable.
 - For example, the value 12,345,678 takes 10 characters in a data file. The equivalent binary representation could be encoded into three "byte" values 0xBC 0x61 0x43. This uses the same storage space as three letters, in this case an unintuitive value of ¼aC.

ACTIVITY 2–5

Correcting Data Formats

Before You Begin

Extracting, Transforming, and Loading Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Identify data types that need correcting**. Select **Cell**→**Run All Above**.

Scenario

Thankfully, it seems that most of the data types in the dataset are being cast correctly. However, you'll need to convert string objects to Boolean values where appropriate, since some columns should be `True` or `False`. Also, dates and times can often cause problems when they're pulled into a programming environment, especially since they're often just cast as standard strings. While this isn't necessarily a problem, it's much easier to work with dates and times when they're cast as `datetime` objects. So, you'll convert the relevant column from a string object to a `datetime`.

1. Identify data types that need correcting.

- Scroll down and view the cell titled **Identify data types that need correcting**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_cleaned.info()
```

- Run the code cell.

- d) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45214 entries, 0 to 45215
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          45214 non-null   object  
 1   number_transactions 35215 non-null   float64 
 2   total_amount_usd   35215 non-null   float64 
 3   age               45214 non-null   int64  
 4   job               44926 non-null   object  
 5   marital            45214 non-null   object  
 6   education          43357 non-null   object  
 7   default             45214 non-null   object  
 8   housing             45214 non-null   object  
 9   loan                45214 non-null   object  
 10  contact             32196 non-null   object  
 11  duration            45214 non-null   int64  
 12  campaign            45214 non-null   int64  
 13  pdays              45214 non-null   int64  
 14  previous            45214 non-null   int64  
 15  poutcome            8255 non-null   object  
 16  term_deposit        45214 non-null   object  
 17  date_joined         45184 non-null   object  
 18  device              45120 non-null   object  
dtypes: float64(2), int64(5), object(12)
memory usage: 6.9+ MB
```

You can see that `date_joined` has a data type of `object` (a string) instead of `datetime64` (a datetime format). A datetime format will make the column easier to work with. Also, none of the columns are of a Boolean type, and you know that at least some of them should be based on their "yes" and "no" values.



Note: `number_transactions` should be an integer (a whole number), but because there are missing values, it defaults to a float (a number with decimal points).

- e) Select the next code cell, then type the following:

```
1 users_cleaned.default.value_counts()
```

This code will print the values for the `default` variable and their frequencies.

- f) Run the code cell.
g) Examine the output.

```
no      44398
yes     816
Name: default, dtype: int64
```

The `default` variable has only "yes" or "no" values in string object form. Other variables follow this pattern, including `housing`, `loan`, and `term_deposit`. It would be better if these were cast as Booleans.

2. Convert the relevant variables to a Boolean type.

- a) Scroll down and view the cell titled **Convert the relevant variables to a Boolean type**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 users_cleaned_1 = users_cleaned.copy() # Work with a new object.
2
3 users_cleaned_1.default = \
4 users_cleaned_1.default.map(dict(yes = 1, no = 0)).astype(bool)
5
6 users_cleaned_1.default.value_counts()
```

- c) Run the code cell.
d) Examine the output.

```
False    44398
True     816
Name: default, dtype: int64
```

The values that were "yes" and "no" are now `True` and `False` for the `default` variable.

- e) Select the next code cell, then type the following:

```
1 # Do the same for the other Boolean variables.
2
3 bool_vars = ['housing', 'loan', 'term_deposit']
4
5 for var in bool_vars:
6     users_cleaned_1[var] = \
7         users_cleaned_1[var].map(dict(yes = 1, no = 0)).astype(bool)
8
9 print(f'Converted {var} to Boolean.')
```



Note: The `print()` function should be indented within the `for` loop.

- f) Run the code cell.
g) Examine the output.

```
Converted housing to Boolean.
Converted loan to Boolean.
Converted term_deposit to Boolean.
```

The `housing`, `loan`, and `term_deposit` variables have also been converted.

- h) Select the next code cell, then type the following:

```
1 users_cleaned_1.info()
```

- i) Run the code cell.

- j) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45214 entries, 0 to 45215
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          45214 non-null   object  
 1   number_transactions 35215 non-null   float64 
 2   total_amount_usd   35215 non-null   float64 
 3   age               45214 non-null   int64  
 4   job               44926 non-null   object  
 5   marital            45214 non-null   object  
 6   education          43357 non-null   object  
 7   default             45214 non-null   bool    
 8   housing             45214 non-null   bool    
 9   loan               45214 non-null   bool    
 10  contact             32196 non-null   object  
 11  duration            45214 non-null   int64  
 12  campaign            45214 non-null   int64  
 13  pdays              45214 non-null   int64  
 14  previous            45214 non-null   int64  
 15  poutcome            8255 non-null   object  
 16  term_deposit        45214 non-null   bool    
 17  date_joined         45184 non-null   object  
 18  device              45120 non-null   object  
dtypes: bool(4), float64(2), int64(5), object(8)
memory usage: 5.7+ MB
```

As you can see, all four variables are now of the Boolean (`bool`) data type.

3. Convert `date_joined` to a datetime format.

- Scroll down and view the cell titled **Convert `date_joined` to a datetime format**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_cleaned_2 = users_cleaned_1.copy() # Work with a new object.
2
3 users_cleaned_2['date_joined'] = \
4 pd.to_datetime(users_cleaned_2['date_joined'],
5                 format = '%Y-%m-%d')
```

The `to_datetime()` function will convert the `date_joined` column. The `format` argument specifies that it will follow the YYYY-MM-DD format, which matches how it appeared when it was a string object.

- Run the code cell.
- Select the next code cell, then type the following:

```
1 users_cleaned_2.info()
```

- Run the code cell.

- f) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45214 entries, 0 to 45215
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          45214 non-null   object  
 1   number_transactions 35215 non-null   float64 
 2   total_amount_usd   35215 non-null   float64 
 3   age               45214 non-null   int64  
 4   job               44926 non-null   object  
 5   marital            45214 non-null   object  
 6   education          43357 non-null   object  
 7   default             45214 non-null   bool    
 8   housing             45214 non-null   bool    
 9   loan               45214 non-null   bool    
 10  contact            32196 non-null   object  
 11  duration            45214 non-null   int64  
 12  campaign            45214 non-null   int64  
 13  pdays              45214 non-null   int64  
 14  previous            45214 non-null   int64  
 15  poutcome            8255 non-null   object  
 16  term_deposit        45214 non-null   bool    
 17  date_joined         45184 non-null   datetime64[ns] 
 18  device              45120 non-null   object  
dtypes: bool(4), datetime64[ns](1), float64(2), int64(5), object(7)
memory usage: 5.7+ MB
```

The `date_joined` column is now in a datetime format.

4. Keep this notebook open.

Deduplication

Deduplication is the process of identifying and removing duplicate entries from a dataset.

Duplicate entries can lead to difficulties with interpreting the data and can impair a model's ability to learn patterns from that data. So, you must find and address them wherever they exist in the dataset.

In the most common cases, duplicated data refers to rows in a table that appear more than once when they should not. This can occur for various reasons, but is usually due to some error in recording the data or consolidating the data from multiple sources. The most obvious indication of this is when two or more rows share the same exact values for every column, like so:

Customer ID	Last Name	First Name	Country	Gender
1056	Williams	Emily	CA	F
1056	Williams	Emily	CA	F
1056	Williams	Emily	CA	F

Most programming libraries provide functions that can automatically recognize fully duplicated rows and drop all of them except for one.

However, some duplicates might exist where only one or two values are the same across multiple rows. If at least one of those repeated values happens to be the primary key, then you know you have a duplicate. The discrepancies with the rest of the columns are likely due to recording error. It may not be clear which row is the "correct" one and which is not, so it might be safer to just drop both rows depending on how frequently they appear and how large the dataset is.

Deduplication Without a Key

Some evidence of duplication can be deceptive. Consider the following table:

Customer ID	Sale Price	Tax	Quantity	Product
1056	799.99	63.99	1	TV
1056	78.99	6.32	2	Speakers
1056	199.99	15.99	1	Subwoofer

In this case, `Customer ID` is *not* the primary key, since this is a transactional dataset. The same customer can initiate multiple transactions. So there are no duplicates here. If there were a column called `Transaction ID` that had the same values across multiple rows, then you'd be dealing with duplicates, since that feature would act as the primary key. In the absence of any kind of common key, it may be even less clear what is a recording error and what is a legitimate observation. This is why it's important to have a deep understanding of your dataset; you may need to solve problems like this manually, since automated programming libraries won't always know what to do.

Deduplication of Columns

Duplication of data can also refer to columns, not just rows. A column could literally appear twice and hold the same values for each row, for example. Or, two columns might hold different raw values, but still be redundant.

Name	Place	Time in Minutes	Time in Hours
Romano	2	176	2.93
Alvarez	3	201	3.35
Matthews	1	168	2.80

In this example, the column `Time in Hours` is redundant with the column `Time in Minutes` since they're both measuring the same thing, just on different scales. So the process of deduplication might involve removing one of these columns. Again, knowledge of your data is an important factor in determining what columns are duplicated and can be removed.

Guidelines for Deduplicating Data

Follow these guidelines when deduplicating data.

Deduplicate Data

When deduplicating data:

- Keep in mind that data that has been duplicated between different files can be difficult to discover and correct after it has been transformed.
- Be systematic when collecting data. Keep track of which dates have been exported to prevent exporting the same data twice.
- Deduplicate data if there is a value that is guaranteed to be unique within a data record. Some examples of this are a unique database record identifier, timestamp, or a combination of these and other indicators to determine uniqueness.
- Deduplicate records by using ad hoc scripts, third-party libraries, or applications.
- Deduplicate data as early as possible in the data pipeline to conserve storage space and redundant processing operations.
- Perform checks to ensure that no duplicate records exist. Often these checks require deep knowledge of the data being collected. This can be done by calculating different aggregates and

verifying that the results are reasonable, such as comparing the count of records with the elapsed collection time.

Do Not Duplicate or Distribute

ACTIVITY 2–6

Deduplicating Data

Before You Begin

Extracting, Transforming, and Loading Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Identify all duplicated data**. Select **Cell→Run All Above**.

Scenario

Another issue that plagues relatively large datasets is the presence of duplicates, and the GCNB data is no different. You'll identify any potentially duplicated rows and then remove them from the dataset. That way, the analysis and modeling you eventually perform won't be skewed by repeated data.

1. Identify all duplicated data.

- Scroll down and view the cell titled **Identify all duplicated data**, then select the code cell below it.
- In the code cell, type the following:

```
1 duplicated_data = \
2 users_cleaned_2[users_cleaned_2.duplicated(keep = False)]
3
4 print('Number of rows with duplicated data:',
5     duplicated_data.shape[0])
```

You're using the `duplicated()` function to find any duplicates in the current dataset.

- Run the code cell.
- Examine the output.

```
Number of rows with duplicated data: 10
```

There are 10 rows with duplicated data.

- Select the next code cell, then type the following:

```
1 duplicated_data
```

- Run the code cell.

- g) Examine the output.

	user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration
15456	cba59442-af3c-41d7-a39c-0f9bffbba0660	2.0	1218.0	57	management	married	tertiary	True	True	False	cellular	317
15457	cba59442-af3c-41d7-a39c-0f9bffbba0660	2.0	1218.0	57	management	married	tertiary	True	True	False	cellular	317
22006	1e826721-b38c-41c2-88f4-4c28b335b1e6	4.0	159.0	31	technician	single	secondary	False	False	False	cellular	129
22007	1e826721-b38c-41c2-88f4-4c28b335b1e6	4.0	159.0	31	technician	single	secondary	False	False	False	cellular	129
35415	a2fb8264-d55a-437b-a8e7-9ec4116b76f4	2.0	676.0	34	management	married	tertiary	False	False	False	cellular	156
35416	a2fb8264-d55a-437b-a8e7-9ec4116b76f4	2.0	676.0	34	management	married	tertiary	False	False	False	cellular	156
35623	f49ac08f-b872-4d57-acb2-9b8a9144020d	4.0	117.0	38	blue-collar	married	secondary	False	True	False	cellular	54
35624	f49ac08f-b872-4d57-acb2-9b8a9144020d	4.0	117.0	38	blue-collar	married	secondary	False	True	False	cellular	54
36296	ae3b92a2-cad8-434f-8037-9815e2228839	2.0	426.0	43	admin.	single	secondary	False	True	False	cellular	76
36297	ae3b92a2-cad8-434f-8037-9815e2228839	2.0	426.0	43	admin.	single	secondary	False	True	False	cellular	76

You can see the 10 duplicated rows, where each unique row appears to have one duplicate. So, there are actually five rows that can be removed.

2. Remove the duplicated data.

- a) Scroll down and view the cell titled **Remove the duplicated data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_cleaned_final = \
2 users_cleaned_2[~users_cleaned_2.duplicated()]
3
4 users_cleaned_final[users_cleaned_final['user_id'] == \
5 'cba59442-af3c-41d7-a39c-0f9bffbba0660']
```

This will create a new DataFrame equal to the existing one, except for the identified duplicates. Also, you'll retrieve a specific user ID that was duplicated before to make sure it only appears once in the new dataset.

- c) Run the code cell.
 d) Examine the output.

	user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	campaign
15456	cba59442-af3c-41d7-a39c-0f9bffbba0660	2.0	1218.0	57	management	married	tertiary	True	True	False	cellular	317	6

This particular user ID only appears once, so it seems the duplicates have been successfully removed.

- e) Select the next code cell, then type the following:

```
1 users_cleaned_final.shape
```

- f) Run the code cell.

- g) Examine the output.

(45209, 19)

As expected, five rows have been removed from the overall dataset.

3. Keep this notebook open.

Word Embedding

Some datasets include textual content. Consider a dataset where one of the features is `Review`. This is a user's written review of the product they purchased. This feature is not quite categorical, and even though the order of words is important, it's not quite ordinal in the sense that one word is necessarily more important than another. You need some way to transform this data so that it's easier to analyze and more conducive to modeling.

Embedding is the process of condensing a language vocabulary into vectors of relatively small dimensions. For example, the average adult American knows the meaning of about 40,000 words in the English language. Rather than have one feature for every one of these 40,000 words, which would present a huge problem for performance, embedding represents each word as its own vector, usually no more than a few hundred dimensions. These vectors constitute the overall embedded space. A word's vector occupies a certain point within this space. When trained on certain models, particularly neural networks, words with similarities are placed close to each other in the embedded space so that the meaning of a clause or sentence is more easily identified.

Many practitioners before you have already trained useful word embeddings, so you should consider leveraging these pre-trained embeddings in your own projects. However, you may wish to train your own embeddings. There are several tools that you can use to do this. Some of the most popular include:

- **Word2vec**, which takes a large corpus of text as input and produces an embedded space in which each word is a vector. This is the traditional approach.
- **fastText**, an extension of Word2vec that partitions words into n -grams, a form of tokenization. For example, the n -grams for the word "phone" might be "pho," "hon," and "one." The vector for "phone" is the sum of these n -grams. The n determines the length of each partition, e.g., the example just given has partitions of length three—also called *trigrams*. If the partitions were two letters in length, they'd be *bigrams*, and so on. The n -gram approach has the advantage of generating more effective embeddings for rare words, as these words may have some n -grams in common with other, more common words.
- **TF-IDF**, or term frequency-inverse document frequency, represents the proportion of words that appear in each document, while also considering how many documents in the entire corpus contain those words.
- **GloVe**, or global vectors, a technique that incorporates matrix factorization to calculate the frequency of words within a certain context, like a document or corpus.

Text Data Transformation Techniques

There are several other techniques for handling text data that you should be aware of. The following table uses a line from Walt Whitman's poetry collection *Leaves of Grass* as an example:

'The shadows, gleams, up under the leaves of the old sycamore-trees'

Technique	Description
Bag-of-words	<p>A bag of words is a list of each individual word in a document without respect to grammar, punctuation, or any other language component that may be extraneous to the word itself. It is usually represented in a collection object like an array or dictionary. In the case of the latter, the key is the word and the value is the number of times it appears in the document.</p> <p>A bag of words using the example might produce a dictionary like:</p> <pre>{'the': 3, 'shadows': 1, 'gleams': 1, 'up': 1, 'under': 1, 'leaves': 1, 'of': 1, 'old': 1, 'sycamore': 1, 'trees': 1}</pre>
Tokenization	<p>Tokenization partitions a document into smaller units. In fact, n-grams are a form of tokenization. There is also word tokenization, where a document is simply broken up into words, and character tokenization, where each character is partitioned.</p> <p>An example of character tokenization applied to the word "gleams" and put into a list:</p> <pre>['g', 'l', 'e', 'a', 'm', 's']</pre>
Stop words	<p>A stop word is any word that is very common in text and whose inclusion typically provides little to no value. Words like "the," "a," "an," and so on, are usually excised from a document and put into a list of stop words. When an algorithm operates on text, it can refer to this list and ignore any such words. Other than universally common words, a stop word can also be any word that appears very frequently in the document in question. So, a list of stop words might be created by taking the frequency of all words in a document and extracting the top results.</p> <p>Considering the entire poetry collection rather than just a single line, the stop words from this line might be:</p> <pre>['the', 'up', 'under', 'of']</pre> <p>The word "leaves" is also very common, so it might be a candidate for a stop word. Though, because of its thematic importance, it may be best to leave it alone.</p>
Stemming	<p>Stemming removes the affix of a word in order to reduce the inflection of that word. In most cases this means a suffix, but stemming can also be applied to prefixes. The result of the operation is the word stem, the base form of the word itself. Suffixes include plurals like "s" and "es"; gerunds ("ing"); past tense ("ed"); adverbs ("ly"); and so on.</p> <p>If run through a stemming program, the word "shadows" becomes:</p> <pre>'shadow'</pre> <p>However, stemming is somewhat limited. For the most part, it simply chops off the suffix without regard to the true dictionary root word. For example, stemming the irregular plural "leaves" produces:</p> <pre>'leav'</pre>

Technique	Description
Lemmatization	<p>Lemmatization addresses the shortcomings of stemming through morphology; that is, the analysis of word parts and structures and how they change. Lemmatization attempts to derive the <i>lemma</i>, or the canonical dictionary form of a word. The lemma can be derived through cutting off a suffix, like in stemming, but also through more intelligent methods, like analyzing word tense and grammatical mood.</p> <p>When "leaves" is lemmatized, the outcome will be:</p> <pre>'leaf'</pre> <p>This is the base dictionary form of the word. However, lemmatization isn't perfect, as "leaves" in a different context might be a verb whose lemma is actually "leave"—to depart.</p>

Image Data Representation

Just like textual data, image data must be transformed before it can be used effectively. Typically, raw images are represented as an array of numeric values, where each cell represents a pixel. In a grayscale image, the number in a cell represents that pixel's intensity, where 0 is black and 255 is white. Simple color images are often represented with RGB (red, green, blue) values for each pixel. Most deep learning algorithms like convolutional neural networks (CNNs) compress this kind of data to eliminate "noise" in the image and only retain the features of the image that are most useful for determining patterns. This is called a *latent representation* of an image because the patterns being learned are "hidden" within the network. When building a CNN, you can extract these latent representations from the neural network as one-dimensional vectors. The vectors include all of the relevant features and can therefore be used in place of the original image in your dataset.

You might want to extract these vectors if you plan to input the image data into another model, as it will speed up the process and potentially improve results if the new model doesn't create its own latent representations. You can also measure the similarity between feature vectors using a distance metric like cosine similarity, which calculates the cosine of the angles between each vector. As the angle decreases, cosine increases, as does the level of similarity. So, to streamline your image data, you might remove any feature vector from the dataset that is highly similar to another feature vector.

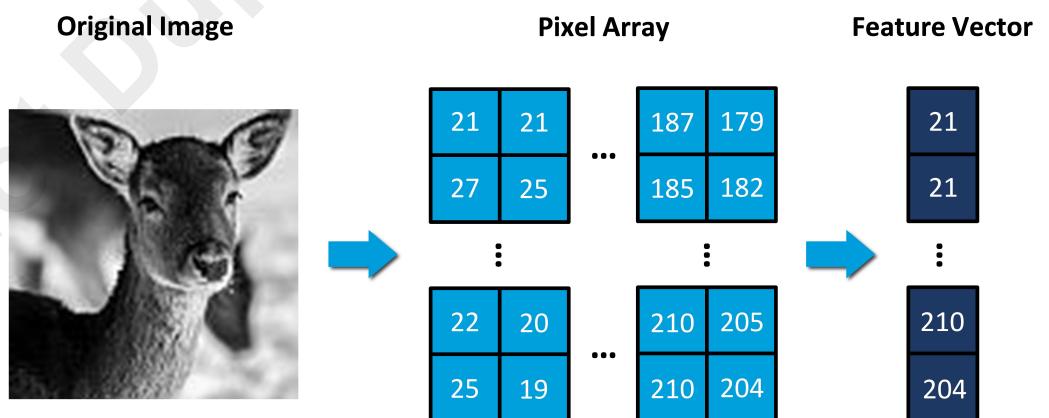


Figure 2–4: Extracting a feature vector from a grayscale pixel array.

Guidelines for Transforming Data

Follow these guidelines when transforming data.

Transform Data

When transforming data:

- Keep in mind that the goal of transforming tabular data is to make data formats uniform between columns and cells in a given column.
- Keep in mind that most transformations can be done with third-party tools. For example, ad hoc scripts in Python, R, or other languages; custom-written software; or even general-purpose worksheets like Excel with custom formulas or VBA macros.
- Include documentation, tests, and versioning for conversion scripts. It is likely that these conversion scripts will be used multiple times for long-running projects.
- Maintain and update conversion scripts as needed. They should account for changes to the format of incoming data.
- Ensure that dates have a common representation. It's common to use ISO 8601, but others are valid as well.
- Determine whether or not currency should have leading or trailing symbols.
- Add a column that indicates a currency code using a standard like ISO 4217.
- Remove symbols for grouping digits by thousands, millions, etc.
- Consider expanding scientific notation to use a floating point representation.
- Keep in mind that some numbers have a prefix or suffix indicating magnitude.
 - "K" and "m" can mean thousands.
 - "BP" means "basis points," i.e., 100th of 1%.
 - "L" or "lakh" means 100,000 in Indian numbering systems, also written as 1,00,000.
- Consider that most data science tools assume an ASCII character set. If the content is in a different character representation like EBCDIC or Unicode, the text should be transformed to ASCII. Some tools will automatically transform the characters when given a "from" and "to" character set. If this feature is not available, there are third-party applications or libraries available that can do these character set transformations.
- Consider that strings can be surrounded by:
 - Single or double straight quotes (' and ").
 - Chevrons/guillemets (« and »).
 - Single or double smart quotes (‘ ’ and “ ”) inserted by some word processing programs.



Note: Most data science tools will recognize and process strings surrounded by straight quotes, but you may need to transform guillemets or smart quotes into straight quotes. Smart quotes can be particularly challenging to transform because they are often hard to distinguish from straight quotes.

ACTIVITY 2-7

Handling Textual Data

Data Files

/home/student/CDSP/Text/Handling Textual Data.ipynb
 /home/student/CDSP/Text/data/consumer_loan_complaints.csv

Scenario

The consumer complaints data that you worked with earlier may still be of value to the project, especially if the team plans to develop natural language processing (NLP) models sometime in the future. However, handling the textual data inside this file requires a much different approach than the numeric and categorical data that you've mostly been working with. You need a way to process the text so that it's more conducive to analysis and machine learning. There are many techniques for doing so, and you'll apply several of them to the complaint data.

1. Open the notebook.

- In the Jupyter Notebook web client, navigate to the **CDSP/Text** directory.



Note: Make sure that the **Extracting, Transforming, and Loading Data.ipynb** notebook is still open and that you're opening this new notebook in a different tab.

- Select **Handling Textual Data.ipynb** to open it.

2. Import the relevant software libraries.

- View the cell titled **Import software libraries**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.

3. Read and preview the text data.

- Scroll down and view the cell titled **Read and preview the text data**, then select the code cell below it.
- In the code cell, type the following:

```
1 complaints_data = pd.read_csv('data/consumer_loan_complaints.csv')
2
3 complaints_data.head()
```

You'll load the data and get another look at its structure.

- Run the code cell.

- d) Examine the output.

	user_id	Date received	Product	Issue	Consumer complaint narrative	State	ZIP code	Submitted via	Date sent to company	Company response to consumer	Timely response?	Consumer disputed?	Complaint ID
0	44feldad-7045-4be5-890e-12e84ae6fdc9	01/27/2016	Consumer Loan	Account terms and changes	NaN	AL	35180	Phone	01/27/2016	Closed with explanation	Yes	No	1760486
1	c49d5d60-909f-406b-b7ff-511431cb650b	08/26/2014	Consumer Loan	Account terms and changes	NaN	NC	278XX	Phone	08/29/2014	Closed with non-monetary relief	Yes	No	1001740
2	9b2cd5d2-900e-4052-831f-6489f6d568af	08/22/2012	Consumer Loan	Account terms and changes	NaN	TN	37205	Referral	08/23/2012	Closed with non-monetary relief	Yes	No	140039
3	b7e5b324-268a-4502-81a1-1a025673c2a0	05/07/2013	Consumer Loan	Problems when you are unable to pay	NaN	OH	43081	Web	05/08/2013	Closed with explanation	Yes	Yes	401541
4	684eeb4c-c9c3-4a97-8213-f3962a6c0aba	06/15/2016	Consumer Loan	Managing the line of credit	NaN	NC	27216	Phone	09/08/2016	Closed with non-monetary relief	Yes	No	1970341

Recall that each user that filed a complaint is listed in a row, and each column is a different aspect of the user or their complaint.

4. Check the shape of the data.

- a) Scroll down and view the cell titled **Check the shape of the data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 | complaints_data.shape
```

- c) Run the code cell.
 d) Examine the output.

```
(1824, 13)
```

The dataset has 1,824 rows and 13 columns.

5. Retrieve information about the data.

- a) Scroll down and view the cell titled **Retrieve information about the data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 | complaints_data.info()
```

- c) Run the code cell.

- d) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1824 entries, 0 to 1823
Data columns (total 13 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   user_id          1824 non-null   object  
 1   Date received    1824 non-null   object  
 2   Product          1824 non-null   object  
 3   Issue            1824 non-null   object  
 4   Consumer complaint narrative 44 non-null   object  
 5   State             1801 non-null   object  
 6   ZIP code          1789 non-null   object  
 7   Submitted via    1824 non-null   object  
 8   Date sent to company 1824 non-null   object  
 9   Company response to consumer 1824 non-null   object  
 10  Timely response? 1824 non-null   object  
 11  Consumer disputed? 1824 non-null   object  
 12  Complaint ID     1824 non-null   int64  
dtypes: int64(1), object(12)
memory usage: 185.4+ KB
```

There are only 44 values in the Consumer complaint narrative feature that aren't null. In other words, most of this data is missing.

- e) Select the next code cell, then type the following:

```
1 complaints_data.Issue.value_counts()
```

- f) Run the code cell.
g) Examine the output.

Managing the line of credit	806
Account terms and changes	484
Shopping for a line of credit	301
Problems when you are unable to pay	233
Name: Issue, dtype: int64	

There are several different types of issues, which makes this a good candidate for encoding, so there's no need to perform any textual processing on this feature.

- h) Select the next code cell, then type the following:

```
1 complaints_data['Company response to consumer'].value_counts()
```

- i) Run the code cell.
j) Examine the output.

Closed with explanation	1291
Closed with non-monetary relief	184
Closed with monetary relief	182
Closed without relief	75
Closed	65
Closed with relief	19
Untimely response	8
Name: Company response to consumer, dtype: int64	

Likewise, this feature can be encoded since it's just categorical in nature. Ultimately, the Consumer complaint narrative feature is the only one that needs special handling.

6. Extract a subset of data to consider only consumer complaints.

- Scroll down and view the cell titled **Extract a subset of data to consider only consumer complaints**, then select the code cell below it.
- In the code cell, type the following:

```
1 print('Number of users with no complaints data:',  
2      complaints_data['Consumer complaint narrative'].isnull().sum())
```

- Run the code cell.
- Examine the output.

Number of users with no complaints data: 1780

As expected, most records weren't filed with the actual text of the user's complaint.

- Select the next code cell, then type the following:

```
1 # Remove records with missing complaint narratives.  
2  
3 text_data = complaints_data[~complaints_data \  
4                               ['Consumer complaint narrative'].isnull()] \  
5                               [[user_id', 'Consumer complaint narrative']]  
6  
7 text_data.head(n = 3)
```

You just need to focus on the records that actually have a complaint narrative, so you'll reduce the dataset.



Note: The first character within the bracket is a tilde (~).

- Run the code cell.
- Examine the output.

	user_id	Consumer complaint narrative
53	1a1448a4-bfe5-455f-bc29-dc79ec5fb2c0	NONE OF YOUR " MY LOAN IS A " below apply to ...
59	5fede48c-096e-4f82-997d-8229007d8318	XX/XX/2014 I received a letter from the IRS st...
65	fd9fc5ff-19bc-424c-880e-c159c110d21f	This was a revolving account in which I paid W...

The reduced dataset has the user's ID and the text of their complaint.

- Select the next code cell, then type the following:

```
1 text_data.shape
```

- Run the code cell.
- Examine the output.

(44, 2)

As expected, there are 44 rows and 2 columns.

7. Preview an example of the consumer complaints.

- Scroll down and view the cell titled **Preview an example of the consumer complaints**, then select the code cell below it.
- In the code cell, type the following:

```
1 sample_text = text_data['Consumer complaint narrative'].iloc[0]
2 sample_text
```

This code will print the first complaint text.

- Run the code cell.
- Examine the output.

```
'NONE OF YOUR " MY LOAN IS A \'\' below apply to this situation! This was a car loan but the company is providing
fraudulent information this is damaging my credit! \n\nRE : MidAtlantic Finance Company Account No. XXXX - NOT TO
BE CONFUSED with my current MAF loan MidAtlantic Finance Company has reported several false items to all XXXX cred
it reporting agencies, and continues to do so. It is damaging my credit so much so that I was told I did n't qual
ify for a mortgage. \n\nMost recently, I settled this account per agreement on XXXX XXXX, XXXX, yet MAF reported i
t is a payment on the amount claimed owed ( which has been disputed since XXXX XXXX ). But that is just the most r
ecent false information that was reported. It is showing a debt of {$250.00} per month along with XXXX different a
mounts charged off of the {$950.00} ( plus interest ) and another {$5100.00} that INCLUDES the {$950.00}. Please r
efer to the following as I NEVER owed MAF {$8100.00} as it reported. That was the original amount financed in XXXX
XXXX with XXXX XXXX, and I made payments of {$250.00} a month through XXXX XXXX to XXXX XXXX. \n\n Car was purcha
sed in XXXX XXXX and was financed IN HOUSE until XXXX XXXX per MAF 's statement of XXXX XXXX, XXXX, with my first
payment being due to MAF XXXX XXXX. Therefore, I was not responsible for any prior late payments ( MAF recorded my
first delinquency as XXXX XXXX ) and that should never have been on my credit report. I do not know why or HOW it
can be considered owned by MAF since YYYY YYYY \n\n There was NEVER a charge off of section 601 for the following
```

The complaint is full of characters like punctuation, numbers, and common words. You'll need to streamline this text to make it more amenable to analysis.

8. Tokenize the sample text into sentences.

- Scroll down and view the cell titled **Tokenize the sample text into sentences**, then select the code cell below it.
- In the code cell, type the following:

```
1 nlp = spacy.load('/home/student/spacy_data/' +
2                     'en_core_web_sm/en_core_web_sm-3.0.0/')
3
4 document = nlp(sample_text)
```

This code uses a third-party text processing library called spaCy to create a document from the sample text.

- Run the code cell.
- Select the next code cell, then type the following:

```
1 for sentence in document.sents:
2     print(sentence)
```

This code will tokenize the entire document text into sentences.

- Run the code cell.

- f) Examine the output.

```
NONE OF YOUR " MY LOAN IS A '' below apply to this situation!
This was a car loan but the company is providing fraudulent information this is damaging my credit!

RE : MidAtlantic Finance Company Account
No.
XXXX - NOT TO BE CONFUSED with my current MAF loan MidAtlantic Finance Company has reported several false items to
all XXXX credit reporting agencies, and continues to do so.
It is damaging my credit so much so that I was told I did n't qualify for a mortgage.

Most recently, I settled this account per agreement on XXXX XXXX, XXXX, yet MAF reported it is a payment on the am
ount claimed owed ( which has been disputed since XXXX XXXX ).  

But that is just the most recent false information that was reported.  

It is showing a debt of $250.00 per month along with XXXX different amounts charged off of the $950.00 / plus
```

Each sentence identified by spaCy is printed. Note that the line breaks (\n characters in the document) are being treated as different sentences.

9. Tokenize the sentences into words.

- a) Scroll down and view the cell titled **Tokenize the sentences into words**, then select the code cell below it.
- b) In the code cell, examine the following:

```
1 sentence = nlp('It is showing a debt of {$250.00} per month along '
2                 'with XXXX different amounts charged off of the '
3                 '{{$950.00}} ( plus interest ) and another {{$5100.00}} '
4                 'that INCLUDES the {{$950.00}}.')
```

To demonstrate the tokenization of individual words, this code is taking a single sentence from the overall document.

- c) Run the code cell.
- d) Select the next code cell, then type the following:

```
1 for token in sentence:
2     print(token.text)
```

- e) Run the code cell.
- f) Examine the output.

```
It
is
showing
a
debt
of
{
$ 
250.00
}
per
month
along
with
XXXX
different
```

Each individual "word" (as defined by spaCy) is printed. Note that numbers and punctuation are counted as words.

10. Identify the parts of speech for each token.

- Scroll down and view the cell titled **Identify the parts of speech for each token**, then select the code cell below it.
- In the code cell, type the following:

```

1 pos = []
2
3 for token in sentence:
4     pos.append({'Word': token,
5                  'Part of Speech': token.pos_
6                         })
7
8 pd.DataFrame(pos)

```

Text processors like spaCy can actually identify each token's role in a sentence, including non-words like numbers and punctuation.

- Run the code cell.
- Examine the output.

	Word	Part of Speech
0	It	PRON
1	is	AUX
2	showing	VERB
3	a	DET
4	debt	NOUN
5	of	ADP
6	{	PUNCT
7	\$	SYM
8	250.00	NUM
9	}	PUNCT
10	per	ADP

The table lists what part of speech each token is. For example, It is a pronoun, is is an auxiliary verb (a verb that affects the tenses and moods of other verbs), { is punctuation, 250.00 is a number, and so on.

11. Identify stop words.

- Scroll down and view the cell titled **Identify stop words**, then select the code cell below it.
- In the code cell, type the following:

```

1 stop = []
2
3 for token in sentence:
4     stop.append({'Word': token,
5                  'Stop Word?': token.is_stop
6                         })
7
8 pd.DataFrame(stop)

```

This code will identify whether or not a token qualifies as a stop word, or a word that is so common that it should be excluded from the text so as not to exert undue influence.

- Run the code cell.

- d) Examine the output.

Word	Stop Word?
0 It	True
1 is	True
2 showing	False
3 a	True
4 debt	False
5 of	True
6 {	False
7 \$	False
8 250.00	False
9 \	False

Words like `It` and `is` are considered stop words, whereas `showing` is not.

12. Stem the text.

- a) Scroll down and view the cell titled **Stem the text**, then select the code cell below it.
 b) In the code cell, examine the following:

```
1 text = 'This was a car loan but the company is providing ' \
2     'fraudulent information this is damaging my credit!'
3
4 print(word_tokenize(text))
```

This is another sample sentence from the larger document. You're going to be using the Natural Language Toolkit (NLTK) to perform stemming on this tokenized sample. Stemming obtains the affix of a word to reduce that word to a base form, making it easier to work with.

- c) Run the code cell.
 d) Examine the output.

```
['This', 'was', 'a', 'car', 'loan', 'but', 'the', 'company', 'is', 'providing', 'fraudulent', 'information', 'this',
 'is', 'damaging', 'my', 'credit', '!']
```

Each word is an item in a list.

- e) Select the next code cell, then type the following:

```
1 stemmer = SnowballStemmer(language = 'english')
2
3 for token in word_tokenize(text):
4     print(token, '-->', stemmer.stem(token))
```

There are multiple stemming algorithms in NLTK, and each one can produce different results. In this case you're using `SnowballStemmer()` to see how it does.

- f) Run the code cell.

- g) Examine the output.

```
This --> this
was --> was
a --> a
car --> car
loan --> loan
but --> but
the --> the
company --> compani
is --> is
providing --> provid
fraudulent --> fraudul
information --> inform
this --> this
is --> is
damaging --> damag
my --> my
credit --> credit
! --> !
```

You can see that some tokens were properly stemmed, but that the resulting stem is not quite a real word. For example, company became compani, and damaging became damag. That's why a more advanced technique like lemmatization is usually preferred.

13. Lemmatize the text.

- Scroll down and view the cell titled **Lemmatize the text**, then select the code cell below it.
- In the code cell, type the following:

```
1 parsed_text = nlp(text)
2
3 for token in parsed_text:
4     print(token, '-->', token.lemma_)
```

This code uses the same sentence sample, but this time retrieves each word's lemma. A lemma is the canonical dictionary form of a word.

- Run the code cell.

- d) Examine the output.

```
This --> this
was --> be
a --> a
car --> car
loan --> loan
but --> but
the --> the
company --> company
is --> be
providing --> provide
fraudulent --> fraudulent
information --> information
this --> this
is --> be
damaging --> damage
my --> my
credit --> credit
! --> !
```

The lemmatization did a better job of obtaining root words. For example, company didn't change because it's already in the canonical form, and damaging became damage, which is the canonical form. Also, verbs like was and is became be.

14. Transform the text.

- a) Scroll down and view the cell titled **Transform the text**, then select the code cell below it.
 b) In the code cell, examine the following:

```
1 def spacy_cleaner(original_text):
2     """Cleans text data to be processed.
3     Removes punctuation, whitespace, numbers, stopwords from the text
4     and lemmatizes each token."""
5
6     nlp = spacy.load('en_core_web_sm')
7
8     final_tokens = []
9     parsed_text = nlp(original_text)
10
11    for token in parsed_text:
12        if token.is_punct or token.is_space or token.like_num or token.is_stop:
13            pass
14        else:
15            if token.lemma_ == '-PRON-':
16                final_tokens.append(str(token)) # Keep pronouns as they are.
17            else:
18                sc_removed = re.sub('[^a-zA-Z]', '', str(token.lemma_))
19                if len(sc_removed) > 1:
20                    final_tokens.append(sc_removed)
21            joined = ' '.join(final_tokens)
22            preprocessed_text = re.sub(r'(\.)\1+', r'\1\1', joined)
23
24    return preprocessed_text
```

Now it's time to actually transform the text data. This function will:

- Use spaCy to tokenize the text.
- Remove punctuation, spaces, numbers, and stop words.
- Obtain the lemma of each word, except for pronouns, which will be kept as is.
- Replace each word with the lemma using a regular expression.
- Remove any other punctuation like parentheses and backslashes using a regular expression.
- Compile the results in a string.

- c) Run the code cell.

- d) Select the next code cell, then type the following:

```
1 # Apply transformation to sample.
2
3 spacy_cleaner(sample_text)
```

First you'll try the transformation out on the single complaint text example.

- e) Run the code cell.
f) Examine the output.

```
'LOAN apply situation car loan company provide fraudulent information damage credit MidAtlantic Finance Company Account xx confused current MAF loan MidAtlantic Finance Company report false item xx credit reporting agency continue damage credit tell qualify mortgage recently settle account agreement XX XX XX MAF report payment claim owe dispute XX XX recent false information report show debt month xx different amount charge plus interest include refer following owe MAF report original finance XX xx XX XX payment month XX xx xx XX car purchase XX XX finance HOUSE X X XX MAF statement XX XX XX payment MAF XX XX responsible prior late payment MAF record delinquency xx XX credit report know consider own MAF XX XX charge follow reason car total XX XX XX pay xx payment plus additional interest fee XX XX XX insurance company pay XX XX XX leave balance MAF dispute XX XX give payoff XX xx expiration date dispute month give finally MAF send accounting XX xx support claim payment wrongfully charge amount XX xx XX account p av MAF letter date xx yy yy MAF comply verbal agreement representative xx xx remove negative credit apply payment
```

As expected, the text of the review has been streamlined.

- g) Select the next code cell, then type the following:

```
1 # Compare to sample before transformation.
2
3 sample_text
```

- h) Run the code cell.
i) Examine the output.

```
'NONE OF YOUR " MY LOAN IS A \'\' below apply to this situation! This was a car loan but the company is providing fraudulent information this is damaging my credit! \n\nRE : MidAtlantic Finance Company Account No. XXXX - NOT TO BE CONFUSED with my current MAF loan MidAtlantic Finance Company has reported several false items to all XXXX credit reporting agencies, and continues to do so. It is damaging my credit so much so that I was told I did n't qualify for a mortgage. \n\nMost recently, I settled this account per agreement on XXXX XXXX, XXXX, yet MAF reported it is a payment on the amount claimed owed ( which has been disputed since XXXX XXXX ). But that is just the most recent false information that was reported. It is showing a debt of {$250.00} per month along with XXXX different amounts charged off of the {$950.00} ( plus interest ) and another {$5100.00} that INCLUDES the {$950.00}. Please refer to the following as I NEVER owed MAF {$8100.00} as it reported. That was the original amount financed in XXXX XXXX with XXXX XXXX, and I made payments of {$250.00} a month through XXXX XXXX to XXXX XXXX. \n\n Car was purchased in YYYY YYYY and was financed IN HOUSE until YYYY YYYY for MAF 's statement of YYYY YYYY YYYY with my first
```

You can see significant differences between the original form of the complaint and the complaint after it underwent text processing.

- j) Select the next code cell, then type the following:

```
1 # Apply transformation to entire dataset.
2
3 text_data['consumer_complaints_cleaned'] = \
4 text_data['Consumer complaint narrative'].apply(lambda x: spacy_cleaner(x))
5
6 text_data.head(n = 3)
```

You'll apply the transformation to the entire dataset.

- k) Run the code cell.



Note: This may take a few moments to execute.

- I) Examine the output.

	user_id	Consumer complaint narrative	consumer_complaints_cleaned
53	1a1448a4-bfe5-455f-bc29-dc79ec5fb2c0	NONE OF YOUR " MY LOAN IS A " below apply to ... LOAN apply situation car loan company provide ...	
59	5fede48c-096e-4f82-997d-8229007d8318	XX/XX/2014 I received a letter from the IRS st... XX XX receive letter IRS state owe agency ask ...	
65	fd9fc5ff-19bc-424c-880e-c159c110d21f	This was a revolving account in which I paid W... revolving account pay Wells Fargo National Ban...	

The first three records are printed, as is the original narrative and the transformed narrative for each complaint.

15. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel→Shutdown**.
- b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
- c) Close the **Handling Textual Data.ipynb** tab in Firefox, but switch back to the **Extracting, Transforming, and Loading Data.ipynb** notebook.

TOPIC C

Load Data

The last step in the ETL process is loading. In this topic, you'll take the data you transformed and put it into a destination format and location, where it will be ready for you to work on as the project progresses.

Data Loading Considerations

Data loading in this context doesn't just mean pulling raw data into a working environment. It refers to putting cleaned data into an ultimate destination from which the rest of the data science process can take place. In other words, this is the prepared dataset that you'll use to analyze your domain problem and develop models to address that problem.

Before you begin loading data into a destination, you need to consider a few things:

- **Choice of data format.** The dataset must be stored in some form, whether it's a file, a database server, or some other repository. The choice of format will depend on what tools are available to you, what database technologies you're familiar with, and what solutions are already in place within the organization's IT infrastructure.
- **Storage capacity.** Large datasets, particularly those that qualify as big data, will need to be loaded into a destination that has high storage capacity. You should also consider that your storage needs may be elastic, meaning that they shrink or grow over time. For example, you might add more data to the dataset later, which will require more storage capacity.
- **Memory capacity.** Like any other computer operation, data analysis and modeling takes place within memory. However, these tasks tend to be very memory intensive, and can quickly consume all available memory in a computer or cluster. So you need to make sure you have enough memory to fit your dataset. This includes memory like RAM that is accessed directly by the CPU, but it can also include GPU memory, which is a common bottleneck when training machine learning models. You might be able to avoid out-of-memory errors in large datasets by dividing the overall data into smaller chunks that you load and analyze on an individual basis.
- **Pipeline integration.** If you have a pipeline that helps automate data science tasks for you, you'll need to ensure that your destination dataset is compatible with that pipeline. It must be able to load data into the desired format without a lot of manual guidance.
- **Collaboration.** As a member of a data science team, you need to be able to share data between multiple team members. Although you may be comfortable with a certain data loading process, your colleagues may not. You need to work with your team to develop a solution that benefits everyone, not just you.

Data Loading: Databases

One common destination for a prepared dataset is a database. That might be an SQL database for relational data, or a NoSQL database for non-relational data. In either case, databases are usually the preferred choice for larger organizations and/or data science projects that deal with large volumes of data. Databases are designed to be as efficient as possible in both storing data and manipulating that data via queries. The actual process of loading data into a database usually takes the form of one or more queries that can select all of the relevant data in your workspace and update or insert that data into the destination database. This can be relatively easy if the data is prepared well, but you may run into difficulties if the current data structure doesn't match the destination database's structure.

Most databases run on one or more servers. This has several advantages, including the ability for the organization to ensure a certain level of availability and reliability of data. Data is centralized and monitored continuously, so IT professionals like database administrators can guarantee access

within a network. Compare this to storing data in files, which can fracture that data among multiple host machines and is difficult to monitor. Database servers also support authentication and authorization mechanisms so that only users who absolutely need access to the data are given permission. Likewise, strict access controls help uphold the integrity of data, preventing tampering or accidental modification. Database servers also commonly implement encryption services to uphold the confidentiality of data.

It's important that you work with your IT team to set up databases into which you can load the data. You should clearly explain your needs as far as capacity, access rights, and the overall structure or technology of the database.

Guidelines for Loading Data into Databases

Follow these guidelines when loading data into databases.

Load Data into Databases

When loading data into databases:

- Look at data files and the targeted database's schema to determine which database tables and columns the data will be loaded to.
- Decide where every piece of data in your data file will be loaded to.
- Load data with a custom-written program, data science tool, or a bulk-loading program like mysqldump for MySQL or SQL*Loader for Oracle Database.
- Consider writing a small program to load data if it will be loaded regularly.
- Collect and analyze missing data or erroneous data and decide if this will violate any data integrity rules and make data inconsistent.

ACTIVITY 2–8

Loading Data into a Database

Before You Begin

Extracting, Transforming, and Loading Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Load data into an SQL database**. Select **Cell→Run All Above**.

Scenario

Now that the GCNB data is in a relatively clean state, you want to begin packaging it for the forthcoming analysis and modeling tasks. There are many formats that you can load this data into; instead of choosing just one, you'll try out multiple formats to get a feel for how they differ in terms of storage and integration. First, you'll load the dataset into an SQL database.

1. Load data into an SQL database.

- Scroll down and view the cell titled **Load data into an SQL database**, then select the code cell below it.
- In the code cell, type the following:

```

1 conn = sqlite3.connect('users_data_cleaned.db')
2
3 users_cleaned_final.to_sql('users_cleaned_final',
4                             conn,
5                             if_exists = 'replace',
6                             index = False)

```

First, you're connecting to an SQLite database file that doesn't yet exist, but will be the save point for the dataset. Then, the `to_sql()` function actually saves the data to the open database file.

- Run the code cell.

2. Confirm that data was loaded into the database.

- Scroll down and view the cell titled **Confirm that data was loaded into the database**, then select the code cell below it.
- In the code cell, type the following:

```

1 query = 'SELECT * FROM users_cleaned_final'
2
3 pd.read_sql(query, conn).head()

```

- Run the code cell.

- d) Examine the output.

	user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	campaign	p
0	9231c446-cb16-4b2b-a7f-ddfc8b25aaf6	3.0	2143.0	58	management	married	tertiary	0	1	0	None	261	1	
1	bb92765a-08de-49e3-b432-496524b39157	NaN	NaN	44	technician	single	secondary	0	1	0	None	151	1	
2	573de577-49ef-42b9-83da-d3cfb817b5c1	2.0	2.0	33	entrepreneur	married	secondary	0	1	1	None	76	1	
3	d6b66b9d-7c8f-4257-a682-e136f640b7e3	NaN	NaN	47	blue-collar	married	None	0	1	0	None	92	1	
4	fade0b20-7594-4d9a-84cd-c02f79b1b526	1.0	1.0	33	None	single	None	0	0	0	None	198	1	

The first few rows of the cleaned dataset appear, indicating that the data was successfully loaded into the SQLite database.

3. Close the database connection.

- Scroll down and view the cell titled **Close the database connection**, then select the code cell below it.
- In the code cell, type the following:

```
1 conn.close()
```

- Run the code cell.

4. Confirm the database file was saved to the local drive.

- In Firefox, select the **CDSP/ETL** tab to view the file structure.
- Verify that a file called **users_data_cleaned.db** exists.
You can't open this file directly, but you already confirmed that you were able to load it into Jupyter Notebook.

5. Return to the active notebook and keep it open.

Data Loading: DataFrames

Not every data science project requires a database architecture for loading data. In fact, you may find that keeping data in a format that closely integrates with your programming environment is the best choice. There's no need to issue queries and send data over the network to a database server, since all of the data is already in an acceptable form. For example, in a Python environment, a lot of cleaning and manipulating of data during the ETL process is done through a pandas DataFrame object. You can keep the data in that object format if it suits your needs.

Of course, you'll want to avoid keeping that data in an executable state only. Otherwise, system failures or power outages can cause you to lose all of your data. Thankfully, libraries like pandas enable you to save your DataFrame object as a binary file. For example, Python has a binary file format called a *pickle* file. The file can live on your local drive, or you can move it to some other storage location if you prefer. When you're ready to work with the data again, you simply load the binary file back in your programming environment, and your DataFrame will appear exactly as you left it. That's actually one of the advantages that a binary file has over a simple text file—it is an exact representation of the DataFrame at the time of saving. There's no need to parse the data or do anything else to get it back to the desired state.

Of course, loading data into a DataFrame has its disadvantages. Compared to a database, you run into the problem of data being fractured and difficult to manage. You can certainly share your binary files with other team members, but there won't necessarily be an authoritative source of data

like you'd get by connecting to a database server. Binary files like pickle files can also pose a security risk if they come from an untrusted source, so you need to be careful about what files you open. Likewise, there's no centralized authentication or authorization mechanisms in place to keep a binary file from being accessed by unwanted users.

Compared to text files, binary files introduce compatibility issues. For example, a binary object created from a pandas `DataFrame` in Python won't open in a different programming environment without some additional effort. Text files, on the other hand, are extremely portable and can be read by just about any data science tool.

Guidelines for Loading Data into `DataFrames`

Follow these guidelines when loading data into `DataFrames`.

Load Data into `DataFrames`

When loading data into `DataFrames`:

- Use one `DataFrame` for each dataset being loaded.
- Decide which columns are in common between datasets so data from different datasets can be combined.
- Combine `DataFrames` after loading based on joined columns.
- Keep in mind that most data science tools read all data from a file into memory at once, potentially causing slowness or errors when files are very large.

ACTIVITY 2–9

Loading Data into a DataFrame

Before You Begin

Extracting, Transforming, and Loading Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Write the DataFrame as a pickle file**. Select **Cell→Run All Above**.

Scenario

SQL databases are common storage platforms for relational datasets, but they aren't the only option available to you. In fact, if you plan to do a lot of work in a programming language like Python, you may want to preserve your data in a DataFrame structure so that it's easier to read from and write to. This is possible by saving the DataFrame as a binary pickle file, which you'll do in this activity.

1. Write the DataFrame as a pickle file.

- Scroll down and view the cell titled **Write the DataFrame as a pickle file**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_cleaned_final.to_pickle('users_data_cleaned.pickle')
```

- Run the code cell.
- Pickle files are in a binary format and preserve the content and structure of a DataFrame.

2. Confirm that the data was written to the pickle file.

- Scroll down and view the cell titled **Confirm that the data was written to the pickle file**, then select the code cell below it.
- In the code cell, type the following:

```
1 pd.read_pickle('users_data_cleaned.pickle').head()
```

- Run the code cell.

- d) Examine the output.

		user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	campaign	
0		9231c446-cb16-4b2b-a7f7-ddfc8b25aa6		3.0	2143.0	58	management	married	tertiary	False	True	False	None	261	1
1		bb92765a-08de-4963-b432-496524639157	NaN		Nan	44	technician	single	secondary	False	True	False	None	151	1
2		573de577-49ef-42b9-83da-d3cfb817b5c1		2.0	2.0	33	entrepreneur	married	secondary	False	True	True	None	76	1
3		d6b669a9-7c8f-4257-a682-e136f640b7e3	NaN		Nan	47	blue-collar	married	None	False	True	False	None	92	1
4		fade0b20-7594-4d9a-84cd-c02f79b1b526		1.0	1.0	33	None	single	None	False	False	False	None	198	1

As expected, the DataFrame was loaded from the pickle file with its structure intact.

- e) Select the next code cell, then type the following:

```
1 pd.read_pickle('users_data_cleaned.pickle').info()
```

- f) Run the code cell.
g) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45209 entries, 0 to 45215
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          45209 non-null   object 
 1   number_transactions 35210 non-null   float64 
 2   total_amount_usd  35210 non-null   float64 
 3   age              45209 non-null   int64  
 4   job              44921 non-null   object 
 5   marital           45209 non-null   object 
 6   education         43352 non-null   object 
 7   default            45209 non-null   bool   
 8   housing            45209 non-null   bool   
 9   loan              45209 non-null   bool   
 10  contact           32191 non-null   object 
 11  duration           45209 non-null   int64  
 12  campaign           45209 non-null   int64  
 13  pdays             45209 non-null   int64  
 14  previous            45209 non-null   int64  
 15  poutcome           8252 non-null   object 
 16  term_deposit        45209 non-null   bool   
 17  date_joined        45179 non-null   datetime64[ns]
 18  device              45115 non-null   object 
dtypes: bool(4), datetime64[ns](1), float64(2), int64(5), object(7)
memory usage: 5.7+ MB
```

As you can see, the date_joined column is still in datetime format. No additional parsing or querying is necessary when loading a DataFrame directly from a pickle file.

3. Confirm the pickle file was saved to the local drive.

- a) In Firefox, select the CDSP/ETL tab to view the file structure.
- b) Verify that a file called users_data_cleaned.pickle exists.

You can't open this file directly, but you already confirmed that you were able to load it into Jupyter Notebook.

4. Return to the active notebook and keep it open.

Data Loading: Text Files

Loading data into text files, a process also called exporting data, is an alternative to using databases and binary files. It's less common when dealing with large volumes of data, or data that is very complex. Text files are generalized to hold any type of character-encoded values and therefore are not designed to accommodate any specific data structure. However, you can implement an ostensible structure in a text file by separating values with delimiters, as with a CSV file that uses commas to delimit values.

The flexibility of a text file can be a strength, since they are readable by just about any person or program and can be shared across many different operating systems and environments. However, there are many clear disadvantages. First, the lack of a true structure means that text files may be searched, but not queried. You won't have an easy time of extracting specific portions of data from a raw text file. Another disadvantage is that, unlike a database or binary file, the values in a text file are not formatted in a deliberate way. To a text file, all of the data is flat; if you want a date value to take on a datetime type, you need to parse it in some other environment. Text files also struggle to efficiently store large datasets. Text editors, for example, can struggle to open text files with tens of thousands of records or more. Programming libraries do a better job of reading data from text files, but the data is still not read very efficiently. From a security standpoint, text files can introduce similar risks to binary files. They have no inherent authentication and authorization mechanisms, and they are likely to be scattered across many machines instead of centralized.

Ultimately, you should avoid loading data into flat text files unless it is relatively simple, low in volume, and doesn't include sensitive information.



Note: File formats like XML and JSON can also be considered text files, but they differ from flat text files in that they have a more well-defined structure that is meant to be interpreted by a software program.

Guidelines for Loading Data into Text Files

Follow these guidelines when loading data into text files.

Load Data into Text Files

When loading data into text files:

- Choose the file format that is best for your needs. Common text-based formats are CSV (comma-separated values), JSON (JavaScript Object Notation), and XLS/X (Excel).
- Prefer CSV for the widest compatibility with all applications. Nearly every application and data science workbench supports importing and exporting CSV files.
- Consider the advantages and disadvantages of CSV files:
 - **Advantages:** near-universal compatibility for importing and exporting, excellent library and language support, easy to parse, human readable, easy to edit, and forgiving of some minor errors in formatting.
 - **Disadvantages:** not very space efficient, only one dataset can be stored per CSV file, and support for embedded delimiters (quotes, commas, etc.) depends on the data science tool used.
- Consider the advantages and disadvantages of JSON files:
 - **Advantages:** human readable, easy to edit, good library and language support, many data sources store data in native JSON or can export in this format, structured so one file can source more than one dataset, and tools and libraries can enable querying like a database.
 - **Disadvantages:** not space efficient, not as widely supported as CSV, and limited and non-standard support for binary data.
- Consider the advantages and disadvantages of XLS/X files:
 - **Advantages:** binary data is supported, very space efficient, and some formats automatically incorporate compression.

- **Disadvantages:** not human readable, and varying support for formats.
- Store binary data for textual file formats (JSON, CSV) in Base64.
- Keep in mind that smaller files are read from and written to faster, but add the complexity of maintaining multiple files.
- When writing CSV files, store less than 1,048,576 rows and 16,384 columns, and keep each column to less than 255 characters to maintain compatibility with Excel.
- Select the proper options when exporting data if the data includes Unicode/UTF-8 characters.

Do Not Duplicate or Distribute

ACTIVITY 2-10

Exporting Data to a CSV File

Before You Begin

Extracting, Transforming, and Loading Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Write the data to a CSV file**. Select **Cell**→**Run All Above**.

Scenario

Some of the other team members on the project don't use Python and aren't particularly comfortable with SQL. But they still need a way to work on the same GCNB data that you've been preparing. So, you'll export the dataset as a comma-separated values (CSV) file, which preserves the data and its general structure in a text file. That way they can more easily integrate the data into whatever environments they are using.

1. Write the data to a CSV file.

- a) Scroll down and view the cell titled **Write the data to a CSV file**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_cleaned_final.to_csv('users_data_cleaned.csv',
 2                           index = False)
```

- c) Run the code cell.

Saving to a CSV will preserve the content of the DataFrame, but not the structure. However, this makes the data more extensible, as not every tool can read binary DataFrame objects, whereas most can read text-based CSV files.

2. Confirm that the data was written to a CSV file.

- a) Scroll down and view the cell titled **Confirm that the data was written to a CSV file**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 pd.read_csv('users_data_cleaned.csv').head()
```

- c) Run the code cell.

- d) Examine the output.

		user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	campaign		
0		9231c446-cb16-4b2b-a7f7-ddfc8b25aa6		3.0	2143.0	58	management	married	tertiary	False	True	False	NaN	261	1	
1		bb92765a-08de-4963-b432-49e524d39157		NaN		Nan	44	technician	single	secondary	False	True	False	NaN	151	1
2		573de577-49ef-42b9-83da-d3cfb817b5c1		2.0	2.0	33	entrepreneur	married	secondary	False	True	True	NaN	76	1	
3		d6b669a9-7c8f-4257-a682-e136f640b7e3		NaN		Nan	47	blue-collar	married	NaN	False	True	False	NaN	92	1
4		fade0b20-7594-4d9a-84cd-c02f79b1b526		1.0	1.0	33	NaN	single	NaN	False	False	False	NaN	198	1	

The structure of the DataFrame appears to be intact, despite loading from a text file. However, there may be some more subtle changes that will require you to parse the data in order to get it into an ideal state.

- e) Select the next code cell, then type the following:

```
1 pd.read_csv('users_data_cleaned.csv').info()
```

- f) Run the code cell.
g) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45209 entries, 0 to 45208
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   user_id          45209 non-null   object 
 1   number_transactions 35210 non-null   float64 
 2   total_amount_usd  35210 non-null   float64 
 3   age              45209 non-null   int64  
 4   job              44921 non-null   object 
 5   marital           45209 non-null   object 
 6   education         43352 non-null   object 
 7   default            45209 non-null   bool   
 8   housing            45209 non-null   bool   
 9   loan              45209 non-null   bool   
 10  contact           32191 non-null   object 
 11  duration           45209 non-null   int64  
 12  campaign           45209 non-null   int64  
 13  pdays             45209 non-null   int64  
 14  previous           45209 non-null   int64  
 15  poutcome           8252 non-null   object 
 16  term_deposit       45209 non-null   bool   
 17  date_joined        45179 non-null   object 
 18  device              45115 non-null   object 
dtypes: bool(4), float64(2), int64(5), object(8)
memory usage: 5.3+ MB
```

As you can see, the `date_joined` column is now in a plain string format instead of datetime. So, keep in mind that text files may require additional parsing when you load them back into your programming environment.

3. Confirm the CSV file was saved to the local drive.

- a) In Firefox, select the CDSP/ETL/ tab to view the file structure.
- b) Verify that a file called `users_data_cleaned.csv` exists.
- c) Select the file name to open it.

- d) Verify that you can see all of the data in a textual format, separated by commas.
- e) When you're done, close this tab and return to the active notebook.

4. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel->Shutdown**.
- b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
- c) Close the **Extracting, Transforming, and Loading Data** tab in Firefox, but keep a tab open to **CDSP/ETL** in the file hierarchy.

ETL Endpoints

One of the challenges of the ETL process is maintaining an authoritative source for the prepared dataset without interrupting other business operations or creating internal project issues. As mentioned earlier, the IT team can provision access to a database that you store the data in. But rarely does the ETL process go from point A to point B in a straight line without any issues along the way. You may make mistakes, or you may simply want to experiment in different ways of doing things. So, it's common to test your ETL tasks in a development environment, especially if those tasks are repeatable through the use of scripts. Directly reading from and writing to a single database isn't a good way to go about this. Instead, you should consider leveraging an ETL endpoint.

An endpoint in an ETL system refers to a development environment that can be configured and provisioned on the fly for use with ETL tasks. The endpoint acts as a sort of intermediary between your programming workspace and the actual database. It can feed you the data from the database so that you can run your scripts iteratively, without directly accessing the backend server. This enables you to fine tune your ETL tasks and run them against many different types of scenarios. For example, you might run your data cleaning scripts against different representations of the same overall data to see which is more compatible. You and your colleagues therefore don't need to all be accessing and updating the same authoritative data source in order to get your work done.

Most ETL endpoints are implemented using RESTful web services. API calls can retrieve data in file formats like XML and JSON, like any other web service. Endpoints are also a common feature of major cloud services like AWS and Azure.

Guidelines for Configuring ETL Endpoints

Follow these guidelines when configuring ETL endpoints.

Configure ETL Endpoints

When configuring ETL endpoints:

- Consider how data will be used when deciding on output formats.
- Find out how to send data to the ETL tool. Some tools support RESTful, textual, and/or proprietary formats and libraries.
- Obtain permissions to use endpoints and tools.
- Write custom applications for repeatability and to help with the automation of regular loads.
- Consider how often data needs to be sent.
- Determine how long a data load will take to complete.
- Work with the ETL provider or team to determine what data transformations will occur before data is sent to the ETL provider and what will be done by the ETL system.
- Keep in mind that working with ETL systems can be challenging because they perform complex transformations between enterprise systems, often requiring a dedicated team to support them.

Data Loading: Visualization Tools

While you can certainly visualize data programmatically by reading the raw data, you may choose to load that data directly into a visualization tool instead. This is particularly helpful if your ultimate goal is to perform exploratory analysis on the data. Or, in some cases, visualization tools can perform simple modeling of data and demonstrate that model in charts and graphs. In addition, loading data into visualization tools can make it easier to present findings to a non-practitioner audience.

There are many data visualization tools out there, and each has different support for both open and proprietary data formats. One popular visualization tool is the suite of Tableau products, which can subsume many different data formats like relational databases and spreadsheets and present the data visually. The advantage of software products like these is that you don't need to be a programmer or write any code to visualize your data. The disadvantage is the other side of the same coin: as powerful as visualization programs can be, they are not necessarily optimized for complex tasks like machine learning.

Guidelines for Loading Data into Visualization Tools

Follow these guidelines when loading data into visualization tools.

Load Data into Visualization Tools

When loading data into visualization tools:

- Keep in mind that some visualization tools are their own data science platforms, and may integrate with other data science tools.
- Only load the data to be visualized and any supporting data, as the volume of data can affect rendering or animation times.
- Consider what data needs to be aggregated before visualization.
- Look at the data formats required by the visualization tool and format appropriately.
- Format data as much as possible before sending it to the visualization tool.

ACTIVITY 2-11

Exploring Data Visualization Tools

Scenario

While most of your work will be done in a programming language, you may find it easy to load the data into a tool that is specifically designed to generate neatly formatted visuals. So, you'll search the web for data visualization tools to see if you can find any that might be beneficial to the project.

1. Search for data visualization tools.

- a) Open a new browser tab in Firefox and navigate to your search engine of choice.
- b) Search for **data visualization tools** or search for a specific tool you're interested in, like Tableau or Microsoft Power BI.
- c) Open one or more of the results that look interesting to you, then briefly read the reviews and/or explanations of the tool(s).

2. Choose one or more tools to discuss with the class.

What types of features do these tools offer? How do they compare to other data visualization tools?

3. Close any new browser tabs that you opened, and return to Jupyter Notebook.

Summary

In this lesson, you extracted data by pulling it in from disparate sources and consolidating it, transformed data to make it more conducive to analysis, and loaded the data into a final format. The ETL process is one of the most complex and involved parts of the data science process, but it's nonetheless crucial to ensuring your data can actually be useful in producing a business solution.

What kind of transformation tasks would be most applicable to the type of data you work with?

What format do you currently use or plan to use to load prepared data into?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

3

Analyzing Data

Lesson Time: 6 hours, 45 minutes

Lesson Introduction

In the previous lesson, you conducted extract, transform, and load (ETL) to ensure your data was ready for the next phase of the data science process: analysis. In some cases, an analysis of the data may be the actual final goal of the project, or it may be an important intermediary step on the road to machine learning. In either case, analyzing your data using various techniques will help you obtain useful insights into that data and what it represents. It'll also give you a better understanding of how the data needs to undergo more processing to prepare it for machine learning.

Lesson Objectives

In this lesson, you will:

- Examine the basic characteristics of a dataset to gain general insights into the data.
- Use statistical analysis methods to explore the underlying distribution of data.
- Use visualizations such as histograms, scatter plots, and maps to analyze data.
- Use preprocessing techniques to further transform data and prepare it for machine learning.

TOPIC A

Examine Data

You'll begin your analysis efforts by exploring the nature of your dataset and the relationships it contains.

Exploratory Data Analysis

Exploratory data analysis (EDA) is a data science approach in which the practitioner closely examines the data in order to reveal new information. The primary purpose of EDA is to maximize the amount of insights gleaned from the data so the practitioner knows as much as possible about what they're studying. In addition, EDA has several objectives:

- Test and evaluate prior assumptions about the data.
- Reveal the underlying structure of the data.
- Determine important features and factors about the data.
- Identify unwanted elements of the data.
- Determine the best path forward for modeling the data.

The emphasis on exploration means that EDA is a flexible approach to learning more about the data. As a result, EDA most often employs visualizations. While not as quantitative as pure numerical statistics, visualizations offer that desired flexibility, enabling the practitioner to examine their data in a more open-ended fashion. Typically, EDA visualizations incorporate graphical plots of both raw data and statistics that summarize the raw data. It's even more useful to incorporate multiple plots and arrange them in such a way that it becomes easier to recognize patterns.

EDA is not a one-time event, either. It's a valuable tool at pretty much any step of the data science process. Every time you make a significant change to the data, or apply that data in a new way, you could benefit from EDA.

Dataset Content and Format

A good place to start when analyzing a dataset is to get familiar with the content and format of the various columns that the dataset contains, as well as the data type of each column. Familiarizing yourself with the data content and format will enable you to start thinking about what you can do with the data, as well as what sort of preparation you must do so that it's ready for modeling. You should try to identify:

- The number of columns.
- The name of each column.
- The data type of each column.
- The number of rows.
- The primary identifier of each row, if applicable.
- The representation of values for each row (e.g., whether or not the data has been normalized).
- The presence/number of missing values.

If you have the dataset loaded in a Python `DataFrame`, you can call the `info()` function on the `DataFrame` to list all of the dataset's attributes and the data type of each attribute. If you would also like to examine samples of various data values within the dataset, you can call the `head()` function to view the first five rows of data, or you can provide the function with a number value to see more rows, as in `head(10)`.

```
data_raw.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object  
 9   Fare          891 non-null    float64 
 10  Cabin        204 non-null    object  
 11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Figure 3-1: Retrieving the attributes of a DataFrame to learn more about the data itself and how it's formatted.

Analysis of Feature Types

As discussed before, you can think of types of data in two ways: specific machine-readable data types like strings, floats, integers, etc., and the three high-level feature types (quantitative, categorical, ordinal). There are various tools you can use to quickly identify the former, like the `DataFrame.info()` function, but it's not always so easy to identify the latter. That's because feature types are not always inherently definable by their values alone. A feature's values sometimes determine its type, but this isn't always the case.

Consider an example where you have a feature column called `color`. It's formatted as an integer, and its values can either be 1, 2, 3, or 4. To a machine, this may look like an ordinal feature, or even quantitative. However, in this example, `color` has just been encoded so that, instead of strings of actual color names, it just has arbitrary numbers assigned to each color. Since the feature name is descriptive enough, you could guess that the correct feature type is actually categorical. Or, if you had prior domain knowledge, you could know this for certain (and what color maps to which number). But you won't always have this advantage.

Analyzing feature types is therefore an important part of ensuring you truly understand what the data actually represents. In the absence of prior domain knowledge and obvious clues, you may need to assess how the values for the unknown feature do or do not relate to other, known features in the dataset. You can also explore the content of a few or even many examples to look out for potential mistakes in data preparation. For example, a feature column might be formatted as a string, implying that it's categorical, but if all of its values follow the pattern '34.42', '56.12', '78.95', etc., you might decide that this column should have instead been formatted as a float, meaning the feature is likely quantitative.

There is no fool-proof method for addressing these issues, other than making sure your dataset is clean and well prepared ahead of time. Critical thinking and thorough reviews of your datasets can go a long way, however.

Target Features

In data science, a **target feature** is the variable you're interested in learning more about. Often, this means the variable you're trying to make estimations for, like the value of a house. Or, it might describe a way of categorizing something, like what species an animal belongs to. A target feature can also simply be used to determine how to modify the dataset itself to optimize it for some other purpose. In any of these cases, the target feature is of primary importance to the dataset. Recall that, in supervised machine learning, datasets are *labeled*. The labeled feature is the same as the target feature—it's what you want the machine to make decisions about.

Even if you have a good idea of what business issue you're trying to address through data science, and have gone through problem formulation, it may not be clear what the target feature(s) actually are. This can be true even when you're working with a clean dataset. Part of your analysis of such a dataset will be to actually identify one or more target features. A thorough understanding of the problem you're trying to solve is the best way to start, as this will reveal good candidates for target features in most situations. For example, if you know you want to use data science to determine how best to allocate computing resources for on-premises servers, the target features could be: server uptime as a percentage; frequency or proportion of dropped connections; average floating point operations per second (FLOPS); and so on. All of these features can measure the performance of your servers in different ways, which is the point of determining optimal resource allocation—to know where to send resources in order to improve performance.

In situations where your domain knowledge doesn't always inform what target features to choose, you should consider researching real-world case studies or examples that are similar to the problem you're trying to solve or use similar datasets. The work done by others in the industry can give you the insights you need to choose the best target features.

Likewise, the choice of target features might become more obvious when you take into account what kind of model you're trying to build. If you want to build a model that can classify data, and your dataset only has one categorical variable, there's a good chance that variable is the target feature you're looking for. So, knowing your feature types is once again important.

Feature Relevance

Even after you've identified feature types and target features in your dataset, you're not necessarily done. As you've seen, datasets are rarely in a perfect state. One thing you need to consider is that the features themselves have certain degrees of relevance to the problem at hand. While the feature `age` might seem like a great predictor of the target feature `income`, there may be subtleties that your assumptions don't account for. Some features are less relevant than others at serving whatever purpose they are meant to. This often prompts a data science practitioner to remove features, consolidate them, split them apart—anything that will make the dataset more viable. That's because an irrelevant feature can significantly impact the power of a model to make predictions or estimate something about the real world.

Again, an understanding of your dataset and the problem you're trying to solve can help you identify irrelevant features. This can only help you so much, since the relationship between features is the very thing you're trying to study, and in the case of machine learning, trying to get a machine to determine for you. There are various techniques for determining the relevance of features, but at this point, just know that packing your datasets full of features is not always guaranteed to produce better results. It might actually have the opposite effect, in fact.

Representative Data

An idea similar to feature relevance is that of representative data. Sample data from a population is representative if it does well to approximate the characteristics of that wider population. Let's say you work for a general retailer, and you have a dataset where each row is a customer who you surveyed. If you have 40 customers as your sample, and 32 of them are female whereas 8 are male, your sample is really not representative of the wider population of customers, especially if you know

that your entire customer base is closer to a 50/50 split. So, whatever results you get from using this sample may be suboptimal since you've misrepresented a key demographic of customers.

The sample size may also lead to unrepresentative data. It's possible that 40 customers is too few to capture a wide enough variety of customer demographics, even if each member of that sample is sufficiently different from every other member. Of course, there are limitations to representative sampling, since you cannot possibly account for every single variation in most scenarios. Likewise, there needs to be at least some common ground between the members of your sample, otherwise there will be no patterns to recognize or trends to estimate.

A common solution to the problem of representation is ***stratified random sampling***. For this solution, the data science practitioner identifies key groups (e.g., demographics like gender, race, age, etc.) that are relevant to the problem, and then divides the population into these groups (strata). Within each stratum, members are chosen randomly. Then, all of these members are compiled to form the sample. This ensures that the ultimate sample will contain data that is more representative of the population than just a normal random sample or a hand-picked sample.

So, when you analyze your datasets, you should look at the categorical features to identify those that have the potential to be key strata. Once you identify these strata, you can determine what proportion of your data examples are in each stratum, and how this compares to the wider population. Note that representation does not always imply an even split. If 80% of your total customer base is female, your sample of 32 female and 8 male customers is actually representative in that regard. Also, you may be in a position where the proportions of each stratum in the population are not known, making it difficult or even unfeasible to know what is or is not representative.

Additional Sampling Techniques

Other than stratified random sampling, there are some additional sampling techniques that can address the effects of imbalanced data, including:

- ***Random oversampling***, where values from the dataset are selected at random to add to a new dataset. The selection happens *with replacement*, meaning the same value may be selected over and over. So, the same data examples from the minority class can be duplicated multiple times in the new dataset, ensuring that the minority class has more representation. However, the dataset does not contain any new information from which a model can learn.
- ***Random undersampling***, where data examples in the majority class are removed in order to achieve balance. The disadvantage of this approach is that removing useful examples could greatly impair a model's ability to learn.
- ***Synthetic minority oversampling technique (SMOTE)*** attempts to address the lack of new data involved in random oversampling by creating synthetic data examples. This synthetic data is not naturally observed, but instead generated by selecting a location that is close to the real data in the feature space. So, if two real points are close in the feature space, the synthetic data might be created between those real points.

Imbalanced Datasets

The most common definition of an imbalanced dataset is one that has a disproportional frequency of each value in a categorical variable, particularly when it's the target variable. As an example, say you have a dataset of people and you want to build a model that can classify any person as either left handed or right handed. About 90% of the world is right handed, and 10% is left handed. For the sample dataset to be truly representative of the population, it would be split this way as well. However, this creates a class imbalance. Only 10% of all data examples in the set are labeled as left handed.

Imbalanced datasets can present a challenge, particularly when it comes to machine learning. The model must learn from the dataset, and its usefulness may be negatively impacted if most of what it sees is one category and not another. For this particular scenario, it's not as much of an issue, since the whole purpose is to classify handedness—something that is, by nature, imbalanced. But what if

your dataset is not (and due to any number of factors, cannot be) truly representative? Say you have a dataset where each row is a timestamp, and the target feature is whether or not a redundant storage system is being written to. Most of the timestamps are labeled class 0 (no) whereas very few are labeled class 1 (yes). If this is the first such sample you've collected from your environment, you may have no idea whether or not this is truly representative of the population. Maybe you collected this data when system activity was unusually low. So, your resulting model might not perform as well in future situations.

Imbalanced datasets also have an effect on the metrics you use to evaluate classification models. Some metrics are better suited than others when it comes to evaluating such models. So, it's important that you identify the frequency of all data examples in your sample set that belong to each class. This is pretty trivial for most data analysis environments to do for you, and it can be done using pure numbers or even visually.



Note: It's also possible to describe datasets with continuous target variables as being imbalanced. Most examples might be within a certain range, whereas a few examples might be in a drastically different range. This is where a discussion of errors, outliers, and noise comes into play.

Errors, Outliers, and Noise

Specific values within a dataset may be inaccurate, faulty, or otherwise undesirable. These values can slip through the cracks even after you've taken care to perform cleanup procedures on your dataset during the ETL process. If left unaddressed, they can lead to quality issues with models or mislead you during your analysis.

For example, a dataset may contain:

Issue	Description	How to Identify
Errors	Errors are incorrect or missing values. Too many can negatively impact a model's ability to learn patterns from the data. This includes failing to learn patterns or just learning the wrong patterns.	Incorrect values can be very difficult to identify since you probably trust that the dataset is authoritative, and you may not have a second source to use as verification. Some incorrect values are easy to spot because they deviate significantly from the rest of the data examples, but this isn't always the case. Outright missing values are typically easy to spot.
Outliers	Outliers are values outside of the main distribution, or spread of values. They deviate significantly from other data examples. Outliers can be the result of mistakes in observation or they can occur naturally in the population. In either case, they can cause problems with pattern recognition, whether it's a machine learning model or a human analyst.	You can use various statistical summary methods to identify outliers, but it's usually easier to spot them through graphs that depict the spread of values. Outliers will appear far outside of the main group.

Issue	Description	How to Identify
Noise	Noise is irrelevant or irregular data that makes it difficult for a model to "hear" patterns revealed by data that is actually relevant. This is because noise includes values, features, or examples that are not necessary to make estimations, or that outright hinder estimation. Noise can also add to the complexity of data analysis and model creation, leading both to become more difficult and time consuming.	There are several statistical and machine learning techniques for identifying the unnecessary complexity of a dataset, particularly irrelevant features. In some cases you can also graph values and see that they exhibit no discernible pattern.

Correlations

In statistics, a **correlation** is a mathematical association between two variables. When variables correlate, it suggests that each variable has some kind of effect on the other. The nature of this effect can vary, but all correlations reveal at least some relationship between the variables. Many data analysis approaches, and even machine learning algorithms, look for correlations between data values. Such correlations can enhance the predictive and decision-making power of both the practitioner and the model(s) they build.

For example, let's say you're developing a statistical model to predict the rise and fall of city populations over time. You might observe that there is a correlation between job availability and the total population. On average, more available jobs might mean a greater population. Of course, the correlation may not be perfect. A city with a high job availability index might have a smaller population than a city with a lower job availability index. But, in general, if you find that cities with higher job availability tend to have larger populations, you could say that there is a positive correlation between the variables `job_availability` and `total_population`. So, your model might do a better job of predicting the future population of a city based on its job availability index. Note that both of these variables are also features of a dataset (`total_population` also being a label in a supervised problem). In practicing data science, it's common to explore correlations between features.

It's important to keep in mind that correlations may or may not be causal. In other words, the change in one variable *may or may not* be the direct cause of a change in another variable. While correlations can certainly indicate a causal relationship, they do not always do so. Other variables or even other factors can be affecting the variables you're studying. Or, the cause and the effect may be the opposite of what you think. Consider the city population example. Does an increase in job availability really lead to a larger population, or does a larger population lead to industry growth, which in turn leads to more available jobs? When you analyze data for correlations, it's a good idea to exercise critical thinking and not jump to conclusions about the power of one variable to change another.

Correlation Coefficient

The **Pearson correlation coefficient (PCC)** provides a measure of the linear correlation between two variables commonly called x and y . It produces a value between $+1$ and -1 that shows the strength of their dependence on each other.

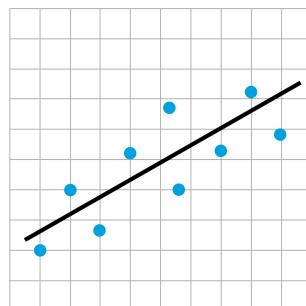
- 1 signifies the strongest possible positive correlation between x and y . As x increases, y increases. Conversely, as x decreases, y also decreases. If you plot a set of data on a chart with dots showing where each value of x (on the horizontal axis) would produce a value of y (on the

vertical axis), the values would all land on a straight line heading from the lower-left to the upper-right.

- **0** signifies no correlation between x and y .
- **-1** signifies the strongest possible negative correlation between x and y . As one increases, the other decreases. An x - y plot would show all values landing on a straight line heading from the upper-left to the lower-right.

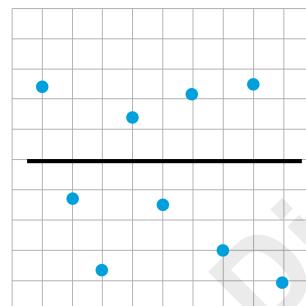


Note: Pearson's correlation coefficient goes by several other names. It is also called Pearson's r , r value, Pearson product-moment correlation coefficient (PPMCC), and the bivariate correlation.



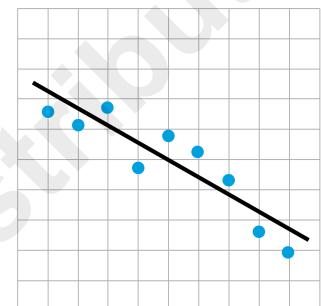
Positive Correlation

$$r = 0.6$$



No Correlation

$$r = 0$$



Negative Correlation

$$r = -0.8$$

Figure 3-2: Correlations shown in an x - y plot.

The farther the correlation coefficient is from zero, the stronger its correlation, signifying that x and y are dependent variables, and that one may be a good predictor of the other.



Note: Typically, correlations above 0.90 or below -0.90 would imply that the variables are similar enough in effect that you might remove them from the datasets without any loss of useful information.

Correlation Strength

In many cases, you will not have a perfect positive or negative correlation. It's more likely that you'll have a coefficient that is between 0 and either 1 or -1. The closer the coefficient is to the absolute strongest correlation value, the stronger the correlation is said to be. The closer the coefficient is to 0, the weaker it is said to be. Calling a correlation "strong" or "weak" is more of a subjective judgment, but it's useful in describing the degree to which two variables relate, as well as comparing two different correlations. Remember, correlation does not always imply causation, so its strength (or lack thereof) may indicate that other factors are affecting the relationship between variables.

The strength of a correlation is not necessarily determined by the angle of the trend line, but by the dispersion of the points. As more points get closer to the line, the correlation is stronger; as the points are spread out from the line, the correlation is weaker. The actual trend of the relationship between two variables is more obvious when the correlation is strong.

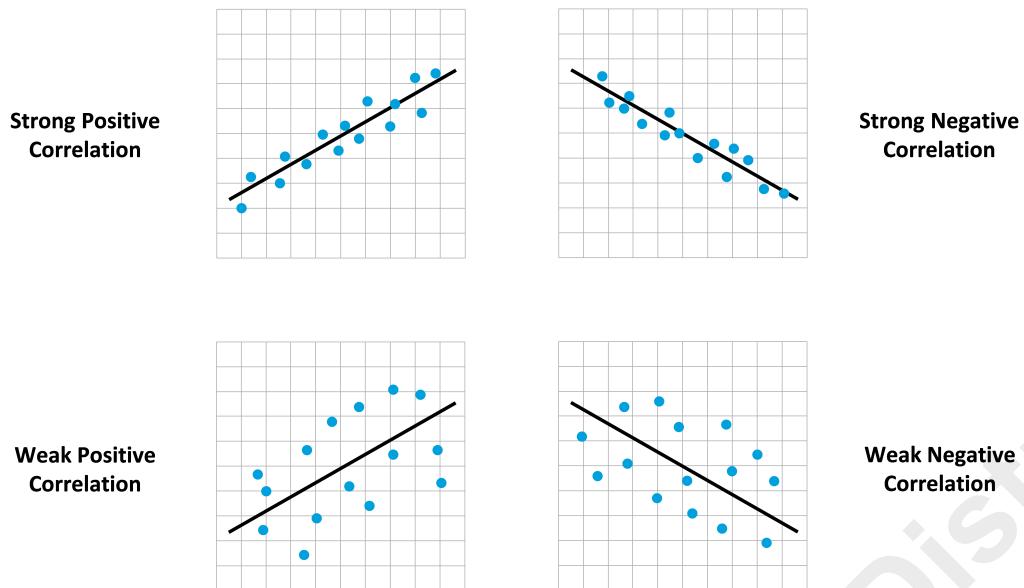


Figure 3-3: Strong vs. weak correlations.

Guidelines for Examining Data



Note: All Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Follow these guidelines when examining data.

Examine Data

When examining data:

- Spend a fair bit of time interacting with the data you will be working with.
- Use charts to begin to understand the data, but don't rely too heavily on them.
- Use multiple correlated charts of different fields to determine correlation between these fields.
- Examine a number of small portions of a dataset to understand relationships. Simply scrolling through the data is one technique to use, as is looking at every n^{th} record.
- Resist the urge to start drawing conclusions from the data without a thorough manual review of the data.
- Bring the data into a data science tool or even Excel for ad hoc searching and plotting.

ACTIVITY 3-1

Examining Data

Data Files

/home/student/CDSP/Analysis/Analyzing Data.ipynb
/home/student/CDSP/Analysis/data/users_data_cleaned.pickle

Scenario

Now that you've gone through the ETL process on the GCNB users data at least once, you can start taking a closer look at the characteristics of that data. There's likely still many more ways the data can be improved before you use it to start building machine learning models. You'll begin your analysis by examining the data from a high level, just to get a sense of what state it's in and what you have to work with.

1. Open the notebook.

- In the Jupyter Notebook web client, navigate to the **CDSP/Analysis** directory.
- Select **Analyzing Data.ipynb** to open it.

2. Import the relevant software libraries.

- View the cell titled **Import software libraries**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.

3. Load and preview the data.

- Scroll down and view the cell titled **Load and preview the data**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data = pd.read_pickle('data/users_data_cleaned.pickle')
2
3 users_data.head(n = 5)
```

- You're loading the pickle file of the cleaned users data table you saved earlier.
- Run the code cell.

- d) Examine the output.

	user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	campaign	previous
0	9231c446-cb16-4b2b-a7f7-ddfc8b25aaef	3.0	2143.0	58	management	married	tertiary	False	True	False	None	261	1	
1	bb92765a-08de-4963-b432-496524b39157	NaN	NaN	44	technician	single	secondary	False	True	False	None	151	1	
2	573de577-49ef-42b9-83da-d3cfb817b5c1	2.0	2.0	33	entrepreneur	married	secondary	False	True	True	None	76	1	
3	d6b66b9d-7c8f-4257-a682-e136f640b7e3	NaN	NaN	47	blue-collar	married	None	False	True	False	None	92	1	
4	fade0b20-7594-4d9a-84cd-c02f79b1b526	1.0	1.0	33	None	single	None	False	False	False	None	198	1	

This is an overview of all columns in the table, and the first five rows. You can see some of the types of values in each column, including NaN (missing data).

4. Check the shape of the data.

- a) Scroll down and view the cell titled **Check the shape of the data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 | users_data.shape
```

- c) Run the code cell.
 d) Examine the output.

```
(45209, 19)
```

There are 45,209 rows and 19 columns in the dataset.

5. Check the number of unique users.

- a) Scroll down and view the cell titled **Check the number of unique users**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 | len(np.unique(users_data.user_id))
```

- c) Run the code cell.
 d) Examine the output.

```
45209
```

As expected, the number of unique users is the same as the number of rows.

6. Check the data types.

- a) Scroll down and view the cell titled **Check the data types**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 | users_data.info()
```

- c) Run the code cell.

- d) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45209 entries, 0 to 45215
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          45209 non-null   object  
 1   number_transactions 35210 non-null   float64 
 2   total_amount_usd   35210 non-null   float64 
 3   age               45209 non-null   int64  
 4   job               44921 non-null   object  
 5   marital            45209 non-null   object  
 6   education          43352 non-null   object  
 7   default             45209 non-null   bool    
 8   housing             45209 non-null   bool    
 9   loan               45209 non-null   bool    
 10  contact             32191 non-null   object  
 11  duration            45209 non-null   int64  
 12  campaign            45209 non-null   int64  
 13  pdays              45209 non-null   int64  
 14  previous            45209 non-null   int64  
 15  poutcome            8252 non-null   object  
 16  term_deposit        45209 non-null   bool    
 17  date_joined         45179 non-null   datetime64[ns]
 18  device              45115 non-null   object  
dtypes: bool(4), datetime64[ns](1), float64(2), int64(5), object(7)
memory usage: 5.7+ MB
```

The data types for each column are as expected.

- e) Select the next code cell, then type the following:

```
1 users_data.columns.to_series().groupby(users_data.dtypes).groups
```

- f) Run the code cell.

- g) Examine the output.

```
{bool: ['default', 'housing', 'loan', 'term_deposit'], int64: ['age', 'duration', 'campaign', 'pdays', 'previous'],
float64: ['number_transactions', 'total_amount_usd'], datetime64[ns]: ['date_joined'], object: ['user_id', 'job',
'marital', 'education', 'contact', 'poutcome', 'device']}
```

Each column is grouped by data type.

7. Check for correlations.

- a) Scroll down and view the cell titled **Check for correlations**, then select the code cell below it.
b) In the code cell, type the following:

```
1 users_data.corr().abs()
```

- c) Run the code cell.

- d) Examine the output.

	number_transactions	total_amount_usd	age	default	housing	loan	duration	campaign	pdays	previous	term_deposit
number_transactions	1.000000	0.163409	0.008813	0.138838	0.030429	0.075319	0.017220	0.026431	0.030751	0.023046	0.053390
total_amount_usd	0.163409	1.000000	0.095839	0.065390	0.066857	0.084526	0.022586	0.017274	0.006435	0.016952	0.050785
age	0.008813	0.095839	1.000000	0.017875	0.185552	0.015641	0.004645	0.004767	0.023745	0.001297	0.025168
default	0.138838	0.065390	0.017875	1.000000	0.006020	0.077232	0.010017	0.016819	0.029982	0.018331	0.022421
housing	0.030429	0.066857	0.185552	0.006020	1.000000	0.041341	0.005041	0.023583	0.124197	0.037087	0.139161
loan	0.075319	0.084526	0.015641	0.077232	0.041341	1.000000	0.012395	0.009972	0.022762	0.011048	0.068193
duration	0.017220	0.022586	0.004645	0.010017	0.005041	0.012395	1.000000	0.084551	0.001549	0.001213	0.394549
campaign	0.026431	0.017274	0.004767	0.016819	0.023583	0.009972	0.084551	1.000000	0.088636	0.032860	0.073179
pdays	0.030751	0.006435	0.023745	0.029982	0.124197	0.022762	0.001549	0.088636	1.000000	0.454817	0.103616
previous	0.023046	0.016952	0.001297	0.018331	0.037087	0.011048	0.001213	0.032860	0.454817	1.000000	0.093232
term_deposit	0.053390	0.050785	0.025168	0.022421	0.139161	0.068193	0.394549	0.073179	0.103616	0.093232	1.000000

This presents a table where the correlation coefficient of each variable pairing is calculated. Most of the coefficients are on the low end, implying weak correlations. It's usually easier to spot correlations when they're plotted visually, so you'll investigate this more later.

8. Correlations aside, take a look at the list of variables and consider their data types, feature types, and what they describe.

At this point in the process, which of these variables do you think might make good candidates for target features?

9. Keep this notebook open.

TOPIC B

Explore the Underlying Distribution of Data

One of the key factors in data analysis is determining how values are spread out within each of the different features. This will give you a deeper understanding of how the data is represented and how it might need to change.

Frequency Distributions

Distributions are a common starting point for a discussion of probabilities and statistics. A **frequency distribution** is a method of demonstrating the frequency of outcomes for a particular sample of a random variable. The random variable being studied can be either discrete or continuous. For example, let's say a grocery store wants to tally how many fruits were sold in one day. So, "fruits" is your discrete random variable, and how many of each type of fruit (apples, oranges, bananas, etc.) were sold represents the frequency. Typically you'd plot a frequency distribution of a categorical variable as a bar chart.

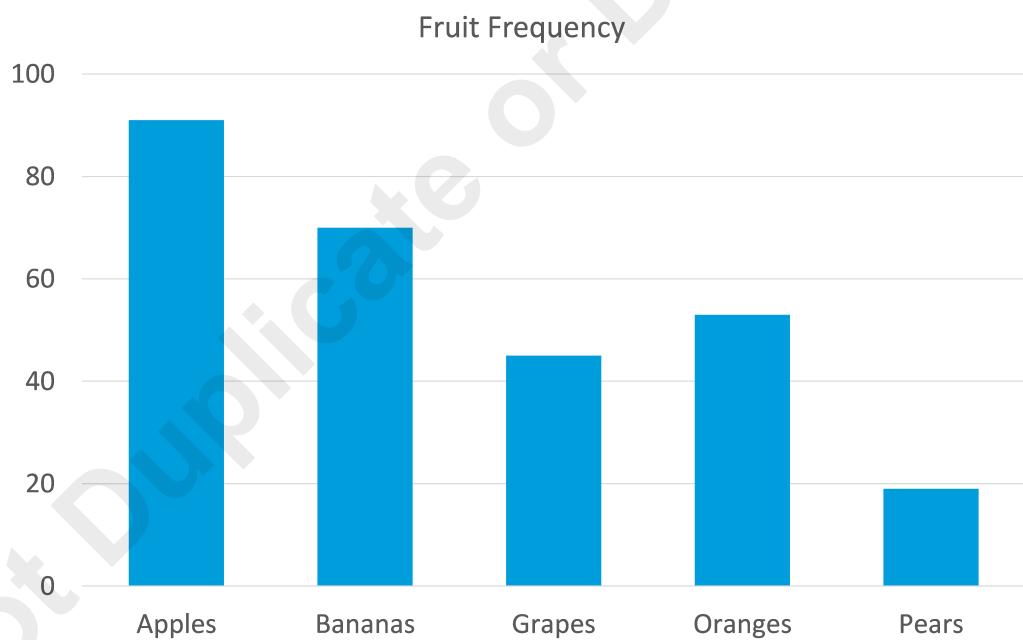


Figure 3-4: A frequency distribution of fruit categories.

You can also plot a frequency distribution of a continuous variable by binning the range of its values. For example, if you want to plot the frequency of people's heights from a sample set, you could place those heights into bins and then plot the distribution in a histogram.

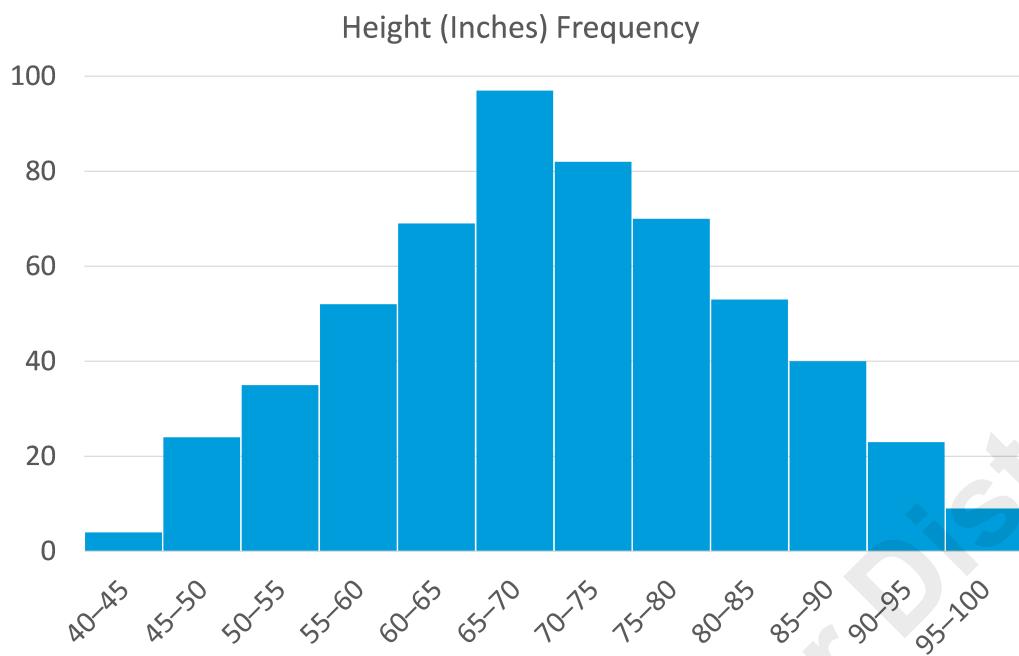


Figure 3-5: A frequency distribution of height ranges.

Frequency distributions can help reveal insights about your data. For example, you can see what the most common fruits sold are, or what the most common height ranges are, and you can compare them to the frequency of other values.

Probability Distributions

A similar type of distribution is a probability distribution. A **probability distribution** is a statistical function that demonstrates the probability of outcomes for a random variable. The difference is that a frequency distribution shows the number of observed outcomes in a sample set, whereas a probability distribution shows a proportion of all a variable's values. This proportion adds up to 1. The probability of one value occurring is equal to its proportion.

For example, if you were to plot the fruit example as a probability distribution, you can see that the probability of selling an apple in any given transaction is around 0.3, or 30%.

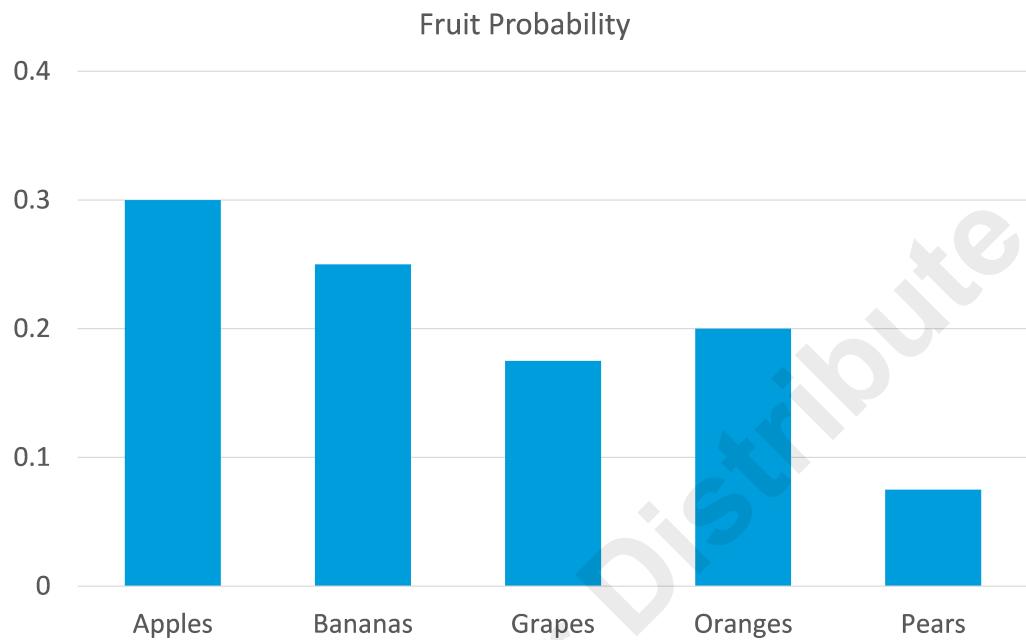


Figure 3–6: A probability distribution of fruit categories.

Likewise, the probability of any given person's height being between 65 and 70 inches is about .175, or 17.5%.

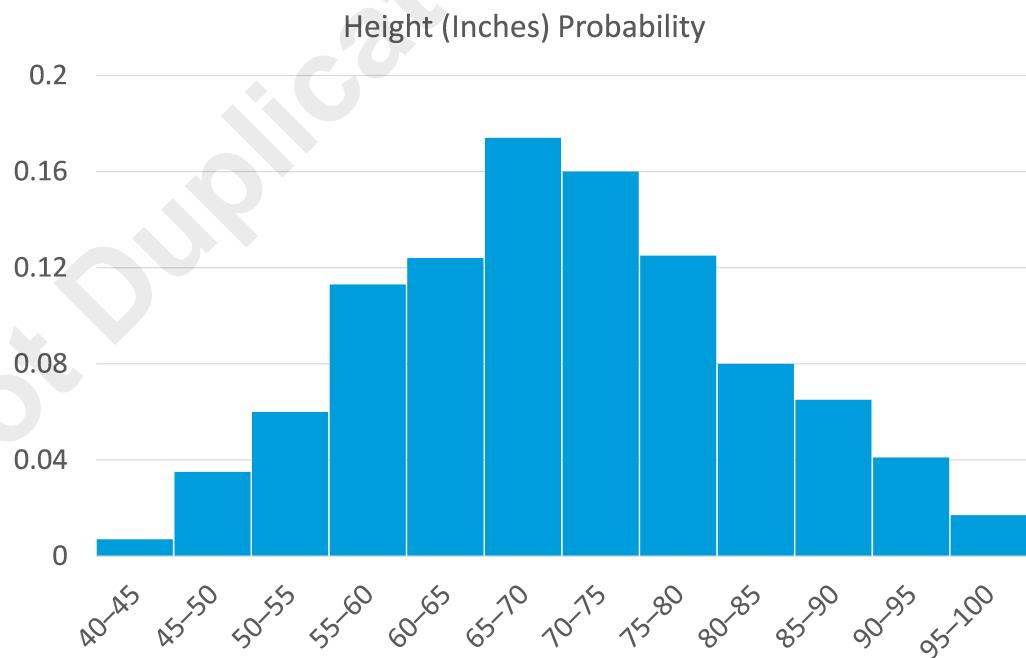


Figure 3–7: A probability distribution of height ranges.

Probability distributions are useful for extrapolating sample data to a population. So instead of limiting your analysis to the exact frequencies of one sample set, you can get a general idea of the probabilities inherent in the population. In the examples, you can see that the proportion of each

bar/bin is somewhat different in the sample sets (frequency distribution) than in the populations (probability distribution).

Both types of distributions are useful for statistical analysis of data and are of fundamental importance to the data science practitioner.

Normal Distribution

There are several ways that data can be distributed, and a distribution may follow one of several commonly recognized patterns or shapes. A **normal distribution** is a distribution whose data is distributed in such a way that most cases are bunched toward the center, whereas fewer cases are present at either end. This creates a distribution that, when viewed visually, appears to form a bell shape; i.e., there is one "hump" at the top middle that gradually tapers downward.

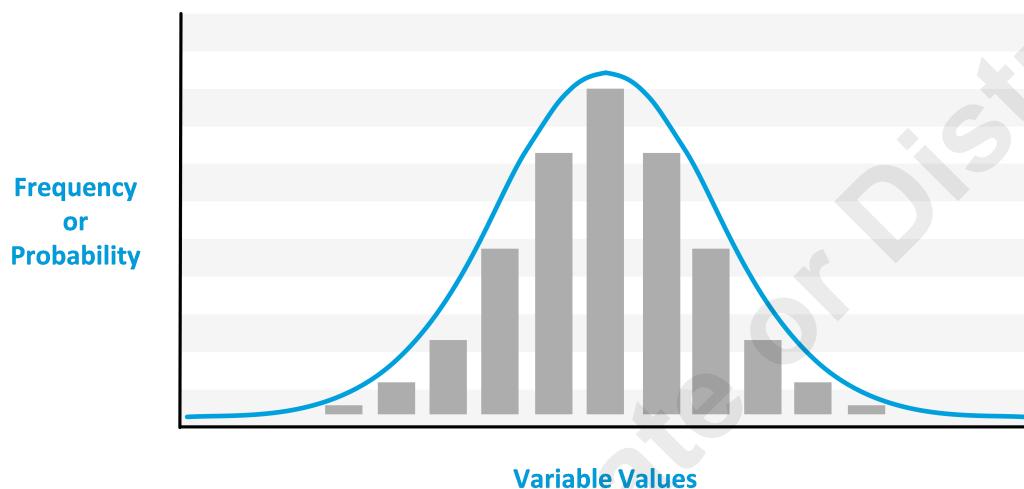


Figure 3-8: The bell shape that forms with normally distributed data.

The normal distribution (also called a **Gaussian curve**) has several visual properties:

- Bell shaped
- Symmetrical
- Centered
- **Unimodal** (only one hump)

In addition, the normal distribution has some mathematical properties that you'll see shortly.

Before moving on, consider the probability distribution of people's heights shown earlier.

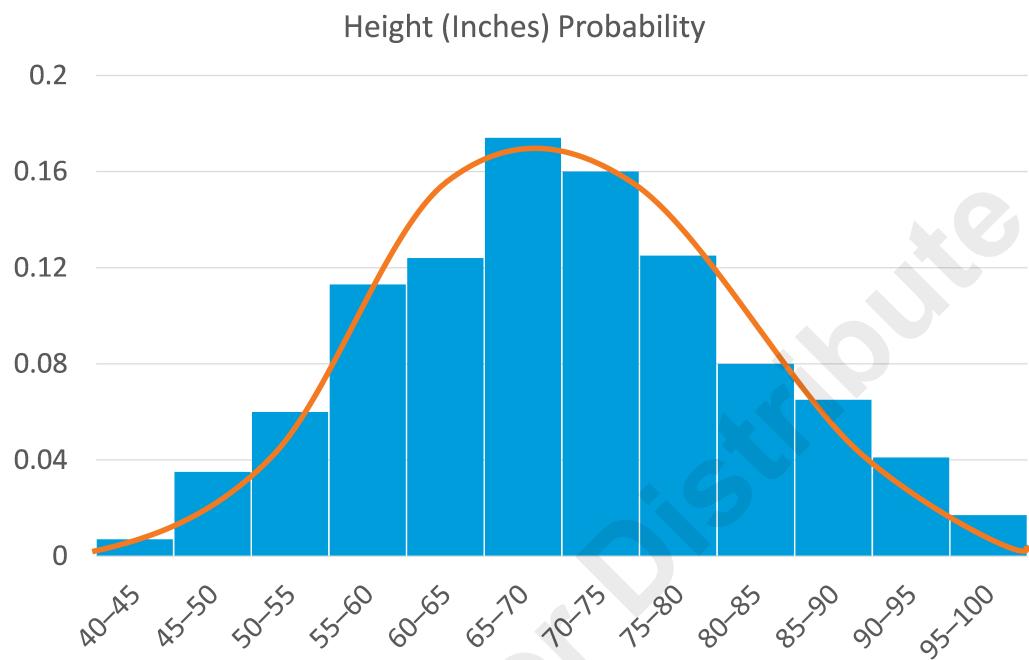


Figure 3–9: The height distribution can be said to approximate a normal distribution.

As you can see, this distribution seems to approximate normally distributed data. It isn't normally distributed to a perfect degree, but few things in nature are. Nevertheless, physical attributes like height and weight are examples of variables that come close to having normally distributed data.

Non-Normal Distributions

Of course, not every distribution is normal. There are various ways a distribution might depart from the normal curve. For example, a distribution might **skew**, with a high density of values distributed toward the lower or higher end of the x-axis, perhaps with a long tail on the other end.

While a normal distribution is unimodal, having only one hump in the middle, a distribution might be **bimodal** or **multimodal**—not having just one hump, but two or more.

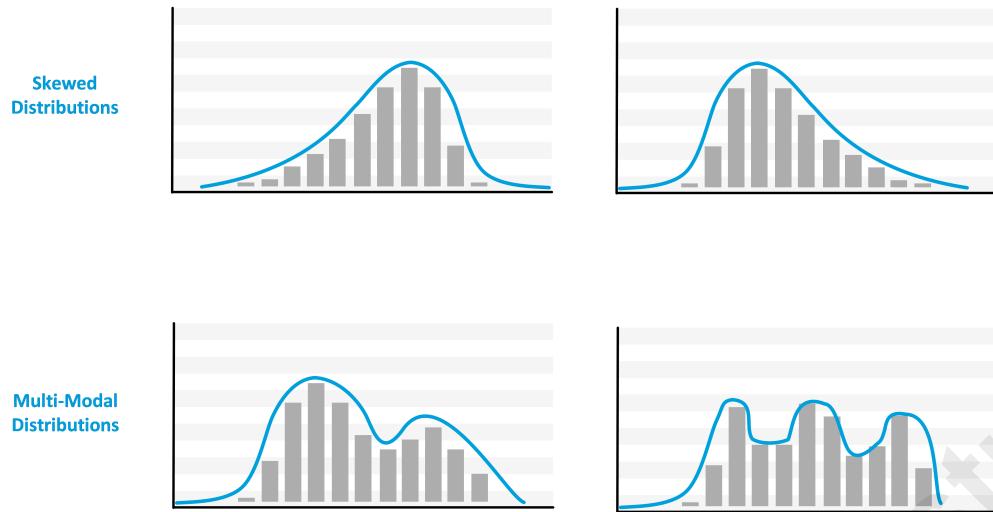


Figure 3-10: Non-normal distributions, including those that are skewed and multi-modal.

Using probability to make estimations is foundational to machine learning, so many data science tasks require data samples that are normally distributed. You must be careful to avoid applying statistical algorithms to data that is not distributed in a normal curve, since it could lead to the wrong conclusions. You may have to perform certain operations on the data to "normalize" it before you can use it effectively.

Descriptive Statistical Analysis

Descriptive statistical analysis, also called descriptive statistics, involves various measures or descriptions you can use to summarize patterns and relationships in data, using *numbers* produced by mathematical calculations, as well as *visualizations* such as graphs or tables that help to reveal significant information in those numbers. In addition to referring to the analysis method itself, the term "descriptive statistics" is also used to refer to the measurements that result from this analysis. Descriptive statistics are also called **summary statistics** since they summarize data quantitatively.

By looking at data the way a statistical model "looks at" data, you can gain insights that will help you improve the dataset's usefulness in developing such models. This will lead to more effective models that are much better at fulfilling whatever purpose you have set out for them.

Concepts described through descriptive statistics include, but are not limited to:

- Central tendency
- Variability
- Variance
- Standard deviation
- Skewness
- Kurtosis

Central Tendency

Measures of central tendency are descriptive statistics that summarize the "middle" portions of sample data. Using the following example numbers:

[10, 10, 10, 24, 28, 38, 62]

The measurements are:

- **Mean**—This simple average of all numbers in the set is sometimes also called the *arithmetic mean*. This value is calculated by adding together all numbers in the set, and then dividing by the total count of numbers.
In the example set, the mean is **26**.
- **Median**—The number value located in the true middle of a dataset that has been sorted from smallest value to largest value (or vice versa). If there is an even number of values, the average of the two middle values is the median.
In the example set, the median is **24**.
- **Mode**—The number that occurs most often within a set of numbers.
In the example set, the mode is **10**.

In a perfect normal distribution, the mean, median, and mode are all the same. This is one of the fundamental mathematical characteristics of a normal distribution, beyond just its visual shape. However, since many datasets are not perfectly normally distributed, each measure of central tendency is useful in describing the middle of a dataset. Differing central tendency statistics lead to skewed distributions, which you'll learn more about soon.

When to Use Different Measures of Central Tendency

While you might expect these three measures to produce a similar result within a normal distribution (some value near the center of the distribution), these different measures may identify very different values as the center of a non-normal distribution. The type of measure that will be most useful depends on the data within the distribution, as shown in the following table.

Measure	Most Useful for Describing
Mean	Continuous data in a symmetric distribution with no outliers.
Median	Distributions that are skewed or that have outliers.
Mode	Qualitative and categorical data—where data values are selected from a limited set such as ["for sale", "sale pending", "sold"] or ["public", "private"].

Variability

Variability is the extent to which data varies across all values in the dataset. Other words used to describe this aspect include dispersion, scatter, or spread. Comprehending the variability of a dataset includes a sense of whether the distribution is stretched out widely among a range of different values or condensed among a group of very similar values. Common measures of statistical dispersion include:

- Interquartile range
- Variance
- Standard deviation

Range Measures

Range is the difference between the smallest and largest values in the data. For example, suppose you are analyzing a quarterly sales report that has been cleaned to remove any outliers. If one sales rep made \$50,000 in sales, and another sales rep made \$320,000 in sales, then the range would be \$270,000—the difference between the lowest sales figure and highest sales figure. Identifying the minimum and maximum values gives you a sense of where the data is spread.

In conjunction with the minimum, maximum, and range, the central tendency measures (mean, median, and mode) start to give you a good sense of the distribution of values. Another way of

measuring range, **interquartile range (IQR)**, helps to show where the majority of values lie. In a quartile range measurement, you divide the range into four subsets, where each subset contains the same number of values. The quartiles are:

- **Q1**—The lowest 0–25% of values.
- **Q2**—The lowest 25–50% of values.
- **Q3**—The highest 50–75% of values.
- **Q4**—The highest 75–100% of values

The two inner sets (Q2 and Q3) are the interquartile range, where the middle half of data values lie. You can measure the total range of the values in these two sets to get the IQR value. A relatively low IQR indicates data that is densely packed around the center, whereas a high IQR indicates that data is more sparse.

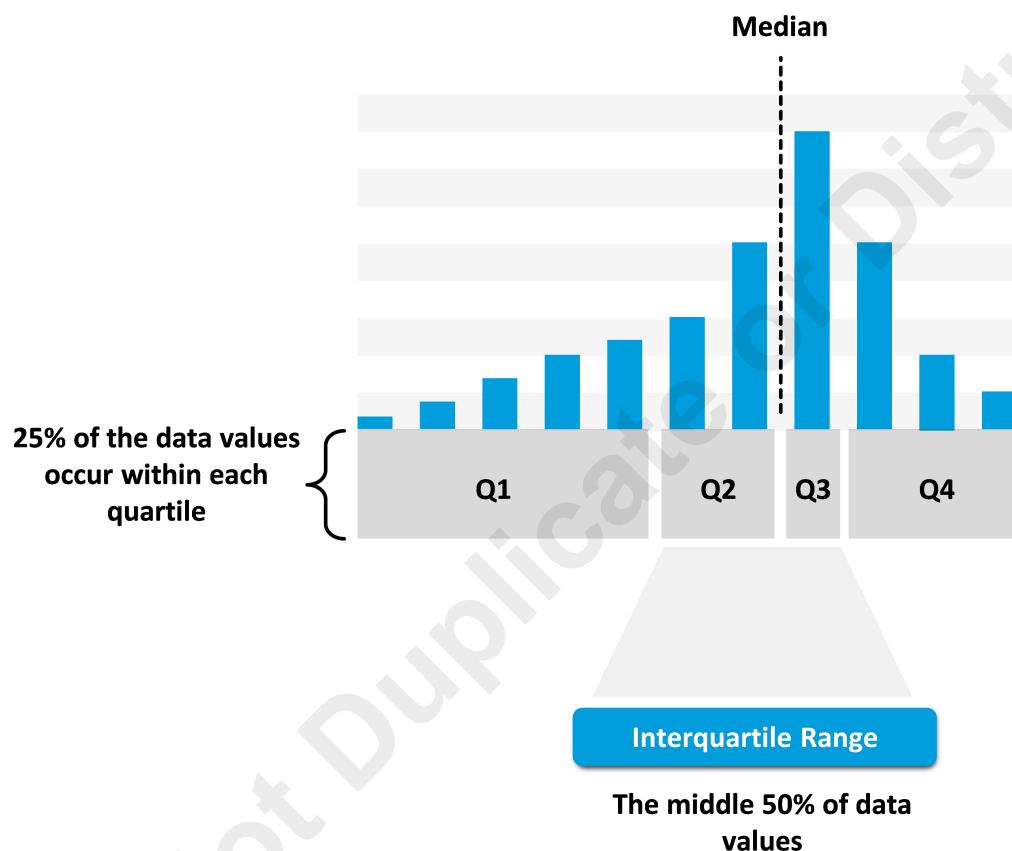


Figure 3-11: Interquartile range.

Variance and Standard Deviation

When comparing the variability of data in different datasets, you might need to compare datasets containing different numbers of samples or datasets with a different central tendency. But when focusing on variability, it is necessary to have a way to measure it independent of these other measures. To be most useful, then, a measure of variability:

- Is dependent on the amount of variability (dispersion, scatter, or spread) in the data.
- Is *not* dependent on:
 - The number of samples in the data.

- The central tendency of the data.

Both **variance** and **standard deviation** (which is based on variance) meet these requirements.

Calculation of Variance

Variance is a measure of how far each value in the dataset is from the mean.

The formula for *variance in a population* is expressed as:

$$\sigma^2 = \frac{\sum(X - \mu)^2}{N}$$

Where:

- σ^2 represents variance of a population.
- Σ is the summation operator, meaning "add the following numbers."
- X is the quantity measured in the data, such as sales per rep or the number of prospects.
- μ is the mean, the average of all the values in the population.
- N is the number of values included in the set.

	Note: In most data science scenarios, you can use software functions to perform this and other calculations described in this course. However, understanding how the calculation is performed will help you understand what it is measuring, so it is provided here for your reference.
	Note: Σ is the capital Greek letter sigma, σ is the lowercase version of sigma (also called "little sigma"), and μ is the Greek letter mu.

Variance is calculated for a column of data by:

1. Finding the mean (average) of all values in the set.
2. For each number in the set, subtracting the mean and then squaring the difference.
3. Adding up all of the values you calculated in step 2, and dividing by the number of values in the set to get the average.

In step 2, the process of squaring the difference provides two benefits. It gives more weight to values farther away from the mean. It also ensures that numbers less than the mean (which produce a negative value when subtracted) won't cancel out numbers greater than the mean (positive differences). Values that are less than the mean or greater than the mean will both have a positive influence on the measure of variance.

Sample vs. Population

In many cases, you'll be working with a sample set instead of an entire population. However, the difference will come down to the problem domain you're dealing with. If your task is to estimate the ages of a specific university's students, then the population would be every student at the university —data you may very well have. But if your task is to estimate the ages of any college students in the country, then you may just have one or more samples instead of an entire population.

Variance in a Sample Set

Often you must work with a **sample set**, a smaller group than the population. In such cases, you should use a slightly different formula. You should subtract 1 from the total number of values in the sample when performing the calculation. This is called **Bessel's correction**, and it is done to compensate for the lack of information that would be provided by the larger population.

The formula for *variance in a sample set* is expressed as:

$$s^2 = \frac{\sum(X - \bar{X})^2}{n - 1}$$

Where:

- s^2 represents variance of a sample.
- Σ is the summation operator, meaning "add the following numbers."
- x is the quantity measured in the data, such as sales per rep or the number of prospects.
- \bar{x} is the mean, the average of all of the values in the sample.
- n is the number of values included in the sample set.



Note: The n is shown in lowercase here to signify that the number of items refers to the count in a *sample* set, not the number of items in the *entire population*.

Calculation of Standard Deviation

You may wonder why variance is commonly shown as sigma *squared* (σ^2). The σ symbol represents *standard deviation*, a different measure of variability closely related to variance. In fact, standard deviation is simply the square root of variance. These two measures of variability are used for different purposes.

Variance is calculated by squaring the differences between each value and the mean. Because of this squaring operation, the resulting measure of variability may end up being much larger than the actual values in the dataset. While the resulting measure is quite useful for math operations, it is not intuitive for people trying to get a sense of the average amount of deviation in relation to the actual data values.

Standard deviation reverses the squaring operation to express the measure of variability in the same scale as the actual data values. This is helpful, since other descriptive measures such as the mean, median, mode, min, and max, are all in the same scale as the data itself. So standard deviation is often used for explaining or reporting purposes. It is easy for people to understand the amount of deviation in relationship to the scale of the actual values.

The formula for *standard deviation for an entire population* is expressed as:

$$\sigma = \sqrt{\frac{\sum(X - \mu)^2}{N}}$$

Where:

- σ represents the standard deviation of a population.
- Σ is the summation operator, meaning "add the following numbers."
- x is the quantity measured in the data, such as sales by rep or the number of prospects.
- μ is the mean, the average of all the values in the set.
- N is the number of values included in the set.

If you have already obtained the variance, you can obtain the standard deviation simply by getting the square root of the variance.

Standard Deviation in a Sample Set

When you are working with a sample, you must adjust the formula for standard deviation like you adjust the variance formula. Thus the formula for *standard deviation for a sample* is expressed as:

$$s = \sqrt{\frac{\sum(X - \bar{X})^2}{n - 1}}$$

Uses for Standard Deviation

The result you get from plugging data into the standard deviation formula tells you about the spread of data irrespective of the mean. Larger standard deviation values indicate data points spread out from the mean, whereas smaller standard deviation values indicate data points are grouped closer to the mean. Two samples can have the same mean but different standard deviations, so, as mentioned before, you can use this as a way to compare different distributions. If two samples of people's heights both have 67 inches as the mean, the sample with a standard deviation of 20 will have its data more dispersed than the sample with a standard deviation of 5.

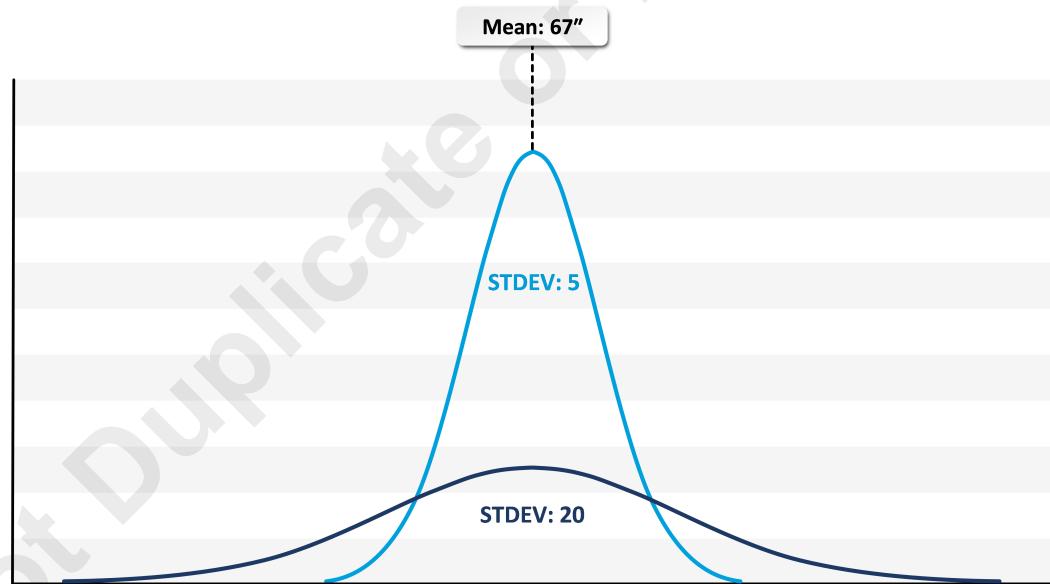


Figure 3-12: Two samples with the same mean, but different standard deviation values.

Standard deviation is also commonly used to identify dispersion in a normal distribution. In a normal distribution, approximately 68% of all data examples are within one standard deviation of the mean in both directions. So, with a standard deviation of 10, 68% of all people in the sample are between 57 and 77 inches tall. Likewise, 95% of the sample is within two standard deviations of the mean. And, 99.7% of the sample is within three standard deviations of the mean. You can use these values (called the *empirical rule*) to possess a degree of certainty that the population's height values are within a specific range.

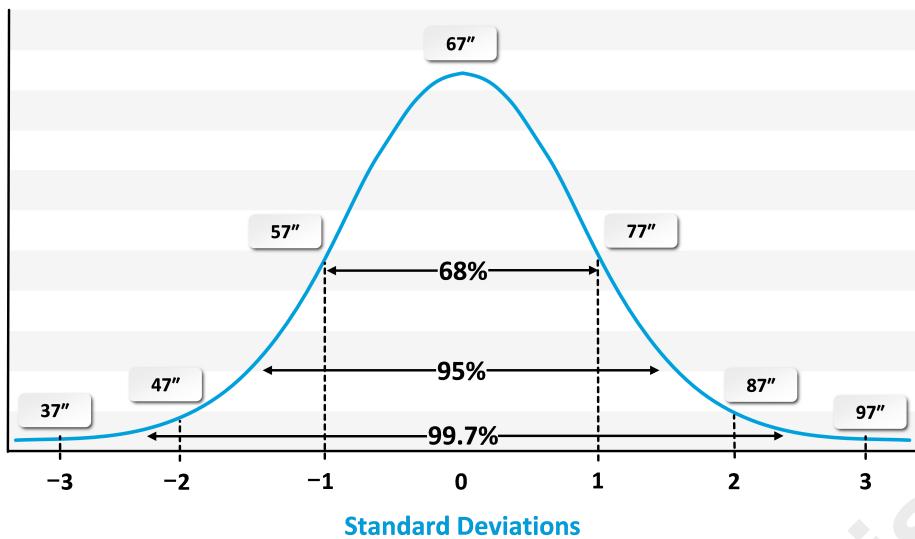


Figure 3-13: Probabilities of data spanning three standard deviations in a normal distribution.



Note: It's common to remove any outliers that are three or more standard deviations from the mean, which would be 0.3% of data in a normal distribution.

Skewness

Skewness describes the extent to which a distribution's symmetry differs from a normal distribution (zero skew, or symmetrical). Right skew, also called positive skew, shows data that tapers off to the right side of the distribution. Left skew, also called negative skew, shows data that tapers off to the left side of the distribution. In a skewed distribution, the mean, median, and mode are not the same.



Note: By default, the "left" and "right" directions indicate where the tail is tapering off, not where the peak is. If you want to refer to the direction of the peak, you should be explicit, e.g., "the peak in a positive skew is shifted to the left."

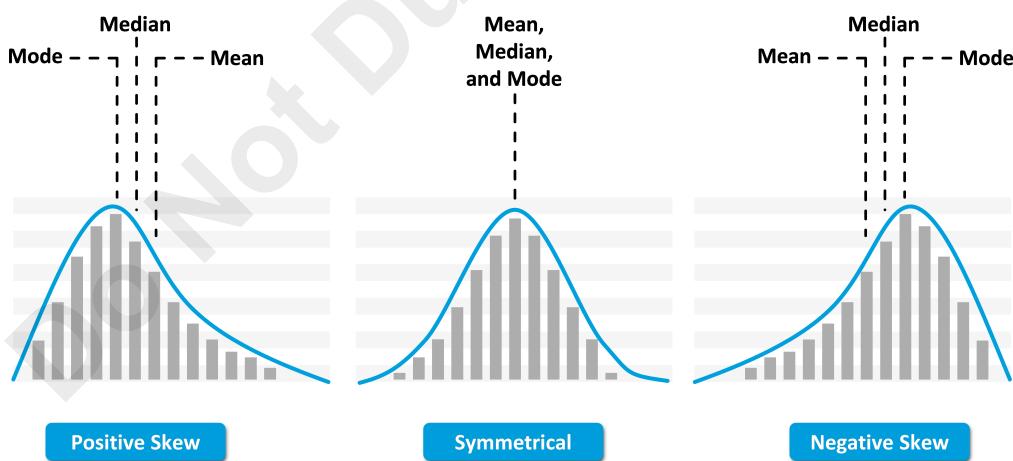


Figure 3-14: Three types of skewness.



Note: Positive skews tend to occur more frequently than negative skews.

There are various numerical measures of skewness, including Pearson's first coefficient of skewness, which is based on the mode, and Pearson's second coefficient of skewness, which is based on the median. While the first method is sufficient for large datasets, for small datasets, the second method may be better because mode may not be the best measure of central tendency when there are a small number of values in the set.

Calculation of Skewness

The formulas for Pearson's skewness measures are shown here.

$$Sk_1 = \frac{\bar{X} - Mo}{\sigma}$$

$$Sk_2 = \frac{3(\bar{X} - Md)}{\sigma}$$

Where:

- \bar{X} is the mean.
- σ is the standard deviation.
- Mo is the mode.
- Md is the median.



Note: Pearson's skewness measures are named after mathematician Karl Pearson.

While the two formulas produce different values, you interpret the result the same way.

- The sign of the result (positive or negative) identifies the direction of skewness. A positive value means the peak is shifted to the left, and a negative value means the peak is shifted to the right. Zero means no skewness at all.
- The size of the result (how much it differs from zero) represents the degree of skewness from a normal distribution. Values farther away from zero represent more skewness.

So, a right-skewed distribution indicates that there are more data examples at the lower end of the distribution than at the higher end. Conversely, a left-skewed distribution indicates that more data examples are at the higher end of the distribution than the lower end. Both examples of skewness can reveal the presence of extreme values in one side of the distribution as compared to the other.



Note: A good rule of thumb is that values greater than 1 or less than -1 indicate a significant degree of skewness that you might want to address.

Kurtosis

In a normal bell-shaped distribution, there are tails on the left and right sides. The **kurtosis coefficient** is a measure of the shape of these tails. It measures the combined weight of the tails relative to the center of the distribution. A normal bell-shaped distribution would have a kurtosis coefficient of 3. A distribution with the classic bell shape is described as **mesokurtic**.

A distribution may be bunched to the center, dropping off quickly at the sides, resulting in heavy tails (i.e., tails that include a lot of data). The peak of this distribution is said to be narrow (or skinny). A kurtosis measurement greater than 3 would result from a heavy-tailed distribution, described as **leptokurtic**.

On the other hand, a distribution may be flat with light tails (i.e., tails that don't include a lot of data), described as **platykurtic**. The peak of this distribution is said to be flat. This sort of distribution would have a kurtosis coefficient less than 3.

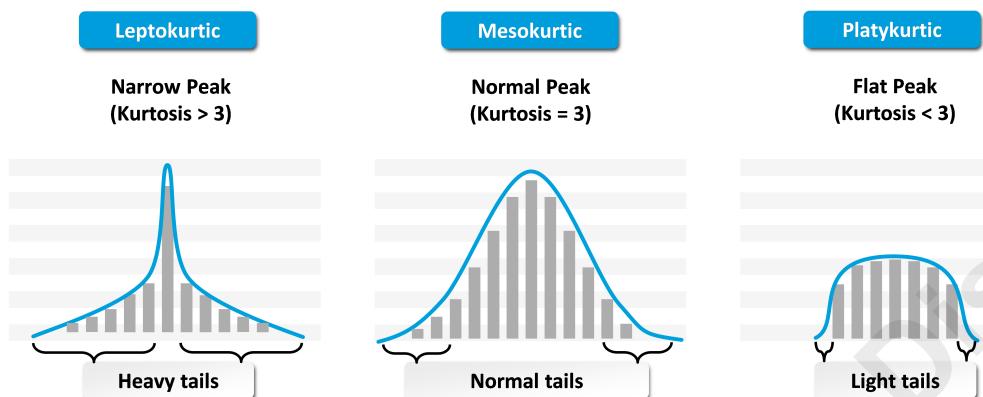


Figure 3-15: Three types of kurtosis.

Calculation of Kurtosis

Kurtosis for a sample set is defined as:

$$a_4 = \sum \frac{(X_i - \bar{X})^4}{ns^4}$$

Where:

- a_4 represents the measure of kurtosis.
- X_i is the i^{th} X value in the dataset.
- \bar{X} is the mean.
- n is the sample size.
- s is the sample standard deviation.

For a normal (mesokurtic) distribution, this formula would produce a kurtosis coefficient of 3. Values less than 3 are platykurtic, and values greater than 3 are leptokurtic. However, to make it simpler to identify distributions that are not normal, 3 is often subtracted from the kurtosis result, producing a value called *excess kurtosis*, which shows zero for a normal distribution. Platykurtic distributions would then be negative, and leptokurtic distributions would be positive.

Distributions that are leptokurtic may indicate the presence of outliers, whereas platykurtic distributions usually indicate a lack of outliers. One is not necessarily better than the other in all cases, however. When you are training a statistical model, the kurtosis measurement may reveal problems in your training data that you need to address before you fit the model. Kurtosis can also be used in conjunction with skewness, standard deviation, and other descriptive measures to give you a bigger picture of the data you're working with.



Note: If you use applications or programming functions to calculate the kurtosis coefficient, you need to find out whether the result shows *kurtosis* or *excess kurtosis*, so you can make the right judgement based on the result. Read the documentation for the function to determine what is actually calculated, as functions named *kurtosis* may actually return the value of *excess kurtosis*.

Statistical Moments

Various measures covered so far describe the characteristics of a distribution. Collectively, these measures are sometimes referred to as statistical *moments*.

- **First moment**—The *mean*, which identifies where numbers typically lie.
- **Second moment**—The *variance*, which describes the spread or scale of the distribution.
- **Third moment**—The *skewness*, which describes the symmetry of the distribution.
- **Fourth moment**—The *kurtosis*, which describes the flatness or peakedness of the distribution.

Guidelines for Exploring the Underlying Distribution of Data

Follow these guidelines when exploring the underlying distribution of data.

Explore the Underlying Distribution of Data

When exploring the underlying distribution of data:

- Use histogram, scatter, and other plots to get a sense of the shape of the data. Plot lines for statistical measures like mean and median. One or two standard deviations can be included too.
- Look at the distribution of different fields as each may have a different distribution.
- Always cross-check assumptions about distributions by comparing and finding correlation with standardized distributions and calculated values like skew and kurtosis, as your initial perception of plots may not always be accurate.
- Prefer to use standard deviation over variance when explaining the variability of data. This keeps the results in the same scale as the data.
- Ensure you're using measures of central tendency that apply to the feature type you're exploring; e.g., don't try to obtain the mean of a categorical variable.

ACTIVITY 3-2

Exploring the Underlying Distribution of Data

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Generate summary statistics for all of the data**. Select **Cell**→**Run All Above**.

Scenario

Now that you've explored the structure and format of the GCNB dataset, you can begin to dive a little deeper into the nature of the values themselves. You'll focus on exploring the distribution of these values, which should give you a sense of how they are spread out and how they can be described using various summary statistics.

1. Generate summary statistics for all of the data.

- Scroll down and view the cell titled **Generate summary statistics for all of the data**, then select the code cell below it.
- In the code cell, type the following:

```
1 | users_data.describe(datetime_is_numeric = True, include = 'all')
```

- Run the code cell.
- Examine the output.

	user_id	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	balance	day	month	year
count	45209	35210.000000	35210.000000	45209.000000	44921	45209	43352	45209	45209	45209	32191	45209.000000	0.000000	0.000000	0.000000	0.000000
unique	45209	NaN	NaN	NaN	11	3	3	2	2	2	2	2	2	2	2	2
top	729b0249-3961-4672-81cd-14b5ec6bdd39	NaN	NaN	NaN	blue-collar	married	secondary	False	True	False	cellular	0.000000	0.000000	0.000000	0.000000	0.000000
freq	1	NaN	NaN	NaN	9731	27212	23202	44394	25128	37965	29285	0.000000	0.000000	0.000000	0.000000	0.000000
mean	NaN	3.977052	1369.417751	40.935853	NaN	NaN	NaN	NaN	NaN	NaN	NaN	258.100000	0.000000	0.000000	0.000000	0.000000
min	NaN	1.000000	-8019.000000	18.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000
25%	NaN	2.000000	73.000000	33.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	103.000000	0.000000	0.000000	0.000000	0.000000
50%	NaN	3.000000	451.000000	39.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	180.000000	0.000000	0.000000	0.000000	0.000000
75%	NaN	4.000000	1438.000000	48.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	319.000000	0.000000	0.000000	0.000000	0.000000
max	NaN	20.000000	102127.000000	95.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4918.000000	0.000000	0.000000	0.000000	0.000000
std	NaN	3.814329	3063.412688	10.618653	NaN	NaN	NaN	NaN	NaN	NaN	NaN	257.500000	0.000000	0.000000	0.000000	0.000000

Various statistics such as mean, frequency, min, max, standard deviation, and more are printed. This can get a little overwhelming, so you'll get more granular with how you retrieve them.

2. Generate summary statistics for numerical data only.

- Scroll down and view the cell titled **Generate summary statistics for numerical data only**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data.describe()
```

- Run the code cell.
- Examine the output.

	number_transactions	total_amount_usd	age	duration	campaign	pdays	previous
count	35210.000000	35210.000000	45209.000000	45209.000000	45209.000000	45209.000000	45209.000000
mean	3.977052	1369.417751	40.935853	258.153067	2.763897	40.199651	0.580349
std	3.814329	3063.412688	10.618653	257.525446	3.098076	100.130586	2.303489
min	1.000000	-8019.000000	18.000000	0.000000	1.000000	-1.000000	0.000000
25%	2.000000	73.000000	33.000000	103.000000	1.000000	-1.000000	0.000000
50%	3.000000	451.000000	39.000000	180.000000	2.000000	-1.000000	0.000000
75%	4.000000	1438.000000	48.000000	319.000000	3.000000	-1.000000	0.000000
max	20.000000	102127.000000	95.000000	4918.000000	63.000000	871.000000	275.000000

This output is a little neater, and focuses just on numeric features. Some things to point out include:

- The mean of each variable is on a very different scale. The mean age is 40, whereas the mean total_amount_usd is around \$1,369.
- The standard deviation for each variable is roughly on the same scale as its mean, though total_amount_usd has quite a large standard deviation, indicating the values are spread out. This is partially due to the fact that some transactions are deposits (positive) whereas some are withdrawals (negative). The minimum value of -\$8,019 confirms this.
- The 50% statistic is the same as the median. So, it looks like the median age is close to the mean age, which could suggest a normal distribution.

3. Generate modal values for all data.

- Scroll down and view the cell titled **Generate modal values for all data**, then select the code cell below it.
- In the code cell, type the following:

```
1 # Drop user ID since it's unique.
2
3 users_data.drop(['user_id'], axis = 1).mode()
```

This will retrieve the mode for each variable, except for user_id, which is unique (and not particularly relevant).

- Run the code cell.

- d) Examine the output.

	number_transactions	total_amount_usd	age	job	marital	education	default	housing	loan	contact	duration	campaign	pdays	previous	poutcome
0	2.0	0.0	32.0	blue-collar	married	secondary	False	True	False	cellular	124.0	1.0	-1.0	0.0	failure
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In this particular dataset:

- The most common age is 32.
- The most common type of job is a blue collar job.
- The most common marital status is married.
- The most common level of education is secondary education.
- Most users didn't default on their loans.
- Most users contact the bank using their mobile phones.
- And so on.



Note: There are two rows because the `date_joined` variable is bimodal. You can see both modes by scrolling all the way to the right.

4. Generate skewness and kurtosis measurements.

- Scroll down and view the cell titled **Generate skewness and kurtosis measurements**, then select the code cell below it.
- In the code cell, type the following:

```
1 | users_data.skew()
```

It's easier to interpret skewness and kurtosis visually, which you'll do a bit later. For now, you can just get a quick look at the numeric results.

- Run the code cell.

- d) Examine the output.

number_transactions	2.704543
total_amount_usd	8.596128
age	0.684861
default	7.245206
housing	-0.224686
loan	1.852545
duration	3.144556
campaign	4.898555
pdays	2.615635
previous	41.845672
term_deposit	2.383403
dtype:	float64

Positive skewness numbers indicate the peak of the distribution is to the left (at the lower end of values) and the tail tapers off to the right; negative implies the opposite. Values closer to zero indicate a lack of skewness. So, it looks like:

- All of the features have a positive skew except for housing.
- Features like age and housing are close to zero, so they are closer to being normalized.
- The previous feature (number of times user was contacted prior to the current campaign) has by far the highest skew value.



Note: The `skew()` function takes the skewness measurements of Boolean variables as well as numeric variables. It considers `False` as 0 and `True` as 1. So, for example, `default` has such a highly positive skew because there are many more `False` (0) values (users who didn't default on a loan) than `True` (1) values (users who did default on a loan). The peak is therefore shifted to the left, closer to 0.

- e) Select the next code cell, then type the following:

```
1 | users_data.kurt()
```

- f) Run the code cell.

g) Examine the output.

```
number_transactions      6.659034
total_amount_usd        150.790967
age                      0.319760
default                  50.495241
housing                 -1.949602
loan                     1.431987
duration                18.155941
campaign                39.248145
pdays                    6.934713
previous                4506.684640
term_deposit             3.680770
dtype: float64
```

The `kurt()` function in pandas uses excess kurtosis, so zero implies a normal distribution whereas negative values are platykurtic and positive values are leptokurtic. So:

- Most of the features are leptokurtic, implying that there are outliers.
- The `total_amount_usd` and `previous` features have the highest leptokurtosis, so they may have some extreme outliers.
- The `housing` feature is slightly platykurtic, meaning it likely isn't as affected by outliers.



Note: Just like `skew()`, the `kurt()` function considers Boolean values as the numbers 0 and 1. Since there are really only two numbers to consider, a feature like `default` is highly leptokurtic because one of those numbers is much more frequent than the other, leading to a peak. A feature like `housing` is much more evenly distributed, so it's mostly flat without a significant peak.

5. Keep this notebook open.

TOPIC C

Use Visualizations to Analyze Data

In this topic, you'll look at your data from a visual perspective in order to reveal insights that raw numbers alone may not provide.

Visualizations

Data visualization involves representing data in charts that clearly depict patterns, trends, and correlations that might go unnoticed when the data is shown as a list of numbers, text, and other data values. Data science practitioners have long understood the value of good visualization tools to provide insights and to help communicate data outcomes, and have used charts for centuries, long before computers came into use.

Visualization tools can be useful at various points throughout a data science project, including when you communicate your results to stakeholders. At this point, however, you'll focus on generating visualizations for you or your team members to analyze, rather than present them to a general audience.

There are many tools out there that you can use to generate data visualizations. Some are external programs that can ingest your data in various ways and output a number of different visualizations, like Microsoft® Excel® or Tableau®. Other visualization software comes in the form of libraries for whatever programming language you're using, like Matplotlib and Seaborn for Python™, or ggplot2 for R. Which tool you use will be based on your own business needs and level of comfort with using those tools. However, most of these tools can generate the same kinds of visualizations, so the choice usually comes down to whatever one (or more) best integrates with your existing or planned environment.

Histograms

A **histogram** compares different frequencies of one continuous numeric variable. In other words, it represents the distribution of a continuous variable. Since a continuous variable can extend infinitely and has no defined gaps between each value, the variable is placed into multiple bins that divide the entire range of the variable along the horizontal axis. The vertical axis shows the frequency of observations in each bin. So, a continuous variable can be something like price, revenue, velocity, time, and so on. The histogram will show how many observations fit into each bin.

The charts used to demonstrate probability distributions earlier were histograms. So, histograms provide a visual summary that can be quickly interpreted to understand where values are concentrated, where the extremes are located, the general skewness of the distribution, and whether there is multimodality.

The following figure is based on a dataset of responses to a survey conducted by a fast food restaurant. Respondents filled out the survey after each purchase over a seven-day period. Each respondent is a row, and `sale_price` (the total price for the respondent's transaction) is a feature, as is the respondent's age.



Figure 3-16: Two histograms that show different distributions.

In the histogram on the left, the distribution of sale prices is skewed to the right, indicating that fewer sales prices are on the higher end. It appears that most sales are in the \$5–\$15 range. The distribution also reveals the presence of some outliers, like the sale price approaching \$70. In the histogram on the right, the age of the respondents appears to be multi-modal, with noticeable spikes for respondents in their early late 20s/30s and later 40/early 50s. Despite being multi-modal, there isn't much of a skew, so the histogram doesn't reveal any major outliers when it comes to age.

Guidelines for Analyzing Data Using Histograms

Follow these guidelines when analyzing data using histograms.

Analyze Data Using Histograms

When analyzing data using histograms:

- Manually adjust the number of bins using observation and experimentation.
- Plot multiple groups of numerical data on the same chart to identify overlap between those groups.
- Use histograms to reveal multimodality in data.

ACTIVITY 3–3

Analyzing Data Using Histograms

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Plot histograms for all numerical columns**. Select **Cell→Run All Above**.

Scenario

You want to start exploring the GCNB dataset visually to aid in your interpretation. There are many chart types you can construct from this data, but you'll start with histograms in order to view the distributions of the several features.

1. Plot histograms for all numerical columns.

- Scroll down and view the cell titled **Plot histograms for all numerical columns**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data_for_hist = \
2 users_data.select_dtypes(exclude = ['bool'])
```

Before drawing histograms, you'll need to temporarily drop the categorical Boolean data from the dataset, as that data is not continuous.

- Run the code cell.
- Select the next code cell, then type the following:

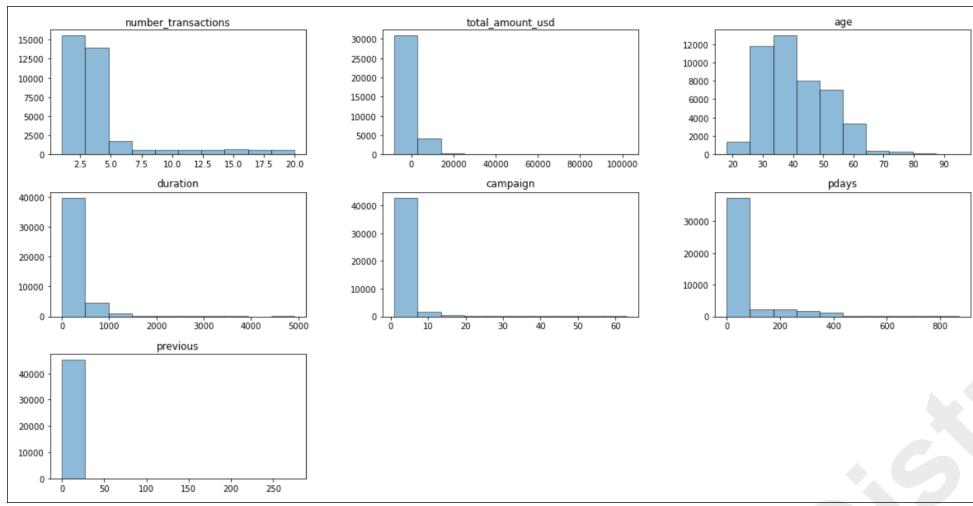
```
1 users_data_for_hist.hist(figsize = (20, 10), alpha = 0.5,
                           edgecolor = 'black', grid = False);
```



Note: In Jupyter Notebook, the semicolon (;) at the end of the last function call in a code cell suppresses that function's text output.

- Run the code cell.

- f) Examine the output.



These histograms visually confirm the skewness measurements you saw earlier. As you can see, most of the data is heavily skewed to the right (positive). The only variable that comes close to a normal distribution is `age`. You'll do some transformations for these skewed variables later on.

2. Keep this notebook open.

Box Plots

A **box plot**, also called a box-and-whisker plot, is another method of displaying the distribution of numerical data. It visually represents the range measurements mentioned previously, including the quartile ranges and minimum and maximum. Either the horizontal or vertical axis shows the range of the numeric variable, and the range measurements are drawn to match the appropriate values along that range. The measurements are:

- **Median**—A line is drawn at the median value.
- **Q1**—A box is drawn before the median to represent the first quartile range.
- **Q3**—A box is drawn after the median to represent the third quartile range.
- **Minimum**—A line is drawn at the minimum value, also called the lower bound. This is the lowest value in the data *excluding outliers*. It is calculated as $Q_1 - 1.5 \times IQR$.
- **Maximum**—A line is drawn at the maximum value, also called the upper bound. This is the highest value in the data *excluding outliers*. It is calculated as $Q_3 + 1.5 \times IQR$.

The following figure represents the spread of sale prices in the fast food survey again, but uses a box plot this time instead of a histogram.

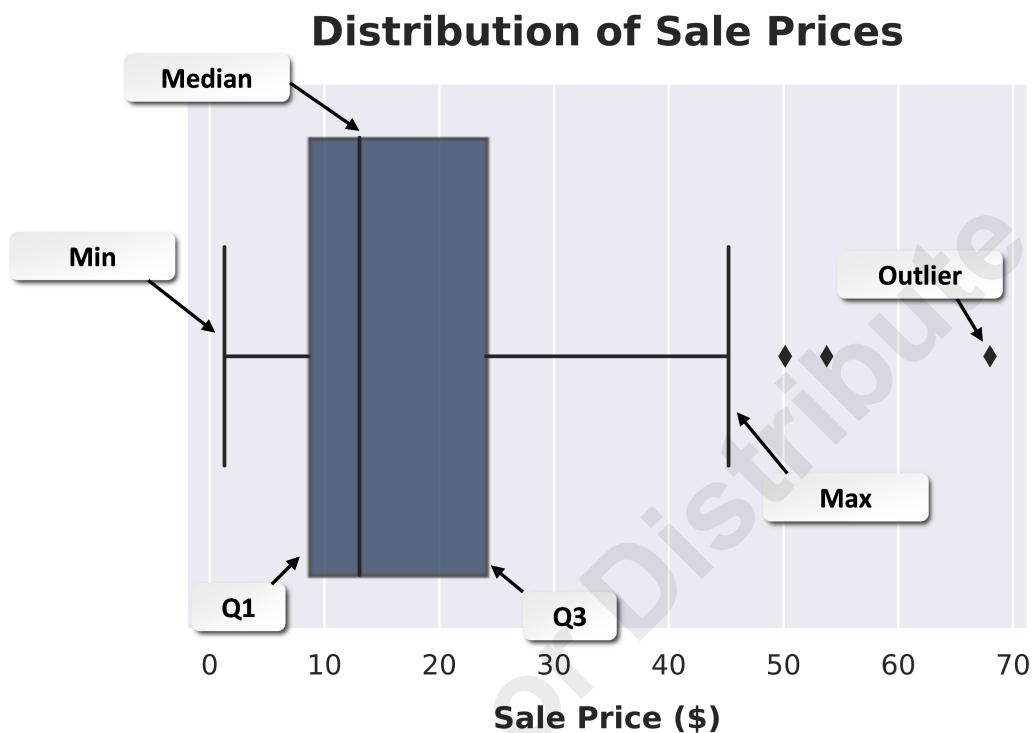


Figure 3–17: A box plot of sale prices.

Like the histogram, this confirms that the majority of sale prices are on the lower end of the distribution. However, the box plot is a little better at showing outliers, which you can see as the dots that extend past the maximum value to the right of the graph.

Violin Plots

A **violin plot** is a method of showing the distribution of a numerical value through probability density. It is similar to a box plot in that it can show a summary statistic like IQR; however, it can also reveal more about where the data tends to fall within the range of values. The distribution is computed using a kernel density estimation (KDE), which differs from how a histogram distribution is computed. The distribution of a KDE is smoothed into curves, rather than represented as rectangular bins like a histogram. This helps alleviate some of the issues of a histogram, such as the difficulty of choosing a good bin size. A violin plot gets its name from the fact that the KDE distribution extends from both sides of the range line, forming a shape that is reminiscent of a violin. The wider areas of the distribution indicate a greater probability of data samples being at that value, whereas thinner areas indicate lower probability.

Violin plots are most often used to compare distributions across similar data samples. In the figure, the mean sale price for two different days are plotted, where each day is its own violin.

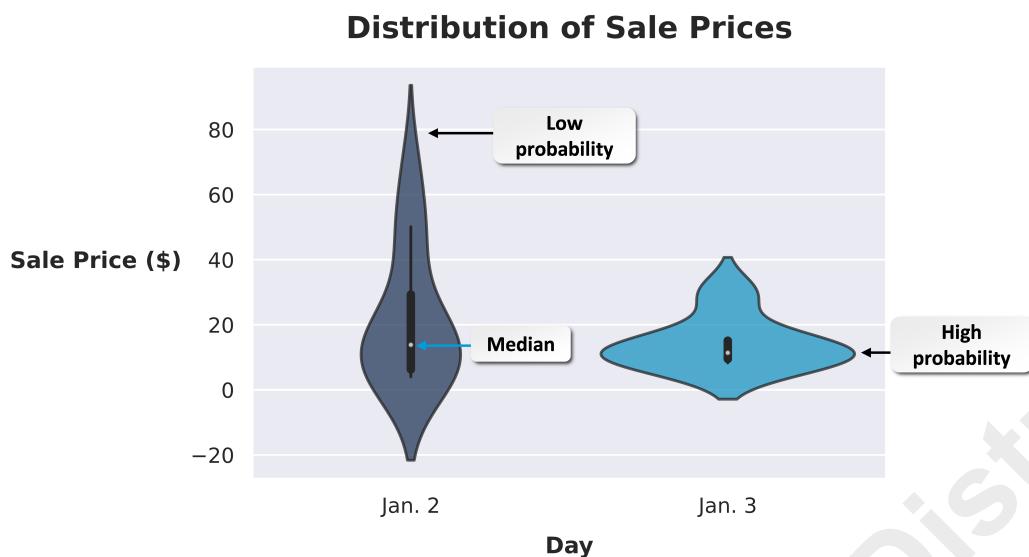


Figure 3-18: Violin plots comparing the distribution of sale price for each day.

The sale prices for January 2nd are very dispersed; it seems that quite a few customers placed larger orders, causing the violin to spike toward the higher end. Also, the \$10 value seems to have the highest probability. The violin for January 3rd is less dispersed overall, but it exhibits a significant degree of variation within that dispersion. Sale prices are much more likely to be around \$10, and much less likely to be slightly above or below that.



Note: The violins extend below \$0 because of how KDE is calculated. This does not mean that some sale prices were below \$0, only that there is a *probability* of them being so, as KDE is not automatically aware of minimum or maximum possible values. Depending on the library you use, you can configure a cutoff point for the violins so that they do not extend into improbable territory.

Guidelines for Analyzing Data Using Box Plots and Violin Plots

Follow these guidelines when analyzing data using box plots and violin plots.

Analyze Data Using Box Plots and Violin Plots

When analyzing data using box plots and violin plots:

- Use a series of box plots to show many statistical measures (mean, median, min, max, etc.) across a number of datasets.
- Use violin plots to show an extra dimension of value densities that box plots can't show.
- Experiment with different colors for each violin plot.
- Create violin and box plots when there are multiple datasets to display at once, plotting them on the same axes.

ACTIVITY 3–4

Analyzing Data Using Box Plots and Violin Plots

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Generate a box plot for age**. Select **Cell**→**Run All Above**.

Scenario

Histograms are great for showing general distribution shape, but there are other ways to visualize the spread of values. You'll create box plots and violin plots to help you identify outliers that you may want to remove so that they don't unduly influence any models you plan on building.

1. Generate a box plot for age.

- a) Scroll down and view the cell titled **Generate a box plot for age**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_data['age'].describe()
```

First you'll call up some of the descriptive statistics for the `age` feature, just as a refresher.

- c) Run the code cell.
- d) Examine the output.

count	45209.000000
mean	40.935853
std	10.618653
min	18.000000
25%	33.000000
50%	39.000000
75%	48.000000
max	95.000000
Name:	age, dtype: float64

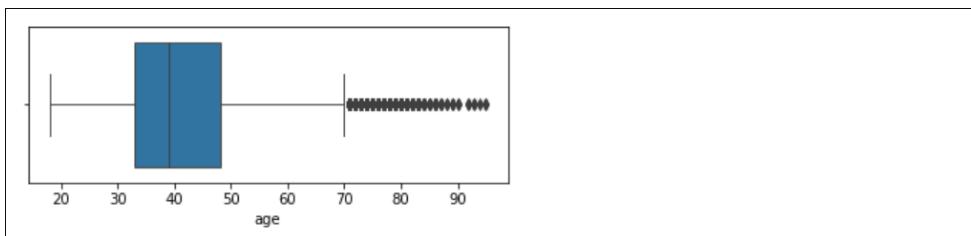
The minimum age is 18 and the maximum is 95. The standard deviation is 10 and the mean is 40, so some of these ages might be outliers.

- e) Select the next code cell, then type the following:

```
1 plt.figure(figsize = (6, 2))
2 sns.boxplot(x = users_data['age'], linewidth = 0.9);
```

- f) Run the code cell.

- g) Examine the output.



The result of the box plot shows the minimum and maximum as vertical lines at the end of each whisker. Note these values exclude outliers. In fact, the box plot clearly shows that there are several outliers on the higher end of the distribution. It seems that users past the age of 70 are skewing the distribution. You may want to drop these outliers from the dataset, but for now, you'll leave them be.

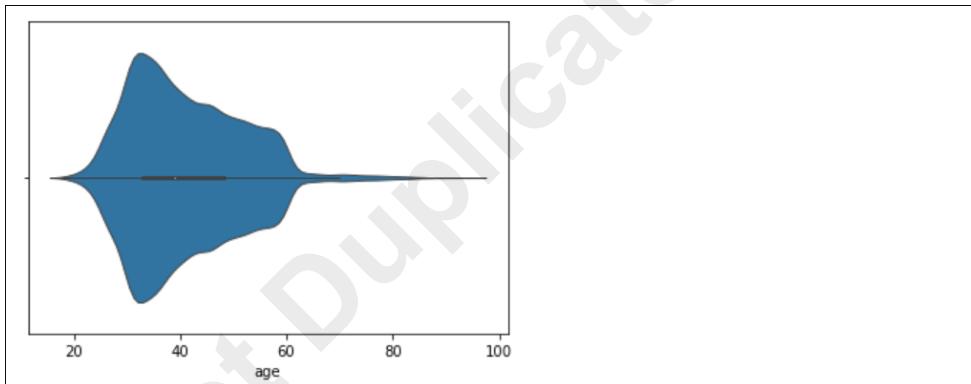
2. Generate a violin plot for age.

- Scroll down and view the cell titled **Generate a violin plot for age**, then select the code cell below it.
- In the code cell, type the following:

```
1 sns.violinplot(x = users_data['age'], linewidth = 0.9);
```

A violin plot is another way of showing variable distribution. It uses kernel density estimation (KDE) to map the distribution.

- Run the code cell.
- Examine the output.



The wider portion of the violin demonstrates a greater probability of values occurring at the point of the distribution. So ages around 30–35 are pretty likely in this dataset, whereas any age above 60 or below 20 is pretty unlikely.

3. Generate a box plot for number_transactions.

- Scroll down and view the cell titled **Generate a box plot for number_transactions**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data['number_transactions'].describe()
```

- Run the code cell.

- d) Examine the output.

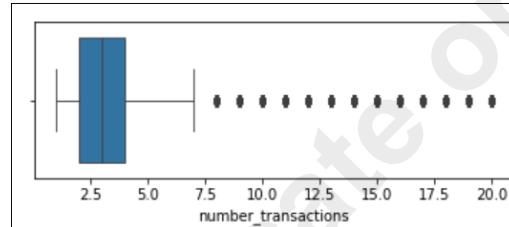
```
count    35210.000000
mean     3.977052
std      3.814329
min     1.000000
25%     2.000000
50%     3.000000
75%     4.000000
max     20.000000
Name: number_transactions, dtype: float64
```

This feature could also be susceptible to outliers, especially at the higher end, so you'll generate a box plot for it as well.

- e) Select the next code cell, then type the following:

```
1 plt.figure(figsize = (6, 2))
2 sns.boxplot(x = users_data['number_transactions'],
3              linewidth = 0.9);
```

- f) Run the code cell.
g) Examine the output.



As suspected, there are also outliers at the higher end of this feature's distribution. Some customers had an unusually large number of transactions, which is defined as being more than about 7 transactions.

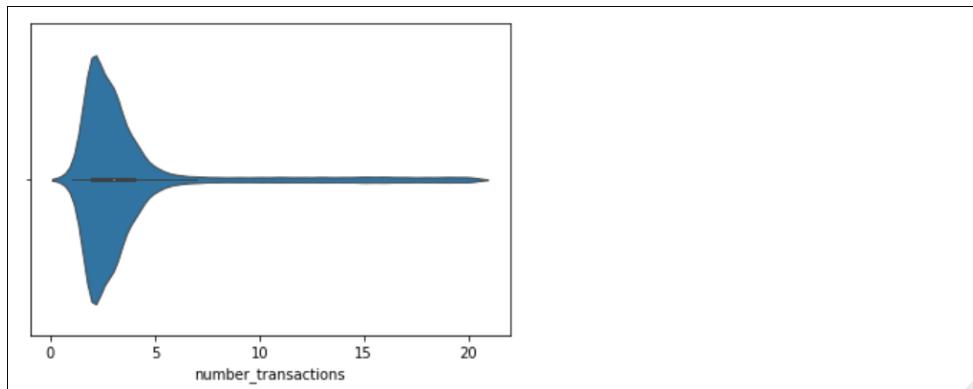
4. Generate a violin plot for number_transactions.

- a) Scroll down and view the cell titled **Generate a violin plot for number_transactions**, then select the code cell below it.
b) In the code cell, type the following:

```
1 sns.violinplot(x = users_data['number_transactions'],
2                  linewidth=0.9);
```

- c) Run the code cell.

- d) Examine the output.



The probability of the number of transactions being around 2–3 is pretty high, whereas anything above 5 has a low probability.

5. Keep this notebook open.

Scatter Plots

A **scatter plot** visually represents the relationship between two variables through the use of points on a graph. It is typically plotted in two dimensions, where the horizontal axis (the x-axis) corresponds to variable x and the vertical axis (the y-axis) corresponds to variable y . A dot is placed for each data example, where that example's values in x and y intersect.

Scatter plots are useful in determining how two numeric variables correlate. So, as the values in one feature increase, the values in another feature may also increase—a positive correlation, as you've seen. Or, as the values in one feature increase, the values in another feature may decrease—a negative correlation. Of course, it's also possible that there is no discernible pattern to how the points are spread, which indicates that there is no real correlation between the variables.

In the fast food survey, respondents were also asked to provide their age. So, you might be interested in studying how (or if) age and sale price correlate. You can easily see this using a scatter plot, as in the follow figure.

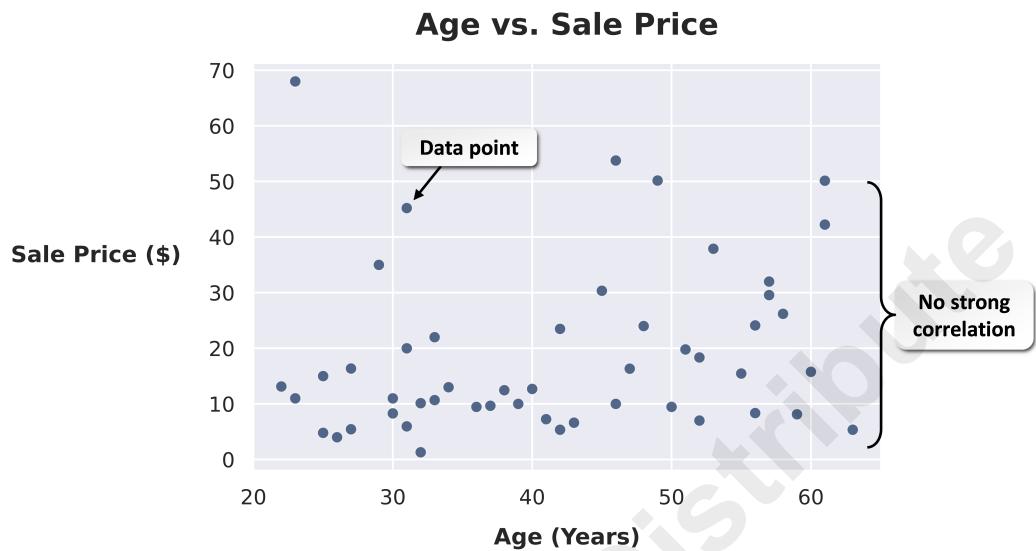


Figure 3–19: Using a scatter plot to compare age and sale price.

In this case, there doesn't appear to be much of a correlation between age and sale price. The dots don't steadily rise or fall as age increases. Older customers don't necessarily make more expensive purchases, and vice versa. Even if you can't identify a correlation, it's still useful to create scatter plots, as it's often just as important to know when two variables don't correlate as when they do. You can also use scatter plots as another way to spot outliers, this time in relation to some other variable. So it seems that the ~\$70 purchase was made by someone in their earlier 20s.

Line Plots

A **line plot** is similar to a scatter plot, except that the points on the graph are connected by a series of straight lines. Line plots are most commonly used to visualize some trend over time, where time is along the x-axis and another variable is along the y-axis. The lines are drawn from the first x-axis value to the next, then to the next, and so on. Therefore, it's common to sort the x-axis data first if it isn't already in order.

The following figure plots both the mean product and service satisfaction scores, each one a separate line, for each of the seven days in the survey.

Mean Satisfaction Scores Over One Week

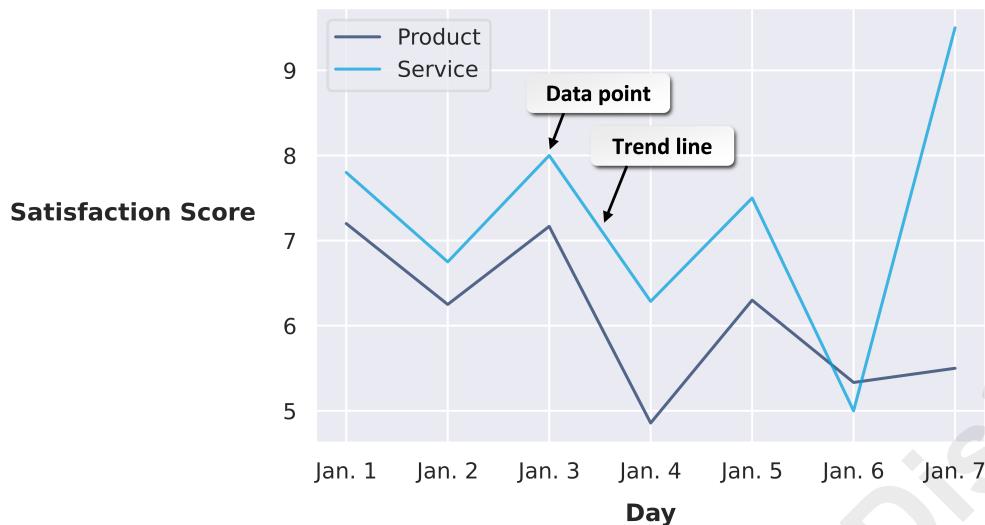


Figure 3–20: Satisfaction scores plotted as trend lines.

Within the week, both scores seemed to fluctuate, with the product scores actually trending downward somewhat. However, the service score seems to have jumped on the last day. Besides that jump, both scores appear to follow a similar trend. You can also see that, throughout most of the week, the service scores are higher than the product scores. Line graphs are useful for comparing trends of similar variables over the same time period, like in this example.

Area Plots

An **area plot** is a type of line plot in which the space below the line is filled in with some color or texture. Like a standard line plot, an area plot is typically used to represent change over time, but with an added emphasis on the general trend of the data, rather than the specific data points. They are also commonly used in a stacked format, in which the same measurements are compared across different contexts. For example, you could create an area plot in which sale price per day is shown, but with each area stack representing a different month.

In this figure, the mean of sale prices for the first week of the month is plotted. The dark blue stack on the bottom is for January, whereas the light blue stack on the top is for February. Note that stacked area plots can be misleading. In the figure, the mean sale price for February 1st is *not* ~\$43, despite the fact that the stack reaches that high on the y-axis. Only the lowest stack correctly aligns with the y-axis; the rest of the stacks are the same relative size, just placed on top of each other. You have to interpret the stacks in relation to one another, rather than on an absolute scale. For example, the peak for February 4th is slightly higher than the peak for January 4th.

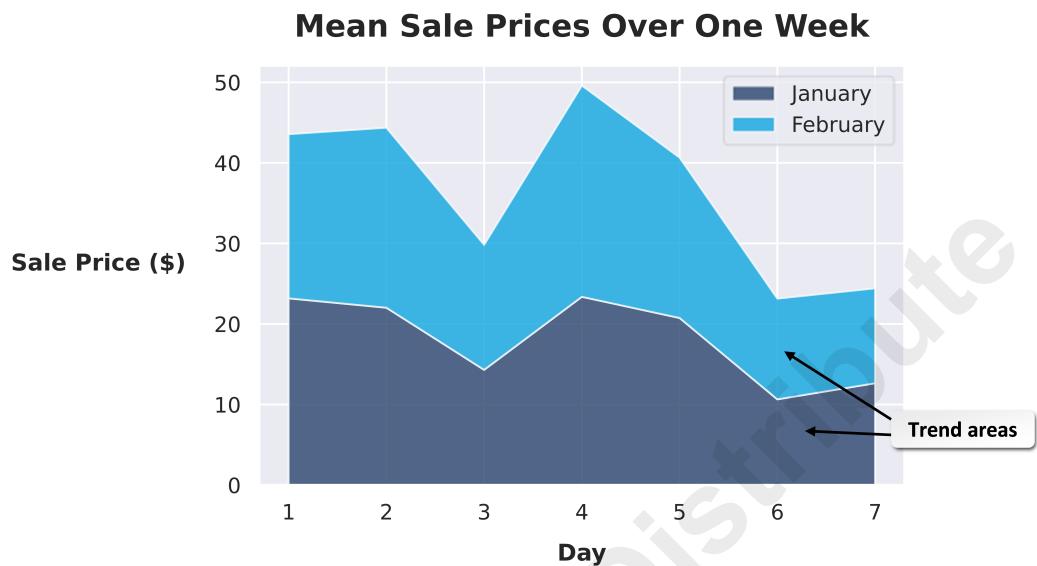


Figure 3-21: A stacked area plot of sale price over the first week in two months.

The area plot underscores the overall downward trend of individual sale prices over the week (for both months), though with obvious alternating peaks and valleys.

Guidelines for Analyzing Data Using Scatter Plots, Line Plots, and Area Plots

Follow these guidelines when analyzing data using scatter plots, line plots, and area plots.

Analyze Data Using Scatter Plots, Line Plots, and Area Plots

When analyzing data using scatter plots, line plots, area plots:

- Use labels for important data points only, as too many labels clutter plots.
- Consider using a color and pattern for each line on a line plot so plots are readable if rendered in black and white.
- Use scatter plots where neither axis is categorical data, one of the axes is a time or date, or both axes have irregular intervals.
- Use a line plot when one axis represents time or date.
- Consider an area plot instead of a line plot when multiple aspects of data need to be shown.
- Plot the largest area first (i.e., at the back) so that the smallest area is not obscured by other plots.

ACTIVITY 3–5

Analyzing Data Using Scatter Plots and Line Plots

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Generate scatter plots comparing total_amount_usd to number_transactions**. Select **Cell→Run All Above**.

Scenario

Other than using distribution plots, you can also get a sense of how data appears when two or more features are compared to each other. You want to see if the total amount of all of a user's transactions is related to the number of transactions they engage in. Perhaps the more transactions the user makes, the more of a total net positive they have in terms of total amount. Or, perhaps it's the opposite. There may even be no correlation whatsoever between the number of transactions and the total amount. So, you'll generate a scatter plot to find out.

You also want to see how account age is affecting the total amount, if at all. Are long-time users more likely to have a net positive in their account, or not? So, you'll generate a line plot comparing the years users signed up and the average total amount for each user in a sign-up year.

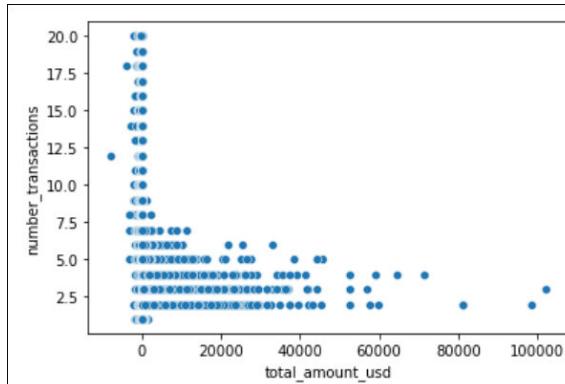
1. Generate scatter plots comparing total_amount_usd to number_transactions.

- Scroll down and view the cell titled **Generate scatter plots comparing total_amount_usd to number_transactions**, then select the code cell below it.
- In the code cell, type the following:

```
1 sns.scatterplot(data = users_data, x = 'total_amount_usd',
2                  y = 'number_transactions');
```

- Run the code cell.

- d) Examine the output.



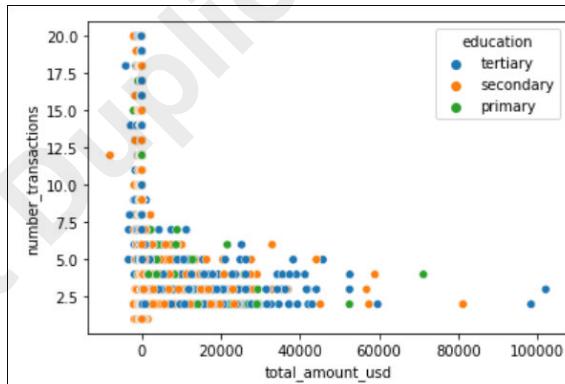
The scatter plot shows `total_amount_usd` on the x-axis, compared to `number_transactions` on the y-axis. By looking at the scatter plot, you can see that there is not necessarily a strong correlation between either feature. However, you can conclude that most users with a high number of transactions also have a relatively low total transaction amount. This could be due to alternating deposits and withdrawals. Also, the highest transaction amounts seem to be part of a low number of transactions.

- e) Select the next code cell, then type the following:

```
1 sns.scatterplot(data = users_data, x = 'total_amount_usd',
2                   y = 'number_transactions', hue = 'education');
```

This code adds an extra dimension to the scatter plot: education level.

- f) Run the code cell.
g) Examine the output.



You can now see each education label as a separate color dot. This helps you compare a third feature to the primary two. In this case, a user's education level doesn't necessarily exhibit a pattern when the number of transactions is compared to the total amount. Though, it does appear that the two users with the highest total amounts have a tertiary education.

2. Generate a line plot for `total_amount_usd`.

- a) Scroll down and view the cell titled **Generate a line plot for `total_amount_usd`**, then select the code cell below it.

- b) In the code cell, type the following:

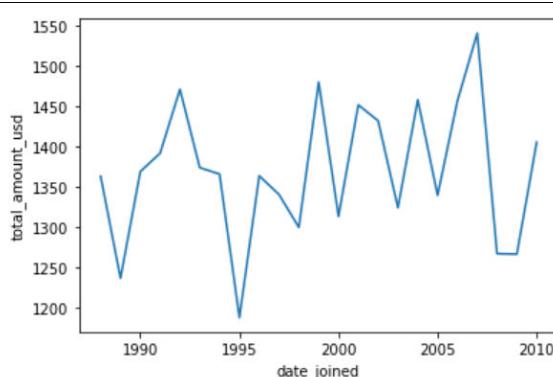
```

1 years = users_data['date_joined'].dt.year
2
3 sns.lineplot(data = users_data, x = years,
4                 y = 'total_amount_usd',
5                 estimator = np.mean);

```

This code takes a slice of `date_joined` so that it only considers the years. These year values are plugged into the plotting function.

- c) Run the code cell.
d) Examine the output.



This line plot shows the trend of the mean of total transaction amount for each year that users signed up for an account. For example, for all users who signed up in 1995, the mean of each user's total transaction amount is slightly less than \$1,200. The graph indicates that the total transaction amount seems to fluctuate quite a bit between each sign-up year. Some years, like 1995, show an obvious decline, whereas others, like 2007, show a dramatic increase.



Note: This graph is not showing the mean amount of all transactions in each year, but the mean amount for all users who signed up in each year.

3. Keep this notebook open.

Bar Charts

A **bar chart** represents the proportional measurement of categorical values by using either horizontal or vertical bars. In a vertical bar chart, the categorical values lie along the horizontal axis, and the measurement of each categorical value lies along the vertical axis. The opposite is true for a horizontal bar chart. The purpose of a bar chart is to compare the measurement of each discrete category value with other values in that category.

Bar charts look similar to histograms, but remember, histograms plot the probability distributions of continuous variables, divided up by some arbitrary number of bins. Bar charts deal in frequencies of categorical variables. In the fast food survey, respondents were categorized according to what branch of the restaurant they ordered from (i.e., the city). They were also categorized by gender. So, the following bar chart compares the mean product satisfaction scores of each city, as well as those for gender.

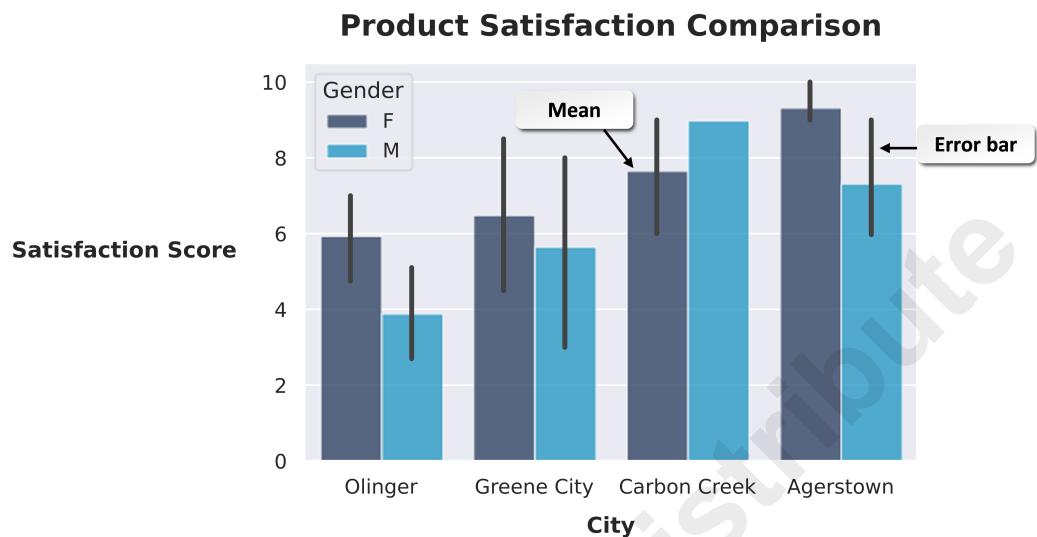


Figure 3–22: The generated bar chart.

Looking at this chart, it appears that female patrons of the Agerstown branch were more satisfied with the food than the patrons of any of the other branches. Overall, it looks like men expressed less satisfaction with their food than women did, the exception being the Carbon Creek branch. Male customers in Olinger were particularly unhappy with their food. Bar charts like this can tell you a great deal about the differences in demographics.

Guidelines for Analyzing Data Using Bar Charts

Follow these guidelines when analyzing data using bar charts.

Analyze Data Using Bar Charts

When analyzing data using bar charts:

- Use bar charts when at least one axis is categorical data or discrete data at regular intervals.
- Consider using bar charts when one of the axes has around twenty or fewer values, as too many categories can be difficult to analyze.
- Show multiple data sources on a bar chart if they share common categories.
- Use one color per dataset when displaying multiple datasets.
- Order bars from highest to lowest when there is no inherent ordering of the categories.

ACTIVITY 3–6

Analyzing Data Using Bar Charts

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Generate bar charts for job**. Select **Cell**→**Run All Above**.

Scenario

The GCNB dataset has many categorical features, which are great candidates for bar charts. You want see the frequency of values in each of these features. So, you'll create several bar charts to compare those frequencies.

1. Generate bar charts for job.

- Scroll down and view the cell titled **Generate bar charts for job**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_job_dist = \
2 users_data['job'].value_counts(dropna = False)
3
4 users_job_dist
```

Before you plot the bar chart, you'll get a look at the frequencies of each job value.

- Run the code cell.
- Examine the output.

blue-collar	9731
management	9457
technician	7597
admin.	5171
services	4154
retired	2264
self-employed	1579
entrepreneur	1487
unemployed	1303
housemaid	1240
student	938
NaN	288
Name: job, dtype: int64	

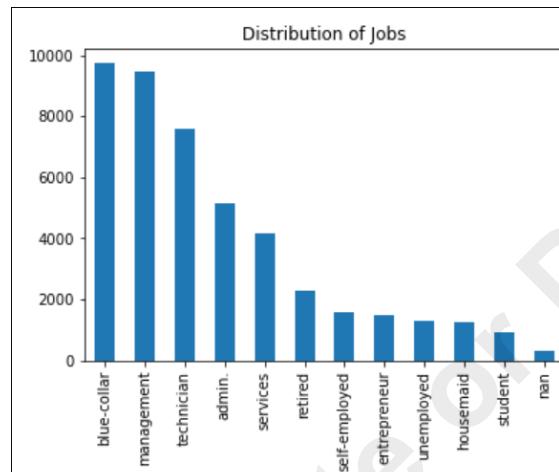
The frequencies of each value are in descending order. So, blue collar jobs and management jobs seem to be the most common, whereas students are the least common (besides missing values).

- e) Select the next code cell, then type the following:

```
1 # Vertical bar chart.  
2  
3 users_job_dist.plot(kind = 'bar')  
4 plt.title('Distribution of Jobs');
```

You'll plot a vertical bar chart first.

- f) Run the code cell.
g) Examine the output.



This visually confirms the numeric frequencies you just saw. You can also see that the frequencies tend to taper off gradually, rather than there being an abrupt change in frequencies.

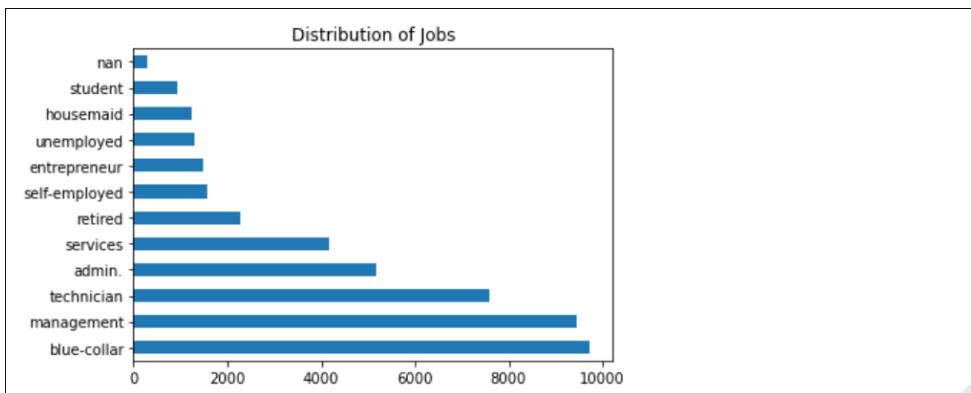
- h) Select the next code cell, then type the following:

```
1 # Horizontal bar chart  
2  
3 users_job_dist.plot(kind = 'barh')  
4 plt.title('Distribution of Jobs');
```

You'll generate the same chart, this time horizontally.

- i) Run the code cell.

- j) Examine the output.



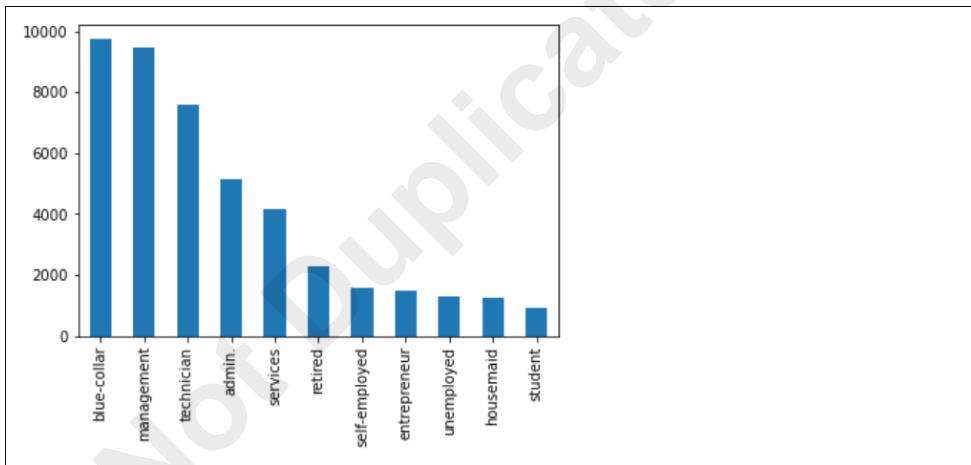
Sometimes, reorienting a chart can make it easier to read.

- k) Select the next code cell, then type the following:

```
1 # Exclude missing values.
2
3 users_data['job'].value_counts().plot(kind = 'bar');
```

This time you'll plot the same data while excluding missing values.

- l) Run the code cell.
m) Examine the output.



You'll deal with these later, but for now it can be helpful to just ignore bad data for analysis purposes.

2. Generate a bar chart for marital.

- a) Scroll down and view the cell titled **Generate a bar chart for marital**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 users_marital_dist = \
2 users_data['marital'].value_counts(dropna = False)
3
4 users_marital_dist
```

As before, you'll get the raw frequencies first.

- c) Run the code cell.
d) Examine the output.

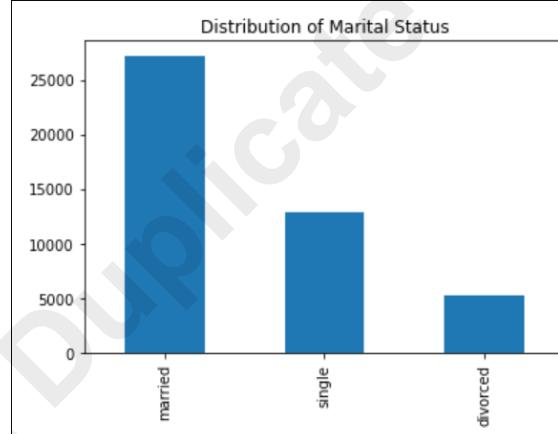
```
married    27212
single     12790
divorced    5207
Name: marital, dtype: int64
```

Married users far outweigh single and divorced users.

- e) Select the next code cell, then type the following:

```
1 users_marital_dist.plot(kind = 'bar')
2 plt.title('Distribution of Marital Status');
```

- f) Run the code cell.
g) Examine the output.



This helps to demonstrate just how differently each marital status is represented in the data.

3. Generate a bar chart for education.

- a) Scroll down and view the cell titled **Generate a bar chart for education**, then select the code cell below it.
b) In the code cell, type the following:

```
1 users_education_dist = \
2 users_data['education'].value_counts(dropna = False)
3
4 users_education_dist
```

- c) Run the code cell.

- d) Examine the output.

```
secondary    23202
tertiary     13300
primary      6850
NaN          1857
Name: education, dtype: int64
```

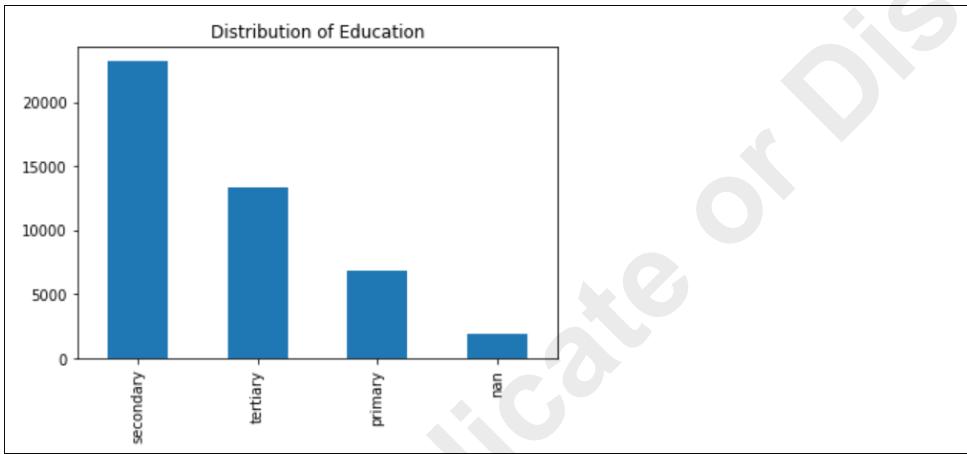
Secondary education seems to be the most common.

- e) Select the next code cell, then type the following:

```
1 users_education_dist.plot(kind = 'bar')
2 plt.title('Distribution of Education');
```

- f) Run the code cell.

- g) Examine the output.



Secondary education is indeed the most common, followed by tertiary, then primary.

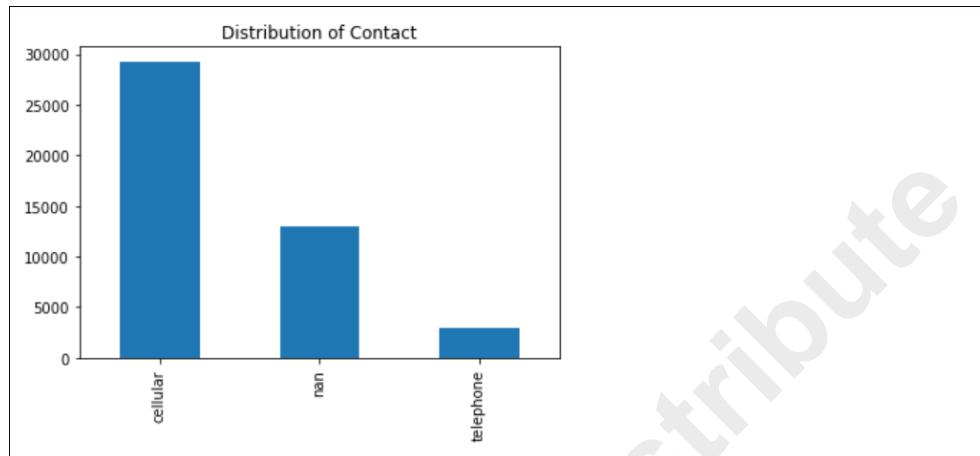
4. Generate a bar chart for contact.

- a) Scroll down and view the cell titled **Generate a bar chart for contact**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_contact_dist = \
2 users_data['contact'].value_counts(dropna = False)
3
4 users_contact_dist.plot(kind = 'bar')
5 plt.title('Distribution of Contact');
```

- c) Run the code cell.

- d) Examine the output.



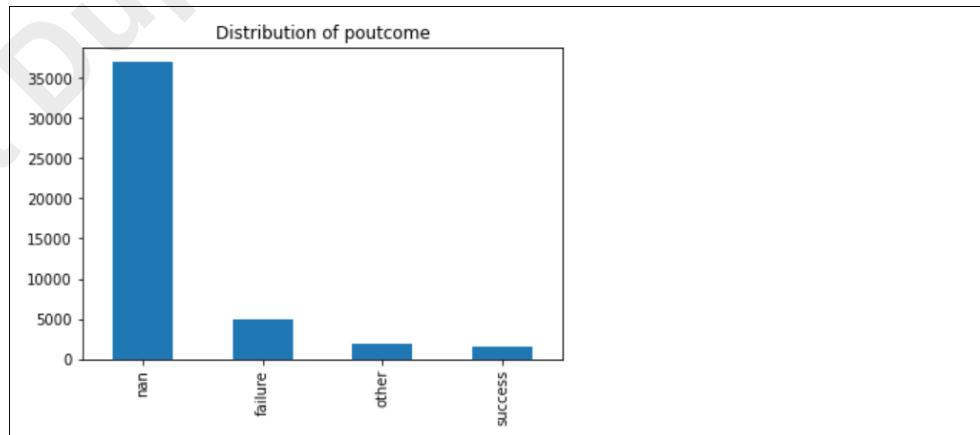
By far, using a mobile phone is the most common way for users to contact the bank. A landline telephone is very uncommon. There are also quite a few missing values for this feature.

5. Generate a bar chart for `poutcome`.

- Scroll down and view the cell titled **Generate a bar chart for `poutcome`**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_poutcome_dist = \
2 users_data['poutcome'].value_counts(dropna = False)
3
4 users_poutcome_dist.plot(kind = 'bar')
5 plt.title('Distribution of poutcome');
```

- Run the code cell.
- Examine the output.



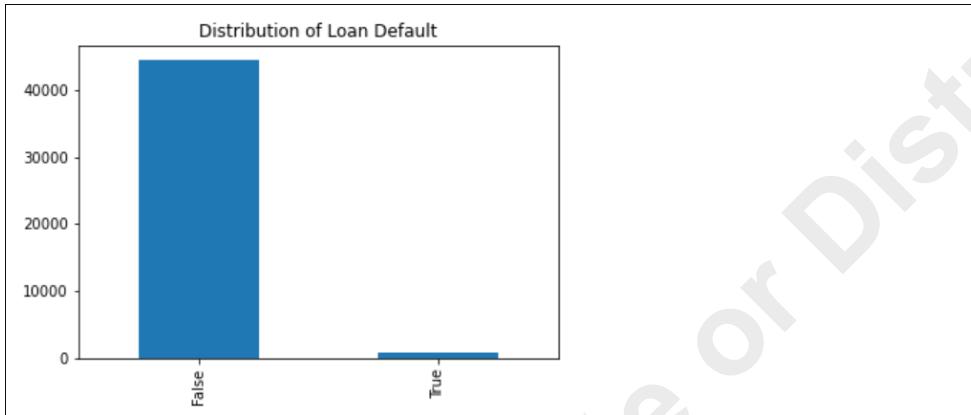
This particular feature (whether or not the previous marketing campaign was successful) has a lot of missing values. You'll handle these soon.

6. Generate a bar chart for `default`.

- Scroll down and view the cell titled **Generate a bar chart for default**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_device_dist = \
2 users_data['default'].value_counts(dropna = False)
3
4 users_device_dist.plot(kind = 'bar')
5 plt.title('Distribution of Loan Default');
```

- Run the code cell.
- Examine the output.



Many more people did not default on a loan than did. The chart also confirms the high level of positive skewness for this Boolean variable that you identified earlier, as well as the high level of leptokurtosis.

7. Generate a bar chart for device.

- Scroll down and view the cell titled **Generate a bar chart for device**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_device_dist = \
2 users_data['device'].value_counts(dropna = False)
```

- Run the code cell.

- d) Select the next code cell, then type the following:

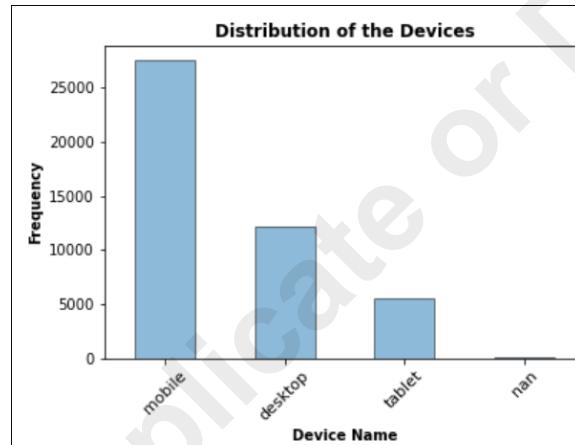
```

1 users_device_dist.plot(kind = 'bar',
2                         alpha = 0.5, edgecolor = 'black')
3 plt.title('Distribution of the Devices',
4            size = 12, weight = 'bold')
5 plt.xticks(rotation = 45, size = 11)
6 plt.xlabel('Device Name', size = 10, weight = 'bold')
7 plt.ylabel('Frequency', size = 10, weight = 'bold')
8 plt.show();

```

Charts can be much more informative and easier to read when you take the time to format them. Here you're:

- Plotting the chart.
 - Adding a title to the chart.
 - Changing the tick labels.
 - Changing the x-axis and y-axis labels.
- e) Run the code cell.
f) Examine the output.



When it comes to using the bank's online services, mobile devices are most popular, while desktops and tablets trail behind. Once again, there are some missing values.

8. Keep this notebook open.

Geographical Maps

Some types of data may describe a particular geographic location, and may therefore be appropriate to analyze using **geographical maps**. The dataset must include some data items that can be used to identify coordinates or locations within the map, such as latitude and longitude or ZIP codes. Although maps are typically thought of in regard to large areas such as a city or country, they may also be used to visualize data points covering a small area or assembly, such as a single building, the anatomy of a person or animal, a piece of equipment (such as the location of structural faults observed within a particular airplane), and so forth.

In the following figure, the price of homes is plotted on a map of Seattle, Washington. The larger the blue marker, the more expensive the home.

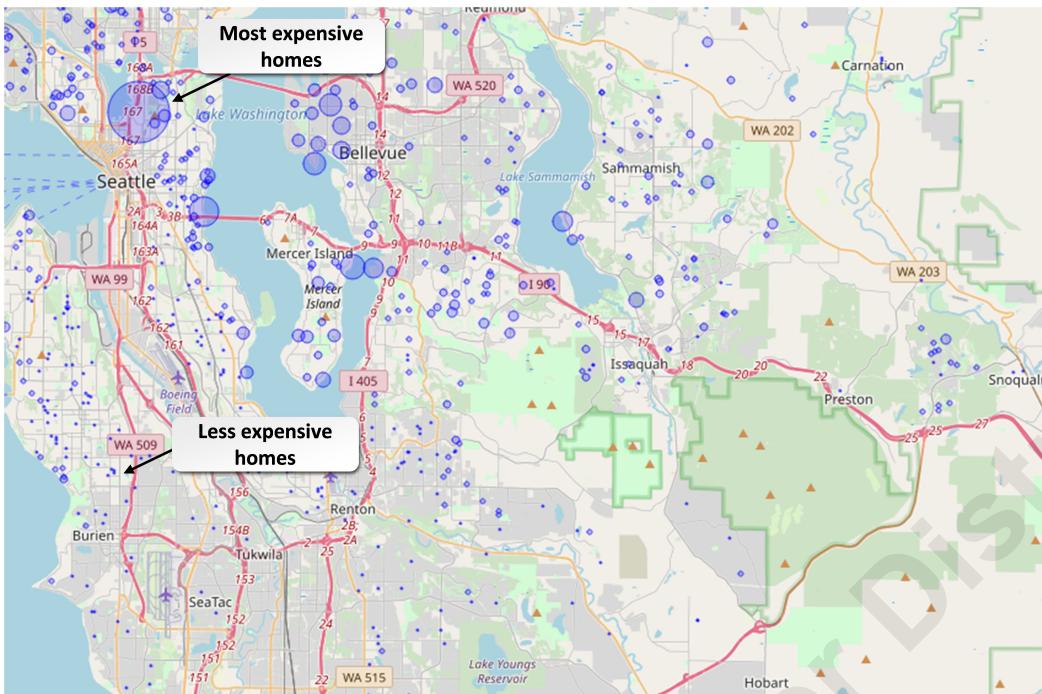


Figure 3–23: Geographical map using larger markers to show more expensive homes.

You could use this to draw some conclusions about each neighborhood and the relative value of its real estate. The area directly east of Lake Union (in the upper-left of the map) has the most expensive homes from this perspective, whereas the Burien suburb to the south has some of the least expensive.

Heatmaps

A **heatmap** plots regions of the chart with an intensity or shade of color based on data values in that location of the chart. This color coding makes it easier for a viewer to identify patterns of values by their color and location within the grid. For example, a heatmap might plot the locations of houses on a map. Parts of the map where the density of houses is high might be highlighted with a particular shade or color. Where the density of houses is low, a different shade or color might be shown. In various parts of the map, the blend of the two shades or colors will reflect how sparse or dense the houses are in that location.

While heatmaps can be used to present a grid that represents physical locations (such as houses located within a city), they can also be used to show conceptual divisions. For example, a heatmap can be used to enhance a correlation matrix. The correlation matrix is a grid that shows how the data items in every column correlate with the data items in every other column. Each cell in the grid shows the correlation coefficient between two columns. Pairs with the highest correlation might be shown in one shade, while pairs with the lowest correlation are shown in another shade. By scanning the intensity of the shades shown, you can quickly identify column pairs with the highest or lowest correlation.

In the following figure, the image on the left adds another dimension to the geographic map, in that the number of houses is indicated by changes to color. The image on the right shows a correlation matrix of features in a housing prices dataset, like number of bedrooms, square footage, number of floors, and, of course, its price. Both of these are considered heatmaps, despite being used in different ways.

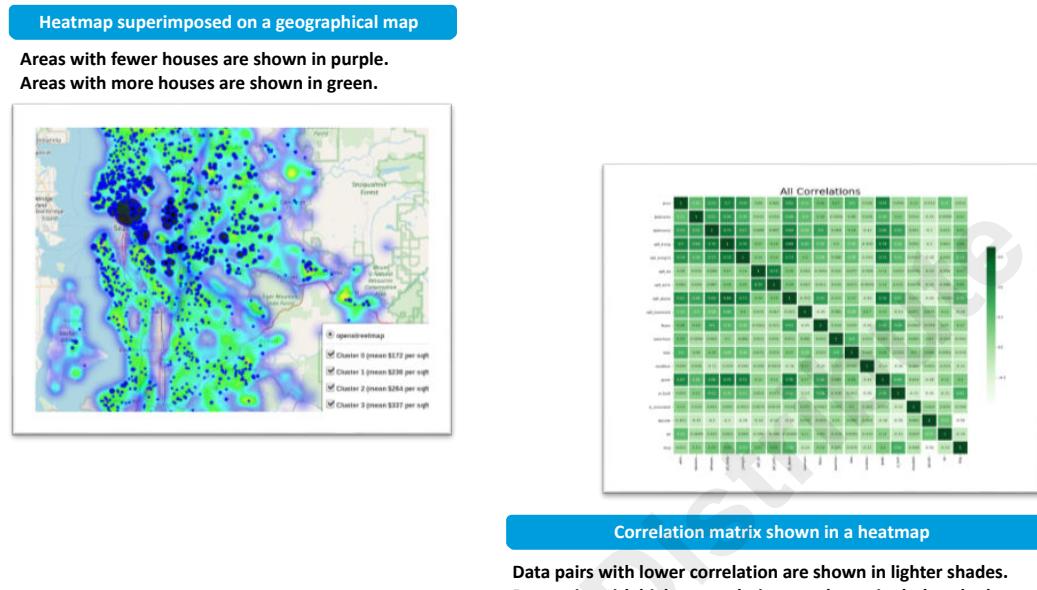


Figure 3-24: Heatmaps using color to represent data values.

Guidelines for Analyzing Data Using Maps

Follow these guidelines when analyzing data using maps.

Analyze Data Using Maps

When analyzing data using maps:

- Consider overlaying data on a topographical map when the data corresponds to physical features such as water levels, vegetation, etc.
- Avoid using too many colors or too many data overlays in your map.
- Add features to pan and zoom or add/remove data layers if presenting the map in an interactive format.
- Use gradients that make sense for the data being displayed, especially if related to a natural feature or physical phenomena, like blue hues for water, green for vegetation, white and blue for precipitation, etc.
- Add major cities and geographical landmarks to help users orient themselves.
- Use maps that relate to the data being presented, not as a background.

ACTIVITY 3-7

Analyzing Data Using Maps

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Generate a heatmap for the feature correlations**. Select **Cell**→**Run All Above**.

Scenario

You've taken a preliminary look at the feature correlations in the dataset, but you want to dive a little deeper. A correlation heatmap is a good way to visually compare these feature correlations, so you'll create one. The correlations or lack thereof may dictate how you drop or keep certain features.

1. Generate a heatmap for the feature correlations.

- Scroll down and view the cell titled **Generate a heatmap for the feature correlations**, then select the code cell below it.
- In the code cell, type the following:

```
1 corr_matrix = users_data.corr()
2
3 corr_matrix
```

- Run the code cell.
- Examine the output.

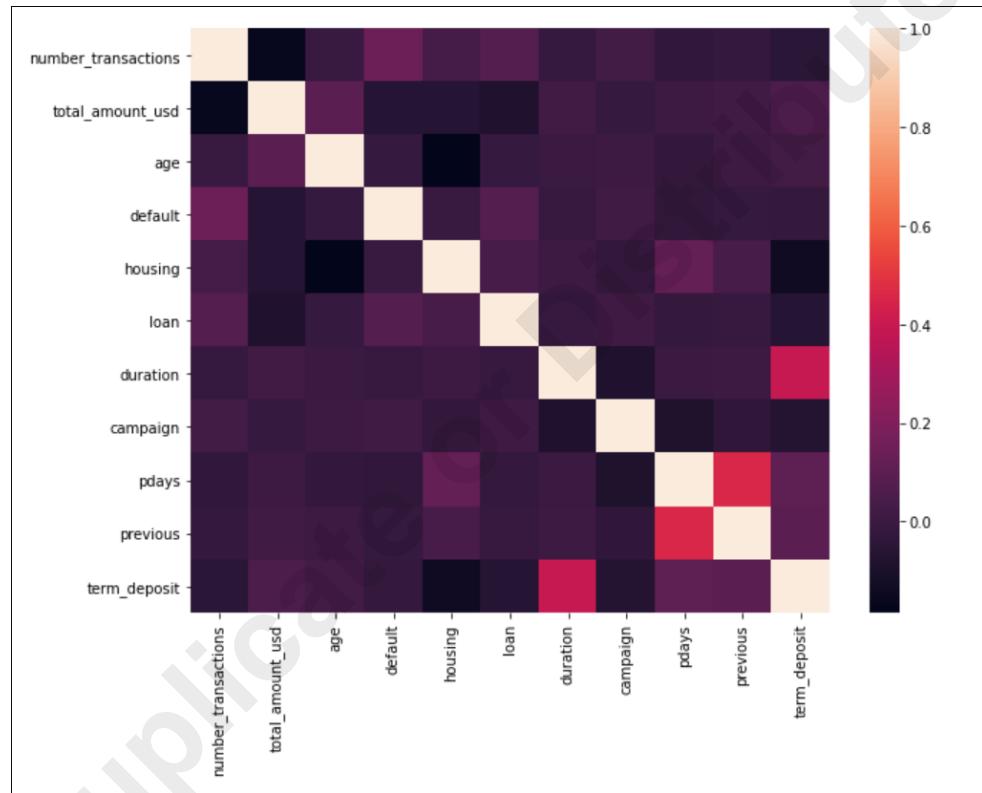
	number_transactions	total_amount_usd	age	default	housing	loan	duration	campaign	pdays	previous	term_dep
number_transactions	1.000000	-0.163409	-0.008813	0.138838	0.030429	0.075319	-0.017220	0.026431	-0.030751	-0.023046	-0.053
total_amount_usd	-0.163409	1.000000	0.095839	-0.065390	-0.066857	-0.084526	0.022586	-0.017274	0.006435	0.016952	0.050
age	-0.008813	0.095839	1.000000	-0.017875	-0.185552	-0.015641	-0.004645	0.004767	-0.023745	0.001297	0.025
default	0.138838	-0.065390	-0.017875	1.000000	-0.006020	0.077232	-0.010017	0.016819	-0.029982	-0.018331	-0.022
housing	0.030429	-0.066857	-0.185552	-0.006020	1.000000	0.041341	0.005041	-0.023583	0.124197	0.037087	-0.139
loan	0.075319	-0.084526	-0.015641	0.077232	0.041341	1.000000	-0.012395	0.009972	-0.022762	-0.011048	-0.068
duration	-0.017220	0.022586	-0.004645	-0.010017	0.005041	-0.012395	1.000000	-0.084551	-0.001549	0.001213	0.394
campaign	0.026431	-0.017274	0.004767	0.016819	-0.023583	0.009972	-0.084551	1.000000	-0.088636	-0.032860	-0.073
pdays	-0.030751	0.006435	-0.023745	-0.029982	0.124197	-0.022762	-0.001549	-0.088636	1.000000	0.454817	0.103
previous	-0.023046	0.016952	0.001297	-0.018331	0.037087	-0.011048	0.001213	-0.032860	0.454817	1.000000	0.093
term_deposit	-0.053390	0.050785	0.025168	-0.022421	-0.139161	-0.068193	0.394549	-0.073179	0.103616	0.093232	1.000

The correlation coefficients are presented as numbers in a table. This can be somewhat difficult to read, so you'll create a heatmap out of this data instead.

- e) Select the next code cell, then type the following:

```
1 fig = plt.figure(figsize = (10, 7.5))
2
3 sns.heatmap(corr_matrix);
```

- f) Run the code cell.
g) Examine the output.



The heatmap shows correlation strength as a gradation of color. Lighter values show a stronger positive correlation. However, negative values are not represented on an equivalent color gradient, which makes it harder to distinguish weak negative correlations from strong negative correlations. You'll therefore create a more useful heatmap in the next step.

2. Format the heatmap to make it easier to read.

- a) Scroll down and view the cell titled **Format the heatmap to make it easier to read**, then select the code cell below it.

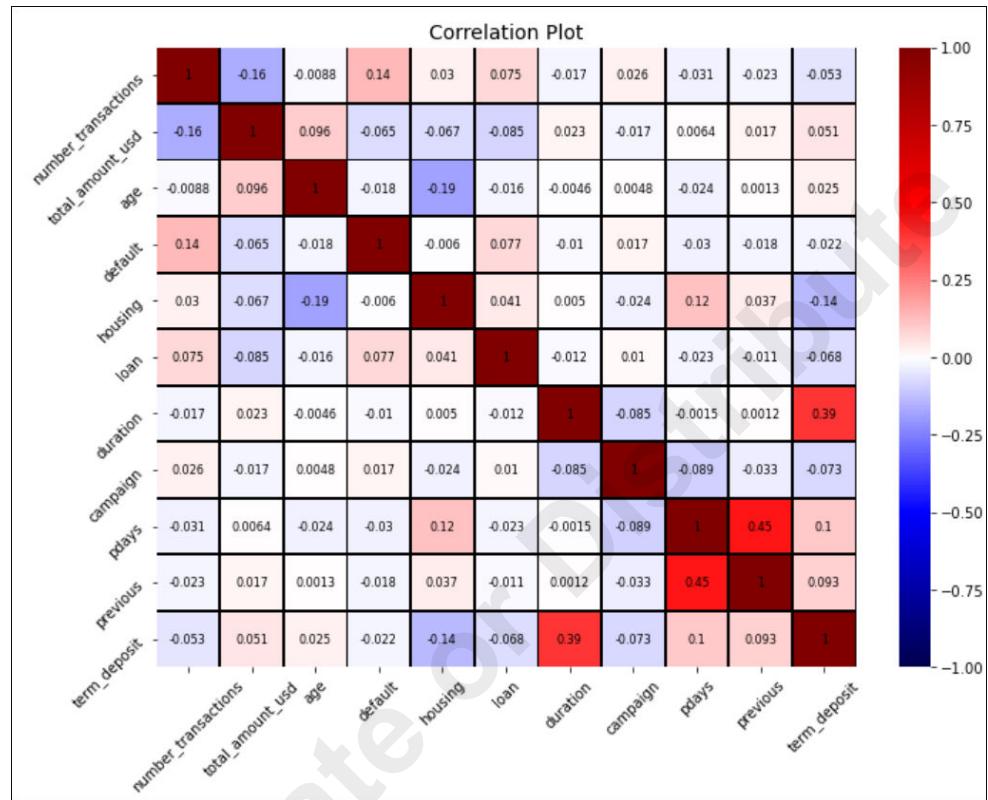
- b) In the code cell, type the following:

```
1 fig = plt.figure(figsize = (11, 8))
2
3 sns.heatmap(corr_matrix,
4             cmap = 'seismic',
5             linewidth = 0.75,
6             linecolor = 'black',
7             cbar = True,
8             vmin = -1,
9             vmax = 1,
10            annot = True,
11            annot_kws = {'size': 8, 'color': 'black'})
12
13 plt.tick_params(labelsize = 10, rotation = 45)
14 plt.title('Correlation Plot', size = 14);
```

This code will show the "heat" as shades of two different diverging colors. Also, this will print the actual correlation values for each cell in the matrix.

- c) Run the code cell.

- d) Examine the output.



The same data is plotted, but in a more informative way. Darker red cells indicate stronger positive correlations, whereas darker blue cells indicate stronger negative correlations. Lightly colored cells exhibit weak correlation in either direction.

Aside from the main diagonal (variables being compared to themselves), the features in this dataset don't show a lot of correlation. However, the strongest positive correlations seem to be at the bottom right of the map, particularly between `duration` and `term_deposit`, and `pdays` and `previous`. So, the length of the last contact between the user and bank seems to have somewhat of a positive correlation with whether or not the user signed up for a term deposit. Likewise, as more days pass since the last time the user was contacted, the more times the user was likely to have been contacted in previous campaigns. These scenarios make sense intuitively, but an increase in one feature doesn't necessarily cause the other to increase.

There's not a lot of strong negative correlations. The strongest (in a relative sense) are between `total_amount_usd` and `number_transactions`, `housing` and `age`, and `housing` and `term_deposit`. So, for example, this suggests that the older a user is, the slightly less likely it is they'll have a housing loan. Again, you can't definitively say that one feature increasing causes another to decrease.

3. Keep this notebook open.

Plots in Combination (Bar Chart Grid)

Creating and analyzing individual plots can certainly be beneficial, but sometimes you may be missing out on important context. Viewing plots in combination can help you reveal even more pertinent information about your datasets. You might set up a plotting space with multiple subplots, each of them related to an overall idea but showing slight variations.

In the following figure, a grid of bar charts compares the mean sale price across three different categorical conditions: city, gender, and whether or not the respondent would recommend the restaurant to friends and family.

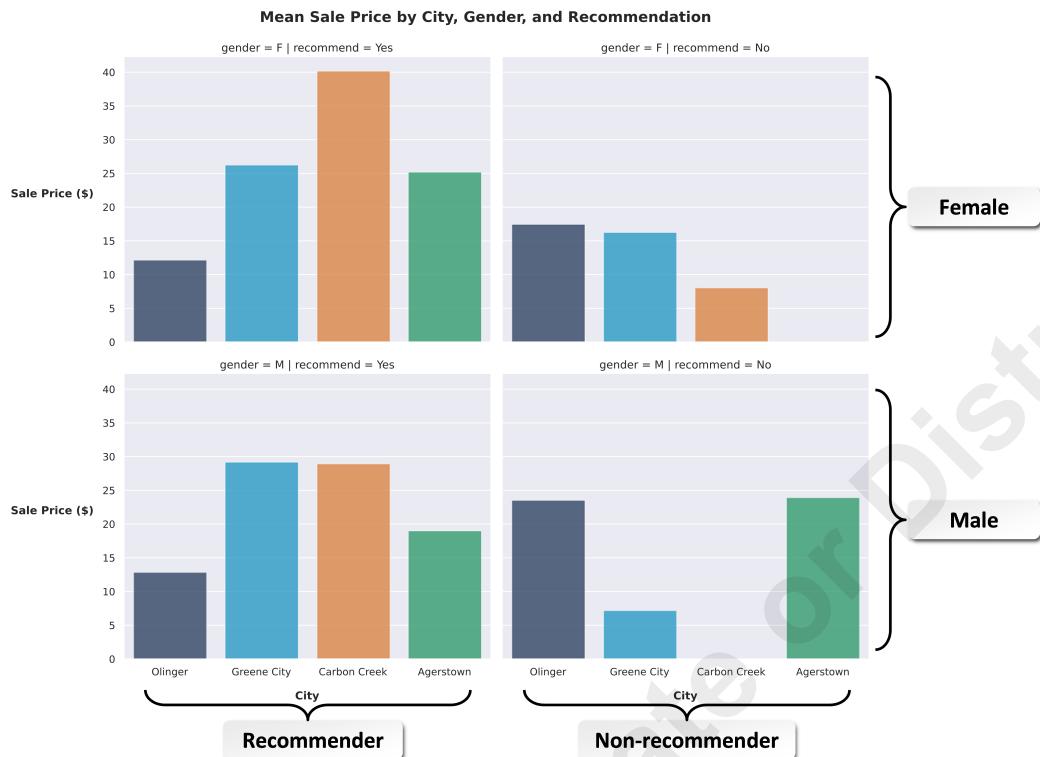


Figure 3-25: A grid of bar charts that compares sale prices for several conditions.

By placing related plots together in a large grid, you can better compare variations across multiple dimensions, without having to analyze these plots in isolation. Each row in the overall grid represents gender, whereas each column in the overall grid represents the recommendation status. For example, the bottom-left plot shows male customers who would recommend the restaurant to others. Within each plot, each bar represents a different city. One conclusion you could draw from this example is that, while female customers who wouldn't recommend the restaurant made smaller purchases than if they would recommend it, male customers were less predictable. In fact, the two cities where recommending male customers made the smallest purchases (Olinger and Agerstown) were the same two cities where non-recommending customers made the largest purchases. This might suggest different recommendation and spending patterns based on gender, at least for this dataset.

Plots in Combination (Pair Plot)

There are many other examples of using plots in combination. For instance, pair plots create a grid where multiple numerical features are plotted as both rows and columns and then each is compared using a scatter plot or something similar. The main diagonal of the pair plot shows the distribution of each feature.

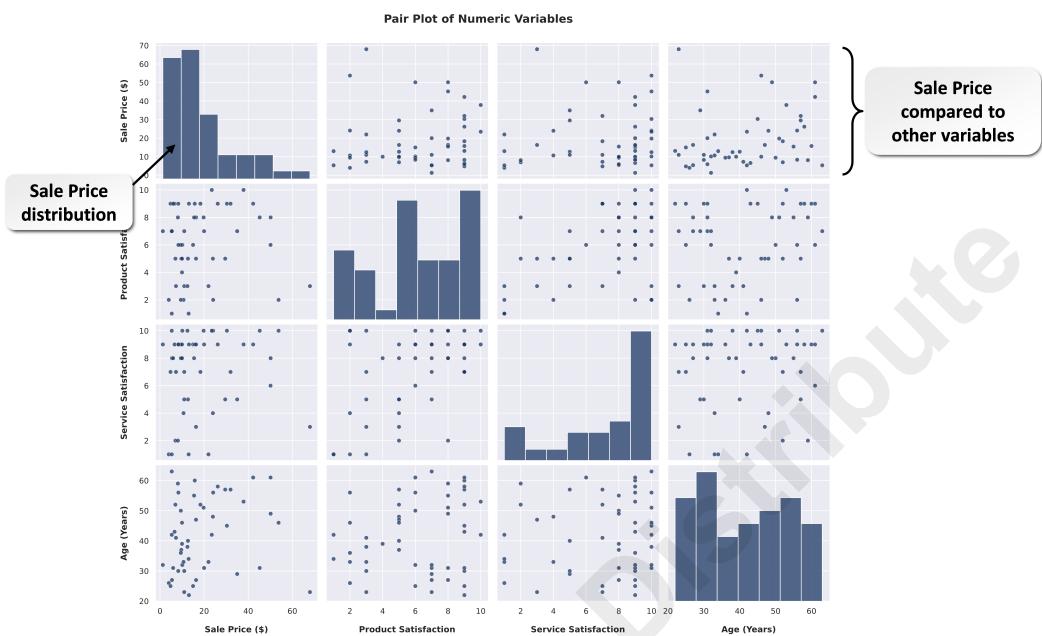


Figure 3–26: A pair plot comparing numeric variables.

In this example, you can see that sale price doesn't strongly correlate with product and service satisfaction or age. At the same time, you can compare this correlation (or lack thereof) to the distribution of sale prices. You can do the same thing for each row of variables, as well as for each column.



Note: This plot was created using Seaborn's `pairplot()` function.

When it comes to plots in combination, you're really only limited by your dataset and your imagination. You should experiment as much as you can so that you're more likely to find patterns that you wouldn't have otherwise.

Guidelines for Using Visualizations to Analyze Data

Follow these guidelines when using visualizations to analyze data.

Use Visualizations to Analyze Data

When using visualizations to analyze data:

- Be mindful of the story you are trying to tell with your data. Visualizations should augment the story.
- Include axis labels, titles, and data legends.
- Scale axes appropriately with major and minor tick marks, and be sure to label tick marks and format tick mark labels.
- Avoid adding too much data that ends up being distracting, making it difficult to "see the forest for the trees."
- Generate enough visualizations that are necessary for revealing important patterns and ideas, but don't overwhelm your workload with too many visuals.

ACTIVITY 3–8

Comparing Visual Analysis Methods

Scenario

You generated many different visualizations through code as a way to facilitate data analysis. Now that you have a feel for these graphs, plots, and charts, you'll compare them to see which ones may have certain strengths or weaknesses within certain contexts. That way you can focus on the ones that are the most pertinent to the job at hand.

1. When it comes to visualizing variable distributions, how would you compare histograms to box plots and violin plots?
2. When it comes to visualizing the intersection between two or more variables, how would you compare scatter plots to line and area plots?
3. How would you compare bar charts to histograms?
4. How do heatmaps compare to other types of visualizations?

TOPIC D

Preprocess Data

Your analysis efforts will most likely prompt you to transform your data further, especially in preparation for machine learning. In this topic, you'll do just that.

Data Analysis Prompts Further Transformation

As explained earlier, the extract, transform, and load (ETL) process is not linear. Instead, it is an iterative approach to improving data. The challenge data science practitioners face is determining how best to transform data. Early in the project's lifecycle, when you first apply ETL, some of these transformations will be obvious. However, your first pass at transforming data is unlikely to be your only one. You need to really dig into the data, analyzing it using the methods described previously, in order to obtain a more comprehensive understanding of what needs to change, and how.

For example, some of your variables might have skewed distributions, which you discover by creating and analyzing histograms. Some machine learning tasks work better when the data fits a normal distribution, so you then take steps to normalize your data. Or, perhaps you identify a lack of correlation between a feature variable and the target variable you're trying to estimate, prompting you to drop that feature. Another example is a box plot revealing outliers that could negatively affect a model's ability to learn, so you identify and then remove those outliers.

The point is, exploratory and visual data analysis can provide you with a lot more ideas about how to change your data. As a data science practitioner, you will most often find yourself making changes to the data, analyzing those changes, making more changes based on that analysis, and so on, until you're satisfied with the results.

Data Preprocessing

Data preprocessing is the task of applying various transformation and encoding techniques to data so that it can be interpreted and analyzed by a machine learning algorithm. The "pre" part of preprocessing implies that the data is being prepared, similar to how ETL prepares data at the early stages. The difference is that, in preprocessing, much of the preliminary transformation is already done, and you're really just focusing on facilitating the machine learning process. Preprocessing is important because machine learning algorithms all have different sets of challenges and requirements, so you need to make sure your data can accommodate them. If you think of your data science project as building toward the creation of a machine learning model, then preprocessing is the final stepping stone.

There are many techniques that fall under the definition of preprocessing, including:

- Identifying and handling missing and null values.
- Scaling variables using transformation functions.
- Removing unnecessary or unhelpful features.
- Engineering new features.



Note: Some practitioners choose to handle null values early in the ETL process.

Identification of Missing Values

In a relational dataset, missing data, or null data, typically refers to any data value that does not exist for a particular intersection between a row and a column. For example:

Student ID	Last Name	First Name	Test Score
0001	Ferard	Catherine	98.5
0002	Murphy	Jeremy	
0003	Haverson	Maria	75.5

Jeremy Murphy's test score is missing from this dataset. Depending on the library you're using, any attempt at measuring the `Test Score` feature (e.g., taking the mean) may fail because of that missing value.

Note that each programming language and library handles missing values differently. A language like Python can assign one of several null types to missing values, the most basic type being `None`. If the table were a CSV file and you pulled it into a pandas `DataFrame`, it would appear as `NaN`, or not a number. This value is a float data type, so the rest of the column will be cast as a float if it isn't already. Another type of null value is `NA`, which can be cast to any data type, enabling greater flexibility for how such values are handled. As of pandas version 1.2.2, `NA` is still in an experimental state.

Many data science programming libraries provide functions for identifying missing values in a dataset. They simply look for the presence of something like `NaN` and then return where this value appears. However, this assumes that the missing value is actually being cast as a null value like `NaN`. If there were a question mark in Jeremy Murphy's test score cell instead of it being completely empty, then the column would be cast a string and the programming function that identifies missing values wouldn't detect any. So you need to pay attention to how your data sources are handling missing values in case they do so in unusual or non-standard ways.



Note: Missing rows and columns cannot be identified so easily, and require more analysis and investigation on your part.

Imputation of Missing Values

Some libraries implement algorithms that can handle missing values on their own, but it is best if you decide exactly how to deal with missing data. If you simply ignore it, the algorithm may cope with the missing data, but the resulting model may not perform as well. When preparing data, you can drop records with missing values, but that may be a bad choice too, depending on how many records you have to delete and what other data they contain. Too many dropped records can limit the model's effectiveness, especially if the missing data only constitutes a small percentage of the overall feature space.

In some cases, the best approach may be to impute missing values. **Imputation** means providing your best estimate to fill in the missing values. There are numerous strategies for doing this. Some examples are described here.

Imputation Method	Description
Mean/mode imputation	Calculate the mean or mode of all items that are not missing in that column, then use the result to fill in missing values. For example, to fill in the missing student grade, you would just take the mean of all student grades minus any missing values. This approach is simple, as it preserves the value of the mean/mode and sample size. However, it may not be as good as other methods listed here.
Substitution	Use data from a new record that is not in the sample. For example, you might find another source of student grades that you can substitute the missing one with.

Imputation Method	Description
Hot deck imputation	Find records in the sample that have similar values on all other data items than the one that is missing, and copy the missing value from one of the similar records. If there is more than one similar record, randomly select the one you copy from. For example, if a student's grade was missing for only one subject (e.g., History class), you could find another student with similar grades in the remaining subjects, then reuse that student's History score as the History score for the student with the missing value.
Cold deck imputation	Similar to hot deck imputation, but instead of pulling from the same sample that the missing value is in, you pull from an external sample. For example, a different sample of students may have been recorded in the recent past, so you'll try to find a similar student in that sample.
Regression imputation	Use a prediction model to identify what the missing value should be, based on data in the record. Leverages patterns established among other records that are not missing the value to determine what the value should be. This is essentially a machine learning task that treats the record with the missing value as new data to predict. Once the prediction is made, the record can be returned to the dataset to use on some other machine learning task.

	<p>Note: The general concept of imputation is simple, but doing it well in practice can be challenging. Some of these methods may be prone to some bias, and in some cases may produce estimated values that produce worse results than you might obtain by simply deleting the record containing the missing value. When it is essential to provide the very best estimates, you may opt to use a multiple imputation approach, combining multiple methods to find the "best guess" and reduce bias.</p>
--	--

Guidelines for Handling Missing Values

Follow these guidelines when handling missing values.

Handle Missing Values

When handling missing values:

- Recognize if the model/algorithm being used is tolerant of some missing data, as many algorithms do not behave well with missing data.
- Replace missing data in continuous values with an average, zero, min, or max depending on the characteristics of a given data series.
- Replace missing category data with the most frequently occurring category.
- Create a new "missing" category if many records are missing a category.
- Consider removing a record if there are a number of variables missing.
- Drop columns missing values for at least 70% of the records.
- When working with an ordered series:
 - Fill in missing records at the start of the series by carrying back the first data available.
 - Fill in missing records at the end of the series by carrying the last data point forward.

ACTIVITY 3–9

Handling Missing Values

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Identify missing values**. Select **Cell**→**Run All Above**.

Scenario

As you've probably seen by now, the GCNB dataset has a lot of missing data. You need to handle that missing data before it can be used to build a machine learning model. Rather than remove all of the missing data wholesale, you'll take a more surgical approach and apply different methods to different instances.

1. Identify missing values.

- a) Scroll down and view the cell titled **Identify missing values**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_data.isnull().sum()
```

This will print the total number of missing values for each column.

- c) Run the code cell.
- d) Examine the output.

user_id	0
number_transactions	9999
total_amount_usd	9999
age	0
job	288
marital	0
education	1857
default	0
housing	0
loan	0
contact	13018
duration	0
campaign	0
pdays	0
previous	0
poutcome	36957
term_deposit	0
date_joined	30
device	94
dtype:	int64

It seems that 8 columns contain missing values, some at higher amounts than others.

2. Identify the percentage of missing values for each feature.

- a) Scroll down and view the cell titled **Identify the percentage of missing values for each feature**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 percent_missing = users_data.isnull().mean()  
2  
3 percent_missing
```

Another way to look at missing values is to see what percentage of the data in that column is missing.

- c) Run the code cell.
- d) Examine the output.

```
user_id          0.000000  
number_transactions 0.221173  
total_amount_usd 0.221173  
age              0.000000  
job              0.006370  
marital           0.000000  
education         0.041076  
default            0.000000  
housing            0.000000  
loan              0.000000  
contact            0.287952  
duration           0.000000  
campaign           0.000000  
pdays              0.000000  
previous            0.000000  
poutcome            0.817470  
term_deposit        0.000000  
date_joined         0.000664  
device              0.002079  
dtype: float64
```

The `poutcome` feature has the highest number of missing values at around 82%, whereas `device` has very few.

3. Generate a missing value report.

- a) Scroll down and view the cell titled **Generate a missing value report**, then select the code cell below it.

- b) In the code cell, verify the following:

```

1 def missing_value_pct_df(data):
2     """Create a DataFrame to summarize missing values."""
3
4     percent_missing = data.isnull().mean()
5     missing_value_df = \
6         pd.DataFrame(percent_missing).reset_index()
7
8     missing_value_df = \
9         missing_value_df.rename(columns = {'index': 'column_name',
10                               0: 'percent_missing'})
11
12     # Multiply by 100 and round to 4 decimal places.
13     missing_value_df['percent_missing'] = \
14         missing_value_df['percent_missing']. \
15         apply(lambda x: round(x * 100, 2))
16
17     missing_value_df = \
18         missing_value_df.sort_values(by = ['percent_missing'],
19                                       ascending = False)
20
21     return missing_value_df

```

This code is provided for you. It's a function that creates a DataFrame to summarize missing values in a readable and repeatable way.

- c) Run the code cell.
d) Select the next code cell, then type the following:

```

1 missing_value_df = missing_value_pct_df(users_data)
2
3 missing_value_df

```

This code calls the function that was just defined on the users dataset.

- e) Run the code cell.

- f) Examine the output.

	column_name	percent_missing
15	poutcome	81.75
10	contact	28.80
2	total_amount_usd	22.12
1	number_transactions	22.12
6	education	4.11
4	job	0.64
18	device	0.21
17	date_joined	0.07
12	campaign	0.00
16	term_deposit	0.00
14	previous	0.00
13	pdays	0.00
0	user_id	0.00
11	duration	0.00
8	housing	0.00
7	default	0.00
5	marital	0.00
3	age	0.00
9	loan	0.00

The output is a DataFrame in which each feature is a row that also has a corresponding percentage of missing values, sorted in descending order by missing percentage. You can see that contact, total_amount_usd, and number_transactions also have a high percentage of missing values.

4. Remove features with a high percentage of missing values.

- a) Scroll down and view the cell titled **Remove features with a high percentage of missing values**, then select the code cell below it.

- b) In the code cell, type the following:

```

1 # Threshold above which to drop feature.
2
3 threshold = 80
4
5 cols_to_drop = \
6 list(missing_value_df[missing_value_df['percent_missing'] \
7 > threshold]['column_name'])
8
9 print('Number of features to drop:', \
10      missing_value_df[ \
11      missing_value_df['percent_missing'] > threshold].shape[0])
12
13 print(f'Features with missing values greater than {threshold} %: ', \
14      cols_to_drop)

```

It's not always feasible to fill in missing values when so many occur for a single feature. Rather than keep that feature in the dataset, it should be dropped so that it doesn't negatively influence a model. In this case, the threshold is 80%, so any feature with a percentage of missing values greater than that will be dropped. This is a somewhat arbitrary figure and something that you can adjust for your own purposes, but it's common to drop features that are missing around 70%–80% of values.

- c) Run the code cell.
d) Examine the output.

```

Number of features to drop: 1
Features with missing values greater than 80%: ['poutcome']

```

As expected, the `poutcome` feature has too many missing values, and should be dropped from the dataset.

- e) Select the next code cell, then type the following:

```

1 users_data_cleaned = users_data.drop(cols_to_drop, axis = 1)

```

This code actually drops the feature.

- f) Run the code cell.
g) Select the next code cell, then type the following:

```

1 # Confirm feature was dropped.
2
3 missing_value_df = missing_value_pct_df(users_data_cleaned)
4
5 missing_columns = \
6 list(missing_value_df[missing_value_df['percent_missing'] \
7 > 0]['column_name'])
8
9 print('Number of features with missing values:', \
10      len(missing_columns))

```

- h) Run the code cell.
i) Examine the output.

```

Number of features with missing values: 7

```

As expected, only the one column was dropped.

5. Identify numerical data with missing values.

- Scroll down and view the cell titled **Identify numerical data with missing values**, then select the code cell below it.
- In the code cell, type the following:

```

1 dtypes = ['int64', 'float64']
2
3 numerical_columns = \
4 list(users_data_cleaned.select_dtypes(dtypes).columns)
5
6 print('Numerical features with missing values:', 
7     list(set(numerical_columns).intersection(missing_columns)))

```

There are still features with missing values that you won't drop completely, so you'll start investigating those.

- Run the code cell.
- Examine the output.

```
Numerical features with missing values: ['number_transactions', 'total_amount_usd']
```

Both `total_amount_usd` and `number_transactions` have missing values.

6. Impute missing values for `total_amount_usd`.

- Scroll down and view the cell titled **Impute missing values for `total_amount_usd`**, then select the code cell below it.
- In the code cell, type the following:

```

1 # Find a sample user with missing value
2
3 sample_user = \
4 users_data_cleaned[users_data_cleaned['total_amount_usd']. \
5     isnull()].sample(1).user_id
6
7 sample_user

```

This will just retrieve a single sample of a user with a missing `total_amount_usd` value.

- Run the code cell.
- Examine the output.

```
12481    d7a5acdd-5943-4ca6-9f31-d23ba1a1317c
Name: user_id, dtype: object
```



Note: Since `sample()` takes a random sample from the dataset, your sample will likely be different than what is shown in the screenshot.

- e) Select the next code cell, then type the following:

```

1 # Print mean of total_amount_usd.
2
3 print('Mean total_amount_usd: ', 
4       round(users_data_cleaned['total_amount_usd'].mean(), 2))
5
6 # Impute missing values for total_amount_usd with mean.
7
8 users_data_cleaned['total_amount_usd']. \
9    fillna(round(users_data_cleaned['total_amount_usd'].mean(), 2),
10        inplace = True)

```

There are many ways to impute missing numeric values, but mean is a common approach. So, in this code, the missing values for `total_amount_usd` will be filled in with the mean of all non-missing values.

- f) Run the code cell.
g) Examine the output.

```
Mean total_amount_usd: 1369.42
```

The mean used to fill in the missing values is \$1,369.42.

- h) Select the next code cell, then type the following:

```

1 users_data_cleaned[users_data_cleaned. \
2                     user_id.isin(sample_user)]['total_amount_usd']

```

- i) Run the code cell.
j) Examine the output.

```
12481    1369.42
Name: total_amount_usd, dtype: float64
```

The sample user from before now has \$1,369.42 as their `total_amount_usd` value, as expected.



Note: The sample's record number should match the random sample you retrieved earlier, which will likely differ from this screenshot.

7. Replace missing values for `number_transactions` with 0.

- a) Scroll down and view the cell titled **Replace missing values for `number_transactions` with 0**, then select the code cell below it.
b) In the code cell, type the following:

```

1 users_data_cleaned['number_transactions']. \
2    fillna(0, inplace = True)

```

It doesn't always make sense to use something like mean imputation to fill in missing numeric values. In this case, it makes more sense to just set the number of transactions to 0.

- c) Run the code cell.
d) Select the next code cell, then type the following:

```

1 users_data_cleaned[users_data_cleaned. \
2                     user_id.isin(sample_user)]['number_transactions']

```

- e) Run the code cell.
- f) Examine the output.

```
12481    0.0
Name: number_transactions, dtype: float64
```

The sample user's `number_transactions` value is now 0.0.

8. Identify categorical data with missing values.

- a) Scroll down and view the cell titled **Identify categorical data with missing values**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 categorical_columns = \
2     list(users_data_cleaned.select_dtypes(['object']).columns)
3
4 print('Categorical features with missing values:', 
5       list(set(categorical_columns).intersection(missing_columns)))
```

Now that you've filled in the missing values for the numeric features, you can move on to the categorical features.

- c) Run the code cell.
- d) Examine the output.

```
Categorical features with missing values: ['job', 'device', 'education', 'contact']
```

The features `job`, `education`, `device`, and `contact` all have missing values.

9. Replace categorical missing values with 'Unknown'.

- a) Scroll down and view the cell titled **Replace categorical missing values with 'Unknown'**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_data_cleaned.device.fillna('Unknown', inplace = True)
2 users_data_cleaned.education.fillna('Unknown', inplace = True)
3 users_data_cleaned.contact.fillna('Unknown', inplace = True)
4 users_data_cleaned.job.fillna('Unknown', inplace = True)
```

For now, you'll simply replace all categorical missing values with the string '`Unknown`'.

- c) Run the code cell.
- d) Select the next code cell, then type the following:

```
1 users_data_cleaned.device.value_counts()
```

- e) Run the code cell.

- f) Examine the output.

```
mobile      27504
desktop    12112
tablet     5499
Unknown      94
Name: device, dtype: int64
```

This confirms that the `device` feature now has 94 values marked as 'Unknown'. The other three features underwent this same transformation.

10. Check if there are any other missing values.

- Scroll down and view the cell titled **Check if there are any other missing values**, then select the code cell below it.
- In the code cell, type the following:

```
1 missing_value_df = missing_value_pct_df(users_data_cleaned)
2 missing_columns = \
3 list(missing_value_df[missing_value_df['percent_missing'] \
4 > 0]['column_name'])
5
6 print('Number of features with missing values:', len(missing_columns))
7 print('Features with missing values:', missing_columns)
```

You're almost done handling missing values, but you need to make sure there aren't any stragglers.

- Run the code cell.
- Examine the output.

```
Number of features with missing values: 1
Features with missing values: ['date_joined']
```

There's still one column with missing values that you'll need to address: `date_joined`.

11. Remove all rows where `date_joined` is missing.

- Scroll down and view the cell titled **Remove all rows where `date_joined` is missing**, then select the code cell below it.
- In the code cell, type the following:

```
1 print('Number of users with corrupted data:',
2     users_data_cleaned[users_data_cleaned['date_joined']. \
3         isnull()].shape[0])
```

You'd expect all users to have a date joined, so the presence of missing values could indicate data corruption. To be safe, you'll drop any rows that have missing join dates.

- Run the code cell.
- Examine the output.

```
Number of users with corrupted data: 30
```

Only 30 users have missing join dates.

- e) Select the next code cell, then type the following:

```
1 # Remove corrupted data.
2
3 users_data_cleaned = \
4 users_data_cleaned[~users_data_cleaned['date_joined'].isnull()]
```

This code will actually remove the corrupted rows.

- f) Run the code cell.
g) Select the next code cell, then type the following:

```
1 # Check to see if any corrupted rows remain.
2
3 print('Number of users with corrupted data:',
4       users_data_cleaned[users_data_cleaned['date_joined']. \
5                           isnull()].shape[0])
```

- h) Run the code cell.
i) Examine the output.

Number of users with corrupted data: 0

All corrupted rows were removed.

12. Perform one last check for missing values.

- a) Scroll down and view the cell titled **Perform one last check for missing values**, then select the code cell below it.
b) In the code cell, type the following:

```
1 missing_value_df = missing_value_pct_df(users_data_cleaned)
2 missing_columns = \
3 list(missing_value_df[missing_value_df['percent_missing'] \
4 > 0]['column_name'])
5
6 print('Number of features with missing values:',
7       len(missing_columns))
```

Just to be sure, you'll do one final check.

- c) Run the code cell.
d) Examine the output.

Number of features with missing values: 0

All of the missing values have been removed from the dataset.

13. Keep this notebook open.

Feature Scaling

Machine learning algorithms find patterns in data using different approaches. In some cases, the distribution of the data is a primary factor, and the actual magnitude of values is either secondary, unimportant, or actively harmful to the process. So, you may need to apply scaling functions to your numeric variables in order to emphasize their distribution, while de-emphasizing the differences in

scales. Otherwise, one feature could exert more influence than another simply because it deals with numbers on a larger scale. Scaling is particularly important when using distance-based algorithms like support-vector machines (SVMs), whereas tree-based algorithms like decision trees and random forests don't require features to be scaled.

Imagine you have a dataset of vehicles with two numerical features: `miles_driven` and `years_old`. The problem is, the former has numbers that probably extend into the hundreds of thousands, whereas the latter will have a maximum value in the tens. Both of these features could be equally useful to a machine learning task, say if you wanted to estimate a car's value on the market. But, since the features are on wildly different scales, distance-based algorithms might treat `miles_driven` as being much more important. You therefore need to ensure the algorithm sees each feature in terms of distribution of values. The two major approaches for this are normalization and standardization.

Normalization and Standardization

Normalization involves transforming a feature such that the lowest value is a 0, and the highest value is a 1. So, whatever vehicle has the fewest miles driven will have a 0 in place of whatever the true value is. And, the vehicle with the most miles driven will have a 1 instead of the true value. The same exact principle applies to the age of the vehicle as well, and any other numeric feature that you believe needs to be scaled. Now the machine learning algorithm will see the distribution of these features rather than the absolute values.

Standardization also provides scaling, but does so in a different way. It calculates a value's ***z-score*** (also called ***standard score***) as the number of standard deviations that the sample is above or below the mean of all values in the sample. You can obtain the *z-score* for any vehicle's miles driven, age, etc. Such scores are standardized so that each feature has a mean value of zero and a standard deviation of 1. Once again, this emphasizes the feature's distribution instead of its absolute scale.

One approach is not necessarily better than the other in all cases. Normalization tends to be useful when the raw data doesn't follow a normal distribution and you want to use the data with an algorithm that doesn't assume any particular type of distribution. It's also good at minimizing the effect of outliers. Since standardization has no upper or lower bounds, it doesn't minimize outliers in the same way. It's often useful for when the raw data is already normally distributed and you just want to ensure two features are on the same scale. Still, it's usually best to experiment with both to determine what's best for your situation.



Note: You scale data used to train the algorithm, but you also use those same scaling parameters (e.g., the mean of a training feature) to scale the equivalent test data.

Additional Transformation Functions

There are other ways to transform features besides normalization and standardization. Some of these transformation functions are described in the following table.

Transformation Function	Description
Log	This function calculates the log base 10 of x , log base e of x ($\ln(x)$), or log base 2 of x , where x is a data example. The log transformation function helps to reduce skewness (particularly positive skewness) in non-normally distributed datasets. Note that it can only be used on positive numbers.
Cube root	This function raises each data example to a power of $1 / 3$. The cube root function helps reduce positive skewness as well, but not as strongly as a log function. However, unlike log, it can be used on negative numbers and zero, in addition to positive numbers.

Transformation Function	Description
Box-Cox	<p>This function raises each data example to a power of some lambda (λ) value, subtracts 1, then divides that result by λ. This calculation happens for all values of λ (usually between -5 and 5) until the transformed data approximates a normal distribution. Note that whenever λ is 0, the function simply takes the log of the data example.</p> <p>Box-Cox, therefore, also helps to reduce skewness and obtain normally distributed data. It leverages both log transformations and power transformations. There is also a version that can be used to transform negative values.</p>



Note: These are just a few common examples. Depending on the circumstances, various other functions may be appropriate, such as sigmoid, hyperbolic tangent, reciprocals, Yeo-Johnson, O'Brien, and so forth.

Guidelines for Applying Transformation Functions to Datasets

Follow these guidelines when applying transformation functions to datasets.

Apply Transformation Functions to Datasets

When applying transformation functions to datasets:

- Test different transformations with the model or algorithm used, as different data can behave differently for a given transformation.
- Recognize if transformations are required for the model or algorithm used.
- Use normalization when different variables need to be between a given range.
- Use standardization to preserve distributions when comparing variables with very different scales.

ACTIVITY 3-10

Applying Transformation Functions to a Dataset

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **View the distribution of age**. Select **Cell**→**Run All Above**.

Scenario

You've seen that a lot of the numeric features in the dataset are skewed. Adjusting this skewness can better facilitate the model training process. These types of adjustments can be dependent on the type of model you're training, so for now, you'll just temporarily transform the `age` feature to get a sense of how such transformations can help mitigate distribution issues. Later, when you start building machine learning models, you'll apply other transformations to the dataset.

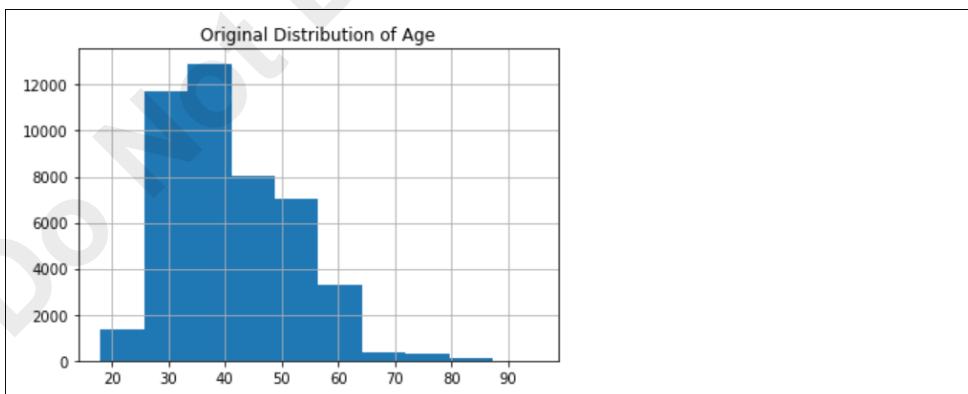
1. View the distribution of age.

- a) Scroll down and view the cell titled **View the distribution of age**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_data_cleaned['age'].hist()
2 plt.title('Original Distribution of Age');
```

While `age` is less skewed than a lot of the other features, it still exhibits some degree of skewness than you can address through transformations. This code will show a histogram of `age` as a refresher.

- c) Run the code cell.
- d) Examine the output.



This shows a left (positive) skew. Recall that you identified outliers above 70 years of age.

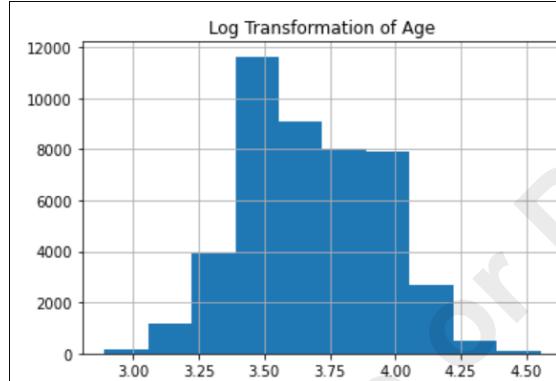
2. Apply a log transformation to age.

- Scroll down and view the cell titled **Apply a log transformation to age**, then select the code cell below it.
- In the code cell, type the following:

```
1 np.log(users_data_cleaned['age']).hist()  
2 plt.title('Log Transformation of Age');
```

First you'll apply a log transformation to the `age` feature.

- Run the code cell.
- Examine the output.



As you can see, `age` is now closer to a normal distribution. The distribution no longer trails off toward the high end of values.

3. Apply a Box–Cox transformation to age.

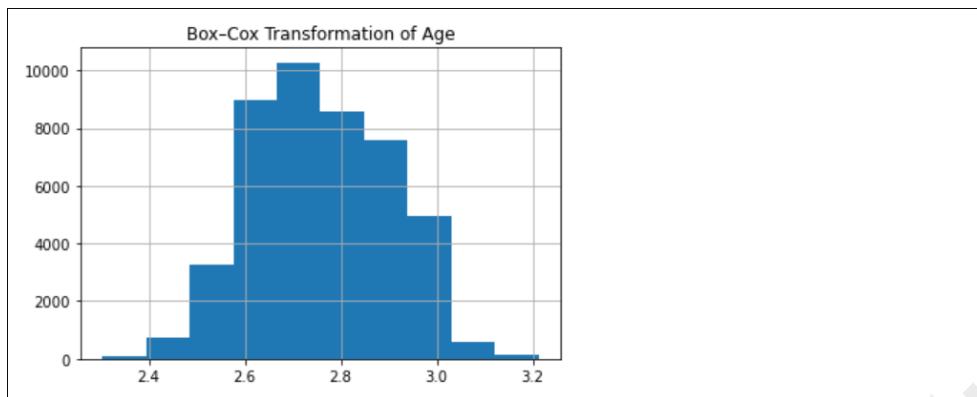
- Scroll down and view the cell titled **Apply a Box–Cox transformation to age**, then select the code cell below it.
- In the code cell, type the following:

```
1 from scipy import stats  
2  
3 pd.Series(stats.boxcox(users_data_cleaned['age'])[0]).hist()  
4 plt.title('Box–Cox Transformation of Age');
```

You'll also apply a Box–Cox transformation to `age` to see how its results differ from the log transformation.

- Run the code cell.

- d) Examine the output.



Like the log transformation, the Box–Cox transformation moved the data closer to a normal distribution. However, there are differences in each technique's outcome. One is not necessarily better than the other in every scenario.

4. Keep this notebook open.

Feature Engineering

Feature engineering is the technique of generating and extracting features from data in order to improve the ability for a machine learning model to make estimations. The features you currently have are not necessarily the best possible features for the problem you're trying to solve. By creating new, more useful features, you have a better chance of producing a model that excels at its given task.

Feature engineering is really an umbrella term for multiple types of tasks that can achieve this goal. Several of these tasks will be discussed shortly, including:

- Encoding categorical data.
- Binning continuous variables.
- Splitting features.
- Selecting and extracting features to reduce dimensionality.

Before you start engineering new features, you should make sure you've completed some of the earlier preprocessing tasks that were mentioned, particularly handling missing data and removing duplicate values. You should determine what features can be outright removed from the dataset because they are either redundant or have no estimative power. For example, `user_id` might make a good primary key in a relational table, but an estimative model is probably not going to learn much from it. Once these preliminary tasks are complete, you can move on to feature engineering.

Data Encoding

Data encoding is the process of converting data of a certain type into a coded value of a different type. In data science, this typically means taking a string of text and converting it into a number. The resulting number becomes a new feature within the dataset, and the original feature is either removed or ignored when a model is built.

Data encoding is important because many machine learning algorithms simply can't handle categorical data that is represented in strings. For example, let's say you have a `city` feature that has the possible values `['Berlin', 'London', 'Rochester']`. This is a categorical variable and should be treated as such by the algorithm. But rather than feed those strings directly to the algorithm, you'll likely need to convert them to numbers first. However, keep in mind that encoding

a categorical variable does not mean that it becomes a numerical feature. The feature is still categorical, even though it is being represented using a number.

Data Encoding Methods

There are various methods for converting categorical features to numbers. Some of the most common encoding methods are listed in the following table.

Encoding Method	Description
Label encoding	<p>Label encoding translates each unique value in a label (the feature you're interested in studying) into separate numbers. So, <code>city_encoded</code> would have <code>[0, 1, 2]</code> as possible values, where 0 is Berlin, 1 is London, and 2 is Rochester.</p> <p>Because machine learning algorithms may perceive an order or ranking to numbered categories, label encoding is most suitable for categories meant to imply a sequence or rank. That's why it's also called ordinal encoding. In this example, a machine learning algorithm might perceive that Rochester (with a value of 2) is ranked the highest, which may not be what you intended.</p>
One-hot encoding	<p>If you don't want to imply a sequence or rank when you encode categorical labels, you can use one-hot encoding.</p> <p>With this method, you create a dummy column for each class of a categorical attribute. For example, you might create three columns named <code>isBerlin</code>, <code>isLondon</code>, and <code>isRochester</code>. The presence of each class is represented by 1 and its absence is represented by 0. This ensures that the machine learning algorithm will give no class (Berlin, London, or Rochester, in this case) more value than the others.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Note: One-hot encoding is sometimes conflated with the very similar technique of dummy encoding. However, dummy encoding creates $n - 1$ columns, whereas one-hot encoding creates n columns, where n is the number of unique values in the categorical variable. </div>
Binary encoding	<p>This encoding scheme converts each categorical value into an ordinal integer, then converts that into a binary digit. Each binary digit becomes a new column. Some information is lost in this conversion, but it's more efficient than one-hot encoding, because with binary encoding, you don't need one column for each possible value. So, binary encoding is preferred when the number of possible categorical values is high.</p>
Effect encoding	<p>This is similar to binary encoding, except the encoded values are either -1, 0, or 1. Effect encoding is a more advanced technique and is less common.</p>
Frequency encoding	<p>With this approach you calculate how frequently each class occurs within the training set, and you use this number as the code for that class, essentially determining the weight of that class within the dataset, and using that value as its code.</p>
Target mean encoding	<p>With this encoder, each class is encoded as a function of the mean of the target variable (i.e., the label in supervised learning). So the target variable must be numeric, whereas the variable being encoded is categorical.</p>

Encoding Method	Description
Hash encoding	This scheme uses a hash encoding algorithm to map a particular text string to a number value. While the resulting hash value appears to be random, it is algorithmic, based on the characters in the text string, and will produce the same number value each time a particular text string is provided to it. This method can be a useful way to generate consistent codes from text values when there are hundreds or thousands of categories.
Base-N encoding	This encoding scheme chooses a number base to convert the categorical values into. Base-1 encoding is the same as one-hot encoding, and base-2 encoding is the same as binary encoding. Otherwise, this particular encoding style confers no additional advantages.

Guidelines for Encoding Data

Follow these guidelines when encoding data.

Encode Data

When encoding data:

- Use integers when assigning labels or categories to discrete non-continuous values even when they appear to be numeric (e.g., ZIP codes).
- Convert categories with an inherent, natural ordering to a sequence using ordinals.
- Maintain a lookup table to generate the category names from the values when reporting or plotting.
- Use one-hot encoding when a machine learning algorithm calls for a vector of categories for each record.
- Consider hashing labels when there are a large number (100+) of categories in order to reduce the number of categories, such as ZIP codes or business merchant category codes. Performance and accuracy are often inversely correlated with the number of categories.

ACTIVITY 3-11

Encoding Data

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Identify categorical features**. Select **Cell→Run All Above**.

Scenario

Since you plan to input this data into multiple machine learning algorithms, you need to make sure the data is actually in a form that those algorithms can read. Many machine learning algorithms can't deal with categorical features that use string values, so those values need to be encoded as numbers. For the most part, you'll use the common one-hot encoding technique to ensure that the features are properly formatted for machine learning.

1. Identify categorical features.

- Scroll down and view the cell titled **Identify categorical features**, then select the code cell below it.
- In the code cell, type the following:

```

1 categorical_columns = \
2 list(users_data_cleaned.select_dtypes(['object']).columns)
3
4 print('The number of categorical features:', 
5      len(categorical_columns))
6 print('The names of categorical features:', 
7      categorical_columns)

```

You'll start by taking stock of your categorical features.

- Run the code cell.
- Examine the output.

```
The number of categorical features: 6
The names of categorical features: ['user_id', 'job', 'marital', 'education', 'contact', 'device']
```

There are 6 total categorical features, though there are actually 5, since you can disregard `user_id`.

2. One-hot encode `job`.

- Scroll down and view the cell titled **One-hot encode job**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data_cleaned.job.value_counts(dropna = True)
```

This code will provide a refresher on the number of values in the `job` feature.

- Run the code cell.

- d) Examine the output.

```
blue-collar      9725
management      9453
technician       7592
admin.           5168
services          4152
retired            2262
self-employed     1577
entrepreneur      1485
unemployed         1301
housemaid          1239
student             937
Unknown              288
Name: job, dtype: int64
```

There are 12 total types of jobs (including unknown).

- e) Select the next code cell, then type the following:

```
1 # Create object for one-hot encoding.
2
3 encoder = ce.OneHotEncoder(cols = 'job',
4                             return_df = True,
5                             use_cat_names = True)
```

The `job` feature has no natural order, so it's a good candidate for one-hot encoding. Each value will be mapped to its own feature, where `0` indicates absence and `1` indicates presence. The `encoder` object will be used to do the actual encoding on the dataset.

- f) Run the code cell.

- g) Select the next code cell, then type the following:

```
1 # Fit and transform data.
2
3 users_data_encoded = encoder.fit_transform(users_data_cleaned)
4
5 # Preview the data.
6
7 users_data_encoded.head()
```

This code will use the `encoder` object to do the actual transformation.

- h) Run the code cell.

- i) Examine the output.

count_usd	age	job_management	job_technician	job_entrepreneur	job_blue-collar	job_Unknown	job_retired	...	housing	loan	contact	duration	campaign	pdays
2143.00	58	1	0	0	0	0	0	0 ...	True	False	Unknown	261	1	
1369.42	44	0	1	0	0	0	0	0 ...	True	False	Unknown	151	1	
2.00	33	0	0	1	0	0	0	0 ...	True	True	Unknown	76	1	
1369.42	47	0	0	0	1	0	0	0 ...	True	False	Unknown	92	1	
1.00	33	0	0	0	0	1	0	0 ...	False	False	Unknown	198	1	

You can see various new columns in the dataset, called `job_x` where `x` is one of the 12 values. Each row has a 0 in every `job_x` column except for one, in which it has a 1 value. For example, the first user in the dataset is a manager.

- j) Select the next code cell, then type the following:

```
1 | list(users_data_encoded)
```

- k) Run the code cell.
l) Examine the output.

```
['user_id',
 'number_transactions',
 'total_amount_usd',
 'age',
 'job_management',
 'job_technician',
 'job_entrepreneur',
 'job_blue-collar',
 'job_Unknown',
 'job_retired',
 'job_admin.',
 'job_services',
 'job_self-employed',
 'job_unemployed',
 'job_housemaid',
 'job_student',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'term_deposit',
 'date_joined',
 'device']
```

Here's another view of all the new features.

- m) Select the next code cell, then type the following:

```
1 print('Shape of data before encoding:',  
2       users_data_cleaned.shape)  
3 print('Shape of data after encoding:',  
4       users_data_encoded.shape)
```

- n) Run the code cell.
o) Examine the output.

```
Shape of data before encoding: (45179, 18)  
Shape of data after encoding: (45179, 29)
```

As expected, 11 new columns were added. (The original `job` column was removed.)

3. Dummy encode `marital`.

- a) Scroll down and view the cell titled **Dummy encode `marital`**, then select the code cell below it.
b) In the code cell, type the following:

```
1 marital_encoded = \  
2 pd.get_dummies(data = users_data_encoded['marital'],  
3                  drop_first = True)  
4  
5 marital_encoded
```

The `marital` feature will undergo a similar treatment as `job`, except the first values will be removed and will not become a new feature. Some practitioners call this dummy encoding in order to distinguish it from one-hot encoding.

- c) Run the code cell.
d) Examine the output.

	<code>married</code>	<code>single</code>
0	1	0
1	0	1
2	1	0
3	1	0
4	0	1
...
45211	1	0
45212	0	0
45213	1	0
45214	1	0
45215	1	0

45179 rows × 2 columns

Since the first value of `marital` is divorced, it did not become a new feature, whereas `married` and `single` did.

- e) Select the next code cell, then type the following:

```

1 # Concatenate the new encoded columns.
2
3 users_data_encoded = \
4 pd.concat([users_data_encoded, marital_encoded], axis = 1)
5
6 # Drop the original variable.
7
8 users_data_encoded.drop(['marital'], axis = 1, inplace = True)
9
10 # Preview the data
11
12 users_data_encoded.head()

```

This code adds the new dummy-encoded features to the dataset, while dropping the original marital feature.

- f) Run the code cell.
g) Examine the output.

job_entrepreneur	job_blue-collar	job.Unknown	job_retired	...	contact	duration	campaign	pdays	previous	term_deposit	date_joined	device	married	single
0	0	0	0	...	Unknown	261	1	-1	0	False	1998-08-23	mobile	1	0
0	0	0	0	...	Unknown	151	1	-1	0	False	2008-07-15	desktop	0	1
1	0	0	0	...	Unknown	76	1	-1	0	False	2002-06-04	mobile	1	0
0	1	0	0	...	Unknown	92	1	-1	0	False	1995-06-29	tablet	1	0
0	0	1	0	...	Unknown	198	1	-1	0	False	1995-08-01	mobile	0	1

At the right side of the table, you can see the two new features.

- h) Select the next code cell, then type the following:

```

1 print('Shape of data after encoding:',
2 users_data_encoded.shape)
3
4 list(users_data_encoded)

```

- i) Run the code cell.

- j) Examine the output.

```
Shape of data after encoding: (45179, 30)

['user_id',
 'number_transactions',
 'total_amount_usd',
 'age',
 'job_management',
 'job_technician',
 'job_entrepreneur',
 'job_blue-collar',
 'job_Unknown',
 'job_retired',
 'job_admin.',
 'job_services',
 'job_self-employed',
 'job_unemployed',
 'job_housemaid',
 'job_student',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'term_deposit',
 'date_joined',
 'device',
 'married',
 'single']
```

There are now a total of 30 columns.

4. One-hot encode the remaining categorical variables.

- Scroll down and view the cell titled **One-hot encode the remaining categorical variables**, then select the code cell below it.
- In the code cell, type the following:

```
1 cols = ['education', 'contact', 'device']
2
3 encoder = ce.OneHotEncoder(cols = cols,
4                             return_df = True,
5                             use_cat_names = True)
```

You're creating another `encoder` object, this time for the remaining three categorical variables.

- Run the code cell.
- Select the next code cell, then type the following:

```
1 # Fit and transform data.
2
3 users_data_encoded = encoder.fit_transform(users_data_encoded)
4
5 # Preview the data.
6
7 users_data_encoded.head()
```

- Run the code cell.

- f) Examine the output.

ie- lar	job_Unknown	job_retired	...	pdays	previous	term_deposit	date_joined	device_mobile	device_desktop	device_tablet	device_Unknown	married	single
0	0	0	...	-1	0	False	1998-08-23	1	0	0	0	1	0
0	0	0	...	-1	0	False	2008-07-15	0	1	0	0	0	1
0	0	0	...	-1	0	False	2002-06-04	1	0	0	0	1	0
1	0	0	...	-1	0	False	1995-06-29	0	0	1	0	1	0
0	1	0	...	-1	0	False	1995-08-01	1	0	0	0	0	1

You can see some of the new `device_x` columns, but since there are so many, they get truncated in output.

- g) Select the next code cell, then type the following:

```
1 print('Shape of data after encoding:',  
2      users_data_encoded.shape)  
3  
4 list(users_data_encoded)
```

- h) Run the code cell.
i) Examine the output.

```
Shape of data after encoding: (45179, 38)  
['user_id',  
 'number_transactions',  
 'total_amount_usd',  
 'age',  
 'job_management',  
 'job_technician',  
 'job_entrepreneur',  
 'job_blue-collar',  
 'job_Unknown',  
 'job_retired',  
 'job_admin.',  
 'job_services',  
 'job_self-employed',  
 'job_unemployed',  
 'job_housemaid',  
 'job_student',  
 'education_tertiary',  
 'education_secondary',  
 'education_Unknown',  
 'education_primary',  
 'default',  
 'housing',  
 'loan',  
 'contact_Unknown',  
 'contact_cellular',  
 'contact_telephone',  
 'duration',  
 'campaign'
```

The dataset now has 38 total columns, which is more than double its original size.

5. Keep this notebook open.

Continuous Variable Discretization

Recall that the difference between a continuous variable and a discrete variable is that the former is uncountable because it has no well-defined gaps between values, whereas the latter does have clear gaps. Some algorithms, like decision trees, have a difficult time working with continuous variables because the tree can just keep splitting over and over again until it becomes too large and inefficient. If you've identified useful features in your dataset that have continuous variables, you may need to engineer discrete features out of them before you input the data to an algorithm.

The process of converting a continuous variable into a discrete variable is called ***discretization***. This can greatly simplify and improve the performance of your models. The primary method of discretization is to take a continuous variable and place its values within specific, discrete intervals—a process also called ***data binning***.

In the example of people running a marathon, the `Time in Minutes` feature is one example of binning a continuous variable. If you assume that the fastest runner finished the race in around 120 minutes and the slowest took around 600 minutes, you could place a time into one of 480 different minute intervals. Now that the variable is discrete, machine learning algorithms like decision trees will be able to handle it.

Bin Determination

Binning requires selecting the number of bins in which to place the values, as well as the range of values between each of the bins—a bin's "width." The choice of number of bins is dependent on the size of the dataset. Typically, with data examples in the thousands, you should choose five or more bins. The 480 bins for `Time in Minutes` are likely too many regardless. A good rule of thumb is to have no more than 20 bins. So, you might instead choose `Time in Hours` (8 bins); or, you might choose a smaller number of bins based on some wider time interval. As the number of bins increases, so too does the complexity of the model, so keep that in mind. Also, no matter how you bin a variable, some amount of information and precision will be lost as compared to the original continuous variable.

As for determining bin width, there are two primary approaches: equal-width and equal-frequency. In equal-width binning, each bin spans the same range of values. For example, if you specify 8 bins for times between 120 and 600 minutes, each bin will have a range of 60 minutes. The range between minimum (120) and maximum (600) values is simply divided by the number of bins. This approach is sensitive to outliers in the data, and it tends to lead to higher levels of information loss. In the equal-frequency approach, each bin contains the same number of values. The distribution of the variable's values is taken into account. So, one bin might be wider than another if it contains fewer values. This approach tends to be better at minimizing information loss, and it is also less sensitive to outliers.

Binning of a feature where the classification label is known (i.e., in supervised learning) can use that label to make more informed determinations about bin number and width. The idea is to measure the relationship between all of the feature's values and the classification labels. The closer the relationship between a feature and a label, the more the complexity of the binning process is justified. A feature that has minimal correlation with the label will be discretized into a small number of bins, even just one bin if there is no significant correlation. In unsupervised tasks, the discretization cannot consider any such relationship, and must use the variable's distribution to determine the number of bins and the width of each bin.

Guidelines for Discretizing Variables

Follow these guidelines when discretizing variables.

Discretize Variables

When discretizing variables:

- Convert continuous variables to discrete ordinals for models or algorithms that cannot handle discrete data or that train faster with a limited number of ordinals for a variable.
- Determine how many bins are required based on the use of the data. Most machine learning algorithms work better with 5 to 10 bins.
- Use a uniform distribution when the shape and other characteristics of the distribution should be preserved.
- Choose a quantile distribution with the same number of samples in each bin when favored by the algorithm being used.
- Recognize that the loss in precision is often offset by the gain in training speed and/or performance of a model.

ACTIVITY 3-12

Discretizing Variables

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Discretize age into bins**. Select **Cell**→**Run All Above**.

Scenario

When you eventually build machine learning models using certain types of algorithms, you'll need to transform your continuous variables into discrete ones. You'll start with `age`, which you'll place into one of several bins. These bins will act as categories to a new feature, which you'll use to replace the original continuous `age` variable.

1. Discretize `age` into bins.

- Scroll down and view the cell titled **Discretize age into bins**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data_encoded.age.describe()
```

You'll get a quick look at the distribution of `age` again to help you determine how to bin it. You could also look at a box plot to get a similar idea.

- Run the code cell.
- Examine the output.

count	45179.000000
mean	40.935103
std	10.618499
min	18.000000
25%	33.000000
50%	39.000000
75%	48.000000
max	95.000000
Name:	age, dtype: float64

Since the minimum age is 18 and the maximum is 95, it might be a good idea to bin in gaps of 10. This is an example of equal-width binning. You can also place the outliers into a wider bin that captures anyone above 75. There's no objectively "correct" way to bin, so this is just one way you could do it.

- e) Select the next code cell, then type the following:

```

1 # Define age bins and labels.
2
3 bins = [18, 25, 35, 45, 55, 65, 75, 110]
4 labels = ['18-24', '25-34', '35-44',
5           '45-54', '55-64', '65-74', '75+']
6
7 # Perform binning using bin list.
8
9 users_data_encoded['age_group'] = \
10 pd.cut(users_data_encoded['age'], bins=bins,
11         labels=labels, right=False)
12
13 # Map bins to integer values.
14 users_data_encoded['age_group_encoded'] = \
15 users_data_encoded['age_group'].cat.codes

```

To save you some typing, the `bins` list has already been created for you. There's also a `labels` list that you'll use to make visuals easier to read. The code you'll type includes a call to `cut()`, which does the actual segmentation of the continuous variable into bins. Below that, you're also mapping the bins to integer values, so that the first bin is 0, the second is 1, and so on.

- f) Run the code cell.
g) Select the next code cell, then type the following:

```

1 # Verify correct binning.
2
3 age_vars = ['age_group_encoded', 'age_group', 'age']
4
5 users_data_encoded[age_vars].sample(10)

```

- h) Run the code cell.
i) Examine the output.

	age_group_encoded	age_group	age
21005	4	55-64	55
19010	2	35-44	42
30593	4	55-64	64
44973	1	25-34	27
8851	0	18-24	24
26435	2	35-44	37
20102	3	45-54	53
19892	1	25-34	34
20748	3	45-54	49
22036	2	35-44	38

The `age_group_encoded` column has integer values for each bin, which map to an `age_group`. Everything should appear correct when compared to the actual `age`.



Note: Your random sampling will likely differ from the screenshot, but it should nonetheless help you verify the binning was performed correctly.

2. Plot the new distribution of `age`.

- Scroll down and view the cell titled **Plot the new distribution of age**, then select the code cell below it.
- In the code cell, type the following:

```
1 user_age_dist = users_data_encoded.age_group.value_counts()
2
3 user_age_dist
```

- Run the code cell.
- Examine the output.

35–44	14524
25–34	14194
45–54	9951
55–64	4892
18–24	809
65–74	510
75+	299
Name:	age_group, dtype: int64

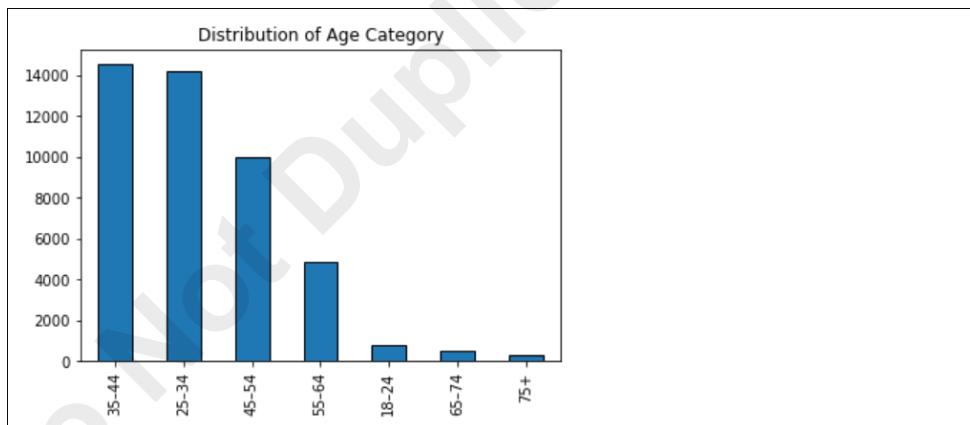
Each bin has a set number of values, with the 35–44 bin being the largest, with 25–34 following close behind.

- Select the next code cell, then type the following:

```
1 user_age_dist.plot(kind = 'bar', edgecolor = 'black')
2 plt.title('Distribution of Age Category');
```

To get a better sense of the distribution, you'll plot it visually using a bar chart. Since you're no longer working with a continuous variable, plotting a histogram wouldn't make sense.

- Run the code cell.
- Examine the output.



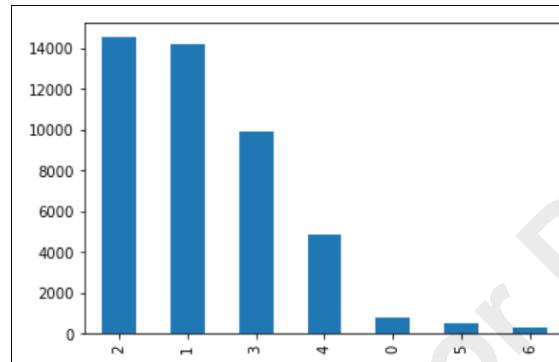
You can see the different age bins that are represented in the dataset. For example, the youngest age range (18–24) seems to have a low amount of representation.

- h) Select the next code cell, then type the following:

```
1 # Check against encoded values.
2
3 users_data_encoded.age_group_encoded. \
4 value_counts().plot(kind = 'bar');
```

Just to make sure the bins are correct, you'll plot the raw integer values for the bins rather than just their labels.

- i) Run the code cell.
- j) Examine the output.



The integer values should line up as expected. For example, bin 0 (18–24) is still third from the right.

3. Drop the age and age_group variables.

- a) Scroll down and view the cell titled **Drop the age and age_group variables**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_data_encoded.drop(['age', 'age_group'],
2                         axis = 1, inplace = True)
3
4 list(users_data_encoded)
```

You'll drop the original `age` variable and the label-based `age_group` variable since they're no longer needed. Only the integer-based `age_group_encoded` variable will remain as a feature.

- c) Run the code cell.

- d) Examine the output.

```
'loan',
'contact_Unknown',
'contact_cellular',
'contact_telephone',
'duration',
'campaign',
'pdays',
'previous',
'term_deposit',
'date_joined',
'device_mobile',
'device_desktop',
'device_tablet',
'device_Unknown',
'married',
'single',
'age_group_encoded']
```

You can see `age_group_encoded` at the bottom of the list of columns.

4. Keep this notebook open.

Feature Splitting

In some cases, a feature in your dataset might benefit from being partitioned into two or more features. For example, let's say you have a feature called `name` that contains both first and last names. So, one of the values might be '`Emily Williams`'. If variations in names have an impact on whatever problem your model is trying to solve (e.g., determining ancestry), then this data might be more useful if first names and last names were split into their own separate features. That way the model can learn from both of these name types independently rather than considering them as just a singular piece of information.

Feature splitting is particularly common with text-based data like names, locations, identifiers, titles —any string that can be compounded with contextually related strings. The choice of whether or not to split such a string will be primarily informed by domain knowledge. Perhaps splitting `location = 'Rochester, NY, US'` into `city = 'Rochester'`, `state = 'NY'`, and `country = 'US'` will lead to better estimations, or perhaps not. You should consider experimenting with analysis and modeling to see how different splits produce different results.

Once you've made a decision to split a feature, there are many ways you can do so. For example, a splitting function like Python's `split()` will divide a string into a list based on a separator, with whitespace being the default. So `name.split()` will produce `['Emily', 'Williams']`. You can also specify a different separator as an argument to the function, which will come in handy with features that separate each piece of the string using something like a hyphen, a comma, a pipe, etc. Another type of function is `strip()` (`trim()` in some languages), which simply removes any leading and trailing whitespace before and after a string. You can use this to extract text in the middle of a string, for instance. You can also use a function like `reverse()` to reverse a list if it happens to be in the "wrong" order (e.g., `['Williams', 'Emily']`).

Feature splitting is also common with date and time values. For example, you might want to take `date = 2021-02-25` and turn it into separate features for `year`, `month`, and `day`. This is much easier if your date feature is already cast as a datetime type—you can simply retrieve each component using a datetime function like `month()`. If it's a string, you'll need to use string splitting functions like those just mentioned, which can be prone to error if the dates are formatted in different ways.

Regular Expressions

Using standard splitting functions will be sufficient in many cases, but sometimes a string is complex enough that you need to extract data using a specific pattern. This is where regular expressions come into play. A **regular expression** (also called regex or regexp) is a group of characters that describe how to execute a specific search pattern on a given text. Search operations using regular expressions use a common syntax, which includes various special characters that have specific uses. This results in the search being able to retrieve granular results that it would otherwise not be able to.

Regular expressions can get very complicated, and a deep dive into how to write a regular expression is beyond the scope of this course. However, the following example can at least give you an idea of the power behind regex. The book feature has values that are formatted like so:

'Gibson1984Neuromancer'. Let's say you want to split this into author, year, and title. You can do this using a regex with a splitting function:

```
df['book'].str.split('([A-Za-z]+)(\d{4})([A-Za-z]+)', expand = True)
```

The parentheses divide the regex into different groups (one for author, year, and title). The first group says to look for any letter in the alphabet, whether uppercase or lowercase ([A-Za-z]). The + says to look for one to an infinite number of these letters. So, it will find the author. The second group says to match any digit (\d) exactly four times (\d{4}), so it will find the year. The third group is the same as the first, so it will match the title. You'll end up with a DataFrame where you now have three individual columns instead of one.

Guidelines for Splitting Features

Follow these guidelines when splitting features.

Split Features

When splitting features:

- Split on punctuation if available in the field.
- Split on whitespace only when data has regular patterns and each split item does not have embedded spaces.
 - For example, whitespace shouldn't be used to split a data item of city name and state name because many cities and states like "St. Louis" and "New Hampshire" have embedded spaces.
 - Use regular expressions when the data format is predictable but doesn't have delimiters.
 - Take advantage of case differences, embedded numbers, punctuation, and other features when using regular expressions.
 - Retain only the new fields that are required, such as country code and area code when splitting a phone number.
 - Process features after splitting by cleaning up or applying transformation functions again as necessary.

ACTIVITY 3-13

Splitting and Removing Features

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Create a month_joined variable from date_joined**. Select **Cell**→**Run All Above**.

Scenario

Although the `date_joined` feature is useful, machine learning algorithms don't always work well with complex dates and times. You'll simplify the data by extracting just the months from this feature, and then make that its own feature. This could be useful for identifying things like churn rate, e.g., perhaps there's a higher churn rate in summer months than in winter.

You also want to see what remaining features you can potentially remove to improve the dataset's viability. Earlier you started identifying correlations between pairs of variables, so now you'll actually start dropping one variable in each of those high-correlation pairs. This will hopefully remove redundancy in the dataset. Likewise, you want to identify any features that exhibit low variance and drop them. Features with very low variance have values that are so similar they may not contain any truly useful information.

Lastly, with all of the preprocessing done on this dataset, you'll finally be ready to load it into a data file that you'll use with machine learning.

1. Create a `month_joined` variable from `date_joined`.

- Scroll down and view the cell titled **Create a month_joined variable from date_joined**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data_encoded['month_joined'] = \
2 users_data_encoded.date_joined.dt.month
```

This code leverages the datetime type to extract just the month values.

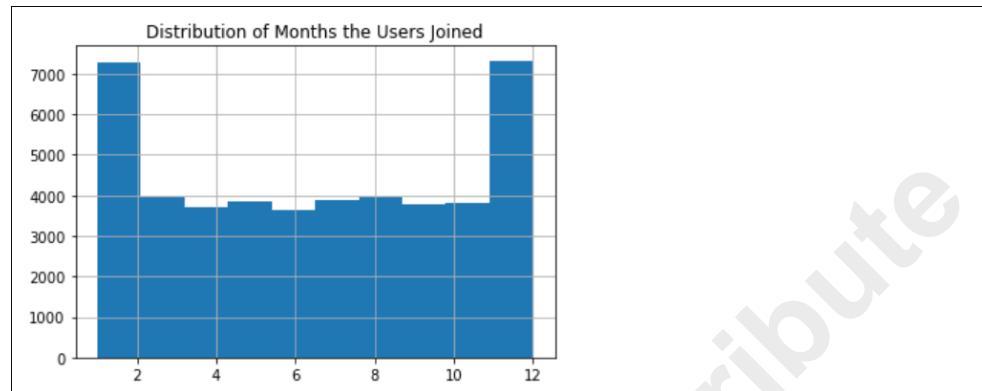
- Run the code cell.
- Select the next code cell, then type the following:

```
1 # View the distribution of data.
2
3 users_data_encoded['month_joined'].hist()
4 plt.title('Distribution of Months the Users Joined');
```

You'll take a look at the distribution of the new `month_joined` feature.

- Run the code cell.

- f) Examine the output.



It looks like there's a spike in January and December, indicating that most users seemed to join the bank in those months.

- g) Select the next code cell, then type the following:

```
1 users_data_encoded.drop(['date_joined'],
                           axis = 1, inplace = True)
2
3
4 list(users_data_encoded)
```

Since the original `date_joined` feature is no longer needed, you'll drop it.

- h) Run the code cell.
i) Examine the output.

```
'duration',
'campaign',
'pdays',
'previous',
'term_deposit',
'device_mobile',
'device_desktop',
'device_tablet',
'device_Unknown',
'married',
'single',
'age_group_encoded',
'month_joined']
```

The `month_joined` feature is in the list of columns.

2. Remove features with low variance.

- a) Scroll down and view the cell titled **Remove features with low variance**, then select the code cell below it.
b) In the code cell, type the following:

```
1 users_data_encoded.std()
```

You're retrieving the standard deviations of each feature. Remember that standard deviation is just the square root of variance.

- c) Run the code cell.

- d) Examine the output.

number_transactions	3.749994
total_amount_usd	2704.291321
job_management	0.406767
job_technician	0.373908
job_entrepreneur	0.178296
job_blue-collar	0.411004
job_Unknown	0.079587
job_retired	0.218087
job_admin.	0.318287
job_services	0.288889
job_self-employed	0.183543
job_unemployed	0.167236
job_housemaid	0.163318
job_student	0.142513
education_tertiary	0.455691
education_secondary	0.499832
education_Unknown	0.198480
education_primary	0.358591
default	0.133095
housing	0.496878
loan	0.366802
contact_Unknown	0.452851
contact_cellular	0.477695
contact_telephone	0.245250

Some features have a high standard deviation, though most seem to have a standard deviation below 1. You want to drop any features with very low values.

- e) Select the next code cell, then type the following:

```

1 # Define standard deviation threshold.
2
3 threshold = 0.1
4
5 # Identify features below threshold.
6
7 cols_to_drop = \
8 list(users_data_encoded.std()[users_data_encoded.std() \
9 < threshold].index.values)
10
11 print('Features with low standard deviation:',
12     cols_to_drop)

```

The threshold is used to define features with very low variance; i.e., any feature with a standard deviation less than 0.1. This is a somewhat arbitrary number, as there is no objectively "correct" threshold for every use case. But it's small enough not to drop too many features that could be useful.

- f) Run the code cell.
g) Examine the output.

Features with low standard deviation: ['job_Unknown', 'device_Unknown']

The job_Unknown and device_Unknown features exhibit low variance. Recall that these are one-hot encoded features from missing categorical values that you filled in with the string 'Unknown'.

- h) Select the next code cell, then type the following:

```

1 # Drop features below threshold.
2
3 users_data_interim = users_data_encoded.drop(cols_to_drop,
4                                              axis = 1)
5
6 list(users_data_interim)

```

- i) Run the code cell.
j) Examine the output.

```

['user_id',
 'number_transactions',
 'total_amount_usd',
 'job_management',
 'job_technician',
 'job_entrepreneur',
 'job_blue-collar',
 'job_retired',
 'job_admin.',
 'job_services',
 'job_self-employed',
 'job_unemployed',
 'job_housemaid',
 'job_student',
 'education_tertiary',
 'education_secondary'
]

```

Both of the identified columns have been removed.

3. Drop highly correlated features.

- a) Scroll down and view the cell titled **Drop highly correlated features**, then select the code cell below it.
b) In the code cell, type the following:

```

1 # Define correlation threshold.
2
3 threshold = 0.75
4
5 corr_matrix = users_data_encoded.corr().abs()
6 high_corr_var = np.where(corr_matrix >= threshold)
7 high_corr_var = [(corr_matrix.index[x],
8                   corr_matrix.columns[y],
9                   round(corr_matrix.iloc[x, y], 2))
10                  for x, y in zip(*high_corr_var)
11                  if x != y and x < y]
12
13
14 high_corr_var

```

You're defining another threshold, this time for correlation coefficient. The `high_corr_var` object will check for pairs of features that exhibit correlation above the threshold.

- c) Run the code cell.

- d) Examine the output.

```
[('contact_Unknown', 'contact_cellular', 0.86),
 ('device_mobile', 'device_desktop', 0.75),
 ('married', 'single', 0.77)]
```

The results indicate that:

- contact_Unknown and contact_cellular are highly correlated with a value of 0.86.
- device_mobile and device_desktop are highly correlated with a value of 0.75.
- married and single are highly correlated with a value of 0.77.

- e) Select the next code cell, then type the following:

```
1 # Tidy up the output.
2
3 record_collinear = pd.DataFrame(high_corr_var). \
4     rename(columns = {0: 'drop_feature',
5                   1: 'corr_feature',
6                   2: 'corr_values'})
7
8 record_collinear = record_collinear. \
9     sort_values(by = 'corr_values', ascending = False)
10
11 record_collinear = record_collinear.reset_index(drop = True)
12
13 record_collinear
```

This code will make the output a little easier to read, helping you identify which feature in the pair to drop, which to keep, and what the correlation coefficient for the pairing is.

- f) Run the code cell.
g) Examine the output.

	drop_feature	corr_feature	corr_values
0	contact_Unknown	contact_cellular	0.86
1	married	single	0.77
2	device_mobile	device_desktop	0.75

- h) Select the next code cell, then type the following:

```
1 cols_to_drop = list(record_collinear['drop_feature'])
2 print(cols_to_drop)
```

- i) Run the code cell.
j) Examine the output.

```
['contact_Unknown', 'married', 'device_mobile']
```

The contact_Unknown, married, and device_mobile features will be dropped.

- k) Select the next code cell, then type the following:

```
1 users_data_final = users_data_interim.drop(cols_to_drop,
2                                         axis = 1)
3
4 list(users_data_final)
```

- l) Run the code cell.
m) Examine the output.

```
['user_id',
 'number_transactions',
 'total_amount_usd',
 'job_management',
 'job_technician',
 'job_entrepreneur',
 'job_blue-collar',
 'job_retired',
 'job_admin.',
 'job_services',
 'job_self-employed',
 'job_unemployed',
 'job_housemaid',
 'job_student',
 'education_tertiary',
 'education_secondary',
 'education_unknown']
```

The three features have been removed.

4. Keep this notebook open.

Dimensionality Reduction

Adding to a dataset's feature space (also called its dimensionality), like when you split a single string into multiple strings, can make a model more effective. The model is given more potentially useful features to learn from, after all. Unfortunately, this is not always the case. Some features may be redundant or make the learning process too noisy, which can complicate the analysis and model building processes and have a negative impact on performance. If the number of data examples stays constant, then at some point, adding to a dataset's dimensionality will actually start reducing the model's ability to learn useful patterns from the data. This is called the *curse of dimensionality*.

Dimensionality reduction is the process of simplifying a dataset by eliminating redundant or irrelevant features. Dimensionality reduction can help reduce the problem of tuning a model so closely to the input data that the model performs poorly on new data samples (a problem called "overfitting"). Dimensionality reduction can also decrease computation time and alleviate storage space issues.

Two categories of dimensionality reduction are feature selection and feature extraction.

In **feature selection**, you select a subset of the original features. This subset includes relevant and/or unique features, and excludes features deemed redundant or irrelevant to the problem. The model learns from this subset rather than the whole dataset. Feature selection is particularly useful in datasets that have a disproportionately large number of features compared to actual data examples.

In **feature extraction**, you derive new features from the original features. This is typically done by combining multiple correlated features into one. For example, if you're trying to predict storage drive failure, the feature `age` probably correlates highly with the features `reads` and `writes`, so the three can be merged into one derivative feature that represents the drive's logical usage. Feature extraction is particularly useful in computer vision applications like image processing.

While the goal is to minimize the loss of useful data as much as possible, there is still a risk of this happening. Nevertheless, dimensionality reduction is almost always worth doing, especially with complicated, feature-rich datasets.

Dimensionality Reduction Methods

You can select and extract features manually, but this becomes tedious in high-dimensional datasets. It's also prone to error, as you might not be making optimal decisions about what to reduce and how. Thankfully, there are many ways to automate the dimensionality reduction process. The following table provides an overview of some of the most common dimensionality reduction methods.

Dimensionality Reduction Method	Description
Principal component analysis (PCA)	PCA performs a type of feature extraction by taking data that is in high dimensions and projecting that data into a space of equal or lower dimensions. It does this by selecting only the features that contribute to the greatest amount of linear variance in the dataset, while dropping the features that contribute very little to the variance.
Singular value decomposition (SVD)	SVD is similar to PCA; both of them decompose matrices of feature values in order to select a subset of features that better represent the data for modeling and analysis. In fact, many implementations of PCA actually use SVD to decompose matrices. SVD tends to work well on sparse matrices, i.e., matrices with mostly zeros.
<i>t</i> -distributed stochastic neighbor embedding (<i>t</i> -SNE)	<i>t</i> -SNE generates probability distributions of data pairs in high dimensions. Similar data examples are given a higher probability, whereas dissimilar examples are given a lower probability. Then, the process is repeated in lower dimensions until the divergence between higher and lower dimensions is minimized. Unlike PCA, <i>t</i> -SNE can retain non-linear variance.
Random forest	Random forests are a type of machine learning algorithm that is commonly used in classification and regression tasks, but they're also effective at feature selection. They can help you determine which features have the largest influence on the model's estimations by returning the weight of each feature as a percentage (out of 100). The higher the percentage, the more important a feature is to the estimator.
Forward feature selection	This method starts a model with no features from the dataset, then iteratively adds features until the performance of the model stops improving.
Backward feature elimination	This method starts a model with all features from the dataset, then iteratively removes features until the performance of the model stops improving.
Factor analysis	Factor analysis measures a latent variable; i.e., a variable that is not directly observed but which is identified through the relationship of other variables. These latent variables are selected as new features because they may exert more influence over a model than the observable variables.

Dimensionality Reduction Method	Description
Missing value ratio	This method removes any features that have a total number of missing values that exceed some predefined threshold. Missing values don't improve a model's performance, and, in some cases, they can actually hinder its performance. You can adjust the threshold to either increase or decrease the number of features that get removed.
Low-variance filter	This method removes any features that have a total variance lower than some predefined threshold. Low variance means the feature values don't change much between each data example, which doesn't help a model learn. As with missing value ratio, you can adjust the threshold as needed.
High-correlation filter	This method calculates the correlation coefficient between pairs of features and removes one of those features if the coefficient exceeds a predefined threshold. Correlated features provide similar information to a model, so they may be redundant for the learning process. Once again, you can tune the threshold as desired.

Guidelines for Performing Dimensionality Reduction

Follow these guidelines when performing dimensionality reduction.

Perform Dimensionality Reduction

When performing dimensionality reduction:

- Remove a variable that has a high correlation with another variable.
- Alternatively, join multiple variables that are closely correlated into a single variable with hyphens or spaces.
- Review the impact to results of removed or reduced variables, as the impact is not always predictable.
- Try different methods of dimensionality reduction for a collection of variables.

ACTIVITY 3-14

Performing Dimensionality Reduction

Before You Begin

Analyzing Data.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Filter by demographics data**. Select **Cell**→**Run All Above**.

Scenario

Your users dataset now has dozens of features, which will likely be useful in many machine learning scenarios. However, some machine learning tasks are more effective when they work with a reduced feature set. Likewise, there are times when you'll want to reduce your dataset's dimensionality to make it more conducive to plotting. There are only so many dimensions you can represent on a scatter plot, for instance.

So, you'll apply principal component analysis (PCA) to a subset of your data—demographics information, in particular—to reduce its dimensionality. Because most of your work will be done on the full users dataset, you'll save this as a separate dataset that you can call up whenever you need it.

1. Filter by demographics data.

- a) Scroll down and view the cell titled **Filter by demographics data**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_data_demographics = \
2 users_data_final.filter(regex = 'education|job|age|single')
3
4 users_data_demographics.head(n = 3)
```

To simplify things a bit, you'll filter the data to only include demographic features such as education level, job type, age, and marital status. The regular expression will capture all of these, so you don't need to specify each column name separately.

- c) Run the code cell.
- d) Examine the output.

	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_housemaid	job_stude
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0

As expected, the resulting DataFrame only includes demographic information.

2. Standardize the demographics data.

- a) Scroll down and view the cell titled **Standardize the demographics data**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 users_data_demographics.describe()
```

- c) Run the code cell.
d) Examine the output.

	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_hous
count	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000
mean	0.209234	0.168043	0.032869	0.215255	0.050068	0.114389	0.091901	0.034906	0.028797	0.0
std	0.406767	0.373908	0.178296	0.411004	0.218087	0.318287	0.288889	0.183543	0.167236	0.1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

All of the variables appear to be on the same scale (ranging from 0 to 1) except for `age_group_encoded`, which ranges from 0 to 6. Remember that these values correspond to the age range bins that you defined earlier. To ensure every feature is on the same scale before you perform PCA, you'll apply standardization.

- e) Select the next code cell, then type the following:

```
1 scaler = StandardScaler()
2
3 scaler.fit(users_data_demographics)
4 users_data_scaled = scaler.transform(users_data_demographics)
5
6 print('New standard deviation: ', users_data_scaled.std())
7 print('New mean: ', round(users_data_scaled.mean()))
```

This code standardizes the data so that the mean of the distribution is 0 and the standard deviation is 1.

- f) Run the code cell.
g) Examine the output.

```
New standard deviation: 1.0
New mean: 0
```

As expected, the data has been scaled using the z -score.

3. Perform PCA to reduce the dimensionality of the demographics dataset.

- a) Scroll down and view the cell titled **Perform PCA to reduce the dimensionality of the demographics dataset**, then select the code cell below it.
b) In the code cell, type the following:

```
1 pca = PCA(n_components = 2, random_state = 1)
2
3 pca.fit(users_data_scaled)
4
5 reduced = pca.transform(users_data_scaled)
```

You'll perform PCA to derive only 2 total features from the existing demographics data, which has 17 features.

- c) Run the code cell.
- d) Select the next code cell, then type the following:

```
1 reduced_df = pd.DataFrame(reduced, columns = ['PCA1', 'PCA2'])
2
3 reduced_df
```

Each of the derived features won't necessarily have a qualitative meaning, so you're assigning them the generic labels PCA1 and PCA2.

- e) Run the code cell.
- f) Examine the output.

	PCA1	PCA2
0	2.557545	1.079613
1	-0.820505	-1.750581
2	-0.576607	-0.551404
3	-0.541647	1.593626
4	0.385598	-0.614768
...
45174	1.160485	0.073978
45175	-0.660738	4.371075
45176	-1.327676	2.367745
45177	-1.575612	1.040206
45178	-0.632467	-0.134990
45179 rows × 2 columns		

All of the rows still represent users, but the columns have changed. The 17 demographics columns have been reduced to only 2. The numeric values in these columns were mathematically derived from the values that were in the 17 demographics columns. The idea is that these two columns now represent an approximation of the initial feature space. The relationships between the records and the features are mostly preserved, but are now in a more compact form. The values may not have much meaning to a human observer, but they can still be meaningful to a machine learning algorithm.

A reduced dataset like this can be applied to a number of different problems. You'll lean on this dataset later, when you start clustering customers based on their demographic information.

4. Load the final dataset.

- a) Scroll down and view the cell titled **Load the final dataset**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_data_final.info()
```

Now it's time to load all of the main user data you just preprocessed into a file. First you'll do a spot check of the data to ensure the columns all have the correct data type and number of values.

- c) Run the code cell.

- d) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45179 entries, 0 to 45215
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          45179 non-null   object  
 1   number_transactions 45179 non-null   float64 
 2   total_amount_usd   45179 non-null   float64 
 3   job_management    45179 non-null   int64  
 4   job_technician    45179 non-null   int64  
 5   job_entrepreneur  45179 non-null   int64  
 6   job_blue-collar   45179 non-null   int64  
 7   job_retired       45179 non-null   int64  
 8   job_admin.        45179 non-null   int64  
 9   job_services      45179 non-null   int64  
 10  job_self-employed 45179 non-null   int64  
 11  job_unemployed   45179 non-null   int64  
 12  job_housemaid    45179 non-null   int64  
 13  job_student       45179 non-null   int64  
 14  education_tertiary 45179 non-null   int64  
 15  education_secondary 45179 non-null   int64  
 16  education_Unknown  45179 non-null   int64  
 17  education_primary  45179 non-null   int64
```

All columns apart from `user_id` are either Booleans, integers, or floats. And, all columns have 45,179 values—so no more data is missing.

- e) Select the next code cell, then type the following:

```
1 users_data_final.to_pickle('users_data_final.pickle')
```

You'll save the data as a pickle file to make it easier to import back into Jupyter Notebook.

- f) Run the code cell.

5. Load the demographics dataset with PCA applied.

- Scroll down and view the cell titled **Load the demographics dataset with PCA applied**, then select the code cell below it.
- In the code cell, type the following:

```
1 reduced_df.to_pickle('users_data_demo_pca.pickle')
```

You'll save the demographics data with PCA applied as its own file so that it can be retrieved separately when you need it. For the most part, you'll be using the `users_data_final.pickle` file to build your models.

- c) Run the code cell.

6. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Analyzing Data** tab in Firefox, but keep a tab open to **CDSP/Analysis/** in the file hierarchy.

Guidelines for Preprocessing Data

Follow these guidelines when preprocessing data.

Preprocess Data

When preprocessing data:

- Automate as much of the process as possible.
- Create ad hoc scripts early in the research to more easily verify with different datasets the design decisions that were made.
- Rerun models and compare with prior executions of the same data to ensure that no code has changed.
- Consider using a single integrated platform for initial investigations before moving different parts to best-of-breed solutions.
- Allocate a generous amount of time when automating a manual process, especially when a number of different tools or data science platforms are being used.
- Consider implementing diagnostic and error reporting for automated processes.
- Recognize that feature engineering decisions will need to be revisited when results are analyzed or new data is received.
- Review data retention policies and identify how long data will need to be stored for verification or modeling purposes.
- Store versions or snapshots of code, programs, workbooks, or workbenches used in the modeling process.

ACTIVITY 3–15

Comparing Data Preprocessing Techniques

Scenario

You performed several different preprocessing techniques on your data to prepare it for machine learning. Now you'll compare those techniques to determine their relative strengths and weaknesses.

1. When it comes to handling missing values, how do imputation techniques compare to dropping the data entirely or filling the data in with an arbitrary value?
 2. How do feature scaling techniques like normalization, standardization, log, cube root, and Box–Cox compare?
 3. How do encoding schemes like label encoding, one-hot encoding, and binary encoding compare?
 4. When it comes to discretizing variables, how does equal-width binning compare to equal-frequency binning?
-

Summary

In this lesson, you performed exploratory data analysis (EDA) on your datasets using summary statistics and different types of visualizations. You also used that analysis to guide the preprocessing of your data, preparing it for the machine learning tasks to come. By analyzing your data, you become more familiar with how it can be used to achieve the project's goals.

What types of target variables do you think you'll be most interested in exploring in your own projects?

What kinds of data visualizations do you think you'll create most often on the job?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

4

Designing a Machine Learning Approach

Lesson Time: 2 hours

Lesson Introduction

Now that your data has undergone a great deal of preparation, it's time to start looking at developing machine learning models. These models will be instrumental in achieving your business objectives because they can intelligently estimate much about the world. But before you start building these models, you need to have a firm grasp on what goes into machine learning and what it means to use machine learning to test a hypothesis.

Lesson Objectives

In this lesson, you will:

- Identify the basic concepts that make up machine learning.
- Use a design of experiments approach to test a model hypothesis.

TOPIC A

Identify Machine Learning Concepts

To lay the groundwork for machine learning, you need to be able to identify its key concepts. You'll gain a deeper understanding of those concepts in this topic.

Machine Learning

Machine learning is a discipline of **artificial intelligence (AI)** and can also be considered a major component of data science. In machine learning, computers make predictions and other decisions based on sets of data, and do so without any explicit instructions provided by a human. This differentiates machine learning techniques from traditional software solutions that are fully programmed to perform some intended function. Independence from human intervention enables machine learning techniques to automate decision-making processes in ways that are faster and more efficient than any traditional software solution. In addition, the other major strength of machine learning is in the second word of its name—its ability to *learn*. In other words, machine learning processes build on themselves over time through experience, becoming more and more effective at solving a problem or set of problems.

The most fundamental concept behind machine learning is the algorithm. An **algorithm** is a set of rules for solving problems. At its core, an algorithm is a mathematical formula or group of formulas that take some input and then output some result. Algorithms also support the "learning" aspect of machine learning since they are designed to update the beliefs and assumptions about the data they're working with over time.

So, how does machine learning relate to data science? In the business world, machine learning is the most common mechanism by which data science practitioners generate automated decision-making and predictive capabilities that fulfill some business need(s). While practices like ETL and data analysis can be done for their own sake, or for non-machine learning purposes, they are typically done as preparation for machine learning. Ultimately, data science encompasses more than just machine learning, but machine learning is a significant part of the data science process. You might spend more time on the earlier stages of the process than on machine learning, but it's still an important concept to understand and skill to have.

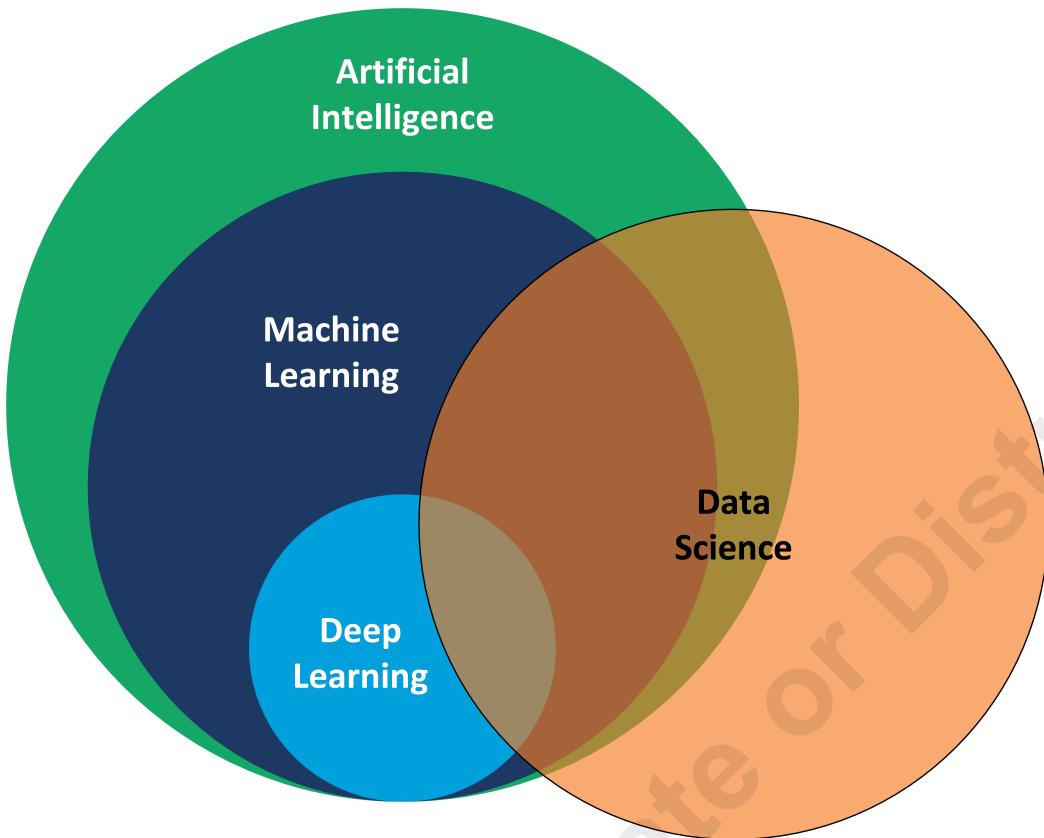


Figure 4-1: A Venn diagram that approximates the relationship between AI, ML, and data science.

Machine Learning Models

A **machine learning model**, usually just called a model, is a mathematical representation of data applied to an algorithm. It is similar to the idea of a traditional statistical model in that it puts forth assumptions about a population based on available sample data. Machine learning models differ in that they are specifically built upon machine learning algorithms (i.e., algorithms that can induce learning), and that they are always designed to output an estimation. The machine learning model is often the ultimate "product" of the machine learning process—it is the thing that you create to actually perform the necessary task, whatever that may be.

The general process for creating and using a model is as follows:

1. The practitioner selects an algorithm to use.
2. The practitioner (or an automated process) feeds data into the algorithm. This is called the *training* data, or training sample.
3. The algorithm outputs a model based on the training data.
4. The practitioner (or an automated process) feeds the model new data that it hasn't seen before.
5. The model makes a prediction or decision about this new data.

So, a model is an implementation of an algorithm. It is specific to whatever problem you are trying to solve, whereas an algorithm can generate many kinds of models if given different data or configured in different ways.

The *training* process in step 3 is of particular importance, as it actually creates the model. This is where the algorithm does all the hard work, taking your input data and performing many calculations on it. You, the practitioner, don't do much at this point, but everything you do leading

up to training (and even things you do after) will have an impact on the success of that training. This is also the point in the process where computational power becomes a factor, as even when run on high-end hardware, certain models can take hours, days, or even weeks to finish training. Preparing to train a machine learning model is therefore crucial to get the results you want out of the process.

Exporting Models

Depending on the environment you use, you may be able to export a model to transfer it into a different environment. For example, if you've trained a model using a cloud service, you may be able to export that model as a file to work with on a local system.

Machine Learning Algorithms

Before you train your initial machine learning model, you'll have to select an algorithm or algorithms that you'll use to produce the outcome you require. For example, if you needed to perform a classification task such as determining whether someone is at risk of contracting a disease based on various inputs—lifestyle factors, age, gender, and so forth—you could use logistic regression, random forest, naïve Bayes, or one of several other algorithms.

The following figure places some example algorithms into the three main machine learning modes (supervised, unsupervised, and reinforcement) and three outcomes/tasks (regression, classification, and clustering). This is not an exhaustive list of algorithms, just some of the most common ones used in the field (aside from neural networks used in *deep learning*, which are beyond the scope of this course).

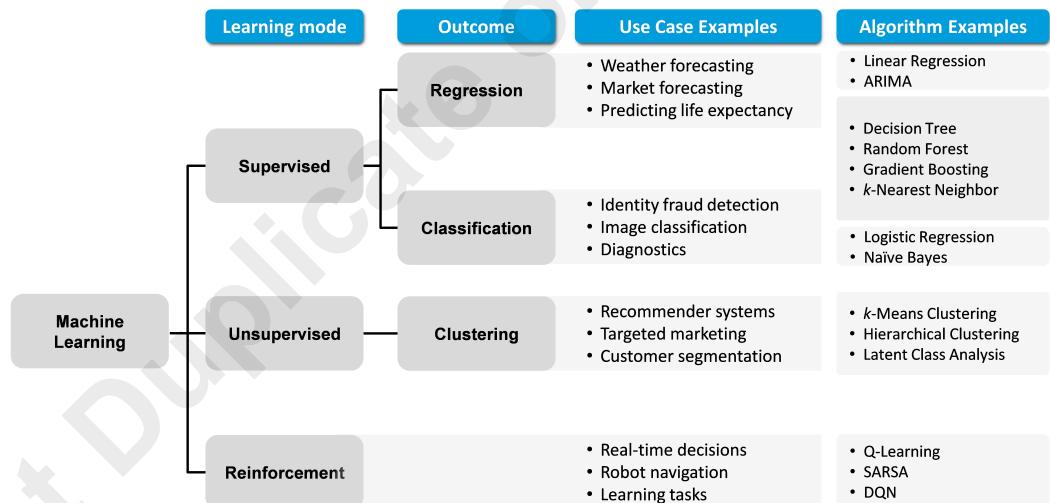


Figure 4–2: Example machine learning algorithms.

Some algorithms, depending on how they are used, can support various types of outcomes. For example, a random forest could be used for various types of classification *or* regression tasks. Also note that some of these terms refer to groups of related algorithms rather than specific algorithms. For example, there are multiple types of decision tree algorithms, but they tend to operate in a similar way and produce the same type of output.

Algorithm Selection

Selecting the right algorithm for a machine learning model can seem overwhelming because there are numerous supervised and unsupervised machine learning algorithms, each one employing its own approach to the learning process. The "right" algorithm depends on the situation, and there may be more than one algorithm that solves a particular problem. Even highly experienced data

science practitioners may not be able to immediately select the best algorithm for a particular situation without some experimentation.

It helps to think about the algorithms you might use even before you are ready to apply them, since it may affect how you need to prepare data earlier in the data science process. Mapping algorithms to the outcomes/tasks that you expect is a good place to start, as you may be able to rule out some algorithms that don't apply to your situation. For example, logistic regression is primarily used for classification tasks, so you'll probably decide not to train a logistic regression model if you just need to forecast sales figures. However, even if the desired task is clear, there may still be many suitable algorithms. Why choose logistic regression instead of a decision tree, for instance? What about using a naïve Bayes model instead?

There are some factors you can look for that differ among algorithms that perform the same basic type of task, even when you use the same datasets. These factors include:

- **Training speed.** Some algorithms take much longer to train a model than others even when you use the same dataset.
- **Model effectiveness.** Some algorithms produce models that perform better than others at their given task.
- **Data preparation requirements.** Some algorithms require more data preparation than others in order to be effective, or they require different kinds of preparation.
- **Complexity.** Some algorithms, especially those in deep learning, can be much more complex and difficult to work with than others.
- **Transparency/explainability/interpretability.** Some algorithms are more closed off or "black box" than others, meaning it is harder to determine why the model made a certain decision.
- **Availability.** Each programming library supports a selection of algorithms, and the library you use may not support the algorithm you're looking for.

Of course, if you have the capacity and the ability, you should consider training multiple models using different algorithms. That way you'll be able to compare each model and choose the one that suits your needs the best.

Iterative Tuning

As you work with machine learning models, you must constantly attend to various challenges that will lead to ineffective and faulty results. Or, results that don't quite measure up to your expectations. After all, machine learning is probabilistic. So a typical approach to improving the effectiveness of a model is to use different samples for training and testing, to reflect the types of challenges the model will encounter when used with real data. As you test the model over multiple iterations, you'll make refinements, and test it again.

In statistics, it is often said that "all models are wrong—but some are *useful*." The point is that statistical models will always contain some degree of error, due to variations based on probability. In machine learning, the goal is to produce a model that is correct *enough* as to be useful, even though the predictions or decisions may not be exactly correct or always correct. A model that is useful for its intended task is commonly described as *skillful*. There are degrees of skill; some models are more useful than others. Improving a model's skill is the ultimate goal of the iterative tuning process.

There are many ways to tune a machine learning model, several of which are discussed throughout the rest of this course. Most of them involve training and retraining the model so that its skill gradually increases until meeting the threshold that satisfies your requirements. Likewise, there are many ways to measure model skill that you'll learn about. The key takeaway at this point is understanding the overall tuning process, which tends to follow this pattern:

- Train the model.
- Test the model's basic purpose (e.g., its ability to predict something).
- Adjust some aspects of the model.
- Evaluate the model using various metrics.

- Adjust the model again.
- Evaluate the model again to see if improvements have been made.
- Repeat as many times as necessary.

Compromises

Tuning a model typically involves making compromises, and finding the "sweet spot" that results in a model that meets business requirements. While you may strive for perfection in your job, as a data science practitioner, this may translate to meeting business requirements and not getting perfect scores for your machine learning models.

For example, you might compromise some of the model's skill in favor of:

- **Simplicity of the data pipeline**—With "just a little more data" you might be able to build an even better model. On the other hand, you might end up creating a complex data pipeline that is time consuming, hard to maintain, and way more complicated than you really need. In many cases it may be sufficient to have just enough skill to get actionable insights from the model.
- **Model interpretability**—When humans delegate decisions to machine learning processes, in many cases humans are still accountable to understand and explain the rationale for those decisions. Some models may produce slightly better performance at the expense of model interpretability. In some cases, the additional performance may not be worth the loss of being able to understand and explain decisions made by the model.
- **Resources needed to train and operate a model**—You may be able to gain some performance points by using additional resources, more time, more memory, more CPUs or GPUs, and so forth. But fewer resources may produce adequate performance while saving time and money.
- **Time and process reliability**—It may be possible to push the iterative tuning process further to get even more skill out of your model, but this might be infeasible if it takes too much time or creates issues with dependent processes elsewhere in the pipeline.

Bias and Variance

When improving the skill of machine learning models, it is necessary to reduce error in those models. Two major types of error that can arise in machine learning are bias and variance.

Bias measures the difference between the model's estimations and the actual ground truth (labeled) values. A model with high bias is simplified to make the **target function** less complex, easier to understand, and easier for a machine to learn. In other words, it makes many assumptions about the target function. Some algorithms, such as linear regression, tend to err on the side of high bias. High bias models tend to have lower estimative performance on complex problems that fail to meet the simplifying assumptions they have made.

Variance, which was defined earlier as being equal to standard deviation squared, measures the variability of the model's estimations. It refers to the extent to which the machine learning algorithm will adapt to a new set of data. A high variance model will be quite complex, perhaps changing its approach significantly from one dataset to another, based on subtle patterns and relationships among the inputs. This will make it very adaptable to different datasets, but it also adds complexity. Generally, algorithms like decision trees are higher in variance than algorithms like linear regression.

Types of Bias

There are some common sources of bias, including:

- **Selection bias** occurs when the training dataset doesn't truly represent the population the model will ultimately be applied to. For example, suppose you are developing a model to predict heating costs for a building. Your selection of training data might bias the results if it covers only two months of a year, or if it came from a single year that was unusually cold.

- **Reporting bias** occurs when the training data is missing observations that were not reported. For example, some types of crimes may be more likely to be reported than others. Using data with a reporting bias to train a machine learning model may skew the outcome.
- **Attrition bias** occurs when the training data excludes participants that dropped out over time. For example, a model might be developed to predict the success rate of a particular treatment over time. Some participants might drop out of treatment, and a well-intentioned data science practitioner might remove their data from the training dataset under the rationale that the outcomes for dropouts would skew results. However, the fact that some participants will not complete the treatment program is an important part of the predictions the model should make.

Model Generalization

When you train a machine learning model, you'll typically want it to make a reasonably good estimation on any new datasets that it might encounter—beyond the dataset that was originally used to train it. This characteristic is called **generalization**. A model that generalizes well is effective beyond the original conditions it was trained under, and therefore can be applied across a spectrum of problems and still be relatively useful. Conversely, a model that does not generalize well is only useful in very limited circumstances.

Both bias and variance relate directly to the concept of generalization. A high amount of either will directly impact the model's ability to generalize.

- **High bias** leads to **underfitting**. An underfit model is too simple to be useful to new data, as it cannot derive pertinent information from the dataset. It simply cannot learn the appropriate patterns from which to make an effective estimation.
- **High variance** leads to **overfitting**. An overfit model is too complex and maps to the training data too tightly. It therefore will perform poorly on new, unseen datasets since it has failed to separate the signal from the noise and learn the true estimation pattern.

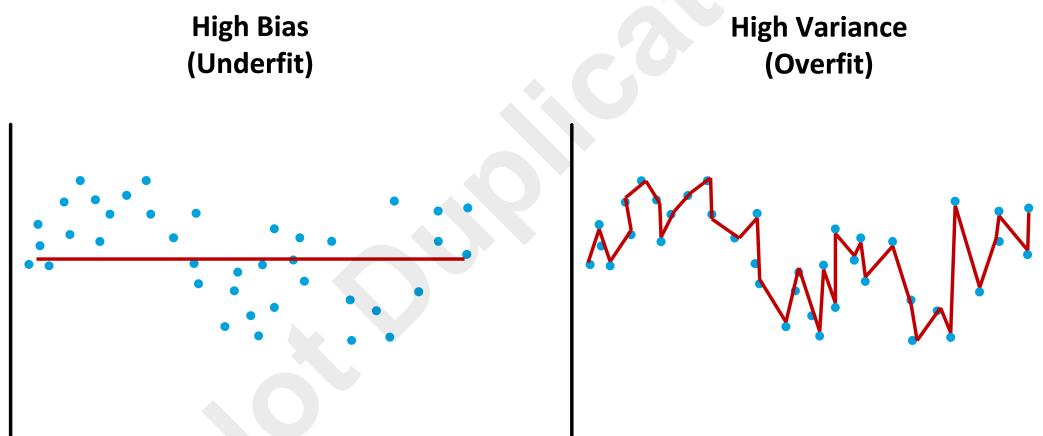


Figure 4-3: A model that underfits training data vs. a model that overfits that same data. Neither model is able to find the curve that is the true pattern.

Underfitting can usually be avoided by choosing more complex algorithms, increasing the number of features, and reducing the effect of techniques like regularization. Overfitting is usually more common, and there are several techniques, including cross-validation and regularization, that can reduce its effects. Adding more data examples and reducing the number of features (among other data preparation methods) can also help your model avoid overfitting.

The Bias–Variance Tradeoff

When you develop a machine learning model, your goal will be to find the "sweet spot" that makes a good compromise between bias and variance. The model may err on the side of high bias or high variance, and these two types of errors are inversely related. In other words, as one increases, the other tends to decrease. The sweet spot is the point where the smallest number of total errors are produced.

Even when you find the sweet spot, a certain amount of error, called **irreducible error**, will always remain. This amount of error cannot be reduced further, due to the way the problem is framed and variables that you never identified or chose not to deal with.

The balance between bias and variance is typically found through experimentation. You configure a machine learning model with controls that you can adjust. By repeatedly running the model, and adjusting these external controls, you can find the best configuration that produces a good balance.

High Bias	The Sweet Spot	High Variance
May <i>underfit</i> the training set	<i>Good enough fit</i>	May <i>overfit</i> the training set
More simplistic	Just complex enough	More complex
Less likely to be influenced by true relationships between features and target outputs	Skillful in finding true relationships between features and target outputs while not overly influenced by noise	More likely to be influenced by false relationships between features and target outputs ("noise")

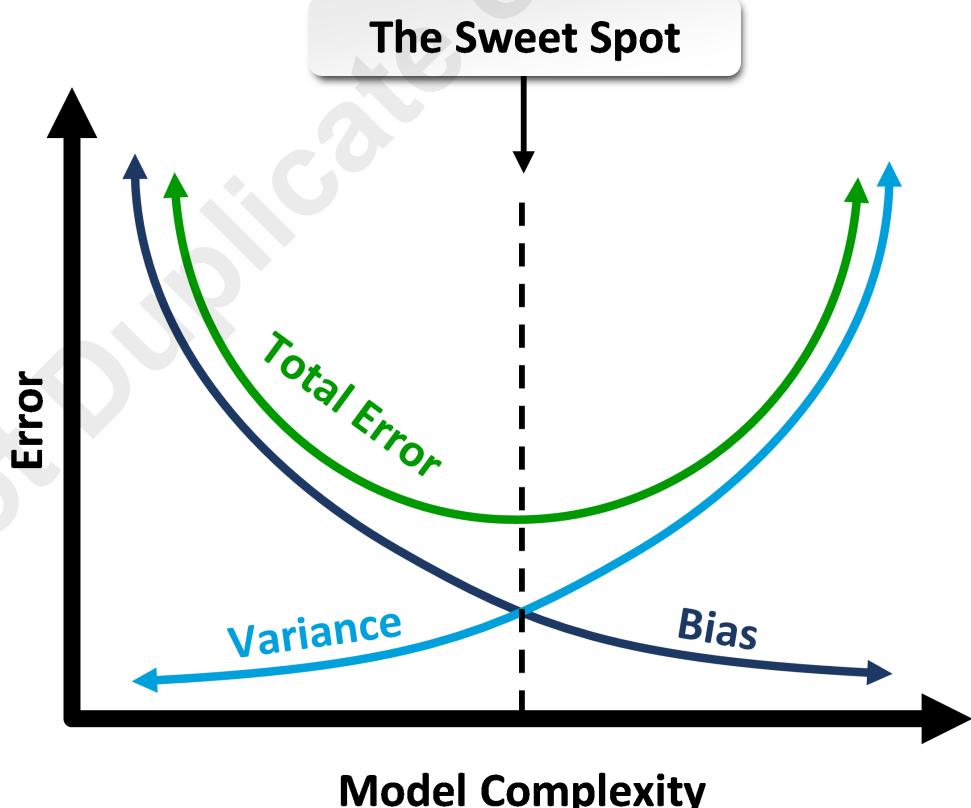


Figure 4–4: Finding a bias-variance balance that doesn't overfit or underfit the model.

Holdout Method

The **holdout** method involves splitting up the original dataset so that you have multiple sets. It is essentially a process of sampling the data, and one of the most basic techniques in addressing the problem of overfitting. You create two or three of these subsets, where each subset is used for a different purpose.

- **Training set:** Data you use to train the model. In supervised machine learning, the learning algorithm operates on the training set that includes target labels for the ground truth.
- **Validation set:** The model doesn't learn from this dataset. You use it to evaluate how well the model can perform on a new dataset that it wasn't trained on. For example, based on the model's performance on the validation set, you may determine that you need to tune some settings of the algorithm. You will be familiar with the data in the validation set, and you will have access to the labels for it. This particular set is optional.
- **Test set:** This is another testing set with data that the model hasn't seen during training. You should not use it to tune performance (like you would use a validation set), but rather only for testing the final fit of the model.

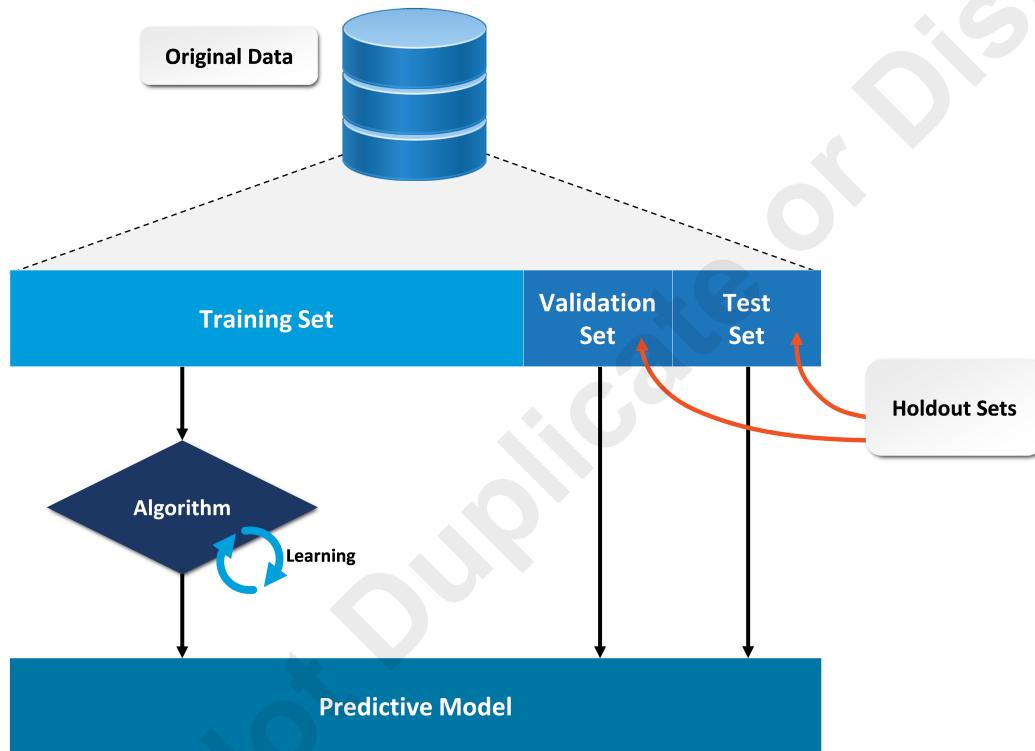


Figure 4–5: Using the holdout method to create a validation and testing set.

A validation set may not be needed in every situation. It's generally only used in situations where there are settings for the learning algorithm (hyperparameters) that you can adjust, since performance tuning is the primary reason for using a validation set. If you use a validation set, typical percentages for splitting the data are 60% training, 20% validation, and 20% testing. If you just use a training and testing set, typical splits are between 70–80% training and 20–30% testing. Either way, you can see that the majority of data is typically used for training, in order to produce a better model. The actual numbers you use depend on the situation, and you may use your judgment to adjust the split amounts to produce the best result.

Cross-Validation

Cross-validation (also called rotation estimation or out-of-sample testing) is a technique for partitioning data in order to improve a model's ability to generalize to new data. There are actually several cross-validation techniques, the most basic of which is the holdout method.

Cross-Validation Method	Description
<i>k-fold cross-validation</i>	<p>The data is split into k groups (folds). One group is the test set, and the remaining groups form the training set. The model trains and then evaluates its performance. Then, the groups rotate: a different group is designated as the test set, and the rest are used in the training set. Once again, the model trains and evaluates its performance. This process repeats k times. Then, the average error across all of these trials is calculated.</p> <p>The advantage of this approach is that it minimizes variance, as every data point is used to both train and test at some point. However, because the training and testing must be done k times, this method adds overhead to both time and processing power. A practical rule of thumb is to set k between 5 and 10.</p>
<i>stratified k-fold cross-validation</i>	<p>Similar to k-fold cross-validation, this alternative helps to minimize variance and bias issues by ensuring that each train/test fold is a good representation of the data as a whole. In binary classification, if 30% of the data is in class 0 and 70% is in class 1, then 30% of the data in each fold will be in class 0, and 70% will be in class 1.</p> <p>The stratification method is therefore most suitable in cases of class imbalance.</p>
<i>Leave-p-out cross-validation (LPOCV)</i>	<p>The k-fold method, but with k equal to all data points in the set (n). So, $n - p$ data points are used in training, and p data points are used to test. This is repeated for all possible combinations of data that fit this split. Like with k-fold, the average error across these trials is then calculated.</p>
<i>Leave-one-out cross-validation (LOOCV)</i>	<p>The same as LPOCV when p is set to 1. It's common to do this because as p increases, the amount of trial combinations increases dramatically, leading to significant performance issues.</p> <p>LOOCV is commonly used to minimize bias in smaller datasets, though it may not perform well with larger datasets due to increased variance and performance concerns.</p>

Parameters

The mathematics of machine learning involves the use of parameters, or configurable values that are of importance to the machine learning process in multiple ways. In machine learning, parameters can be divided into two groups: model parameters and hyperparameters.

A **model parameter** is a parameter that is derived from the machine learning model as it undergoes the training process. In other words, model parameters are what is actually "learned" by the model

while it performs calculations on the training data. These parameters therefore determine how well the model makes predictions and other intelligent decisions. As they are derived from the training process, model parameters are typically not configured by the machine learning practitioner, but by the algorithms and mathematical functions that comprise the machine learning process. One example of a model parameter is a coefficient of an independent variable for linear regression models; i.e., the model parameter is what the variable is multiplied by.

A **hyperparameter** is a parameter that is set on the algorithm itself and not the learning model. This means that a hyperparameter is provided *before* training, typically by the machine learning practitioner. The practitioner tunes hyperparameters so that the eventual model will be better at estimating the model parameters, thereby improving the model's learning performance. For example, in a random forest, the maximum number of decision trees in the forest is a hyperparameter and must be configured before training.



Note: Model parameters are said to be *internal* to the model, whereas hyperparameters are *external* to the model.

Parametric vs. Non-Parametric Algorithms

The algorithms that generate machine learning models can be defined as parametric or non-parametric.

- **Parametric** algorithms generate a fixed number of model parameters. No matter how much data you input to the algorithm for training, it will always produce a model with the same number of model parameters. Algorithms like linear regression and logistic regression are parametric.
- **Non-parametric** algorithms can generate a potentially infinite number of model parameters. The more data you input to the algorithm, the more model parameters it can generate. Algorithms like decision trees are non-parametric.

Guidelines for Training Machine Learning Models



Note: All Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Follow these guidelines when training machine learning models.

Train Machine Learning Models

When training machine learning models:

- **Select the right algorithm for the job.** One machine learning algorithm may be more effective at accomplishing a specific task than another. Or, consider selecting multiple algorithms and comparing the results to choose the right one.
- **Avoid unnecessary complexity.** In general, you should use the simplest or most computationally efficient method that meets your requirements.
- **Prioritize model generalization.** It's important for a model to be able to generalize well to new data. This usually means paying attention to factors like bias and variance and how they can restrict a model's ability to generalize.
- **Learn through experimentation.** Continue to tune your models by modifying factors like the content of datasets used and training and hyperparameters that apply to the relevant algorithms.
- **Select the right hardware.** Make sure the systems you use are appropriate for the tasks you need to accomplish, and they are appropriately configured.
- **Set realistic performance requirements.** You may not need the most skillful model that time and money can produce. Set your performance requirements for the model to align with business requirements. In some cases, you may not need a very high-performing model to get the actionable insights you require.

- **Improve the structure of your datasets.** Some algorithms take much longer to process datasets when data items have high standard deviations or columns are in vastly different ranges. Optimize datasets as needed for the types of tasks you need to perform on them.

Do Not Duplicate or Distribute

ACTIVITY 4-1

Identifying Machine Learning Concepts

Scenario

You've worked quite a bit with Greene City National Bank (GCNB) data, and should have a good sense of the data's format, structure, and characteristics. You also analyzed the data and were able to draw some conclusions about what the data suggests, like how certain variables correlate with others (e.g., call duration in a marketing campaign having some correlation with whether or not the user signed up for a term deposit), and how certain categorical variables imply degrees of representation (e.g., what type of device is most common to use when banking online).

Before you can start training machine learning models, you need to make sure you're up to speed on the fundamentals. You'll use this opportunity to think about how machine learning can support the project's goals.

1. **How might machine learning go above and beyond the ETL and analysis you've done so far to better fulfill the project's goals?**

2. **What factors do you need to consider when selecting machine learning algorithms to fulfill the project's goals?**

3. **How might you ensure that your model is "skillful"; i.e., effective at performing its given task?**

4. Your organization, Rudison Technologies, has a relatively robust set of computing hardware available for use during the project. However, these resources are still limited.

How might this influence the model training process?

5. Your colleagues have expressed concerns about how well the project's machine learning models will generalize to new data.

Why is a model's ability to generalize to new data so important?

6. How does cross-validation help improve a model's ability to generalize?
-

TOPIC B

Test a Hypothesis

You can think of your machine learning models as putting forth something akin to a scientific hypothesis—a hypothesis that must be tested for it to be acceptable.

Hypothesis

In scientific experiments, the hypothesis is a provisional idea or educated guess that requires additional investigation, experimentation, or evaluation to be proven true or false. In machine learning, the *hypothesis* is a candidate machine learning model that you create to test its performance, particularly whether it is able to produce the outcome that you require. You develop the hypothesis to employ a particular function that you hope will produce the intended outcome. Like any hypothesis, you use it as the basis for additional investigation, experimentation, and evaluation.

Through experimentation, you improve upon the hypothesis. You provide the hypothesis with a *sample* of historical data to train it, which produces a model. You then test the model's performance—for example, trying it out on a new dataset and evaluating the effectiveness of its estimations based on that data.

When you have refined the model through experimentation and have a result that can serve as your finished solution, you have essentially produced a *target function*. A target function maps *independent variables* (variables you can directly change, often called input variables or predictor variables) to *dependent variables* (variables that change indirectly, also called output variables or response variables) in a way that best meets your needs and expectations. You can use the target function to find output data (answers) for inputs to real problems.



Note: Remember that models used in data science and machine learning are essentially modeling beliefs or assumptions about a system. This will always lead to some level of uncertainty, which is why experimenting with a hypothesis is so important.

Design of Experiments

Experimentation is an important part of the data science practitioner's job. The kinds of problems addressed through machine learning are often too complex to be solved through the analysis methods typically used in traditional computer programming. Fortunately, you don't have to get it right the first time. With machine learning, you may produce a solution through a process of systematic experimentation.

Following the *design of experiments (DOE)* approach (DOX or *experimental design*)—an approach used by data analysts, medical researchers, and others—you begin with a hypothesis, and then you systematically change the variables that *you can control* to see their impact on the variables you *can't directly control*.

For example, through this type of experimentation, you might:

- **Determine which combination of independent variables will produce the best model for your needs.** You can experiment to see how training the algorithm using different combinations of input variables affects the performance of the model when it is applied to the evaluation data.
- **Select the best machine learning algorithm for your needs.** Use experimentation to compare the performance of different machine learning algorithms to see the impact on the estimative skill of the model.

- **Adjust settings of the learning algorithm to optimize its performance.** You can experiment to see how you can adjust various hyperparameters to tune the performance of the model, and choose hyperparameters that produce the best model for your needs.

Hypothesis Testing

An important part of your role as a data science practitioner is to select the best model for a given task. For example, when you apply two machine learning methods to predict outcomes on a dataset, you have to choose one model over the other, presumably the model whose effectiveness best meets your requirements. In the case of a prediction model, this probably means the model with the best *estimated* effectiveness when making predictions on new data. The problem is, the effectiveness estimate is just that—an estimate. The difference in estimated effectiveness could be real, or it might be due to statistical chance. Fortunately, statistical hypothesis testing can help you determine which one is actually better in practice.

Hypothesis testing focuses on the **null hypothesis**, which is the assumption that there is no statistically significant (i.e., real) difference between the models under comparison. So there are two outcomes of such a test:

- **Insufficient evidence to reject the null hypothesis.** In other words, observed differences in model effectiveness are likely due to statistical chance.
- **Sufficient evidence to reject the null hypothesis.** In other words, observed differences in model effectiveness are likely due to a difference in the models.

Consider how you might train two models on the same data, where one model has undergone additional dimensionality reduction to remove a feature that appears to be noisy. The reduced model may end up giving you a better result (whatever specific result you may be looking for). Due to the probabilistic nature of machine learning algorithms, it's entirely possible that the "improvement" in the reduced model is due to random chance. You must therefore test the models to verify this. If it turns out that the change in results was due to *real* differences, you can reject the null hypothesis. Otherwise, you cannot—and if you can't reject the null hypothesis for the new model, you can't be confident that the model is truly more effective.

You can also test a hypothesis in which one model is evaluated by itself. In other words, if you train one model to make a prediction with some degree of effectiveness, the null hypothesis would state that this effectiveness is no different than if you had made random predictions. Think of a scientific experiment in which a medical treatment is tested by assigning the proposed treatment to one group, and a placebo to a control group. If both groups end up with the same number of people who are successfully treated, then you have failed to reject the null hypothesis.



Note: It's important to understand that you're not *accepting* the null hypothesis; you are merely failing to reject it. You cannot say with absolute certainty that the null hypothesis is true, so it's not correct to say you accepted it.

Hypothesis Notation

You can represent hypotheses using the notation H_0 and H_a , where the former is the null hypothesis and the latter is the alternative hypothesis (i.e., your hypothesis that contradicts the null hypothesis). Let's say your hypothesis is that your organization will hire 12 employees in a given year. You could write the hypotheses as:

$$\begin{aligned} H_0 &: \text{new_hires} < 12 \\ H_a &: \text{new_hires} \geq 12 \end{aligned}$$

Type I and Type II Errors

Because hypothesis testing is done on sample data and not an entire population, the tests are susceptible to errors. These errors are classified as follows:

- **Type I**—You rejected the null hypothesis, but the null hypothesis was actually true.

- **Type II**—You failed to reject the null hypothesis, but the null hypothesis was actually false.

A/B Tests

There are many methods that you can use to conduct a hypothesis test, some of which are more relevant than others when applied to certain problems. One of the most common hypothesis testing methods is an **A/B test**. An A/B test compares two different values of the same variable in order to determine which value is most effective. A common use case is serving online customers two different versions of the same web page, and then choosing the page that leads to the best conversion. Group A would see the new version of the web page, whereas Group B would act as a control group and only see the original web page. So, your hypothesis might be that adding a specific UI widget increases conversion rates, whereas the null hypothesis would state that this widget has no real effect on conversion rates.

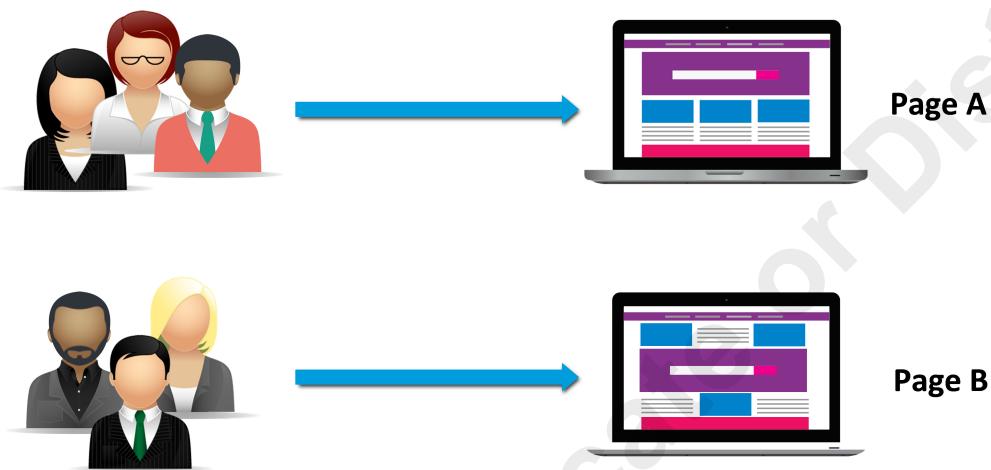


Figure 4–6: An A/B test in which two slightly different web pages are shown to two groups of users.

More A/B Test Use Cases

Besides the common example of changing a web page, some additional use cases for A/B tests include:

- Changing the contents of marketing emails, including the subject line, body, and signature. Some recipients will get one type of email, some will get another type.
- Changing the layout of a physical store. One branch may place its electronics section closer to the front registers, whereas another branch may place its sporting goods section closer to the front.
- Changing the layout of a fast food restaurant's drive-thru menu. One branch may have its menu place chicken sandwiches front and center, and another branch may have french fries front and center.
- Changing the visual design of a book cover. One cover variant may sell in one region, whereas another variant sells in a different region.

A/B Tests and Machine Learning

A/B tests have been around for a while and don't require machine learning. You can simply show half your users Web Page A and the other half Web Page B, calculate whatever metric you're using to evaluate the page's effectiveness (e.g., total length of time spent on page; average conversions, like

filling out a form or purchasing a product; average user ratings; etc.), then find out which web page did the best. However, the real world isn't so simple. There are many complex factors that can contribute to a web page's effectiveness, including:

- **User demographics.** Factors like age, gender, culture, political beliefs, etc., all influence a user's behavior in many aspect of life—web page browsing included.
- **Access paths.** Different users find or access your web page in different ways. Some may come from links on social media, others may have used the results of a search engine. One page may be more effective than another when targeting these paths.
- **Multiple success metrics.** As mentioned, you could just select a single metric to gauge the web page's effectiveness. But you'll almost always obtain more useful results by considering these metrics in combination. Instead of just focusing on total time spent on the page, you could additionally focus on conversions and user satisfaction scores.

Hopefully you're starting to see why a traditional approach to A/B testing isn't the most effective. Enter machine learning. If you collect all of this information, you could compile it into a training dataset and input that data into one or more algorithms to generate machine learning models. These models could be useful in several ways.

For example, you could train a model to identify the features that contribute the most to one or more target metrics. This would help the business focus its efforts on those features. Perhaps age and country of origin contribute the most to purchase conversions on Web Page A, but not Web Page B. So, the web development team might choose not to display Web Page B to anyone whose age group and country of origin is known to the system.

Or, you might create a model that identifies a specific age range and specific country of origin as performing well on Web Page A, but not Web Page B. So, the web development team might tailor the user experience so that people with these demographics are only ever shown Web Page A, whereas others are shown Web Page B (or a mix of A and B).

There are many more ways such an A/B test can help solve business problems. The point is that machine learning can drastically improve the effectiveness of such tests since it does a good job of handling the complexities of the business world (and the world at large).

Additional Hypothesis Testing Methods

Besides A/B tests, there are several other types of hypothesis tests you can conduct alongside machine learning.

Testing Method	Description
<i>z-test</i>	A z-test is used to compare the mean of two distributions when the standard deviation of a population is known. For example, among a population of bank customers, the mean credit score is 700. If you selected a sample of 100 customers from this population, the mean of this sample might be 712. You'd then compare this sample mean to the mean of some other random sampling of 100 customers (e.g., customers in a different market) to see if the increase in score is significant. In other words, the null hypothesis is that the selected 100 customers have comparable credit scores to a random sampling of customers. A z-test is performed by calculating the standard deviation of the sample, then using this to calculate the aforementioned z-score. The z-test is most applicable to larger sample sizes, typically above 30.

Testing Method	Description
t-test	A t-test is an alternative to the z-test in that it compares the mean of two distributions in which the population standard deviation is <i>not</i> known. A t-test estimates the population standard deviation by incorporating the standard deviation of the sample. The t-test is most applicable to smaller sample sizes, typically below 30.
Analysis of variance (ANOVA)	An ANOVA test compares the mean of multiple distributions. In the standard approach, an ANOVA test evaluates the effect that a single independent variable has on three or more sample groups. For example, among a population of crops, you can test the effect of a specific type of insecticide. The null hypothesis would state that the specific insecticide has the same effect on one random sampling as it does on multiple other random samplings. ANOVA as a single test is more useful than conducting multiple t-tests for each sample, as ANOVA considers the variation within and between all of the samples. This minimizes the chance of errors.
Chi-squared test	A chi-squared (χ^2) test compares the effect of categorical variables. For example, you might categorize animals as being either mammals or not mammals. You also categorize animals as having fur or not having fur. A chi-squared test attempts to answer the question, "Does the presence of fur affect whether or not an animal is a mammal?" If yes, then the two categorical variables are said to be dependent. The null hypothesis of such a test is that the variables are independent—i.e., one variable has no significant effect on the other.

p-value

The results of a hypothesis test should help you reject or fail to reject the null hypothesis, all while avoiding type I and type II errors as much as possible. One common testing output is a **p-value**. The p-value is the probability of obtaining a result from the test given that the null hypothesis is true. In other words, the p-value enables you to determine whether or not to reject or fail to reject the null hypothesis if the value is lower or higher than a specified level of statistical significance. This level of significance, also referred to as the alpha value, is something you must determine beforehand. There is no "right" alpha, but it's common to set the alpha at 0.05 (5%). Another way to think about this alpha value is that it sets a level of confidence about the test—in the case of a 5% alpha, this means a 95% level of confidence.

Ultimately, you compare the p-value returned from the test to the alpha value you decided on earlier. This will lead to two possible outcomes:

- If **p-value > alpha**, fail to reject the null hypothesis.
- If **p-value <= alpha**, reject the null hypothesis.

So, if your p-value is 0.03, and you decided on an alpha of 0.05, you'd reject the null hypothesis.

Note that p-value *is not* the probability that the null hypothesis is true or false. Nor does it definitively tell you whether or not the null hypothesis is true or false. It simply provides you with a level of confidence about either rejecting or failing to reject the null hypothesis. It is for this reason that some statisticians have suggested alternative ways to measure confidence.

Alpha and Beta Values

Another way to think about the alpha value is that it is the rate of type I errors. Likewise, there is also a beta value, which is the rate of type II errors. Beta is not as commonly used as alpha, because reducing type I errors is often more important to the practitioner than reducing type II errors. However, you may still want to consider the beta value during hypothesis testing.

Confidence Interval

A **confidence interval** returns a range of plausible values for some unknown variable, most commonly the mean of a population. A related concept is the confidence *level*, which defines what percentage of confidence intervals over multiple random samplings will actually contain the true population mean. As an example, let's say you're taking a survey of people's ages. Ideally, you'd be able to find the mean age of all humans on the planet. Of course, you're limited to only taking random samplings from the entire population. You can use a sample to generate a confidence interval for a certain confidence level—95% being the most common level. The calculation gives you a range of (36.78, 41.22) as the interval. So, this particular confidence interval suggests that the population mean is no less than 36.78 and no greater than 41.22. Then you take another random sample of the population and generate another confidence interval based on that sample. The second interval will probably not be exactly the same as the first. You do this over and over again for as many times as you like. In the end, 95% of your confidence intervals will contain the true population mean, and 5% will not.

Confidence intervals can be confusing because they are often misunderstood to mean that, with a confidence level of 95%, there is a 95% probability that the true population mean lies within the given range of values. This is not necessarily the case. If you were to ask some all-knowing being whether or not your one sample's confidence interval actually contains the population mean, the being would respond either "yes" or "no." The interval either contains the mean or it doesn't; probability no longer factors in. Also, unlike the empirical rule, a confidence interval does not imply that 95% of all values lie within the given range.

Unlike *p*-values, confidence intervals are able to show the likely effects in the population; if a value lies outside the interval, there is good evidence that it does not exist in the population. Confidence intervals and *p*-values are not mutually exclusive, however, and you can certainly choose to use both when deciding whether or not to reject the null hypothesis.

Confidence Interval Visualization

The following figure is a visual representation of how multiple confidence intervals calculated from multiple samples can reveal an approximation of the population mean (μ). The true population mean is not actually known, which is why the vertical line is an approximation based on the confidence intervals. Out of these intervals, 95% will actually contain the population mean. So, if this figure were to plot 100 confidence intervals, 95 of them would bisect the μ line, and 5 would not.

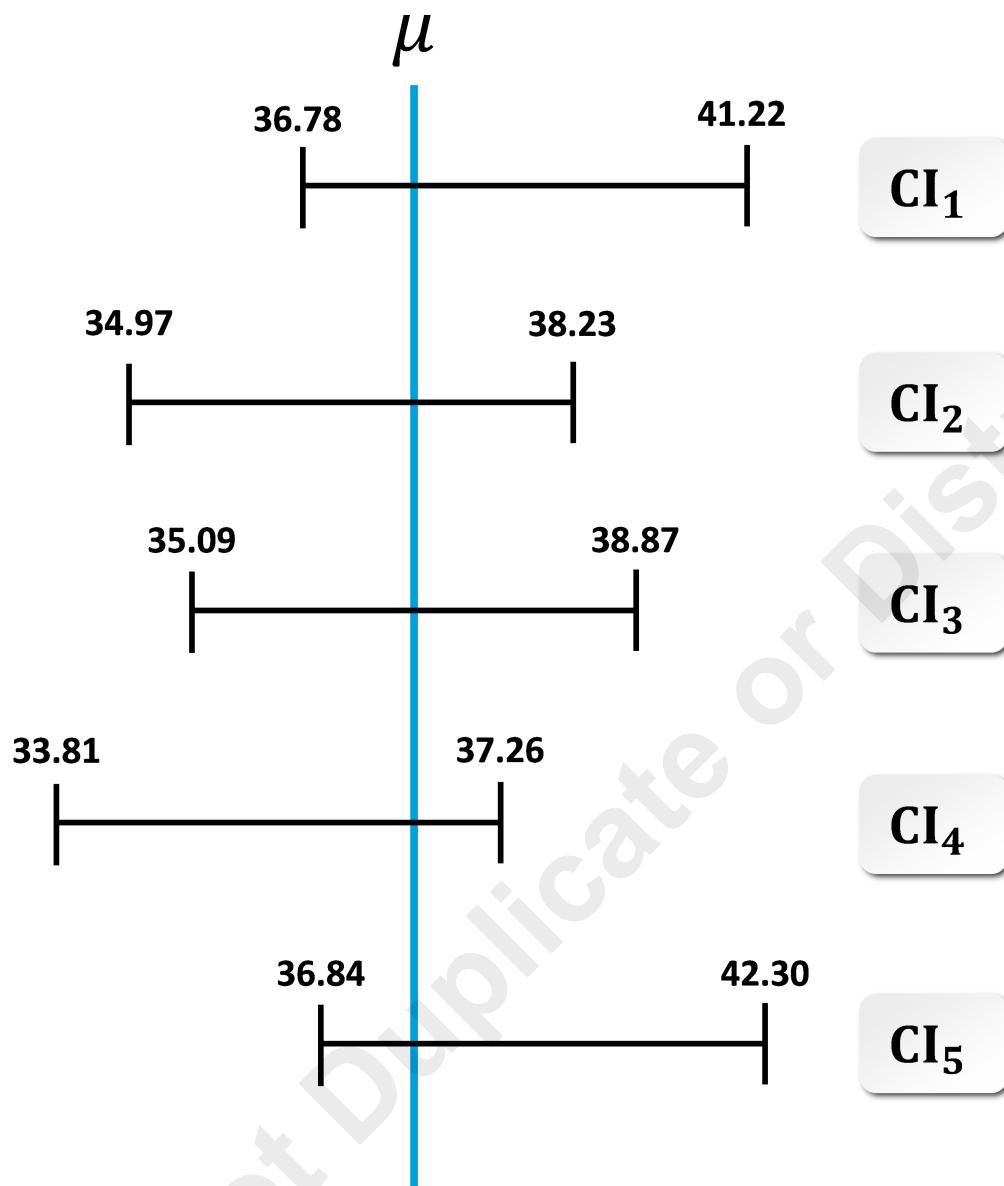


Figure 4-7: A visual representation of confidence intervals for age in a population.

Calculation of Confidence Interval

How you calculate the confidence interval depends on what information you have access to. If the size of the sample is below 30, and you know the population's standard deviation, you can calculate the confidence interval using the z -score. If the sample size is below 30 but you only know the sample's standard deviation, you can use the t -score. When your sample size is above 30, you can just use the z -score and either the standard deviation of the sample or the population. The formula for calculating the confidence intervals using z -score is as follows:

$$\bar{X} \pm z \frac{s}{\sqrt{n}}$$

Where:

- \bar{X} is the mean of the sample set.
- z is the z -score for the confidence level. This score is always the same for a specified confidence level, regardless of the problem. The score for a 95% confidence level is always 1.96.
- s is the standard deviation (in this case, for the sample set).
- n is the number of data examples in the sample.



Note: Dividing the standard deviation by the square root of the sample size is referred to as the *standard error*.

Let's say you have a sample set of 50 people, and the mean age of the people in this sample is 39 years old. You also know that the sample standard deviation is 8. The confidence interval for a confidence level of 95% is therefore **(36.78, 41.22)**. Then, you take new samples of the population and generate new confidence intervals for each. Of those intervals, 95% will contain the true population mean age. This will help you to narrow down the true middle values of the entire population, rather than just one or a few sample sets.

Guidelines for Testing a Hypothesis

Follow these guidelines when testing a hypothesis.

Test a Hypothesis

When testing a hypothesis:

- Consider the accuracy of a model compared to a naïve prediction of all one class. As an example, for an illness predictor you can "predict" that a given individual has no illness and be over 99% accurate, as only a small fraction of a population will have any given illness.
- Keep in mind that the null hypothesis is that nothing has changed in a particular problem set, whereas the alternative hypothesis is that there is some meaningful change.
- Choose a hypothesis testing method that is most applicable to your problem domain and dataset.
- For A/B tests:
 - Focus on testing a hypothesis that is easily measurable.
 - Ensure you have a large enough sample size for the test to be effective.
 - Define a reasonable duration for conducting the test.
 - Ensure your test audience is relevant to the hypothesis being tested.
 - Avoid making changes in the middle of the test.

ACTIVITY 4-2

Testing a Hypothesis

Scenario

As you get closer to training machine learning models, it helps to think of those models in terms of how they put forth a hypothesis. That way, you're not just producing models for the sake of it, but to prove or fail to prove some assertion that directly maps to a business objective and/or data science outcome.

There are many potential hypotheses that you could test, but for now, you'll just focus on one example. In this example, your hypothesis is that a candidate machine learning model will choose the optimal parameters for an email-based marketing campaign for GCNB that will lead to 10% or more of all bank customers signing up for a term deposit. A term deposit is a deposit of funds that is held by the bank for some predetermined period of time, and which accrues interest for the investor at a fixed rate.

1. In the example given, what is the specific null hypothesis? What does this null hypothesis suggest?

2. GCNB's email marketing campaigns consist of many factors, like who the email is sent to, what format the body of the email takes, what kind of content is included in the email, and so on. Some of your colleagues want to collect new data by conducting an A/B test of the marketing emails that GCNB sends out.

What factors might be involved in this A/B test, and how does the test relate to machine learning?

3. The team wants to generate a p -value to help them determine whether or not to reject the null hypothesis. They have selected an alpha (level of statistical significance) of 0.05. The calculation of the p -value results in 0.02.

What does this suggest about the null hypothesis and your alternative hypothesis?

4. The team has also decided to generate confidence intervals from multiple random samplings of GCNB's customer base. In particular, the confidence interval will focus on estimating the mean time in seconds spent reading a marketing email for the entire population. The result is an interval of (145.79, 190.81) using a confidence level of 95%. The team also calculates a 95% confidence level for a different sample and comes up with an interval of (143.12, 189.90).

What does this suggest about how the mean might appear within these confidence intervals?

5. Why is testing a hypothesis through a design of experiments approach so important to the data science process?
-

Summary

In this lesson, you identified the fundamental components that make up the machine learning process, and how your data fits into it. You also recognized the importance of formulating and testing a hypothesis using machine learning techniques. Now you're ready to start developing statistical models that can address many different business needs.

What factors are most important for you when selecting a machine learning algorithm?

What kind of hypothesis testing methods do you think are most applicable to your projects?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

5

Developing Classification Models

Lesson Time: 4 hours, 40 minutes

Lesson Introduction

The first type of machine learning task you'll build models for is classification. Classification has many applications across many different fields, so it's a good starting point. In this lesson, you'll train classification models, tune those models, and then evaluate them as part of a process of iterative improvement.

Lesson Objectives

In this lesson, you will:

- Train models using algorithms that solve classification problems, then tune them to improve performance.
- Evaluate the performance of classification models to inform the tuning process.

TOPIC A

Train and Tune Classification Models

The first step in actually creating a machine learning model is to put it through a round of training. So, in this topic, you'll train models on multiple different classification algorithms to see how they compare. You'll also tune those models by retraining them using different hyperparameters to see if you can get better results.

Binary Classification

Classification is the task of placing data examples into categories, and the simplest type of classification is **binary classification**. In a binary scheme, a data example is determined to either be a 1 or a 0. It either is something, or it is not something. It is on or off. It exists or it doesn't. In other words, there are only two choices.

Binary classification schemes are common in fields like health care, where medical professionals benefit from diagnosing a patient as either having some condition (1) or not having it (0). For example, a binary classifier could diagnose a patient as having heart disease based on a number of features—and the classifier could do so more quickly and/or more accurately than a human doctor. Another area where binary classification excels is in predicting faulty equipment. Certain equipment may show signs of imminent failure, so a binary classifier could determine whether something is about to fail (1) or not (0). There are many such examples of the usefulness of binary classification. There are multiple algorithms that perform binary classification, but you'll start by focusing on just one: logistic regression.

Logistic Regression

Logistic regression is a type of regression in which the output is a classification probability between 0 and 1. It is one of the most common machine learning algorithms used in classification because it can produce results relatively quickly on larger datasets.

The term logistic *regression* can be somewhat confusing. Although technically a type of regression analysis, logistic regression is most often used for classification tasks. When data science and machine learning practitioners use the general term "regression," they are usually referring to the task of estimating the relationships between continuous variables and outputting an estimation that is also a continuous value. For example, predicting the monetary value of a house is a type of regression task—you're not putting anything into categories. So, logistic regression is usually treated as a classification task, since it *does* help you put your data into categories. On the other hand, an algorithm like linear regression is treated as a regression task because it can output a continuous value, like the price of a house.

The value that a logistic regression algorithm outputs is called a **logistic function**. This is a type of *sigmoid function*. Instead of a straight line like in linear regression, the logistic function takes an *S* shape, as shown here:



Figure 5–1: A sigmoid function used in logistic regression.

In the scenario depicted in the graph, you're trying to predict whether or not a customer will return to purchase more products/services within a year. One of the features that determines this is the "satisfaction score" calculated by the customers' answers to survey questions. Notice that function is able to fit the outlier at the top right, and that in doing so, it takes an S shape. If the function were a straight line (as in linear regression), it would not be able to fit this outlier cleanly.

The logistic function can be expressed as the following equation:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Where:

- t is the estimation between negative infinity and positive infinity.
- e is the natural logarithm base.

The estimation value that is output by the logistic function is a value between 0 and 1. The logistic function's goal is to estimate the model parameters—the values that the model "learns" in order to enhance its decision-making capabilities.

Decision Boundary

The division line that separates negative classes and positive classes is the **decision boundary**. This determines what class an example belongs to based on its estimation value between 0 and 1. A decision boundary can be expressed as follows, where \hat{p} is the estimated value and \hat{y} is the classification:

$$\begin{aligned} \text{if } \hat{p} \geq 0.50, \hat{y} &= 1 \\ \text{if } \hat{p} < 0.50, \hat{y} &= 0 \end{aligned}$$



Note: Notice the use of the "hat" symbol above both y and p . This symbol is often used to denote an estimation from a model.

So, if the model calculates an estimation value greater than or equal to 0.50, then it classifies that instance as a 1 (will be a return customer). If the estimation value is less than 0.50, then the model classifies the instance as a 0 (will not be a return customer). When graphed on a sigmoid function, this would look something like the following:

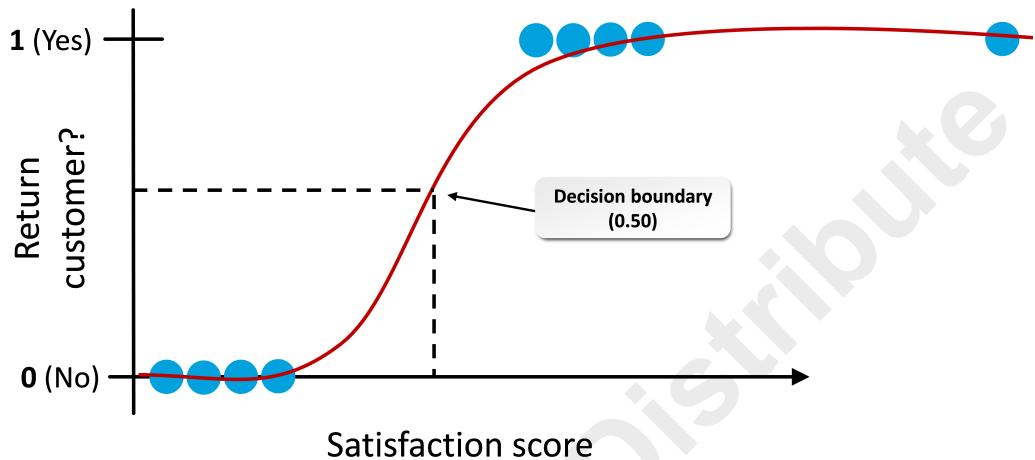


Figure 5-2: A sigmoid function applied to a classification problem. Note that the decision boundary in this case was selected arbitrarily.

The decision boundary on a sigmoid function is able to account for the outlier, improving the algorithm's ability to classify the data examples.

You can configure a different boundary depending on the data and the context of the problem you're trying to solve. You can also analyze and evaluate logistic regression model performance using a technique like ROC curves.

Multi-Label Classification

The logistic regression example given previously works with a simple binary choice—something either *is* x , or is *not* x . Either a customer will return, or they will not. But, you may be faced with a classification problem that involves multiple labels, also called **multi-label classification**. For example, let's say you run a platform that accepts written works from amateur authors and publishes them online. You need multiple ways to classify these works so they're easier to find. In this case, the machine learning model must classify the following:

- Category 1: Whether the work is **fiction** or **non-fiction**.
- Category 2: Whether the work is targeted toward **young children** or **adult** readers.

Assuming your platform uses a rigid classification scheme, all written works are either fiction or non-fiction, and one cannot be both. Likewise, all works are either targeted for young children or for adult readers. While the classes in each category are mutually exclusive, the same is not true *between* categories. You can have a fictional children's book, a fictional adult book, a non-fictional children's book, or a non-fictional adult book. That's four possible classification labels.

Multi-label problems can be solved by training multiple binary classification models, unless the labels themselves are correlated. You could train two models (one for category 1, the other for category 2) and then combine the results.

Multi-Class Classification

Multi-class classification is the process of placing a data example within a single class among three or more choices. This differs from multi-label classification in that the classes are mutually exclusive. An example cannot belong to class A and B and C; it must belong to A *or* B *or* C.

In the example of categorizing written works, you can break down the fictional works into multiple genres:

- Horror
- Fantasy
- Romance
- And so on...

Again, assuming your classification system is rigid, a written work is either in the horror genre or some other non-horror genre; it cannot be both. Unlike with multi-label problems, multi-class problems require a different approach than just combining binary classifiers.



Note: It's also possible to have both a multi-label *and* a multi-class problem. For example, Category 2 in the previous example could include more than just two types of audiences, e.g., children, young adult, adult, etc.

Multinomial Logistic Regression

Multinomial logistic regression, sometimes called softmax regression, is a method for solving multi-class problems. Multinomial logistic regression starts by computing a score represented by $(\mathbf{x})_k$:

$$(\mathbf{x})_k = (\boldsymbol{\theta}^{(k)})^T \cdot \mathbf{x}$$

Where:

- k is the class.
- $\boldsymbol{\theta}$ is the vector of the model parameters.
- \mathbf{x} is the vector of the feature values.



Note: The $\boldsymbol{\theta}$ symbol is the Greek letter theta.

The scores are computed for every class k for vector \mathbf{x} . Then, the scores are plugged into the softmax function. The softmax function calculates the exponent of each score divided by the sum of all exponents:

$$\sigma(\mathbf{x})_k = \frac{\exp(\mathbf{x})_k}{\sum_{j=1}^K \exp(\mathbf{x})_j}$$

Where:

- $\sigma(\mathbf{x})_k$ is the probability that example \mathbf{x} belongs to class k given the score computed earlier.
- K is the total number of classes.

The end result is that, for each data example, the softmax function determines the class with the highest probability, where all probabilities add up to 1.

So, if you were to analyze a written work that mentions dragons and wizards throughout, the softmax function might generate the following probabilities:

- 89% fantasy.
- 8% romance.
- 3% horror.

Thus, the algorithm determines that the story is in the fantasy genre. And, as you can see, all of the percentages add up to 1 (100%).

Guidelines for Training Logistic Regression Models



Note: All Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Follow these guidelines when training logistic regression models.

Train Logistic Regression Models

When training logistic regression models:

- Consider training a logistic regression model to address binary classification tasks.
- Consider using logistic regression on large datasets.
- Consider using logistic regression when avoiding overfitting is a priority.
- Determine if a classification problem may require a data example being placed into two or more classes (multi-label).
 - For multi-label problems, consider training multiple binary classification models, unless the labels are correlated.
- Consider how a classification problem may require a data example being placed into one of three or more possible classes (multi-class).
 - For multi-class problems, consider training a multinomial logistic regression model.

ACTIVITY 5-1

Training a Logistic Regression Model

Data Files

/home/student/CDSP/Classification/Developing Classification Models.ipynb
/home/student/CDSP/Classification/data/users_data_final.pickle

Scenario

The time has come to start building machine learning models that will help GCNB make intelligent decisions about its data and its business operations. One of the potential target variables you and your team identified earlier was the `term_deposit` variable. The more effective a GCNB marketing campaign, the more term deposits the bank can get from its customers, and the more revenue it can generate. Your task is therefore to determine what customers are most likely to sign up for a term deposit, so that the marketing campaigns can prioritize those customers and seek out new customers who have the features that are most important for predicting a term deposit. This is a good task for classification.

Rather than just build one classification model and call it a day, you'll end up building multiple models from various algorithms to determine which one best meets your performance needs. You'll start by training a logistic regression model. Before you train the model, you'll need to do a bit more prep work on the data to make it more conducive to machine learning.

1. Open the notebook.

- In the Jupyter Notebook web client, navigate to the **CDSP/Classification** directory.
- Select **Developing Classification Models.ipynb** to open it.

2. Import the relevant software libraries.

- View the cell titled **Import software libraries**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.

3. Load and preview the data.

- Scroll down and view the cell titled **Load and preview the data**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data = pd.read_pickle('data/users_data_final.pickle')
2
3 users_data.head(n = 5)
```

- Run the code cell.

- d) Examine the output.

	user_id	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_se
0	9231c446-cb16-4b2b-a7f7-ddfb8b25aa6	3.0	2143.00	1	0	0	0	0	0	0
1	bb92765a-08de-4963-b432-496524d39157	0.0	1369.42	0	1	0	0	0	0	0
2	573de577-49ef-42b9-83da-d3cfb817b5c1	2.0	2.00	0	0	1	0	0	0	0
3	d6b6699d-7c8f-4257-a682-e136f640b7e3	0.0	1369.42	0	0	0	1	0	0	0
4	fade0b20-7594-4d9a-84cd-c02f79b1b526	1.0	1.00	0	0	0	0	0	0	0

5 rows × 33 columns

This is the dataset you cleaned and performed preprocessing on earlier. You'll examine the data to make sure it's in a good state to send to a machine learning algorithm.

4. Check the shape of the data.

- a) Scroll down and view the cell titled **Check the shape of the data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_data.shape
```

- c) Run the code cell.
 d) Examine the output.

```
(45179, 33)
```

There are 45,179 records and 33 columns.

5. Check the data types.

- a) Scroll down and view the cell titled **Check the data types**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_data.info()
```

- c) Run the code cell.

- d) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45179 entries, 0 to 45215
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   user_id          45179 non-null   object  
 1   number_transactions 45179 non-null   float64 
 2   total_amount_usd   45179 non-null   float64 
 3   job_management     45179 non-null   int64  
 4   job_technician     45179 non-null   int64  
 5   job_entrepreneur   45179 non-null   int64  
 6   job_blue-collar    45179 non-null   int64  
 7   job_retired         45179 non-null   int64  
 8   job_admin.         45179 non-null   int64  
 9   job_services        45179 non-null   int64  
 10  job_self-employed   45179 non-null   int64  
 11  job_unemployed     45179 non-null   int64  
 12  job_housemaid      45179 non-null   int64  
 13  job_student         45179 non-null   int64  
 14  education_tertiary 45179 non-null   int64  
 15  education_secondary 45179 non-null   int64
```

All of the columns are either floats, integers, or Booleans, except for `user_id`. String objects won't work well with machine learning algorithms, and `user_id` isn't particularly useful, so you'll soon drop it from the dataset.

6. Explore the distribution of the target variable.

- Scroll down and view the cell titled **Explore the distribution of the target variable**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data.term_deposit.value_counts(normalize = True)
```

The target variable is the variable you're interested in learning more about. In this case, you've identified `term_deposit` as a good target variable candidate.

- Run the code cell.
- Examine the output.

```
False    0.883021
True     0.116979
Name: term_deposit, dtype: float64
```

As you can see, most people did not sign up for a term deposit (about 88%), which means that the dataset is imbalanced. You'll soon correct this imbalance through oversampling.

7. Split the data into target and features.

- Scroll down and view the cell titled **Split the data into target and features**, then select the code cell below it.
- In the code cell, type the following:

```
1 target_data = users_data.term_deposit
2 features = users_data.drop(['user_id', 'term_deposit'], axis = 1)
```

You're separating the target variable into its own `DataFrame`, as well as dropping the unnecessary `user_id` column from the main `DataFrame`. This will ensure your models consider `term_deposit` the labeled variable.

- Run the code cell.

8. Split the data into train and test sets.

- Scroll down and view the cell titled **Split the data into train and test sets**, then select the code cell below it.
- In the code cell, type the following:

```
1 X_train, X_test, y_train, y_test = train_test_split(features,
2                                         target_data,
3                                         test_size = 0.3)
```

This code performs the holdout method by dividing each dataset into training and testing sets.

- `X_train` includes all of the features except the label and will be used to train the model. 70% of the rows are in this set, according to the `test_size` parameter.
- `y_train` includes just the label/target variable and will be used to train the model. Once again, 70% of the rows are in this set.
- `X_test` includes all of the features except the label and will be used to test the model. 30% of the rows are in this set.
- `y_test` includes just the label/target variable and will be used to test the model. 30% of the rows are in this set.



Note: `X_train/test` and `y_train/test` is a conventional naming scheme for dataset splits in machine learning. You could use more descriptive variables instead.



Note: You'll perform more advanced forms of cross-validation later on.

- Run the code cell.
- Select the next code cell, then type the following:

```
1 print('Training data features: ', X_train.shape)
2 print('Training data target: ', y_train.shape)
```

- Run the code cell.
- Examine the output.

```
Training data features: (31625, 31)
Training data target: (31625,)
```

There are 31,625 rows in the training sets, which is 70% of the total `user_data` set.

9. Apply oversampling to the data.

- Scroll down and view the cell titled **Apply oversampling to the data**, then select the code cell below it.
- In the code cell, type the following:

```
1 print('Before oversampling: ', Counter(y_train))
```

- Run the code cell.
- Examine the output.

```
Before oversampling: Counter({False: 27963, True: 3662})
```

At the moment, there are 27,963 `False` values for `term_deposit`, and 3,662 `True` values.

- e) Select the next code cell, then type the following:

```

1 # Define oversampling strategy.
2
3 SMOTE = SMOTE()
4
5 # Fit and apply the transform.
6
7 X_train_SMOTE, y_train_SMOTE = SMOTE.fit_resample(X_train, y_train)
8 X_train_SMOTE = pd.DataFrame(X_train_SMOTE,
9                               columns = X_train.columns)
10
11 print('After oversampling: ', Counter(y_train_SMOTE))

```

This code uses the oversampling technique known as SMOTE, which creates synthetic data based on the existing feature space.

- f) Run the code cell.
g) Examine the output.

```
After oversampling: Counter({False: 27963, True: 27963})
```

After performing SMOTE, more samples have been added to the training sets, and both class values for `term_deposit` are now represented equally. In other words, the dataset is now balanced with regard to the target variable.

10. Check the distribution of the test data.

- a) Scroll down and view the cell titled **Check the distribution of the test data**, then select the code cell below it.
b) In the code cell, type the following:

```

1 # Test data should not be oversampled.
2
3 print('Test data features: ', X_test.shape)
4 print('Test data target: ', y_test.shape)

```

- c) Run the code cell.
d) Examine the output.

```
Test data features: (13554, 31)
Test data target: (13554,)
```

The test data still has 13,554 rows, which was 30% of the original dataset.

- e) Select the next code cell, then type the following:

```
1 Counter(y_test)
```

- f) Run the code cell.
g) Examine the output.

```
Counter({False: 11931, True: 1623})
```

Unlike the training data, the test data is not oversampled, as the model won't be learning from this data.

11. Normalize the data.

- Scroll down and view the cell titled **Normalize the data**, then select the code cell below it.
- In the code cell, type the following:

```
1 norm = MinMaxScaler().fit(X_train_SMOTE)
```

Certain machine learning algorithms, including logistic regression, perform best when the input data is scaled. Here, you're using a function that normalizes the data.

- Run the code cell.
- Select the next code cell, then type the following:

```
1 X_train_norm = norm.transform(X_train_SMOTE)
2
3 print('Minimum: ', np.min(X_train_norm))
4 print('Maximum: ', np.max(X_train_norm))
```

- Run the code cell.
- Examine the output.

```
Minimum:  0.0
Maximum:  1.0
```

After the transformation, the minimum value is 0 and the maximum is 1, indicating that the data was normalized. Note that the normalized version of the dataset is in its own variable so that you can retain the non-normalized version for use with algorithms that don't benefit from featuring scaling.

12. Train a logistic regression model.

- Scroll down and view the cell titled **Train a logistic regression model**, then select the code cell below it.
- In the code cell, type the following:

```
1 logreg = LogisticRegression()
2 logreg.fit(X_train_norm, y_train_SMOTE)
```

This code creates a logistic regression object with the default hyperparameters. Then, the `logreg` model is "fit" with the normalized training data (both the features and the label). This initiates the training process.

- Run the code cell.



Note: In most cases, the output of creating the model object and calling `fit()` will just be the model's class name.

13. Make predictions using the logistic regression model.

- Scroll down and view the cell titled **Make predictions using the logistic regression model**, then select the code cell below it.
- In the code cell, type the following:

```
1 logreg_y_pred = logreg.predict(X_test)
2 print(Counter(logreg_y_pred))
```

When `predict()` is called on the model object, it takes the test dataset as the input the model is trying to predict. You can compare the predictions to the actual label values.

- Run the code cell.

- d) Examine the output.

```
Counter({True: 13486, False: 68})
```

The model's predictions include 13,486 True values, and 68 False values. Remember that this is a prediction of just `X_test`—data that the model is seeing for the first time.

- e) Select the next code cell, then type the following:

```
1 results = pd.concat([y_test.iloc[:5], X_test.iloc[:5]], axis = 1)
2 results.insert(1, 'term_deposit_pred', logreg_y_pred[:5])
3 results
```

This code will show the first five records of the test set and compare the actual `term_deposit` labels to the model's predictions.

- f) Run the code cell.
g) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	jo
43308	False	True	0.0	1369.42	1	0	0	0	0	0
32770	False	True	2.0	246.00	0	0	0	1	0	0
17440	False	True	0.0	1369.42	0	0	0	0	0	0
36164	False	True	0.0	1369.42	0	0	0	1	0	0
29218	False	True	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

The first column, `term_deposit`, is the ground truth for each record, whereas the `term_deposit_pred` column shows the model's predictions. This logistic regression model incorrectly predicted True values for all of the first five records.

14. Obtain the logistic regression model's score.

- a) Scroll down and view the cell titled **Obtain the logistic regression model's score**, then select the code cell below it.
b) In the code cell, type the following:

```
1 accuracy_score(y_test, logreg_y_pred)
```

You can use metrics like accuracy to determine how well the model did in its predictions. Most classifiers in scikit-learn use accuracy as the default scoring metric, which is the fraction of predictions that the model got right. However, this is not the only metric, nor is it necessarily the best for any given scenario. You'll learn more about metrics later—for now, verifying the model's accuracy scores will suffice.

- c) Run the code cell.
d) Examine the output.

```
0.1247602183857164
```

The logistic regression model has an accuracy of ~12%. That's very low, and not something you would normally accept. You'll see if you can improve this score later.

15. Keep this notebook open.

***k*-Nearest Neighbor (*k*-NN)**

The ***k*-nearest neighbor (*k*-NN)** algorithm is an alternative classification approach in which a data example is placed in a class based on its similarities to other data examples. These similarities are derived from the feature space (i.e., the combined vectors of training features). For example, you want to classify a body of water as either a pond or a lake. If a body of water is larger by surface area and maximum depth, it is more similar to bodies of water labeled lakes than those labeled as ponds. Therefore, *k*-NN will classify the example as a lake.

The *k* in *k*-NN defines the number of the data examples that are the closest neighbors of the example in question. "Closest" in this case refers to the distance between data points when they are mapped to the feature space. Using *k*, *k*-NN takes a plurality vote of these neighboring points to determine how to classify the example in question. In the following figure, *k* = 3, so the algorithm takes a vote of the three nearest neighbors to the data example (the green X). Since two of the three neighbors are in class 0 (red circles), the data example is classified as a 0.

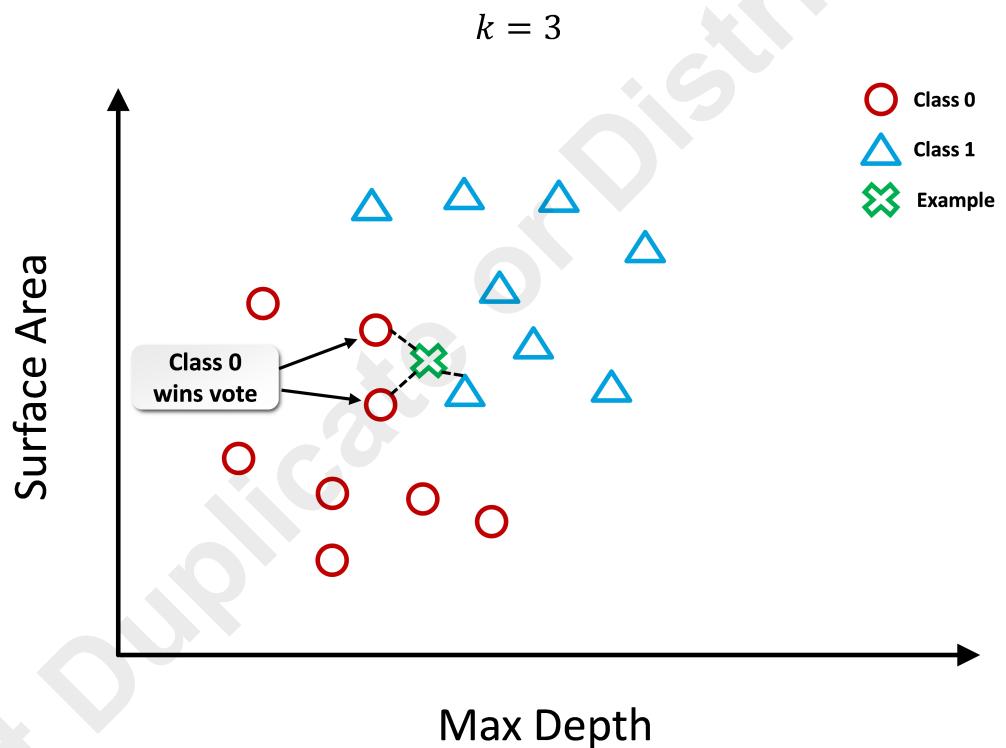


Figure 5–3: *k*-NN classification where *k* = 3.

Compared to logistic regression, *k*-NN is simpler to implement since it doesn't really "learn" in the same way as typical machine learning algorithms; i.e., it doesn't improve its classification ability through training. It also outputs a classification score directly instead of a probability. However, *k*-NN can take a very long time to calculate classifications for large datasets, so logistic regression is preferred in those cases.



Note: *k*-NN can also solve regression problems, but it is most often used in classification.

***k* Determination**

Like many hyperparameters in machine learning, the choice of k in k -NN is ultimately dependent on the nature of the training data and the output you're looking for. There is not necessarily one perfect value for k for all known circumstances. The larger you make k , the less "noisy" the dataset becomes with respect to classification; in other words, the effect of anomalies or mislabeled examples is reduced. However, this also leads to less distinct boundaries between classes, and can increase computational overhead. Lower k values are better at keeping these boundaries distinct, but are less effective at minimizing the effect of noise on the data.

Consider the following example. In the figure, the k value from before (3) has been tuned to a new value (5) on the same training set. The model now classifies the example as a 1 instead of a 0 because there are more 1 votes (blue triangles) than 0 votes between the five nearest neighbors. A change of k like this can completely change the classification a data example receives.

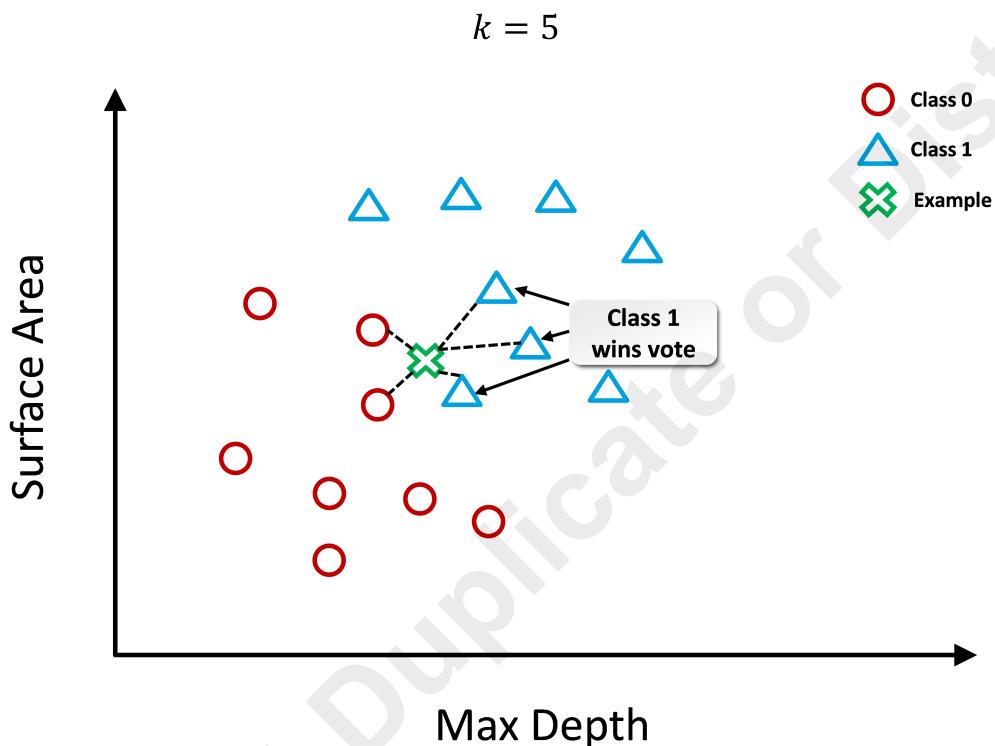


Figure 5-4: Changing k in a k -NN model can lead to different results.

One rule-of-thumb when selecting k is to always make k odd when you have a binary classification problem; this eliminates the chance of there being a tie in the vote. Another approach to selecting k is called bootstrapping, and it involves taking the square root of the total number of data examples in the training set. The resulting value is approximately the k you should consider using. You can also use a performance-testing technique like cross-validation to help you determine an acceptable value for k .

Guidelines for Training k -NN Models

Follow these guidelines when training k -NN models.

Train k -NN Models

When training k -NN models:

- Consider that lower k values make class boundaries more distinct while being less effective at minimizing noise, and vice versa.
- Consider making k odd for binary classification to avoid a tie vote.
- Consider using a bootstrapping method of selecting k by taking the square root of the number of examples in the dataset.
- Consider using cross-validation to help determine a good k value.
- Consider using k -NN over logistic regression when simplicity of implementation is key, and when working with smaller datasets.

ACTIVITY 5–2

Training a k -NN Model

Before You Begin

Developing Classification Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Train a k -nearest neighbor (k -NN) model**. Select **Cell**→**Run All Above**.

Scenario

The initial logistic regression model didn't produce very good results. But, there are plenty more classification algorithms to try. So, you'll build another model, this time using k -nearest neighbor (k -NN), a distance-based algorithm. You'll make predictions using this model and evaluate its accuracy score as well to see how it compares.

1. Train a k -nearest neighbor (k -NN) model.

- a) Scroll down and view the cell titled **Train a k -nearest neighbor (k -NN) model**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 knn = KNeighborsClassifier()
2 knn.fit(X_train_norm, y_train_SMOTE)
```

As with logistic regression, you're creating a model object and fitting the data to it. Since k -NN calculates classifications using distance (i.e., the numeric ranges of features), it's best to use the normalized version of the data.

- c) Run the code cell.

2. Make predictions using the k -NN model.

- a) Scroll down and view the cell titled **Make predictions using the k -NN model**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 knn_y_pred = knn.predict(X_test)
2 print(Counter(knn_y_pred))
```

This code will make predictions on the test data, similar to the code for logistic regression.

- c) Run the code cell.
- d) Examine the output.

```
Counter({True: 10197, False: 3357})
```

The k -NN model predicted that 10,197 records have `True` for `term_deposit`, whereas 3,357 records show `False`.

- e) Select the next code cell, then type the following:

```
1 results['term_deposit_pred'] = knn_y_pred[:5]
2 results
```

- f) Run the code cell.
g) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	jd
43308	False	True	0.0	1369.42	1	0	0	0	0	0
32770	False	True	2.0	246.00	0	0	0	1	0	0
17440	False	True	0.0	1369.42	0	0	0	0	0	0
36164	False	True	0.0	1369.42	0	0	0	1	0	0
29218	False	True	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

The k -NN model also incorrectly predicted the first five rows.

3. Obtain the k -NN model's score.

- a) Scroll down and view the cell titled **Obtain the k -NN model's score**, then select the code cell below it.
b) In the code cell, type the following:

```
1 accuracy_score(y_test, knn_y_pred)
```

- c) Run the code cell.
d) Examine the output.

```
0.2968865279622252
```

The k -NN model's accuracy is ~30%; better than the logistic regression model, but still not great.

- e) Select the next code cell, then type the following:

```
1 print('Number of mislabeled points out of a total %d points: %d'
2   % (X_test.shape[0], (y_test != knn_y_pred).sum()))
```

This code will give you another perspective into the model's performance by identifying how many data examples it predicted incorrectly.

- f) Run the code cell.
g) Examine the output.

```
Number of mislabeled points out of a total 13554 points: 9530
```

9,530 out of 13,554 examples were predicted incorrectly.

4. Keep this notebook open.

Support–Vector Machines (SVMs)

Support–vector machines (SVMs) are supervised learning algorithms that can be used to solve both classification and regression problems. SVMs separate data values using a hyperplane. The

hyperplane includes a decision boundary that is either a line or curve function. On each side of this boundary is a line or curve function parallel and equidistant to the decision boundary called a support-vector margin. The functions on either edge of these margins are called the support vectors.

It's much easier to grasp the concept of SVMs using visuals, so consider the following figure, in which a hyperplane is plotted on a graph.

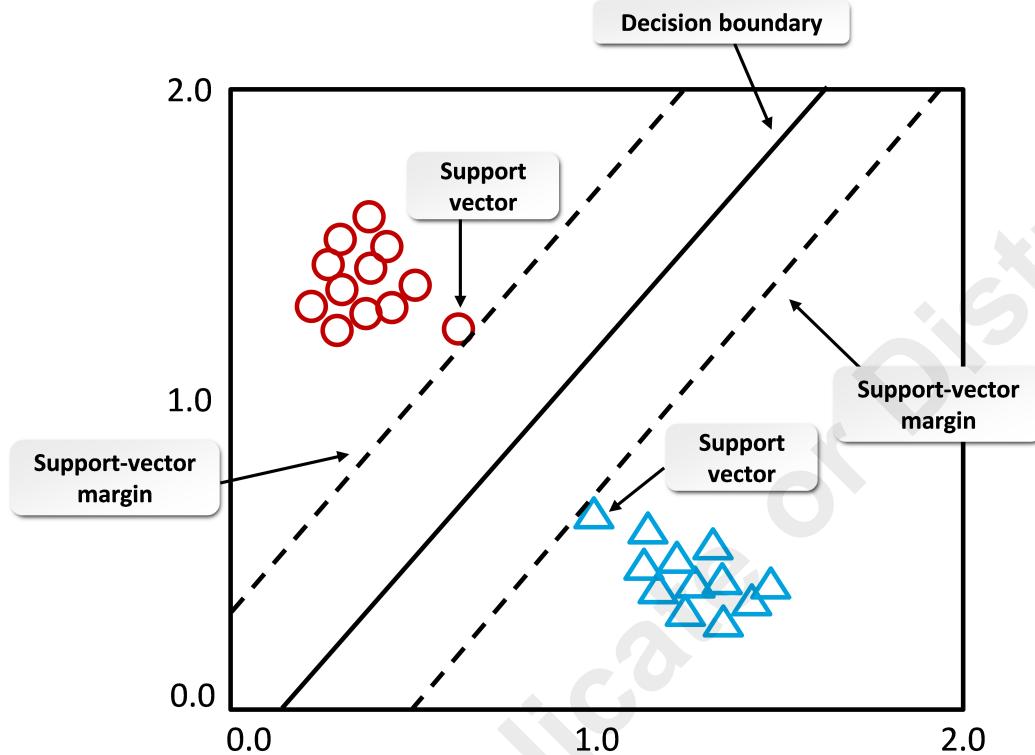


Figure 5-5: An example of a hyperplane used in SVMs.

SVMs for Linear Classification

How a hyperplane is constructed and how it applies to the data are dependent on the type of machine learning problem you're trying to solve. The previous figure demonstrated a hyperplane applied to a linear classification problem; that is, a classification problem whose classes can be reasonably split using a straight line when mapped to a feature space.

In the following figure, the graph on the left demonstrates a decision boundary using a standard linear classification algorithm. The decision boundary in this graph does a good job of separating both classes in the feature space, but it comes very close to the edge data examples. This will likely lead to problems of overfitting, as new test data may not generalize well using this model. Compare this to the model introduced earlier (on the right of this figure), which uses SVMs. The decision boundary is plotted in such a way that it not only splits the classes, but it also stays as far away from the edge examples as possible. The dashed lines create the margins of separation by intersecting the edge examples.

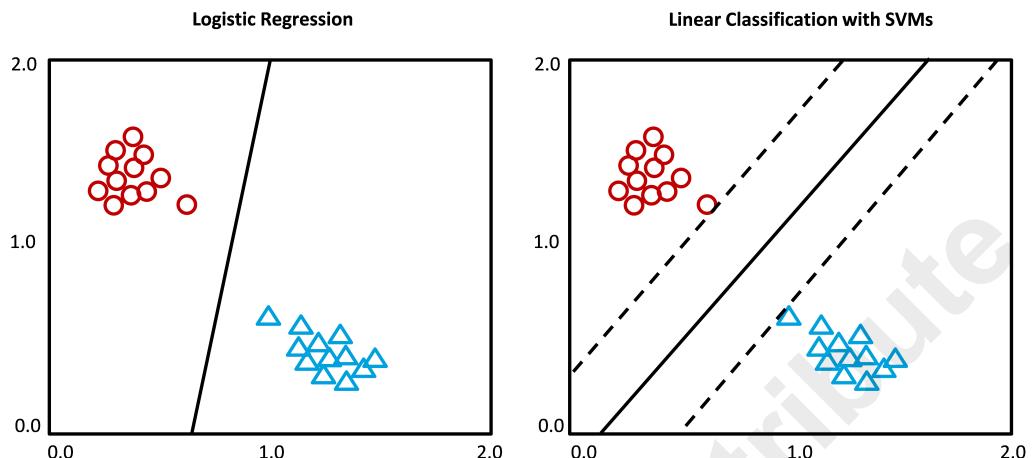


Figure 5-6: A linear classifier without SVMs (left) compared to a linear classifier with SVMs (right).

Now, observe what happens when a new test example is evaluated on the model.

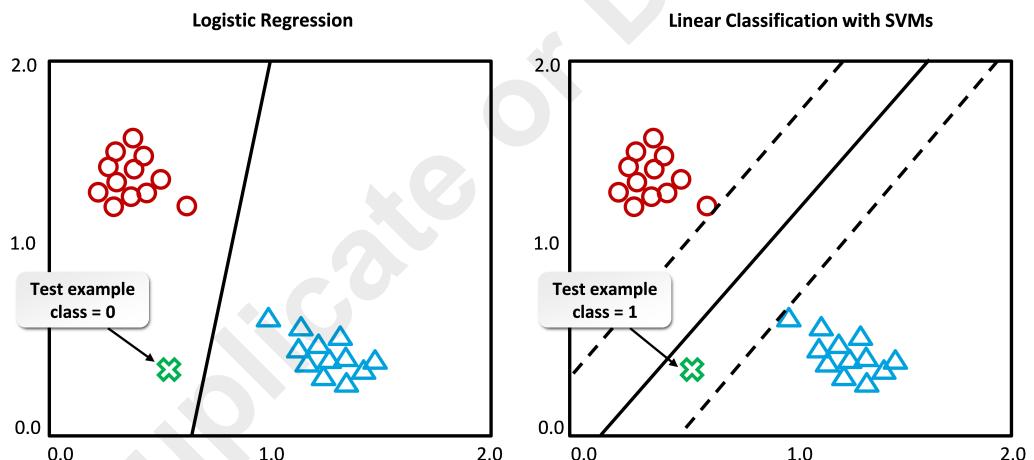


Figure 5-7: Classifying a test example for a model without SVMs (left) and a model with SVMs (right).

Although the test example has the same values when plugged into both models, the model with SVMs classified it differently. SVMs are therefore useful in classification tasks where the training data includes outliers. They tend to outperform logistic regression and other classification algorithms in this regard. However, when you use SVMs, it's very important to scale your data—more so than other classification algorithms. If one feature has much larger values than another, it will have an undue influence on the distances that are calculated between the support vectors and the decision boundary. The distribution of the data points is more important in this case than the range of each feature.



Note: SVMs can also solve non-linear classification problems, but a discussion of this is beyond the scope of this course.

Hard-Margin Classification

The previous figure demonstrated what is known as **hard-margin classification**, or a type of classification in SVMs where all data examples are outside of the margins, and each example is on the "correct" side of the margins. This is sufficient in some situations, but it can cause problems when there are even more extreme outliers. Consider the following figure, in which both Class 0 and Class 1 outliers are plotted close to the data examples in the opposite class.

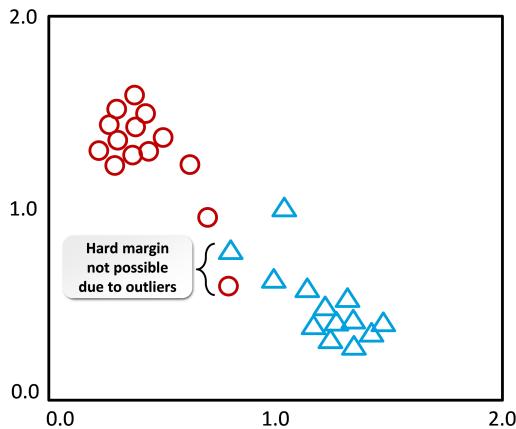


Figure 5-8: Extreme outliers in the training may lead to hard-margin classification failing.

This renders hard-margin classification ineffective; the data examples will either be on the wrong side of the margins, or multiple data examples will be inside the margins. Because there is no way to cleanly separate the data, hard-margin classification fails in this case.

In other cases, an extreme example may lead to a very small distance between the margins, meaning the model will do a poor job at generalizing to new test data.

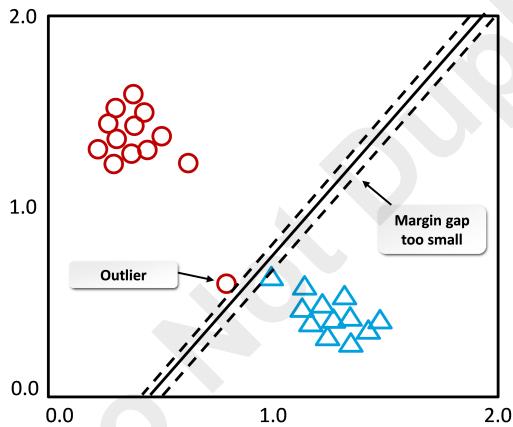


Figure 5-9: A single outlier leading to a poorly fit hyperplane.

Soft-Margin Classification

Soft-margin classification is an approach that strikes a balance between keeping the distance between the margins as large as possible and minimizing the number of examples that end up inside the margins. So, it does not perfectly solve the problem of extreme outliers, but is instead a

compromise. Nevertheless, soft-margin classification ends up being more effective than hard-margin classification in such cases. It helps the model avoid overfitting to the training data.

Depending on your toolset, you may be able to tune a hyperparameter to give more weight to one output over another (i.e., wider margins vs. number of examples inside margins). So, the difference between hard and soft margins is not an absolute; instead, there is a degree of "softness" or "hardness" that you, the practitioner, specify before training.

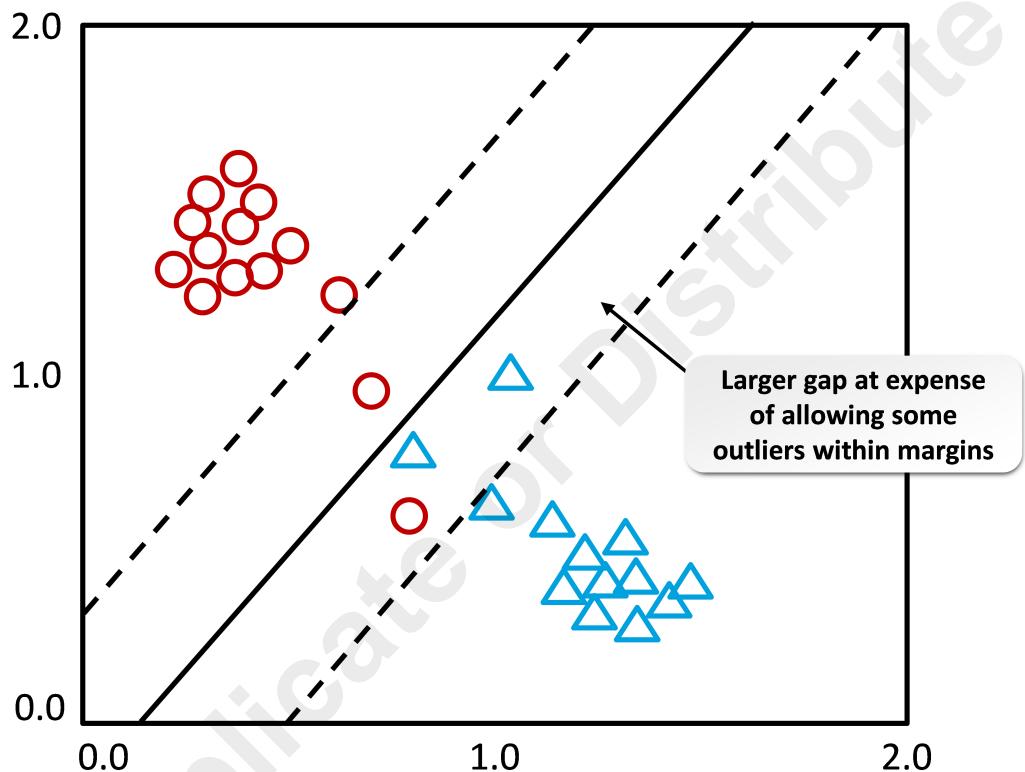


Figure 5–10: A hyperplane plotted using soft-margin classification. The margin gap remains wide at the expense of keeping some outliers within the margins.



Note: In SVMs used in regression, the ideal is to fit as many examples *within* the margins as possible, the opposite of SVMs used in classification.

Guidelines for Training SVM Classification Models

Follow these guidelines when training SVM classification models.

Train SVM Classification Models

When training SVM classification models:

- Consider using an SVM model when the problem you are trying to solve is sensitive to outliers.
- Recognize that the goal of an SVM classification model is to widen the margins as much as feasible, while at the same time keeping data examples outside the margins.
- Tune the regularization hyperparameter to adjust the size of the margins.
- Consider that narrowing the margins too much to keep all examples outside those margins may lead to complications (e.g., overfitting).

- Consider softening the margins to avoid hard-margin overfitting issues.
- Recognize that softening the margins will likely place some examples within those margins, which is often a necessary tradeoff.
- Scale the data used to train an SVM model so the margins are calculated based on the distribution of the feature spaces and not the range of values.

Do Not Duplicate or Distribute

ACTIVITY 5–3

Training an SVM Classification Model

Before You Begin

Developing Classification Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Train a support-vector machine (SVM) model**. Select **Cell→Run All Above**.

Scenario

The *k*-NN model did better than the logistic regression model, but there's still a chance to improve your scores and generate a better model. So, you'll try out your data on a support-vector machine (SVM) algorithm. SVMs excel at modeling data with outliers. Although you dealt with outliers earlier in the ETL process, an SVM model might still perform well. So, you'll train one and find out.

1. Train a support-vector machine (SVM) model.

- Scroll down and view the cell titled **Train a support-vector machine (SVM) model**, then select the code cell below it.
- In the code cell, type the following:

```
1 svm = SVC()
2 svm.fit(X_train_norm, y_train_SMOTE)
```

SVC() is a scikit-learn implementation of an SVM classifier. Because SVMs are also based on distance, you'll train the model using the normalized dataset.

- Run the code cell.



Note: It can take up to five minutes for this model to finish training.

2. Make predictions using the SVM model.

- Scroll down and view the cell titled **Make predictions using the SVM model**, then select the code cell below it.
- In the code cell, type the following:

```
1 svm_y_pred = svm.predict(X_test)
2 print(Counter(svm_y_pred))
```

- Run the code cell.
- Examine the output.

```
Counter({False: 13554})
```

It seems that the SVM model predicted *all* 13,554 examples would be `False` for `term_deposit`. Obviously, this model has some significant limitations.

- e) Select the next code cell, then type the following:

```
1 results['term_deposit_pred'] = svm_y_pred[:5]
2 results
```

- f) Run the code cell.
g) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	jc
43308	False	False	0.0	1369.42	1	0	0	0	0	0
32770	False	False	2.0	246.00	0	0	0	1	0	0
17440	False	False	0.0	1369.42	0	0	0	0	0	0
36164	False	False	0.0	1369.42	0	0	0	1	0	0
29218	False	False	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

The SVM model correctly predicted that the first five records would be False.

3. Obtain the SVM model's score.

- a) Scroll down and view the cell titled **Obtain the SVM model's score**, then select the code cell below it.
b) In the code cell, type the following:

```
1 accuracy_score(y_test, svm_y_pred)
```

- c) Run the code cell.
d) Examine the output.

```
0.880256750774679
```

The SVM model's accuracy is very high: ~88%. This is a good example of why some metrics, particularly accuracy, can be misleading. The model predicted that every example would be False, and since the test set is so imbalanced in favor of False, the model ended up making a lot of correct predictions. However, a model that only predicts False and nothing else is not particularly useful, since any human could have done that without effort. When you learn about metrics, you'll see some other ways that will make the model's limitations more apparent. For now, consider that not every model you train will end up performing better than the last. This is the unavoidable reality of experimentation.

4. Keep this notebook open.

Naïve Bayes

Naïve Bayes classification is another type of classification algorithm used in machine learning. It computes classification probabilities based on Bayes' theorem, which, when applied to a classification problem, can be represented as:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

Where:

- y is the observed classification.
- \mathbf{x} is a vector of the dataset's features.
- $p(y|\mathbf{x})$ is the likelihood of y given \mathbf{x} . It is also called the *posterior* probability. This is what you're trying to determine.
- $p(\mathbf{x}|y)$ is the likelihood of \mathbf{x} given y .
- $p(y)$ is the probability of y independent of the features. It is also called the *prior* probability.
- $p(\mathbf{x})$ is the probability of \mathbf{x} independent of the class.

Naïve Bayes Example

Consider the customer retention example. Here is an example dataset where, for simplicity's sake, there is only one categorical feature: level of customer satisfaction.

Satisfaction	Returning?
Satisfied	Yes
Not satisfied	No
Very satisfied	Yes
Not satisfied	Yes
Not satisfied	No
Satisfied	No
Very satisfied	Yes
Satisfied	Yes

Figure 5–11: Naïve Bayes example.

Using this table, you add up the frequency of each satisfaction value and divide it by the total number of examples to get the probability of each satisfaction value. There's 5 "Satisfied" customers, and 10 total customers, so the probability of a customer being satisfied is 0.50. This and the rest of the feature probabilities ("Not satisfied" = 0.30; "Very satisfied" = 0.20) make up $p(\mathbf{x})$. You also find the frequency of each label value and divide it by the total number of examples to get your prior probabilities ($p(y)$). In other words, there are 7 "Yes" responses, so its prior probability is 0.70. Likewise, there are 3 "No" responses, so its prior probability is 0.30.

Now you have all you need to plug your values into Bayes' theorem. Let's say you want to predict if a "Satisfied" customer is going to return. The theorem can be restated like so:

$$p(\text{Yes}|\text{Satisfied}) = \frac{p(\text{Satisfied}|\text{Yes})p(\text{Yes})}{p(\text{Satisfied})}$$

You already know that $p(\mathbf{x})$ is 0.50, and that $p(y)$ is 0.70. Now you just need to calculate $p(\mathbf{x}|y)$. You do this by taking the number of "Satisfied" examples that resulted in a "Yes" (4) and dividing that by the total number of "Yes" results (7). This comes out to roughly 0.57.

Now plug in the number values to the theorem:

$$p(\text{Yes}|\text{Satisfied}) = \frac{0.57 \cdot 0.70}{0.50}$$

The end result is that $p(y|\mathbf{x})$, the posterior probability, is about **0.80**. This exceeds a decision boundary of 0.50 (since this example is a binary problem), so the naïve Bayes classifier will predict a "Yes" for this customer—in other words, the customer will return.

Naïve Bayes Classification Characteristics

The "naïve" in naïve Bayes classification refers to the fact that the probabilities are calculated with the assumption that the features are conditionally independent. In other words, one feature does not interact with another. If your customer retention dataset had fields like "Purchase price" and "Quantity purchased," it would not be suitable for naïve Bayes classification as those two variables influence each other. In practice, it is rare for a machine learning problem or its datasets to include only independent features. Nevertheless, if you do encounter such a task, naïve Bayes classifiers can significantly outperform logistic regression in both training time and model skill.

One of the most common real-world applications of naïve Bayes classification is in spam filtering. Spam filters can analyze the text of emails so that each textual component (e.g., a word or phrase) is analyzed independently of the others. The presence of certain words or phrases can prompt the model to classify an email message as either "spam" or "not spam," regardless of how these words or phrases appear in a wider context. In addition, naïve Bayes classifiers have found success in sentiment analysis, such as determining whether users are satisfied with a product or service based on the content of their social media posts.

There are actually multiple different types of naïve Bayes classifiers. They mainly differ based on how they treat the distribution of $p(\mathbf{x}|y)$ when determining a classification. For example, Gaussian naïve Bayes assumes a normal distribution of values, whereas multinomial naïve Bayes assumes a multinomial distribution.



Note: Like logistic regression, multinomial forms of naïve Bayes can address multi-class classification problems.

Guidelines for Training Naïve Bayes Models

Follow these guidelines when training naïve Bayes models.

Train Naïve Bayes Models

When training naïve Bayes models:

- Use naïve Bayes when variables have minimal or no dependency or relationship to each other.
- Consider using naïve Bayes when the speed of training and prediction generation are important.
- Encode categorical features (observations and outcomes) to integers.
- Consider discretizing variables that have a large number of distinct observations.
- Create a new feature for a combination of features that does not already exist in the dataset. For instance, if the predictor factors in weather and temperature, you probably won't have a combination for rainy and less than 40 degrees, so you can create a feature for `rainy & temp < 40` with an outcome of True or False.

ACTIVITY 5–4

Training a Naïve Bayes Model

Before You Begin

Developing Classification Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Train a naïve Bayes model**. Select **Cell**→**Run All Above**.

Scenario

The next model you'll train is a naïve Bayes model, which calculates classification probabilities based on Bayes' theorem. These types of models assume that the features do not interact with each other, which they almost certainly do in a dataset like this one. Still, it's worth trying out, so you'll build one and see how it scores.

1. Train a naïve Bayes model.

- Scroll down and view the cell titled **Train a naïve Bayes model**, then select the code cell below it.
- In the code cell, type the following:

```
1 gnb = GaussianNB()
2 gnb.fit(X_train_SMOTE, y_train_SMOTE)
```

You'll be training the model on the non-normalized dataset, as naïve Bayes is not as sensitive to differing numerical ranges as logistic regression and distance-based algorithms.

- Run the code cell.

2. Make predictions using the naïve Bayes model.

- Scroll down and view the cell titled **Make predictions using the naïve Bayes model**, then select the code cell below it.
- In the code cell, type the following:

```
1 gnb_y_pred = gnb.predict(X_test)
2 print(Counter(gnb_y_pred))
```

- Run the code cell.
- Examine the output.

```
Counter({False: 9685, True: 3869})
```

The naïve Bayes model predicted 9,685 False values and 3,869 True values. Unlike the SVM model, this model is able to discriminate between both values.

- e) Select the next code cell, then type the following:

```
1 results['term_deposit_pred'] = gnb_y_pred[:5]
2 results
```

- f) Run the code cell.
g) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	jd
43308	False	False	0.0	1369.42	1	0	0	0	0	0
32770	False	True	2.0	246.00	0	0	0	1	0	0
17440	False	False	0.0	1369.42	0	0	0	0	0	0
36164	False	False	0.0	1369.42	0	0	0	1	0	0
29218	False	False	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

Out of the first five records, all but the second were predicted correctly.

3. Obtain the naïve Bayes model's score.

- a) Scroll down and view the cell titled **Obtain the naïve Bayes model's score**, then select the code cell below it.
b) In the code cell, type the following:

```
1 accuracy_score(y_test, gnb_y_pred)
```

- c) Run the code cell.
d) Examine the output.

```
0.7019330087059171
```

The naïve Bayes model has an accuracy score of ~70%. This is considerably better than the logistic regression and k -NN models.

4. Keep this notebook open.

Decision Tree

Another algorithm that can perform classification is a decision tree. A **decision tree** is an arrangement of conditional statements and their conclusions in a branch–leaf structure. Consider the following figure in which a business wants to identify whether or not someone will be a returning customer.

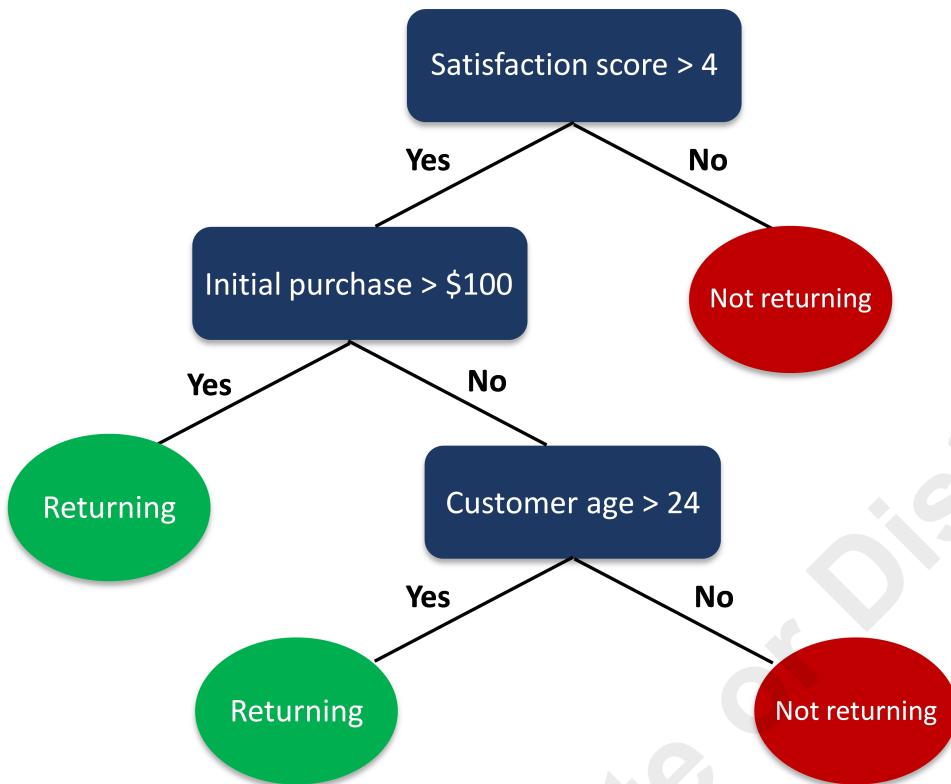


Figure 5-12: A decision tree determining whether a customer will return.

A decision tree looks like a flowchart, where each "branch" represents a decision based on some condition and each "leaf" represents an output. The decision nodes receive input based on prior decisions until that particular branch terminates at a leaf.

In this example, the rectangles are the decision nodes, and the ovals are the output values. The output values in this case are classification labels. The decisions are either "Yes" or "No." Any data example (a customer) must start at the root decision node (satisfaction score). If the customer has expressed low satisfaction, the model immediately predicts that they will not return. If the customer doesn't express low satisfaction, they move on to the next node (initial purchase amount). If they've spent more than \$100 in their initial purchase, the model predicts they will return. If the customer doesn't spend that amount, they continue to the final decision node (age). If they are 24 or younger, the model predicts they will not return. If they are older than 24, the model predicts they will return. Ultimately, any data example put through this decision tree will be given a classification based on its features.

Decision trees are used extensively in machine learning because they are simple to understand and do not require as much data preparation as some other algorithms. This is because the structure of the decision tree will be the same on a dataset regardless of procedures like scaling or feature engineering that you apply.



Note: Decision trees are not only capable of classification, but also regression tasks.

Classification and Regression Tree (CART)

The [classification and regression tree \(CART\)](#) algorithm is one of the most popular decision tree algorithms in machine learning. CART uses the [Gini index](#) as a metric for constructing the decision

tree from the training dataset. The algorithm splits the training set into two based on a single feature with a threshold value. For example, it will first split the dataset based on the "Satisfaction score" feature, and it will choose a decision value—greater than 4, in this case. It makes this choice by using the Gini index cost function to determine the "purity" of the decision node. The purest decision node is one in which all data examples at the decision node belong to one class, and none belong to the other class. The most impure decision node is one in which the data examples are split 50–50 between each class.

The following is the formula for Gini index:

$$G = 1 - \sum_{i=1}^c (p_i)^2$$

Where:

- p_i is the probability of a data example being placed in a class i , using a single feature and the label.
- c is the total number of classes.

The Gini index is calculated for each value of a feature, then a weighted sum is taken for those values to produce the ultimate Gini index for a feature. This process is repeated for the rest of the features in the dataset. The feature with the highest level of purity ($G = 0$), and therefore lowest Gini index, is chosen as the root decision node.



Note: In the context of decision tree splitting, the name "Gini index" is a reference to a measurement of income inequality, also called the Gini index or Gini coefficient. It is named after Corrado Gini, the statistician that developed the inequality measurement.

Gini Index Example

Consider the following dataset based on the customer retention example provided earlier.

Initial purchase > \$100	Satisfaction score > 4	Customer age > 24	Returning
Yes	Yes	No	Yes
No	Yes	No	No
No	No	Yes	No
Yes	No	No	No
No	Yes	Yes	Yes
Yes	Yes	Yes	Yes
Yes	No	No	Yes
No	Yes	No	Yes

First, we'll calculate the Gini index for "Initial purchase." There are 4 "Yes" values for "Initial purchase." There are 3 cases where "Initial purchase" and "Returning" are both "Yes." There is 1 case where "Initial purchase" is "Yes" and "Returning" is "No." Plugged into the Gini index formula, this is:

$$G = 1 - \left((3/4)^2 + (1/4)^2 \right) = 0.38$$

Then, you apply the same formula to where "Initial purchase" is "No." There are 4 "No" values for "Initial purchase." There are 2 cases where "Initial" is "No" and "Returning" is "Yes." There are 2 cases where both "Initial purchase" and "Returning" are "No." Plugged into the Gini index formula, this is:

$$G = 1 - \left((2/4)^2 + (2/4)^2 \right) = 0.50$$

The weighted sum of both indices is equal to:

$$G = (4/8) \cdot 0.38 + (4/8) \cdot 0.50 = 0.44$$

CART then repeats this process for the other two features ("Satisfaction score" and "Customer age"). Fastforwarding through the math, the resulting weighted Gini indices are:

Feature	Gini Index
Initial purchase	0.44
Satisfaction score	0.37
Customer age	0.49

Because "Satisfaction score" has the lowest Gini index, it is made the root decision node.

These Gini index values are only relevant at the root decision node. CART uses the same Gini index cost function to split the data into another subset at each new decision node, recalculating the Gini index for that node and evaluating the purity of data as before. It then splits this into a sub-subset, and so on. The splitting stops when there is no more impurity to reduce, or the tree reaches a depth specified in a hyperparameter.



Note: CART only works with binary values. It cannot construct decision trees in which a single node has three or more child nodes.

Customer Retention Example Tree

The decision tree that was shown earlier was a simplified version. The following is what the tree might look like when the customer retention data is run through the CART algorithm.

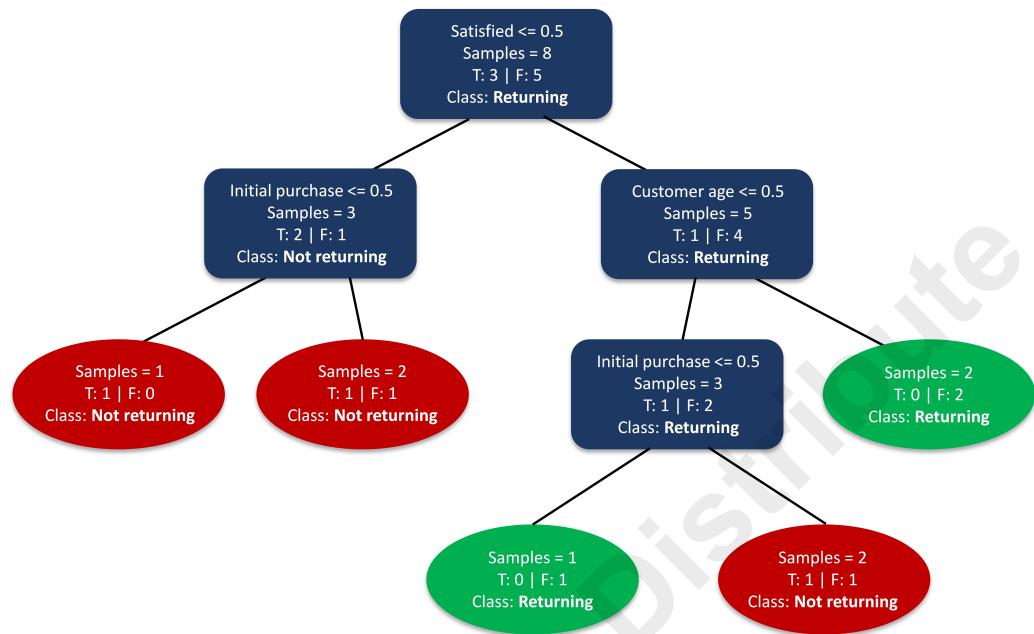


Figure 5–13: A CART model of the customer retention data.

The dataset was encoded so that "Yes" values are 1 and "No" values are 0. So, the root node that says `Satisfied <= 0.5` is referring to the truth value of the feature (i.e., was the customer's satisfaction score above 4?), not the actual level of satisfaction. Since the truth value is either a 1 or 0, less than or equal to 0.5 means it's a 0. In other words, the root node splits based on if the customer was *not* satisfied. This is true for three of the customers, and false for five of the customers. So the node branches off to the left for unsatisfied customers, and to the right for satisfied customers. Note that the root node also has a prediction. This isn't particularly meaningful since, at this point, it's just indicating that the majority of customers in the entire dataset returned. These predictions will occur at each node, but ultimately, they are most useful at the terminating leaves.

Moving down the tree, the decision node on the left checks if the initial purchase was *not* over \$100. Notice that there are only three samples to work with at this point, two of which did not spend over \$100, and one of which did. In this case, the initial purchase amount for unsatisfied customers doesn't really matter, since both branches lead to terminating leaves that make the same decision: the customer will not be returning.

If you move back to the right side of the tree after the root node, you can see that the decision node is checking if the customer's age is *not* above 24. If the customer *was* above 24, the tree terminates at a leaf that predicts they will return. Otherwise, the tree continues to another instance of the initial purchase amount decision node. The remaining decisions can be summed up like so:

- **Question 1:** Will satisfied customers 24 years or younger return to our business if they spend *more than* \$100 in their initial purchase?
 - **Answer:** No.
- **Question 2:** Will satisfied customers 24 years or younger return to our business if they spend *less than* \$100 in their initial purchase?
 - **Answer:** Yes.

Any new data samples that you put through the model will descend this tree and be evaluated at each relevant decision node.

CART Hyperparameters

The following are some of the major hyperparameters associated with CART.

Hyperparameter	Description
max_depth	<p>Use this to specify how "deep" the tree should go, i.e., the number of decision nodes from root to farthest leaf, which relates to the number of times the data is split. If you don't specify a max depth, CART will keep splitting the dataset until ϵ (purity) for all leaves is 0, or until all decision nodes are less than the value of min_samples_split.</p> <p>In their quest for achieving high levels of decision purity, decision trees tend to fall prey to overfitting, especially when the dataset has many features. The max_depth parameter is one way to regularize the training in order to avoid overfitting. There is not necessarily one ideal depth value for all scenarios; you must evaluate the model's performance to determine the ideal value for your specific task.</p>
min_samples_split	Use this to specify how many samples (i.e., data examples) are required in order to split a decision node. By increasing this value, you constrain the model because each node must contain many examples before it can split. This can help minimize overfitting, but take care not to constrain the model too much, and thus underfit it.
min_samples_leaf	Use this to specify how many samples are required to be at a leaf node. Again, increasing this value can reduce overfitting but also has the chance to lead to underfitting.
splitter	The default value for this parameter is "best", which calculates the Gini index as described previously. However, if you specify splitter="random", the CART algorithm splits a feature at random and then calculates its Gini index. It repeats this process and identifies the split with the best Gini index. The advantages of this approach are that it requires less computational overhead and may be better at avoiding overfitting in certain cases. However, random splitting will not always be as good at making optimal decisions as the default setting.



Note: The exact names of the parameters may differ based on the library you're using; the names in the previous table are based on scikit-learn.

Pruning

In keeping with the tree metaphor, **pruning** is the process of reducing the overall size of a decision tree by eliminating nodes, branches, and leaves that provide little value for the classification or regression problem at hand. This helps to mitigate the problem of overfitting, which often plagues larger trees. Even if you specify hyperparameters like max depth (often called "pre-pruning"), you may still wind up with an overly complex tree with many sections. So, pruning is another method for simplifying the model and improving its overall performance and skill.

The following is a list of major pruning methods that are applied after the tree is first generated, also known as "post-pruning":

- **Reduced-error pruning:** This is a relatively simple pruning method that tends to be quicker than the other methods and leads to smaller trees. It has also been shown to exhibit similar levels

of estimative skill as other pruning methods. It is therefore a good first option, especially when combined with pre-pruning techniques.

- **Minimum error pruning:** Unlike with reduced-error pruning, minimum error pruning does not involve a test dataset. However, it assumes that each class is equally likely, and it tends to perform poorly as the number of classes increases.
- **Cost complexity pruning:** Like reduced-error pruning, cost complexity pruning also tends to perform well, though it is usually not as quick.

Ensemble Learning

Ensemble learning is an application of machine learning in which the estimations of multiple models are considered in combination. The purpose is to obtain a more skillful estimation than any one model could in isolation. While a single model trained using a specific set of hyperparameters might appear to solve a classification or regression problem, it is not necessarily the optimal way of solving the problem. Studies have shown that aggregating estimations from multiple models, especially a diverse set of models, tends to lead to better estimations.

For example, you might train several models on the customer retention problem. One of your models might be trained using a logistic regression; another might be trained using naïve Bayes; another using a decision tree; and so on. Each model may achieve a certain level of skill, depending on how you measure that skill. Simply choosing the model that has the best of any one skill measurement is not guaranteed to give you ideal results. Instead, many ensemble learning methods use a majority voting system that selects the class that is determined by the most models. For example, the following models determine a binary classifier for the customer retention problem:

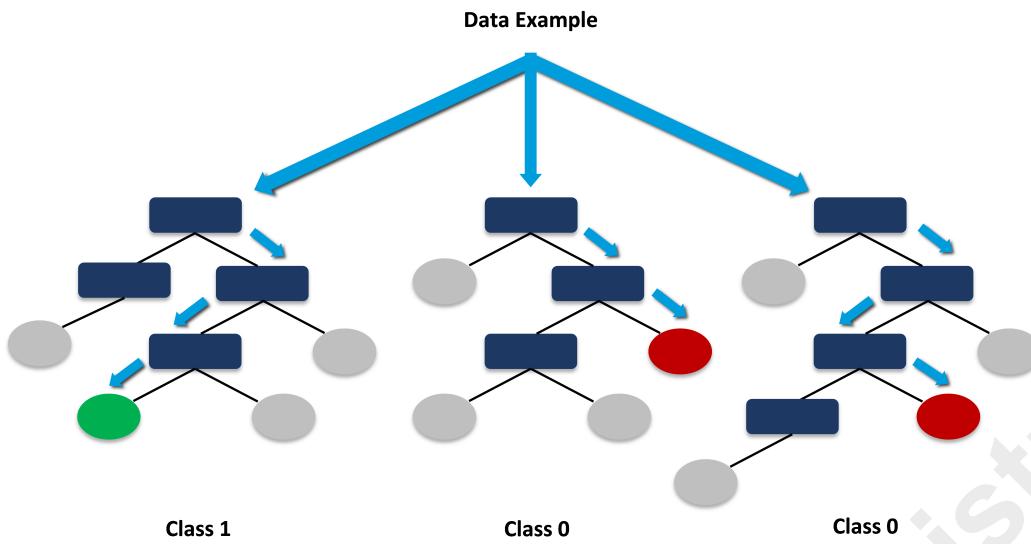
- Logistic regression model
 - Accuracy: **78%**
 - Estimation: **Class 1**
- Naïve Bayes model
 - Accuracy: **86%**
 - Estimation: **Class 0**
- CART model
 - Accuracy: **81%**
 - Estimation: **Class 1**

So, the ensemble method would determine class **1** because it is the majority. Even though the model using naïve Bayes had the highest accuracy, its estimation of 0 is not considered as useful as the majority vote. This might seem counter-intuitive, but consider how probability works: when rolling a six-sided die, the probability of getting any number is roughly 16.66%. However, if you roll the die 100 times, you may not get an even distribution of approximately 16 results for each number. However, the more rolls you perform, the higher your chances are of getting that even distribution. Rolling the die 100,000 times will have a greater chance of getting close to that even 16.66% split. This is similar to why ensemble learning is so effective—more models can lead to better results.

Random Forest

Not all ensemble methods aggregate a diverse set of training models. Some actually aggregate multiple models that use the same algorithm—the only difference is that each model is trained on a different subset of the data. The ensemble method generates these subsets by randomly sampling the overall training data for each model.

A **random forest** is an ensemble method that aggregates multiple decision tree models together and, in a classification task, selects the mode of the classifiers between all decision trees.



Mode = Class 0

Figure 5-14: A random forest selecting a classifier based on the mode of its constituent trees. Note that green ovals indicate class 1, whereas red ovals indicate class 0.

In other words, the classification that gets the most votes among all the decision trees is the classification that the random forest outputs. As with other ensemble methods, this typically results in a more skillful classification than if you had constructed a single tree and used its results, or constructed multiple trees and simply chose the one with the most accuracy. Another way to think about this is that a random forest reduces the tendency of a single model to overfit to the training data because the forest exhibits lower levels of variance.

Most random forests use a data sampling technique called bootstrap aggregating, more commonly shortened to **bagging**. This technique samples the training dataset for each individual tree, *with replacement*. "With replacement" means that a data example can show up in multiple different models. Bagging tends to lead to lower variance, further reducing overfitting. Depending on the library used, bagging may also only use a fraction of the total feature space at each decision node.



Note: Random forests, like other tree-based algorithms, don't require features to be scaled for training.

Gradient Boosting

Another ensemble method that incorporates multiple decision trees is called gradient boosting. **Gradient boosting** algorithms build models in stages, where past decision trees influence how later decision trees are built. This differentiates them from random forests, where each tree is built independently and then randomly sampled, concluding in the majority vote. With gradient boosting, the trees are built in an additive fashion, where each tree attempts to correct the errors of the tree before it. That means its conclusions are developed iteratively rather than determined all at once. The algorithm starts by generating decision trees that have low classification skill (called weak learners) and then combines those weak learners into an eventual decision tree with high classification skill (called a strong learner).

Gradient boosting starts by generating a base classification probability for the initial decision tree. It does this in part by taking the logarithm of the target label's odds. The odds are simply the number of instances in class 0 divided by the number of instances in class 1 (or vice versa). For the customer retention example, assume that you have 5 customers labeled as returning and 3 customers labeled

as not returning. So, $\log(5/3) = \sim 0.51$. To get the probability, you need to put this value through the following formula:

$$p = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

Where:

- p is the base estimator's probability (i.e., the base decision tree).
- e is the natural logarithm base.



Note: This is a logistic function, like the one used in logistic regression

In this example, the probability is **0.625**. So, the initial tree will predict this value for every data example. If the decision boundary is 0.5, then we predict that all customers in the dataset will return. This is obviously not helpful on its own, and it's why this is called a weak learner.

Gradient Boosting: Residuals

After the probability is calculated, the *residual* value for each data example is calculated by subtracting the estimated probability from the actual classification value. So, the residual for a returning customer is $1 - 0.625 = 0.375$. For a customer that didn't return, the residual is $0 - 0.625 = -0.625$. The residuals can also be thought of as the errors in the estimator.

The next step is to build a new decision tree that tries to estimate these residual values. In this new tree, the algorithm tries to estimate the residuals at each leaf of the tree. It does this by using the following transformation:

$$\frac{\sum Residual_i}{\sum [p_{t-1} \cdot (1 - p_{t-1})]}$$

Where:

- $Residual_i$ is the residual score(s) at the current leaf.
- p_{t-1} is the previous tree's probability score.

So, if a leaf has one instance of 0.375 as its residual score, then the new estimated residual for this leaf is $0.375 / (0.625 \times (1 - 0.625)) = 1.6$. This process is repeated for all other leaves in the tree.

Then, the data examples are fed through the tree so that a new set of probabilities are generated. The basic formula for this is as follows:

$$p_0 + (\eta \cdot Residual_i)$$

Where:

- p_0 is the estimated probability from the base estimator.

- η is the learning rate. This is a constant value that helps minimize bias and therefore reduce overfitting. It is a tunable hyperparameter.
- $Residual_i$ is the residual at the relevant leaf that the data example is in.

So, assuming a learn rate of 0.1, a data example that ends up in the same leaf mentioned before would have a $\log(odd)$ equal to: $0.625 + (0.1 \times 1.6) = 0.785$. You'd then plug the $\log(odd)$ value in to the same logistic function as before to get the probability. The result is approximately **0.687**. The new residuals are then calculated based on these new probabilities by subtracting the probabilities from the real label values, as before. Then they are passed to the next tree, and so on, until a stopping criterion is met. The stopping criterion is usually when the residuals have been reduced to a value close to zero, or until a maximum number of trees are produced.

Gradient Boosting: Estimations

Once the model is finished creating its trees, it can start accepting unseen data points in order to make estimations. The estimation is calculated by adding up all of the weighted residual values for each tree (and the base tree's probability), then running that through the logistic function to get the final probability score. If the score is greater than 0.5, then the model predicts the customer will return. Otherwise, the model predicts the customer will not return.

Gradient boosting models can lead to better performance as compared to random forests, particularly for unbalanced datasets. However, they are highly susceptible to overfitting in cases where the data is noisy, and they can also take longer to train than random forests since the trees are built iteratively instead of independently. Gradient boosting models are also difficult to tune. In terms of real-world applications, gradient boosting has found success in tasks like page ranking for search engines, realtime risk assessment, and scientific data analysis.

XGBoost

One of the most popular open source libraries that implements gradient boosting is XGBoost. It is available not only for Python, but for other languages such as R, C++, Java, and more. The scikit-learn package has its own gradient boosting methods, but XGBoost tends to be faster and can lead to more skillful models. XGBoost is also adept at handling missing values without requiring them to be imputed.

Guidelines for Training Classification Decision Trees and Ensemble Models

Follow these guidelines when training classification decision trees and ensemble models.

Train Classification Decision Trees and Ensemble Models

When training classification decision trees and ensemble models:

- For decision trees:
 - Consider using decision trees if you'd like to keep the model simple, easy to understand, and easy to demonstrate visually—especially to non-technical audiences.
 - Consider that, depending on the nature of the dataset and the implementation of the decision tree algorithm, you may not need to prepare that data as much as you would with other algorithms.
 - Consider that you may need to one-hot encode variables before using them with decision tree algorithms.
 - Consider that you may need to discretize continuous variables before using them with decision tree algorithms.
 - Be aware that decision trees are prone to overfitting.
 - Perform pre-pruning by tuning various decision tree hyperparameters, like the maximum depth of the tree, to help reduce overfitting.

- Consider performing various post-pruning methods such as reduced error pruning to further reduce the complexity of trees and minimize overfitting.
- For random forests:
 - Consider using an ensemble learning method like random forests to improve your model's estimative skill and reduce overfitting.
 - Use a bootstrap aggregation (bagging) technique with random forests to perform data sampling.
 - Consider that the bagging process makes cross-validation unnecessary with random forests.
 - Use most of the same pre-pruning hyperparameters in a random forest as you would on a single decision tree.
 - Consider limiting the number of trees to grow in the forest to around a few hundred, as growing more may not be worth the extra training time.
 - Use random forests for feature selection, culling the features that exhibit less importance and thereby reducing the dimensionality of the dataset.
- For gradient boosting:
 - Consider using gradient boosting methods over random forests if you have unbalanced datasets.
 - Be aware of gradient boosting's tendency to overfit data that is noisy.
 - Be aware that a gradient boosting model can take much longer to train than a random forest.

ACTIVITY 5–5

Training Classification Decision Trees and Ensemble Models

Before You Begin

Developing Classification Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Train a decision tree model**. Select **Cell**→**Run All Above**.

Scenario

As you've seen, individually trained models can produce worthwhile results, but you may also get better results with ensemble methods. Ensemble algorithms use an aggregation of multiple decision trees to classify data. So, you'll leverage these algorithms by first training a single decision tree, then training multiple trees within random forests and gradient boosting models. Hopefully, these methods will improve upon the individual models you've built thus far.

1. Train a decision tree model.

- a) Scroll down and view the cell titled **Train a decision tree model**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 | clf_tree = DecisionTreeClassifier()
2 | clf_tree.fit(X_train_SMOTE, y_train_SMOTE)
```

Tree-based algorithms are insensitive to feature scaling, so you'll train these models with the non-normalized data.

- c) Run the code cell.

2. Make predictions using the decision tree model.

- a) Scroll down and view the cell titled **Make predictions using the decision tree model**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 | clf_tree_y_pred = clf_tree.predict(X_test)
2 | print(Counter(clf_tree_y_pred))
```

- c) Run the code cell.
- d) Examine the output.

```
Counter({False: 11596, True: 1958})
```

The decision tree predicted 11,596 False values and 1,958 True values.

- e) Select the next code cell, then type the following:

```
1 results['term_deposit_pred'] = clf_tree_y_pred[:5]
2 results
```

- f) Run the code cell.
g) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	...
43308	False	True	0.0	1369.42	1	0	0	0	0	0
32770	False	False	2.0	246.00	0	0	0	1	0	0
17440	False	True	0.0	1369.42	0	0	0	0	0	0
36164	False	False	0.0	1369.42	0	0	0	1	0	0
29218	False	False	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

The second, fourth, and fifth records were correctly predicted, but the first and third were not.

3. Obtain the decision tree model's score.

- a) Scroll down and view the cell titled **Obtain the decision tree model's score**, then select the code cell below it.
b) In the code cell, type the following:

```
1 accuracy_score(y_test, clf_tree_y_pred)
```

- c) Run the code cell.
d) Examine the output.

```
0.8314150804190645
```

The single decision tree's accuracy is ~83%. This is the highest accuracy score yet, aside from the SVM model.

4. Visualize the decision tree.

- a) Scroll down and view the cell titled **Visualize the decision tree**, then select the code cell below it.
b) In the code cell, type the following:

```
1 text_representation = tree.export_text(clf_tree)
2 print(text_representation)
```

Visualizing a decision tree can make it easier to identify specific decisions that lead to classifications.

- c) Run the code cell.

- d) Examine the output.

```
|--- feature_22 <= 201.50
|   --- feature_22 <= 114.50
|   |--- feature_22 <= 77.50
|   |   --- feature_24 <= 372.50
|   |   |--- feature_23 <= 3.50
|   |   |   --- feature_25 <= 3.50
|   |   |   --- feature_1 <= 10271.10
|   |   |   |--- feature_26 <= 0.50
|   |   |   |   --- feature_14 <= 0.50
|   |   |   |   |--- feature_16 <= 0.50
|   |   |   |   |   --- feature_13 <= 0.50
|   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |--- feature_13 > 0.50
|   |   |   |   |   |--- truncated branch of depth 11
|   |   |   |   |   |--- feature_16 > 0.50
|   |   |   |   |   |--- class: False
|   |   |   |   |--- feature_14 > 0.50
|   |   |   |   |--- feature_30 <= 1.50
|   |   |   |   |   --- feature_24 <= 320.50
```

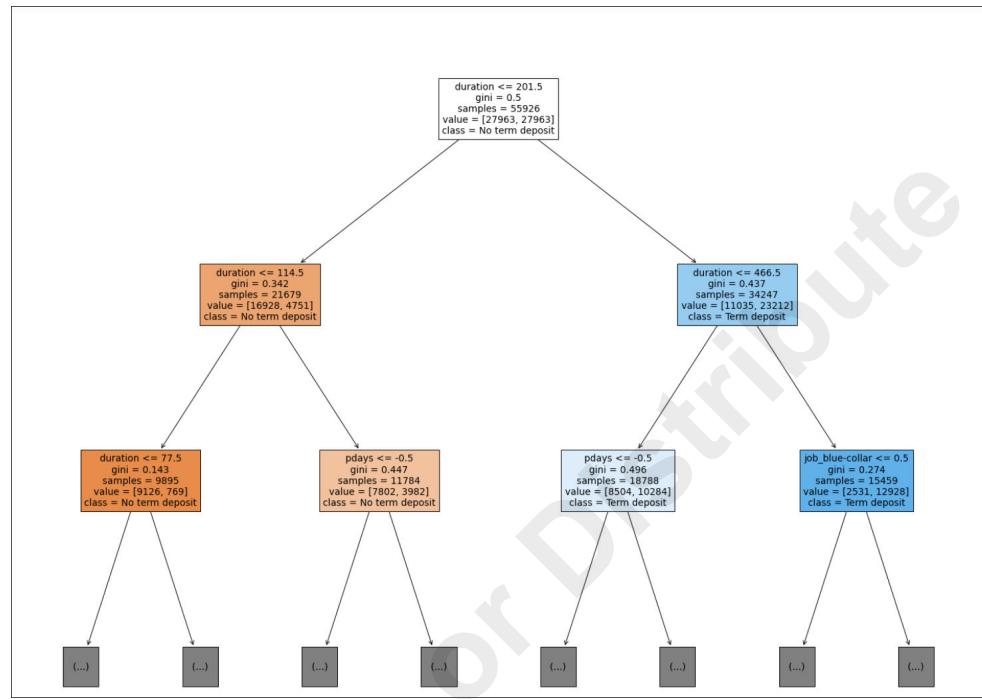
This is a text-based representation of the entire tree. You could trace each branch and leaf, but given the size of the tree, this can take a while. Instead, you'll create a more visual representation of a truncated version of the tree.

- e) Select the next code cell, then type the following:

```
1 fig = plt.figure(figsize = (25, 20))
2 _ = tree.plot_tree(clf_tree,
3                     feature_names = list(X_train.columns),
4                     class_names = ['No term deposit', 'Term deposit'],
5                     max_depth = 2,
6                     filled = True)
```

- f) Run the code cell.

g) Examine the output.



This visualization truncates the tree to a depth of two. Each node includes splitting/decision information at that node. For the root decision node at the top:

- The tree starts by evaluating if the user's contact session duration was less than or equal to 201.5 seconds.
- The Gini index at this point is 0.5. An index of 0 represents purity, so the number should ideally be decreasing as you descend the branches.
- The number of samples at this node is 55,926 (the entire oversampled training set).
- The `value` shows that 27,963 examples are true for this decision, and 27,963 are false. This aligns with the oversampling you performed earlier.
- The class prediction at this point is 0, meaning the user didn't sign up for a term deposit. This prediction will get much more useful as you descend the tree.

For the first "True" branch on the left:

- The decision node here is also evaluating the contact duration.
- The Gini index is 0.342, which, as expected, is becoming more pure.
- The number of samples is 21,679.
- 16,928 samples were not less than or equal to 114.5, and 4,751 samples were.
- The class prediction here is still 0 (no term deposit).

For the first "False" branch on the right:

- The decision node here is also evaluating contact duration.
- The Gini index is 0.437.
- The number of samples is 34,247.
- 11,035 samples were not less than or equal to 466.5, and 23,212 samples were.
- The class prediction is 1 (signed up for a term deposit).

If you had printed the entire tree, you could continue to descend the tree until you get to the bottom, where there are multiple leaves, each with their own class predictions based on the branching logic of their parent nodes.

5. Train a random forest model.

- a) Scroll down and view the cell titled **Train a random forest model**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 rf = RandomForestClassifier()
2 rf.fit(X_train_SMOTE, y_train_SMOTE)
```

This random forest algorithm will use bagging with multiple decision trees and make classification decisions based on the popular vote.

- c) Run the code cell.

6. Make predictions using the random forest model.

- a) Scroll down and view the cell titled **Make predictions using the random forest model**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 rf_y_pred = rf.predict(X_test)
2 print(Counter(rf_y_pred))
```

- c) Run the code cell.
 d) Examine the output.

```
Counter({False: 12488, True: 1066})
```

The random forest model predicted 12,488 False values and 1,066 True values.

- e) Select the next code cell, then type the following:

```
1 results['term_deposit_pred'] = rf_y_pred[:5]
2 results
```

- f) Run the code cell.
 g) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_services
43308	False	False	0.0	1369.42	1	0	0	0	0	0
32770	False	False	2.0	246.00	0	0	0	1	0	0
17440	False	True	0.0	1369.42	0	0	0	0	0	0
36164	False	False	0.0	1369.42	0	0	0	1	0	0
29218	False	False	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

Of the first five records, the random forest correctly predicted all except the third.

7. Obtain the random forest model's score.

- a) Scroll down and view the cell titled **Obtain the random forest model's score**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 accuracy_score(y_test, rf_y_pred)
```

- c) Run the code cell.

- d) Examine the output.

```
0.8826176774383946
```

The random forest achieved an accuracy score of ~88%. This is better than the lone decision tree, and the top-scoring model so far.

8. Train a gradient boosting model.

- Scroll down and view the cell titled **Train a gradient boosting model**, then select the code cell below it.
- In the code cell, type the following:

```
1 xgb = XGBClassifier(eval_metric = 'logloss')
2 xgb.fit(X_train_SMOTE, y_train_SMOTE)
```

Gradient boosting is another ensemble method that uses decision trees. It builds the trees sequentially so that past trees can improve the performance of later trees. This particular implementation of gradient boosting is XGBoost. The `eval_metric` argument refers to the method that the algorithm will use to minimize error. Log loss is the same method used by logistic regression to minimize error.

- Run the code cell.



Note: The output of this algorithm is more verbose than the others, and includes information about its default arguments.

9. Make predictions using the gradient boosting model.

- Scroll down and view the cell titled **Make predictions using the gradient boosting model**, then select the code cell below it.
- In the code cell, type the following:

```
1 xgb_y_pred = xgb.predict(X_test)
2 print(Counter(xgb_y_pred))
```

- Run the code cell.
- Examine the output.

```
Counter({False: 12287, True: 1267})
```

The gradient boosting model predicted 12,287 False values and 1,267 True values.

- Select the next code cell, then type the following:

```
1 results['term_deposit_pred'] = xgb_y_pred[:5]
2 results
```

- Run the code cell.

- g) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	jc
43308	False	False	0.0	1369.42	1	0	0	0	0	0
32770	False	False	2.0	246.00	0	0	0	1	0	0
17440	False	True	0.0	1369.42	0	0	0	0	0	0
36164	False	False	0.0	1369.42	0	0	0	1	0	0
29218	False	False	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

Of the first five records, the gradient boosting model correctly predicted all except the third.

10. Obtain the gradient boosting model's score.

- Scroll down and view the cell titled **Obtain the gradient boosting model's score**, then select the code cell below it.
- In the code cell, type the following:

```
1 accuracy_score(y_test, xgb_y_pred)
```

- Run the code cell.
- Examine the output.

```
0.8854950568097979
```

The gradient boosting model's accuracy score is ~89%. This is slightly higher than the random forest, and the highest one yet.

11. Keep this notebook open.

Hyperparameter Optimization

After you train a model, you'll want to tune its performance so that it can improve and be even more skillful at its assigned task. While there are several data processing methods for improving a model, like re-engineering the training data through cleaning, standardization, normalization, etc., one of the most significant improvement techniques during training itself is hyperparameter optimization. **Hyperparameter optimization** is the process of repeatedly altering the hyperparameters that an algorithm uses to train a model to determine the set of hyperparameters that lead to the best or desired level of model performance.

You could try to optimize the hyperparameters manually, evaluating performance after each iteration, but this can get tedious and is prone to error. There are several automated methods for hyperparameter optimization, including:

- Grid search
- Randomized search
- Bayesian optimization



Note: Hyperparameter optimization methods work for any algorithm that has hyperparameters, not just classification algorithms.

Grid Search

Grid search is a hyperparameter optimization method that simply takes a set (or grid) of parameter combinations, trains a model using each of those combinations, and then returns the combination that best optimizes an evaluation metric that you specify. Grid search also uses cross-validation to derive the optimal parameters by training over several folds. For example, you might construct a basic parameter grid for CART like the following:

```
grid = [ {'max_depth': [5, 6, 7],
          'min_samples_split': [10, 50, 100],
          'min_samples_leaf': [10, 50, 100]} ]
```

In this case, `max_depth`, `min_samples_split`, and `min_samples_leaf` are all separate hyperparameters. Each hyperparameter is used in all possible combinations. So, grid search will first train on the dataset $3 \times 3 \times 3 = 27$ times. The actual search object would look something like this:

```
search = GridSearchCV(model, param_grid = grid, scoring = 'accuracy', cv = 5)
```

Because `cv` is 5, grid search will do 5-fold cross-validation for each combination—in other words, it will train $27 \times 5 = 135$ times. The `scoring` argument is the evaluation metric you want to optimize the model on. So, in this case, you're looking for the combination of hyperparameters that will return the best accuracy score. That optimal combination is returned and can then be used to construct a more ideal model.



Note: If you're not sure what values to add to the grid when it comes to numerical hyperparameters, consider starting at a low value and then exponentially increase that value.

Randomized Search

While grid search will ultimately derive the optimal hyperparameters for a given metric, if your dataset has a very large feature space, grid search can take a very long time to exhaustively evaluate every possible combination of hyperparameters. **Randomized search** is an alternative approach in which random combinations of hyperparameters are used to train and evaluate the model. Rather than a grid of discrete values, hyperparameters in randomized search are typically defined in distributions; the algorithm will select a combination of values from these distributions for a certain number of specified iterations. Consider the following parameter distribution and accompanying randomized search object:

```
dist = { 'max_depth': [5, 6, 7],
          'min_samples_split': [10, 50, 100],
          'min_samples_leaf': sp_randint(10, 100) }

search = RandomizedSearchCV(model, param_distributions = dist, n_iter = 20,
                             scoring = 'accuracy', cv = 5)
```

The `max_depth` and `min_samples_split` hyperparameters are the same discrete values as they were before, but the values for `min_samples_leaf` are now a distribution of random integers that range from 10 to 100 (this is what SciPy's `sp_randint()` method does). For the `search` object, the `n_iter` argument specifies how many times a random combination of the hyperparameters is sampled—in this case, 20. As with grid search, this randomized search method will also perform cross-validation given a number of folds.

The fact that you can control the number of iterations of random hyperparameter sampling gives you more granular control over the training process, and subsequently, enables you to minimize training time in large feature spaces. While randomized search is not as thorough as grid search, and may not find the truly optimal hyperparameters in some cases, it will usually lead to adequate results.

Bayesian Optimization

Even randomized search is susceptible to training time issues, especially if you need to conduct many random sampling iterations over huge datasets. The algorithm needs to find a combination of random hyperparameters for each iteration, all while looking for a combination that optimizes the specified evaluation metric. **Bayesian optimization** is a hyperparameter optimization method that uses past samples to influence where sampling is conducted in subsequent iterations, enabling it to determine the next optimal space to sample from. This makes Bayesian optimization "smarter" than randomized search, because it is able to get to the optimal hyperparameters much faster than truly randomized sampling.

A simplified explanation of the Bayesian optimization process is as follows:

1. Start with an initial random sampling of the hyperparameter distribution space.
2. Evaluate the loss function (i.e., the chosen evaluation metric) for this space.
3. Use this evaluation to compute a posterior distribution of the loss function—in other words, capture the "beliefs" of the prior evaluation(s), then update those beliefs to account for everything that is currently known about the loss function.
4. Sample a new hyperparameter space that optimizes an acquisition function—in other words, given the posterior distribution, find the next space with the best tradeoff between loss optimum estimation and high estimation uncertainty.
5. Evaluate the loss from this new space.
6. Repeat steps 3 through 5 until some stopping criterion is met, such as a specified number of iterations or until the loss no longer improves.

Bayesian optimization is more complicated and harder to implement than grid search or randomized search, as it requires more careful tuning of the search configurations. However, it has been shown to be faster than grid search and randomized search in some scenarios.



Note: The default implementation of scikit-learn does not come with a Bayesian optimization library, but there are several third-party APIs that provide this functionality.

Guidelines for Tuning Classification Models

Follow these guidelines when tuning classification models.

Tune Classification Models

When tuning classification models:

- Use a search technique like grid search to find the optimal hyperparameters for a given model.
- Perform the search based on the metric(s) you're trying to optimize.
- Define a grid that includes the hyperparameters you want to try in combination.
- Prefer randomized search to cut down on search time, especially when there is a large field of values that you'd like to include in the search.
- Adjust the number of iterations in a randomized search, considering the tradeoff between search time and the quality of the results.
- Consider that Bayesian optimization can return results even faster than randomized search, but is more difficult to implement.

ACTIVITY 5–6

Tuning Classification Models

Before You Begin

Developing Classification Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Define the parameter grid used to tune the logistic regression model**. Select **Cell→Run All Above**.

Scenario

You've built several different models to classify the `term_deposit` target variable. Each model is different in how it makes its predictions, as well as its success at doing so. However, effective machine learning employs an iterative tuning process, where such models can get better when they're configured properly. You'll go through this tuning process by determining optimal hyperparameters for two different models: the logistic regression model and the gradient boosting model. The former obviously performed very poorly at first, so it should only get better through tuning. The latter has the highest score thus far, but may be susceptible to overfitting (as tree-based models often are). So, you'll see if you can improve that model as well, while also reducing overfitting through cross-validation.

1. Define the parameter grid used to tune the logistic regression model.

- Scroll down and view the cell titled **Define the parameter grid used to tune the logistic regression model**, then select the code cell below it.
- In the code cell, type the following:

```

1 solvers = ['newton-cg', 'lbfgs', 'liblinear']
2 penalty = ['l2']
3 c_values = [10, 1.0, 0.1, 0.01]
4
5 param_grid = dict(solver = solvers, penalty = penalty, C = c_values)
6
7 print(param_grid)

```

You'll use hyperparameter searching methods to find the optimal hyperparameters for the logistic regression model. In this code, you'll start by defining a grid of the different hyperparameters to try:

- `solvers` defines the type of methods that the model will use to minimize errors and find the optimal model parameters.
 - `penalty` is the type of regularization to use, which is a method of minimizing overfitting.
 - `c_values` define different strength values for the regularization.
- Run the code cell.
 - Examine the output.

```
{'solver': ['newton-cg', 'lbfgs', 'liblinear'], 'penalty': ['l2'], 'C': [10, 1.0, 0.1, 0.01]}
```

The hyperparameter grid is printed.

2. Perform a randomized search for optimal hyperparameters.

- Scroll down and view the cell titled **Perform a randomized search for optimal hyperparameters**, then select the code cell below it.
- In the code cell, type the following:

```
1 model = LogisticRegression()
2 random_search = RandomizedSearchCV(estimator = model,
3                                     param_distributions = param_grid)
4 random_search.fit(X_train_norm, y_train_SMOTE)
```

The first search you'll do is a randomized search, in which multiple models are trained on a random distribution of the specified hyperparameters for a fixed number of iterations. The parameter grid you defined in the previous block will be used, and the metric it will attempt to optimize is, by default, accuracy. This will not always determine the best possible hyperparameters, but it usually gets pretty close, and is much faster than an exhaustive search like grid search. The `RandomizedSearchCV()` function in scikit-learn also performs cross-validation by default.

- Run the code cell.
- Select the next code cell, then type the following:

```
1 # Summarize the results of the randomized search.
2
3 print('Best accuracy score:', round(random_search.best_score_, 4))
4 print('Best parameters: ', random_search.best_params_)
```

- Run the code cell.
- Examine the output.

```
Best accuracy score: 0.8949
Best parameters: {'solver': 'newton-cg', 'penalty': 'l2', 'C': 1.0}
```

The highest accuracy achieved in the search was a model that produced an accuracy of ~89%. This is much higher than the original logistic regression model, and is comparable to the original gradient boosting model you trained. You can also see which hyperparameters were used in the optimal model:

- 'newton-cg' as the solver.
- 'l2' as the regularization penalty.
- 1.0 as the C regularization strength.



Note: 'l2' uses the lowercase letter "L", not the number 1.

3. Perform a grid search for optimal hyperparameters.

- Scroll down and view the cell titled **Perform a grid search for optimal hyperparameters**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 model = LogisticRegression()
2 grid_search = GridSearchCV(estimator = model,
3                             param_grid = param_grid)
4 logreg_fit = grid_search.fit(X_train_norm, y_train_SMOTE)
```

Unlike a randomized search, a grid search will always perform an exhaustive search of the entire parameter grid, so it will always return the optimal hyperparameters from that grid. You'll perform a grid search using the same parameter grid to see if it does any better than the randomized search. Also, `GridSearchCV()` performs cross-validation to minimize overfitting.

- c) Run the code cell.
d) Select the next code cell, then type the following:

```
1 # Summarize the results of the grid search.
2
3 print('Best accuracy score:', round(grid_search.best_score_, 4))
4 print('Best parameters: ', grid_search.best_params_)
```

- e) Run the code cell.
f) Examine the output.

```
Best accuracy score: 0.8949
Best parameters: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
```

As you can see, the results are the same. The randomized search is therefore sufficient for this particular use case.

4. Tune the gradient boosting model to reduce overfitting.

- a) Scroll down and view the cell titled **Tune the gradient boosting model to reduce overfitting**, then select the code cell below it.
b) In the code cell, type the following:

```
1 model = XGBClassifier(eval_metric = 'logloss')
2
3 param_grid = {
4     'n_estimators': [10],
5     'max_depth': [15, 20, 25],
6     'reg_alpha': [1.1, 1.2, 1.3],
7     'reg_lambda': [1.1, 1.2, 1.3]
8 }
```

You'll perform another grid search, but this time on the XGBoost model. This can help you select the optimal hyperparameters to use, and the cross-validation will also hopefully reduce overfitting. Since this model uses a different algorithm than logistic regression, it has a different set of hyperparameters, so you need to create a new grid. In this case, the hyperparameters you'll be testing include:

- `n_estimators` is the number of individual decision trees in the model.
 - `max_depth` specifies the maximum depth of each tree.
 - `reg_alpha` specifies the weights of a certain type of regularization.
 - `reg_lambda` specifies the weights of a different type of regularization.
- c) Run the code cell.

- d) Select the next code cell, then type the following:

```

1 gs = GridSearchCV(estimator = model,
2                     param_grid = param_grid,
3                     n_jobs = -1,
4                     scoring = 'accuracy',
5                     verbose = 2)
6
7 fitted_model = gs.fit(X_train_SMOTE, y_train_SMOTE)

```

As before, this sets up the grid search. The `n_jobs` argument is telling scikit-learn to use all available CPU threads on the machine, as this search is resource intensive.

- e) Run the code cell.



Note: It can take up to 15 minutes for the search to complete.

- f) Examine the output.

```

Fitting 5 folds for each of 27 candidates, totalling 135 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 37 tasks      | elapsed: 3.2min
[Parallel(n_jobs=-1)]: Done 135 out of 135 | elapsed: 12.8min finished

```

Since there are 27 different combinations of hyperparameters specified in the grid, and because the default `k` in the `k`-fold cross-validation that `GridSearchCV()` uses is 5, the model is fit a total of $27 \times 5 = 135$ times.

- g) Select the next code cell, then type the following:

```

1 print('Best accuracy score:', round(gs.best_score_, 4))
2 print('Best parameters:    ', gs.best_params_)

```

- h) Run the code cell.

- i) Examine the output.

```

Best accuracy score: 0.9092
Best parameters:    {'max_depth': 25, 'n_estimators': 10, 'reg_alpha': 1.1, 'reg_lambda': 1.3}

```

The XGBoost accuracy score for the optimal model is now ~91%. The optimal hyperparameters are:

- 10 as the number of decision trees (`n_estimators`).
- 25 as the `max_depth` of each tree.
- 1.1 as the `reg_alpha` strength.
- 1.3 as the `reg_lambda` strength.

5. Keep this notebook open.

TOPIC B

Evaluate Classification Models

After training and tuning machine learning models, you'll need to assess how well they perform. There is not just one way to measure a classification model's skill, but many, and you'll need to know how to use them effectively.

Evaluation Metrics

In data science and machine learning, **evaluation metrics** are used to assess the skill, performance, and characteristics of a model. They do this by measuring some aspect of a model and/or its results. There are many such evaluation metrics, each one based on a different measurement. You're not limited to using just one metric, but each type of machine learning outcome (regression, classification, clustering, etc.) has its own metrics. So if you're training a classification model, it won't do much good to use a clustering metric like silhouette analysis.

Evaluation metrics are crucial to developing any type of machine learning model, regardless of whether the model is supervised or unsupervised. Supervised metrics tend to evaluate the estimations a model makes on the validation and/or test sets. Since these sets are labeled, and were not used to train the model, you can find out how closely the model gets to the ground truth. Metrics usually report this level of performance in the aggregate—in other words, you won't be evaluating how the model performs on a single data example, but all data examples in the set combined. Unsupervised metrics tend to look at the model's characteristics, as there are no labels with which to measure performance.

Evaluating model performance is important for a couple of major reasons. First, you can determine whether or not you are "satisfied" with the model according to whatever expectations you've established or business needs you've been tasked with fulfilling. If the model doesn't meet the desired level of performance, you may abandon it. Alternatively, the other reason to use evaluation metrics is to inform the tuning process. Rather than tune hyperparameters at random and see what happens, you'll be able to tune those hyperparameters in order to maximize one or more metrics. Recall that a method like grid search takes an evaluation metric as input—this is what the search is trying to optimize as it searches through the given hyperparameter field. The ultimate goal of this process is to iteratively improve the model until it meets your expectations.

Goodhart's Law

Goodhart's Law, named after economist Charles Goodhart, can essentially be paraphrased as: "When a measure becomes a target, it ceases to be a good measure." In other words, if you focus too much on achieving good results for a specific measurement, then the measurement itself becomes the goal. If you attempt to exert too much control over a system through one metric, this could lead to a false sense of accomplishment. In the field of machine learning, you might optimize a model so that it performs extremely well on one metric, but it might also perform poorly on other, equally important metrics. The model's overall skill could suffer as a result. So, as you learn about the various metrics in machine learning, keep in mind that there is no one objectively "perfect" metric for every circumstance, and that you may need to consider multiple metrics in the evaluation process.

Classification Model Performance

Let's say you have a classifier that attempts to predict whether or not a physical storage device is going to malfunction within 90 days of being put into production. The model was trained on a historical dataset where each example is a storage device and each feature is that device's characteristics (e.g., type of storage technology, brand, capacity, etc.). Each device is labeled as to

whether or not it malfunctioned after 90 days of use. The model therefore classifies malfunctioning devices as "Yes" (positive), and working devices as "No" (negative). As you train a classifier, there are various ways the model might succeed or fail, as depicted in this table.

True (Correct) Label	Estimated Label	Model Result	Assessment
Yes	Yes	Success	True positive (TP)
No	No	Success	True negative (TN)
No	Yes	Failure	False positive (FP)
Yes	No	Failure	False negative (FN)

Considerations When Choosing Classification Metrics

When training a classifier, you must evaluate how well it performs—not only when it succeeds in identifying positives or negatives, but also when it fails to correctly identify positives and negatives. Ideally, every estimation made by the model would be correct. But in the real world that is seldom possible, so you must tune the model to make compromises that will meet your needs. You tune the model to ensure correct identifications where they are essential, at the expense of allowing failures where they can be tolerated.

For example, in some situations, it might be better to have a model that ensures there will be no false negatives at the expense of allowing perhaps 30% of positive estimations to be false. You'd probably want your machine learning model to identify every storage device that has a problem before it leads to data loss (especially if you don't have adequate backups). Preserving data by replacing the device would outweigh the cost of replacing devices that don't actually need replacing. In other situations, it might be better to ensure there are no false negatives while allowing a small number of false positives.

Having various ways to measure how the model performs will help you optimize it for a particular situation.

Confusion Matrix

A [confusion matrix](#) is a method of visualizing the truth results of a classification problem. It helps you more easily examine the dimensions of a classification problem, as well as enumerate true positives/negatives and false positives/negatives within the estimated data. In a confusion matrix, the number of estimations is compared to the number of actual values in a table format. There are several ways you can structure this table; the following is one example, using a model with two classes (a binary classifier).

		Estimation	
		No	Yes
Actual	No	True negatives	False positives
	Yes	False negatives	True positives

Figure 5–15: A confusion matrix template.

Recall that the example mentioned previously has to do with predicting storage device failure. After training the model on the data, it'll make predictions that end up either being true or false and negative or positive. Plugging those values into a confusion matrix, you might get something like the following.

		Estimation	
		Device didn't fail	Device failed
Actual	Device didn't fail	513	8
	Device failed	4	17

Figure 5–16: A confusion matrix of storage device failure predictions.

Using a table like this, it's much easier to identify the prevalence of errors in the model's prediction. To summarize the example:

- 513 drives were correctly predicted to not have failed (true negatives).
- 8 drives were incorrectly predicted to have failed when they actually did not (false positives).
- 4 drives were incorrectly predicted to not have failed when they actually did (false negatives).
- 17 drives were correctly predicted to have failed (true positives).

Accuracy

Accuracy is a measure of how frequently each classification is correctly deemed positive or negative.

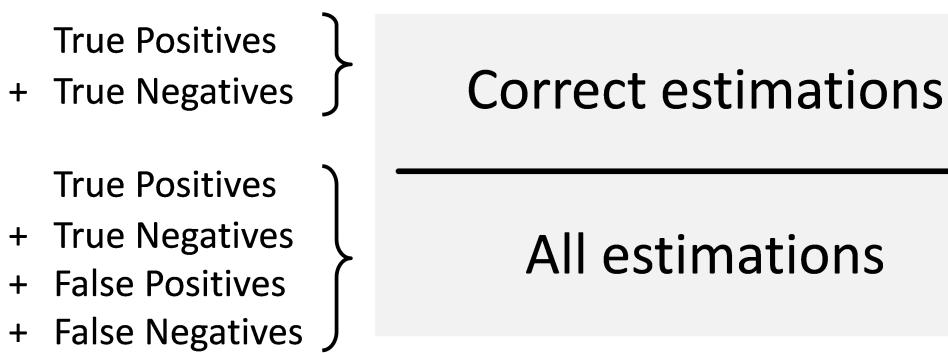


Figure 5–17: How to calculate accuracy.

As shown here, accuracy is the number of correct estimations divided by the number of all estimations made. In other words, accuracy can be expressed as:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

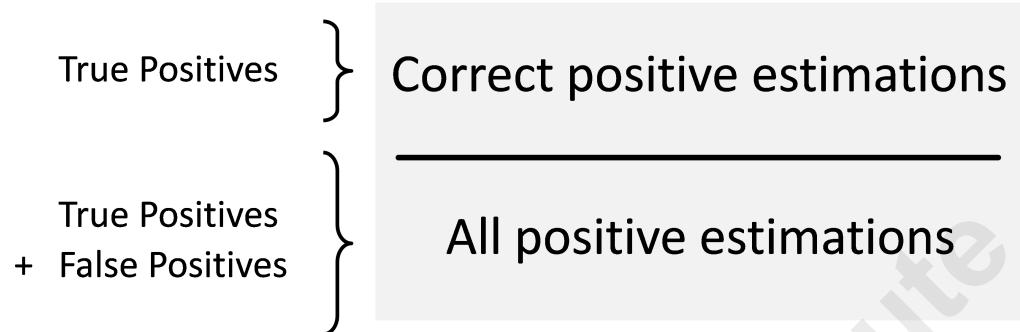
If accuracy is perfect (all estimations are correct), it will be measured as 1.0. If half the predictions are correct, the accuracy will be 0.5. These can be converted into percentages as needed.

Accuracy is intuitive, and the word itself has the connotation of being highly desirable. However, accuracy often proves to be an unreliable measure of model performance. Consider the example of identifying faulty storage drives to predict those that might fail. A model trained on this data would probably end up with a great many true negatives and almost no true positives, false positives, or false negatives, because drive failures within the first 90 days are very rare.

As a result, the accuracy is extremely high—almost 98%. This makes accuracy nearly useless, as the machine learning model will be no better at predicting the rare cases in which a storage device shows signs of imminent failure. Accuracy is therefore only useful in datasets where the label data is balanced.

Precision

Precision is a measure of how often the positives identified by the learning model are true positives.

**Figure 5–18: How to calculate precision.**

As shown here, precision is the number of correct positive estimations divided by the number of all positive estimations made. In other words, precision can be expressed as:

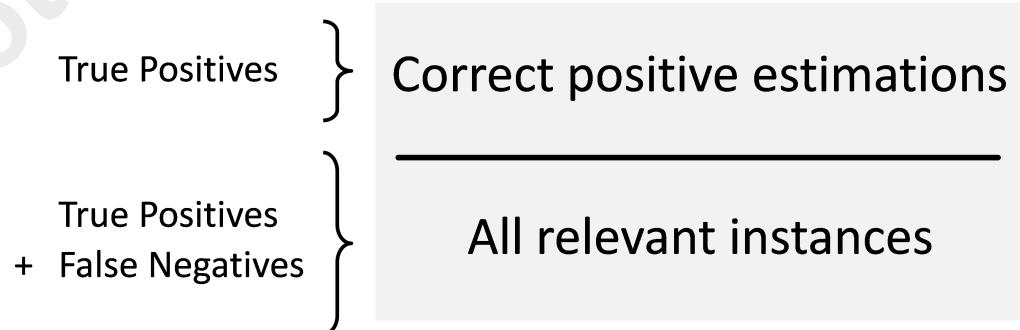
$$\frac{TP}{TP + FP}$$

Like accuracy, precision can be expressed as a number from 0 to 1.0 or as a percentage.

Precision is typically more useful than accuracy, especially in asymmetric datasets, but you still need to consider the context of the data and what outcomes you're looking for. Using the dataset, the model correctly predicted that 17 drives were going to fail within 90 days. It also predicted that 8 devices were about to fail when they actually weren't. So, the precision would be $17 / 17 + 8$, or **68%**. This is more useful than accuracy, because it accounts for the unbalanced nature of the dataset with regard to the label. However, the problem with precision in this case is that it doesn't address cases in which a drive will fail but the model doesn't think it will (i.e., false negatives). Even just a single drive malfunctioning is undesirable because of the potential consequences it can have for the business—if the model doesn't predict that instance, the model may be considered a failure. Even if you set your tolerance higher than just one drive failure, precision will still come up short in assessing this model's performance.

Recall

Recall is the percentage of positive instances that are found by a model as compared to all relevant instances. A "relevant" instance is any instance that is actually true, even if the estimation is wrong.

**Figure 5–19: How to calculate recall.**

As shown here, recall is the number of correct positive estimations divided by the number of correct positive estimations plus the number of incorrect negative estimations. In other words, recall can be expressed as:

$$\frac{TP}{TP + FN}$$

Like accuracy and precision, recall can be expressed as a number from 0 to 1.0 or as a percentage.

As you've seen, precision might not be the most useful way to measure how good a machine learning model is at predicting drive failure. Let's try recall. In the example, assume the model was able to correctly predict that 17 storage devices would malfunction. The model failed to predict that 4 additional devices would malfunction. The recall would be $17 / (17 + 4)$, or around 81%. Now, you have a better idea of how well your model performs with respect to its ultimate purpose—minimizing drive failure and the business impact it can have. This is because recall focuses on false negatives. Technically, the model could improve its recall by predicting that all drives will fail within 90 days, leading to a recall of 100%, but this would make the model useless for prioritizing replacement efforts. Also, recall doesn't minimize false positives as well as precision does.

Precision–Recall Tradeoff

In general, there is tradeoff between precision and recall. As you tune a machine learning model to emphasize *one* of these factors, you do so at the expense of de-emphasizing the *other*, as depicted through this table and chart.

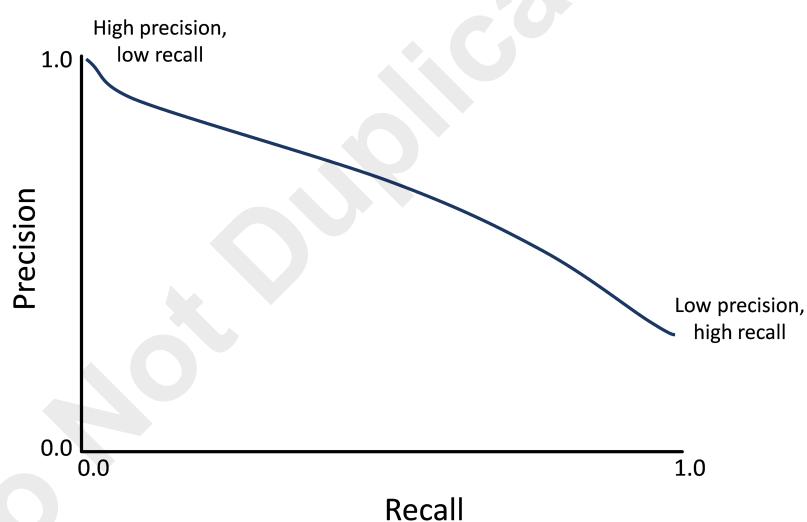


Figure 5–20: The precision and recall tradeoff.

When	False Positives	False Negatives
Precision increases	Tend to decrease	Tend to increase

When	False Positives	False Negatives
Recall increases	Tend to increase	Tend to decrease

Which factor you choose to emphasize depends on your business requirements. The drive failure model may be more conducive to recall, but there are plenty of other datasets and scenarios where precision (or some other metric) would be the better choice.

F₁ Score

As you've seen, precision and recall are more useful in unbalanced datasets, but they come with a tradeoff. Sometimes, like in the storage drive example, it's relatively clear which one is more useful. However, this is not always the case. Consider a machine learning task in which you want to determine whether or not a novel is in the science fiction genre, given a short text sample. A false positive (e.g., classifying a novel as science fiction when it's not) is just as undesirable as a false negative (e.g., not classifying a novel as science fiction even though it is); neither is particularly worse. So, what's the best way to measure performance in this case?

The **F₁ score** helps you find the optimal combination of both precision and recall. The F₁ score essentially just takes a weighted average (more precisely, a harmonic mean) of both precision and recall. The weighted average reduces the effect of extreme values. The F₁ score can be expressed as:

$$F_1 = 2 \left(\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \right)$$

So, assume that 143 novels were correctly classified as science fiction, and 21 novels were incorrectly classified as science fiction. The precision is ~87%. If 38 novels were not classified as science fiction even though they are, then the recall would be ~79%. Plug these values into the formula like so:

$$F_1 = 2 \left(\frac{.87 \cdot .79}{.87 + .79} \right)$$

The resulting F₁ score is ~.828 or around 83%.

To reiterate, the F₁ score is preferred when label values in the dataset are not distributed evenly, *and* neither precision nor recall are more useful than the other.

Specificity

Specificity, also called the **true negative rate (TNR)**, is a measure of how often the model identifies the actual negatives.

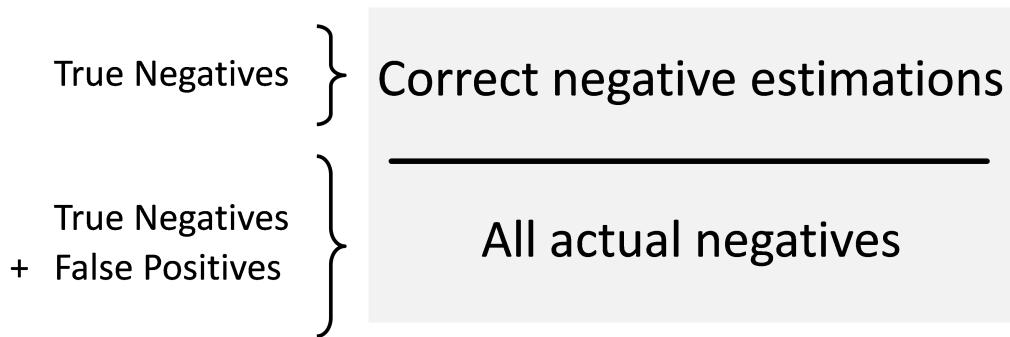


Figure 5-21: How to calculate specificity.

As shown here, specificity is the number of correct negative estimations divided by the total number of actual negatives. In other words, specificity can be expressed as:

$$\frac{\text{TN}}{\text{TN} + \text{FP}}$$

Just like the other classification metrics you've seen thus far, specificity can be expressed as a number from 0 to 1.0 or as a percentage.

In the storage drive example, you would calculate specificity as $513 / 513 + 8$, or around 98%. However, specificity is useful for when you need to maximize the amount of true negatives the model produces. Also, like accuracy, it doesn't do so well when the label in the dataset is imbalanced. So, the storage device example is not really a good candidate for specificity.

Consider a different scenario in which you have a model that predicts whether or not customers will be satisfied by a new food product. The label is positive when the customer is satisfied, and negative when they are not. The labeled responses seem to be balanced, where roughly half of the customers like the new item. In this case, you might want your model to maximize true negatives so that you can more easily offer alternative items to the unsatisfied segment of your customer base, or take some other action that will reduce customer attrition. Therefore, specificity might be a good metric to optimize.

Receiver Operating Characteristic (ROC) Curve

A **receiver operating characteristic (ROC) curve** is a method of plotting the relationship between estimated "hits" versus false alarms. On the y-axis is the **true positive rate (TPR)** (the "hits"), which is the same as the recall. On the x-axis is the **false positive rate (FPR)** (the false alarms), which is expressed as:

$$\frac{\text{FP}}{\text{FP} + \text{TN}}$$

Once these two values for a model are plotted on the graph, a line can be fit to the data. This line starts from the bottom left of the graph (0 FPR, 0 TPR) and continues to the top right of the graph (1 FPR, 1 TPR). Once the line is fit, it becomes the ROC curve.

ROC curves are useful because they help you tune a machine learning model to achieve the desired balance between the TPR and the FPR. In particular, they can help you evaluate two things. First, they can help you compare the performance of two or more models. Consider the following graph:

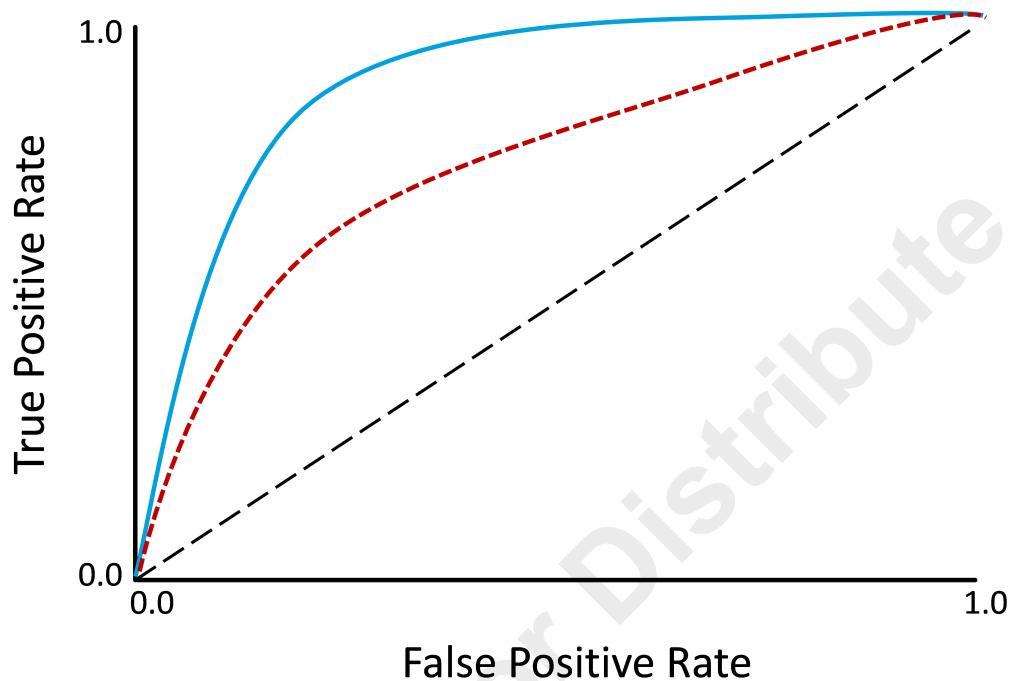


Figure 5–22: ROC curves plotted on a graph.

The diagonal dashed line (0.5, 0.5) represents a random guess. Therefore:

- Any data above and to the left of the dashed line is *better* than a random guess. This is what you want for your model.
- Any data near or at the dashed line is *no better* than a random guess. This is not what you want for your model.
- Any data below and to the right of the dashed line is *worse* than a random guess. This typically means there's a problem with the model.

A perfectly estimative model would be fit from the top-left corner of the graph (0, 1). This is unrealistic, but the more a model's curve approaches this top-left corner, the better it performs. So, in the figure, the model represented by the unbroken blue curve performs better than the model represented by the dashed red curve.



Note: You can use a confusion matrix to easily calculate the TPR and FPR.

Thresholds

The second way to use a ROC curve is to evaluate a model by itself, not in comparison to any other model. After all, one model may perform better than another, but still perform poorly overall. You could do an independent evaluation by configuring a **threshold**. Any data above this threshold is considered positive; anything below, negative. In other words, the threshold creates a decision boundary. In the case of predicting storage device failure, you could set a threshold of 0.70. If the model makes a prediction of device failure with a value of 0.75, that will be considered a 1 (will fail). If the prediction is 0.65, then that will be considered a 0 (will not fail).

The problem with this method is that it can be very difficult to actually decide what a good threshold is for your data. As explained before, you'll want to treat data differently depending on what it represents. In the case of device failure, you would want to put much more weight on

minimizing false negatives rather than false positives. So, what threshold would you set? Set it too low and the model will predict device failure where there is none. This is due to a high true positive rate (TPR)/recall, also called **sensitivity**, which leads to a low true negative rate (TNR)/specificity. Set the threshold too high and the model will miss drives that are about to fail. This is due to a high specificity, which leads to low sensitivity. The problem lies in judging the tradeoff between the two.

In this first example, the threshold is 0.50—right in the middle. The red Xs represent the actual negative values in the dataset, and the green check marks represent the actual positive values in the dataset. So, with this threshold, one false positive and two false negatives are generated by the estimative model.

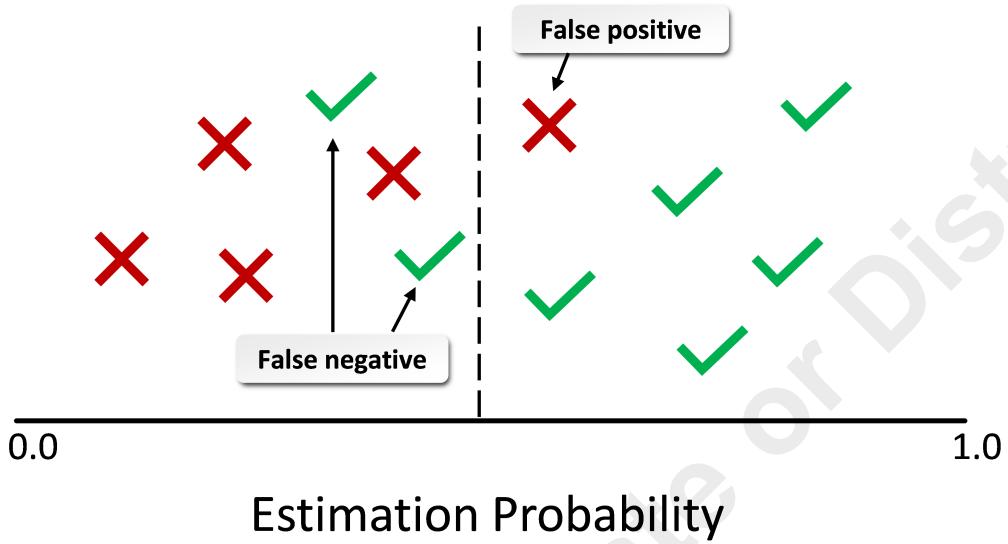


Figure 5–23: Plotting classification values with a threshold of 0.50.

Now, in this second example, the threshold is increased to around 0.70. Because of this, the false positive from before has been eliminated, and its data instance is now a true negative. However, this also added one more false negative.

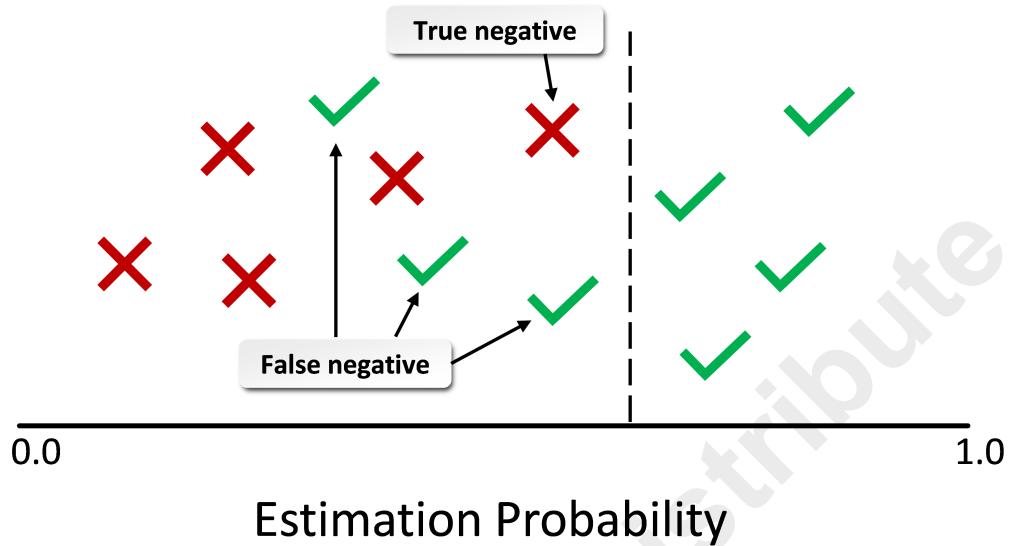


Figure 5-24: Moving the threshold to ~0.70.

Thankfully, there is another approach.

Area Under Curve (AUC)

The [area under curve \(AUC\)](#) is, as its name suggests, the total space that is under a model's ROC curve. The AUC can also be defined as the probability that the model will classify any positive instance higher than any negative instance, from 0 to 1.0. This minimizes the need to decide on a threshold, because the AUC considers *all* possible thresholds. So, AUC is a useful way to evaluate the overall performance of a classification model.

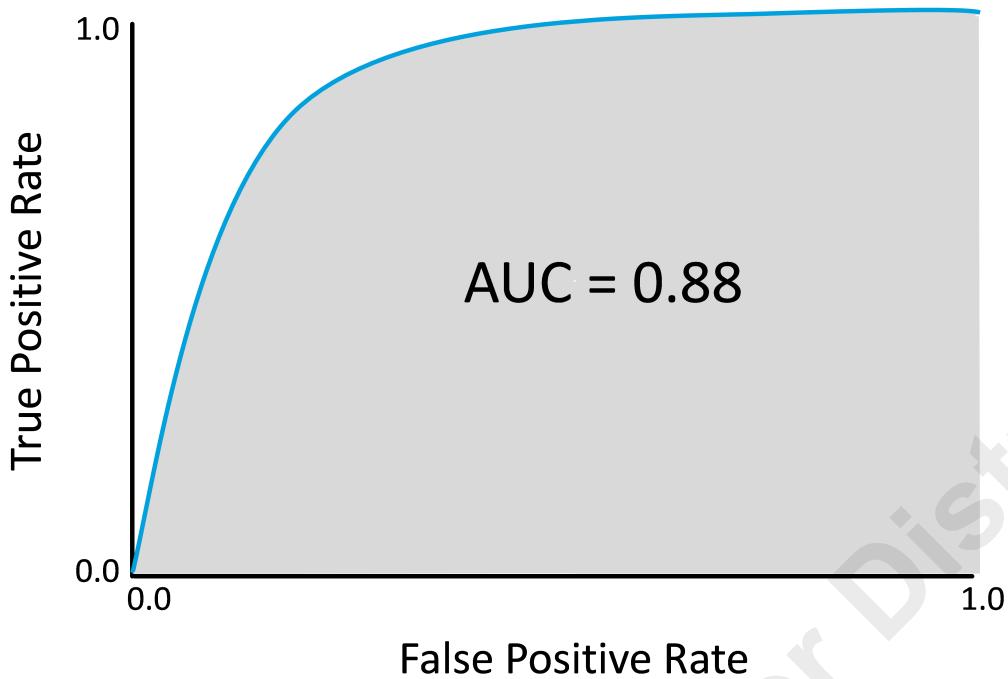


Figure 5–25: The AUC in this example is 0.88.

The AUC must at least be above 0.5 (random guessing), but there is no gold standard that applies to all data. You need to, once again, consider the context of your data and what it would mean to raise or lower the expected AUC. Although storage device failure can have a significant business impact, it's not necessarily a life-and-death situation. You might therefore accept a reasonably high AUC (>0.8), while still leaving some room for error. If you had a model that diagnosed fatal diseases in medical patients, you would only accept a very high AUC (>0.9) due to the highly critical nature of the problem. Consider researching AUC values for your particular industry or for the particular problem you are trying to solve.

It's important to note that the AUC of a ROC curve is not useful in datasets where there is a class imbalance, as it may be better to focus on minimizing one classification error rather than optimizing for multiple errors.



Note: Several math programming libraries can calculate the AUC for you.

Learning Curve

A **learning curve** is a method of visually comparing the number of data examples to the changing score of some metric. Along the x-axis is the number of training examples, and along the y-axis is whatever metric you're evaluating the model with (e.g., accuracy, precision, recall, etc.). Two points are plotted: one for the training score (i.e., the model performs estimations on the training data after being trained on that data), and one for the validation score (typically using cross-validation). These scores are computed for several different sizes of the input data, then all of the points are connected via a line.

The purpose of a learning curve is twofold: it can help you determine whether adding more examples to the training dataset is likely to increase the model's score, and it can help you determine whether the model is more prone to errors of bias or errors of variance. For determining the effect of more data examples, you're looking for both score lines to converge. If they do converge at some

sample size, then that means the score is not really changing between training and validation, implying that adding more data examples won't improve the model. If the lines don't yet converge, it implies that there's still room to improve the model by adding more data.

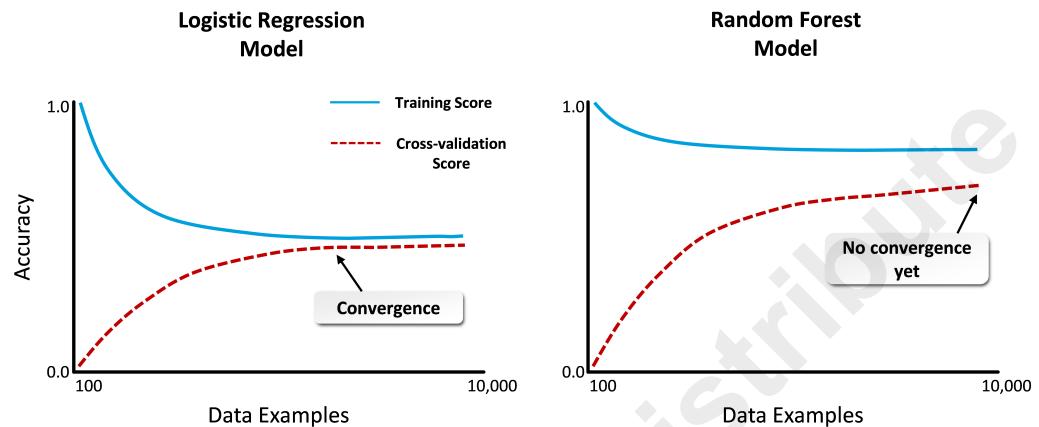


Figure 5-26: Comparing how the number of data examples impacts the skill of two different models.

In the figure, the logistic regression classifier on the left converges at around 6,000 data examples and its accuracy stays constant after that. It therefore won't benefit from more data. On the other hand, the random forest model on the right has a validation score that is still well below the training score after 10,000 examples, so it could benefit from more data.

Note that the training score is almost always going to be higher than the validation score, assuming that the "score" is any metric that evaluates how *well* the model performs. That's because the model is making estimations about the same data it learned from, so naturally it's going to be better at estimating that data. Higher overall training scores indicate low bias, whereas low training scores indicate high bias. Larger gaps between training and validation scores indicate high variance, whereas small gaps indicate low variance. Recall that high bias can lead to underfitting, and high variance can lead to overfitting. So you can use learning curves to help you identify these issues in your models.

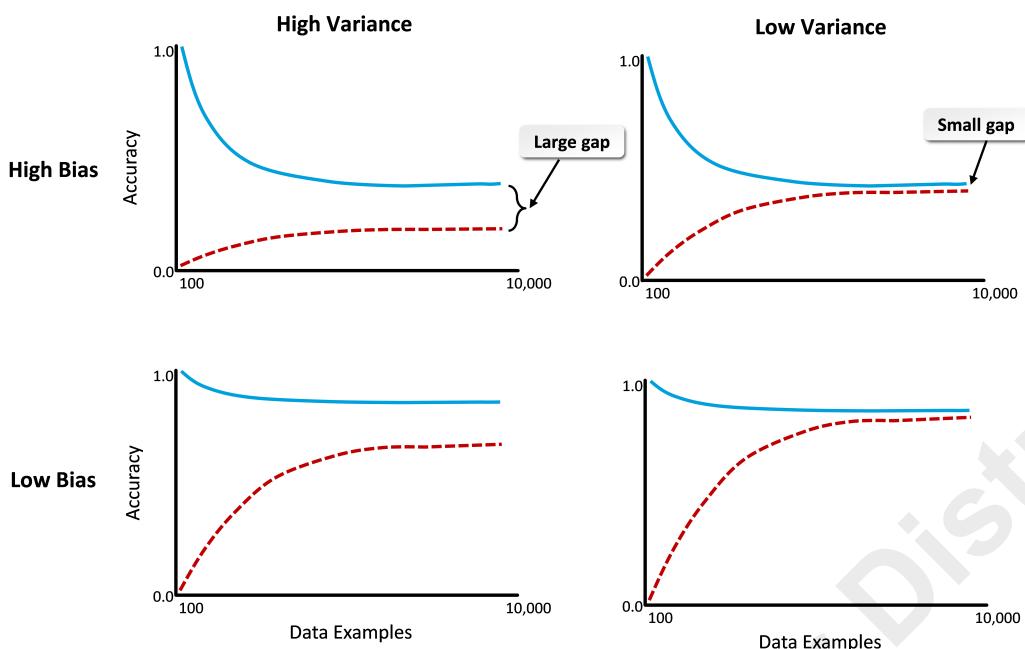


Figure 5–27: How bias and variance issues appear on a learning curve.



Note: Learning curves can evaluate any supervised model, not just classifiers. In the case of regression models, they measure error, so the training error will always be lower than the validation error. High errors indicate high bias, and vice versa.

Guidelines for Evaluating Classification Models

Follow these guidelines when evaluating classification models.

Evaluate Classification Models

When evaluating classification models:

- Consider the significance of the four truth values for a classification problem: true positive, true negative, false positive, false negative.
 - Use a confusion matrix to visualize these truth results for a classification problem.
- For individual metrics:
 - Consider that accuracy may not be useful in datasets with a class imbalance.
 - Prefer precision and recall when the dataset has a class imbalance.
 - Consider that recall may be better than precision when it's crucial that the model avoid false negatives (e.g., in predicting the presence of disease in patients).
 - Consider that there is a tradeoff between precision and recall—as one increases, the other tends to decrease.
 - Use F_1 score when neither precision nor recall is more important than the other.
 - Prefer specificity when you need to maximize the amount of true negatives in a balanced dataset.
 - Generate a ROC curve and its AUC to compare the true positive rate (TPR) with the false positive rate (FPR) for all thresholds.
 - Generate learning curves to identify whether or not adding more data examples will improve a model's skill.

- Generate learning curves to identify whether a model is high in bias, high in variance, or both.
- Overall:
 - Consider that there is no objectively "correct" evaluation metric for all problems, or even for a given type of problem.
 - Consider applying multiple metrics to the same problem.

Do Not Duplicate or Distribute

ACTIVITY 5–7

Evaluating Classification Models

Before You Begin

Developing Classification Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Compare evaluation metrics for each model**. Select **Cell**→**Run All Above**.

Scenario

Now that you've trained and tuned your models, you wanted to evaluate their performance using more than just accuracy. There are many useful classification metrics that might be relevant to your data. In particular, because you balanced the dataset through oversampling, and because neither recall nor precision are more valuable in the case of predicting users signing up for a term deposit, you decide to focus on F_1 score as the primary metric. Still, it's important to consider other metrics as well, so you'll get a more comprehensive look at your models.

1. Compare evaluation metrics for each model.

- Scroll down and view the cell titled **Compare evaluation metrics for each model**, then select the code cell below it.

- b) In the code cell, examine the following:

```

1 models = ['Logistic Regression', 'Naive Bayes', 'SVM', 'k-NN',
2           'Decision Tree', 'Random Forest', 'XGBoost', 'Dummy Classifier']
3
4 metrics = ['Accuracy', 'Precision', 'Recall', 'F1']
5
6 pred_list = ['logreg_y_pred', 'gnb_y_pred', 'svm_y_pred', 'knn_y_pred',
7               'clf_tree_y_pred', 'rf_y_pred', 'xgb_y_pred', 'dummy_y_pred']
8
9 # Baseline algorithm.
10 dummy = DummyClassifier(strategy = 'stratified')
11 dummy.fit(X_train_SMOTE, y_train_SMOTE)
12 dummy_y_pred = dummy.predict(X_test)
13
14 scores = np.empty((0, 4))
15
16 for i in pred_list:
17     scores = np.append(scores,
18                         np.array([[accuracy_score(y_test, globals()[i]),
19                                    precision_score(y_test, globals()[i]),
20                                    recall_score(y_test, globals()[i]),
21                                    f1_score(y_test, globals()[i])]]),
22                         axis = 0)
23
24 scores = np.around(scores, 4)
25
26 scoring_df = pd.DataFrame(scores, index = models, columns = metrics)
27 scoring_df.sort_values(by = 'F1', ascending = False)

```

This code, which has been provided for you, constructs a `DataFrame` that will neatly compare the metric scores for each of the models you've trained so far. In addition, a `DummyClassifier()` model is being trained. This model makes classification decisions based on very simplistic rules, and is purely used as a way to ensure that your actual machine learning models are meeting a baseline level of performance. Also, to simplify things, the tuned models have been left out for now.

- c) Run the code cell.

- d) Examine the output.

	Accuracy	Precision	Recall	F1
XGBoost	0.8855	0.5280	0.4122	0.4630
Random Forest	0.8826	0.5150	0.3383	0.4083
Decision Tree	0.8314	0.3309	0.3993	0.3619
Naïve Bayes	0.7019	0.1876	0.4473	0.2644
Logistic Regression	0.1248	0.1203	1.0000	0.2148
k-NN	0.2969	0.1123	0.7055	0.1937
Dummy Classifier	0.4964	0.1171	0.4904	0.1891
SVM	0.8803	0.0000	0.0000	0.0000

The accuracy, precision, recall, and F₁ scores are listed for every model. The table is sorted by F₁ score. Some conclusions you can draw include:

- The XGBoost model has the highest F₁ score, highest accuracy, and highest precision. For your purposes, you can consider this your "best" model.
- The random forest and SVM models are close behind when it comes to accuracy, but keep in mind that the SVM model was not very helpful since it predicted all False values.
- The dummy classifier and k-NN have very low F₁ scores.
- Although logistic regression has very poor accuracy, it has perfect recall. This means that it had no false negatives; i.e., it never predicted that a user wouldn't sign up for a term deposit when they actually would. This is likely due to the fact that it predicted many True values and very few False values.
- Overall, it seems like the tree-based algorithms, particularly the ensemble methods, performed the best with your data.

2. Generate a confusion matrix.

- a) Scroll down and view the cell titled **Generate a confusion matrix**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 confusion_matrix(y_test, xgb_y_pred)
```

You'll focus on your best model, the XGBoost model, for the rest of your evaluation efforts. This code will show a confusion matrix of the classification truth values.

- c) Run the code cell.
 d) Examine the output.

```
array([[11333,    598],
       [  954,   669]])
```

This confusion matrix shows the raw values for each truth assessment. It'll be easier to read when you use a visualization function to plot the matrix, however.

- e) Select the next code cell, then type the following:

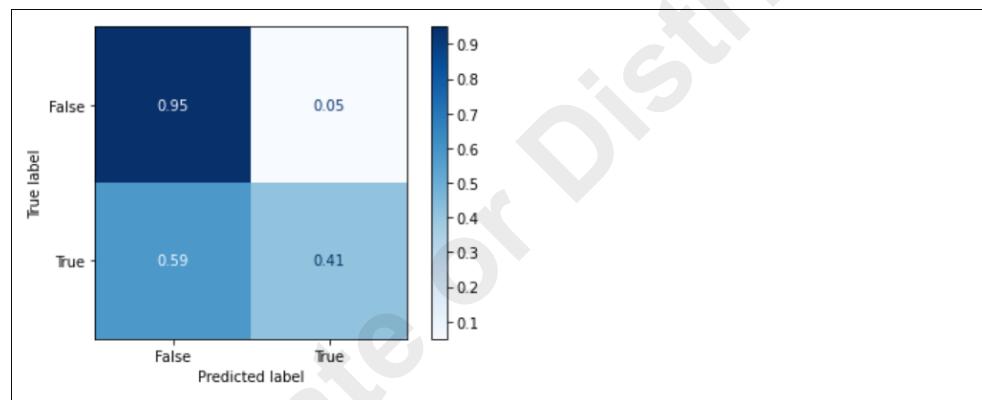
```

1 plot_confusion_matrix(xgb,
2                         X_test,
3                         y_test,
4                         cmap = plt.cm.Blues,
5                         normalize = 'true')
6
7 plt.show();

```

The confusion matrix will be labeled and have a color map to indicate magnitude in each quadrant. Also, the values will be normalized so that they show a percentage of the total row rather than the raw values.

- f) Run the code cell.
g) Examine the output.



The results can be interpreted like so:

- In the top-left quadrant, 95% of actual False values (i.e., user *did not* sign up for a term deposit) were correctly predicted to be False by the model. These are the true negatives.
- In the top-right quadrant, the remaining 5% of actual False values were incorrectly predicted to be True by the model. These are the false positives.
- In the bottom-left quadrant, 59% of actual True values (i.e., user *did* sign up for a term deposit) were incorrectly predicted to be False by the model. These are the false negatives.
- In the bottom-right quadrant, the remaining 41% of actual True values were correctly predicted to be True by the model. These are the true positives.

This supports the scores you saw earlier. The largest "mistake" is in the bottom-left quadrant, which shows the false negatives. As false negatives increase, the recall score decreases.

3. Plot a ROC curve.

- a) Scroll down and view the cell titled **Plot a ROC curve**, then select the code cell below it.
b) In the code cell, type the following:

```

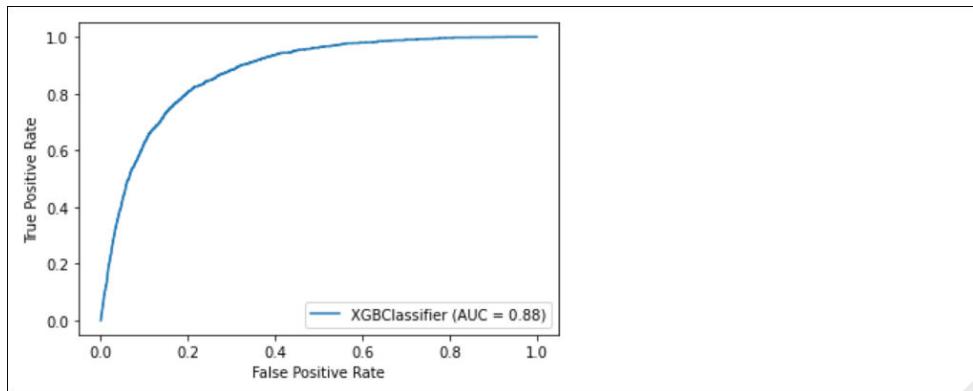
1 plot_roc_curve(xgb, X_test, y_test)
2 plt.show()

```

This will plot a ROC curve and show its AUC.

- c) Run the code cell.

- d) Examine the output.



The ROC curve, since it sweeps up and to the left, has a shape that suggests it is much better than a random guess at distinguishing positive and negative classes. Its high AUC of 0.88 also supports that.

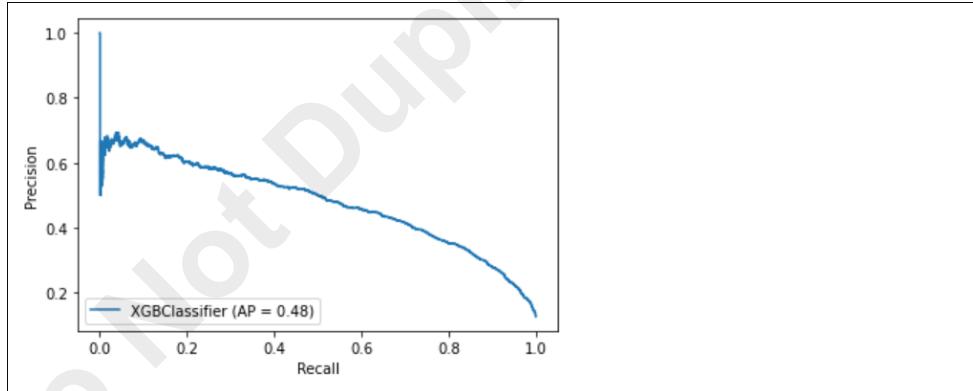
4. Plot a precision–recall curve.

- a) Scroll down and view the cell titled **Plot a precision–recall curve**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 plot_precision_recall_curve(xgb, X_test, y_test)
2
3 plt.show();
```

This plots a PRC to show the balance between precision and recall at different thresholds.

- c) Run the code cell.
 d) Examine the output.



The curve shows that the balance between precision and recall is somewhat low. However, PRCs are most useful in evaluating imbalanced datasets, so this isn't necessarily a problem for the oversampled dataset.

5. Generate a feature importance plot.

- a) Scroll down and view the cell titled **Generate a feature importance plot**, then select the code cell below it.

- b) In the code cell, examine the following:

```

1 def feature_importance_plot(model, X_train, n):
2     """Plots feature importance. Only works for ensemble learning."""
3     plt.figure(figsize = (8, 5))
4     feat_importances = pd.Series(model.feature_importances_,
5                                   index = X_train.columns)
6     feat_importances.nlargest(n).plot(kind = 'barh')
7     plt.title(f'Top {n} Features')
8     plt.show()

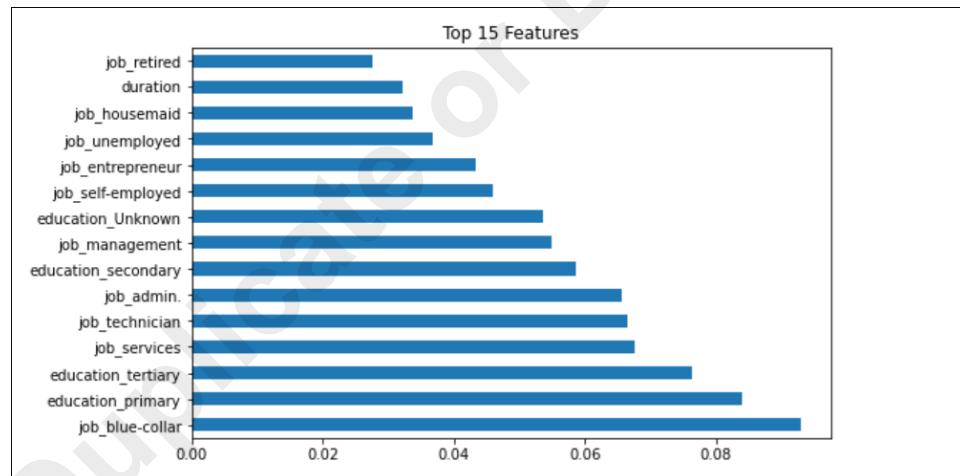
```

This code has been provided for you. It's a function that will plot feature importances for the given model as a bar chart.

- c) Run the code cell.
d) Select the next code cell, then type the following:

```
1 feature_importance_plot(xgb, X_train_SMOTE, 15)
```

- e) Run the code cell.
f) Examine the output.



For the XGBoost model, features like `job_blue-collar`, `education_primary`, and `education_tertiary` contribute the most to the model's predictions. Features with low importance, like `job_retired`, could potentially be removed in order to improve performance.

6. Plot learning curves.

- a) Scroll down and view the cell titled **Plot learning curves**, then select the code cell below it.

- b) In the code cell, examine the following:

```

1 def plot_learning_curves(model, X_train, y_train):
2     """Plots learning curves for model validation."""
3     plt.figure(figsize = (5, 5))
4     train_sizes, train_scores, test_scores = \
5         learning_curve(model, X_train, y_train, cv = 5,
6                         scoring = 'accuracy', n_jobs = -1,
7                         shuffle = True,
8                         train_sizes = np.linspace(0.01, 1.0, 5))
9
10    # Means of training and test set scores.
11    train_mean = np.mean(train_scores, axis = 1)
12    test_mean = np.mean(test_scores, axis = 1)
13
14    # Draw lines.
15    plt.plot(train_sizes, train_mean, '--',
16              color = '#1f77b4', label = 'Training score')
17    plt.plot(train_sizes, test_mean,
18              color = '#1f77b4', label = 'Cross-validation score')
19
20    # Create plot.
21    plt.title('Learning Curves')
22    plt.xlabel('Training Set Size')
23    plt.ylabel('Accuracy Score')
24    plt.legend(loc = 'best')
25    plt.tight_layout()
26
27    plt.show()

```

This function, which has been provided for you, generates learning curves for the XGBoost model. Learning curves generate learning performance over time and can be used to identify models that might benefit from more training examples, as well as models that may be overfitting or underfitting.

- c) Run the code cell.
d) Select the next code cell, then type the following:

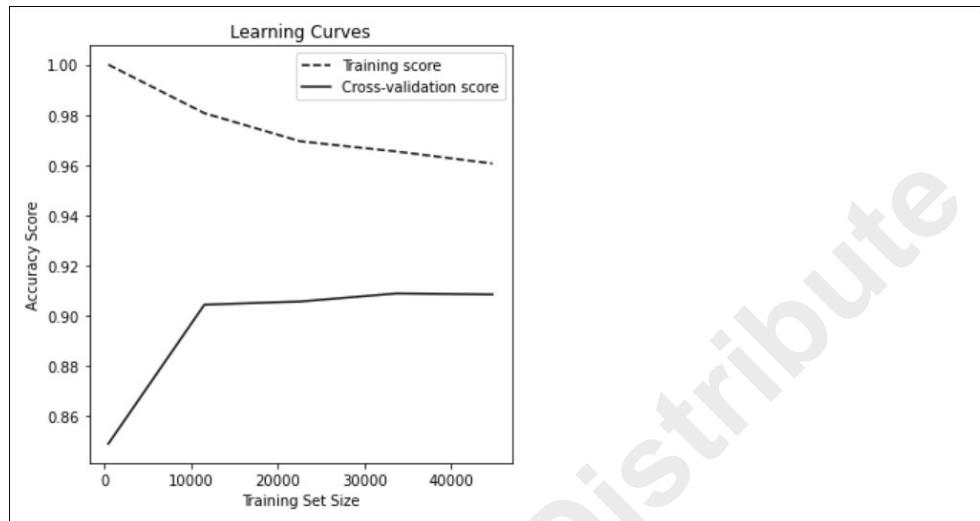
```
1 plot_learning_curves(xgb, X_train_SMOTE, y_train_SMOTE)
```

- e) Run the code cell.



Note: It can take up to five minutes to plot the learning curves.

- f) Examine the output.



Learning curves are plotted for both training and validation datasets. The gap between the two curves indicates that adding more training instances is likely to help, and the cross-validation curve could converge toward the training curve. The gap also indicates that the model suffers from some variance. Although the gap may look large, keep in mind that the scale of the y-axis starts at around 0.85 rather than 0. So, the gap is narrower than it might seem at first, which means that the variance is relatively low. Still, the variance isn't insignificant, so there are some signs of overfitting. Also, because both lines are close to 1 on the y-axis (i.e., they have high accuracy), the bias is also relatively low.

Increasing the number of training instances and applying regularization to the current learning algorithm would likely decrease the variance and increase the bias. You could also consider reducing the number of features to build a more simplified model.

7. Save the best model.

- Scroll down and view the cell titled **Save the best model**, then select the code cell below it.
- In the code cell, type the following:

```
1 pickle.dump(xgb, open('xgboost_classifier.pickle', 'wb'))
```

You'll save the XGBoost model to a binary pickle file so that you can use it later.

- Run the code cell.

8. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Developing Classification Models** tab in Firefox, but keep a tab open to **CDSP/Classification/** in the file hierarchy.

Summary

In this lesson, you trained, tuned, and evaluated machine learning models used in classification. There are many methods to choose from, so it's helpful that you became familiar with the most prominent of them. This will ensure that you are prepared to develop skillful classification models in your own projects.

What type of data in your organization do you think might be conducive to classification?

Given the datasets you're likely to use and classification problems you're trying to solve, what evaluation metrics do you think you'll find most useful when tuning a classification model?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

6

Developing Regression Models

Lesson Time: 3 hours, 30 minutes

Lesson Introduction

The next major machine learning task you'll undertake is regression. Whereas classification is about placing things in categories, regression is about estimating numbers. As with the previous lesson, in this lesson you'll train, tune, and then evaluate models that perform regression.

Lesson Objectives

In this lesson, you will:

- Train models using algorithms that solve regression and forecasting problems, then tune them to improve performance.
- Evaluate the performance of regression and forecasting models to inform the tuning process.

TOPIC A

Train and Tune Regression Models

In this topic, you'll begin developing regression models by inputting training data to multiple algorithms, then tuning those models to improve their effectiveness.

Linear Regression

In the field of statistics, regression analysis is the technique of identifying the relationships between variables. These variables are categorized as either dependent or independent. A dependent variable is one whose variation you are interested in studying. An independent variable has a potential effect on the variation of dependent variables—in other words, it helps explain what is happening to the dependent variable.

Linear regression is the most basic type of regression analysis. It demonstrates a linear relationship between one independent variable and one dependent variable. If plotted on a graph, this relationship would form a straight line, and the slope of that line between any two points on the graph would be the same.

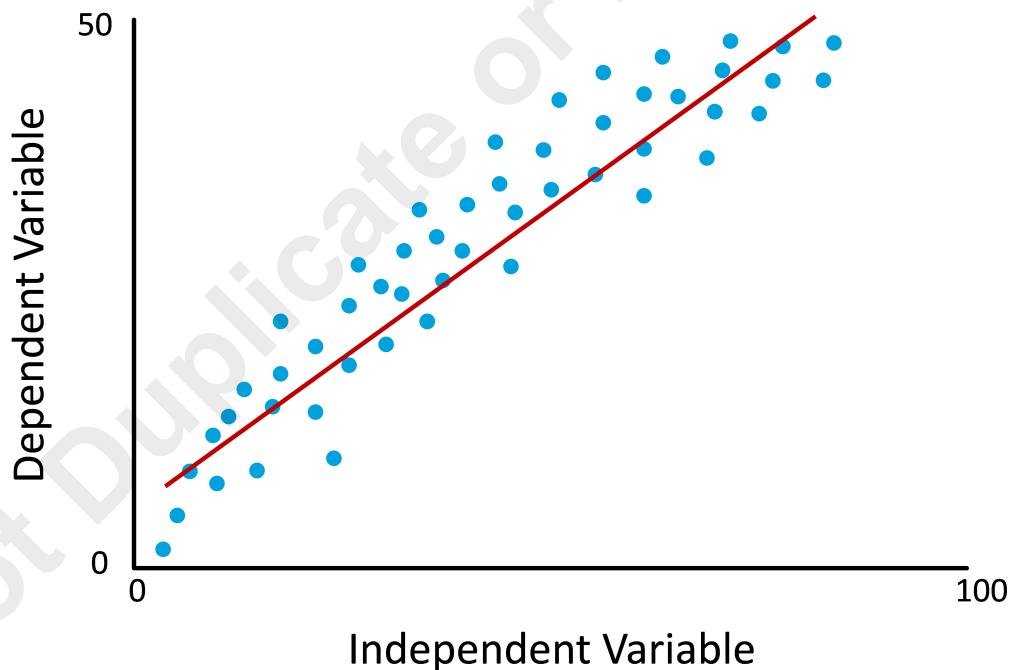


Figure 6–1: A linear regression model in which the data fits to a straight line.



Note: This particular example shows a positive correlation between the two variables. Linear regression can also work with variables that have a negative correlation (i.e., the data points would go from the top left to the bottom right in this graph, and so would the line).

Linear Equation

It helps to understand linear regression by first considering how data may be plugged into a simple linear equation. This equation generates a straight line fit to linear data. You may recognize the linear equation as it is a foundational concept in statistics courses taught in schools:

$$y = mx + b$$

Where:

- y is the y -value for a data example (the dependent variable).
- x is the x -value for a data example (the independent variable).
- m is the slope of the line, calculated by dividing the change in y by the change in x .
- b is the intercept—the value of y when x is 0.

Linear Equation Example Data

Let's say you're trying to analyze products sold by an electronics vendor. In particular, you want to map how long a TV has been on the market (x) and the effect that has on the TV's sale price (y). You have 15 historical data examples to draw from (each a different TV), as recorded in the following table.

Months Since First Release	Sale Price in Dollars
0	849.99
8	819.49
19	775.99
30	699.49
37	720.49
44	745.99
56	625.99
66	560.99
75	580.49
87	520.99
92	590.49
99	440.99
105	389.99
112	405.99
120	299.99



Note: For example purposes, this data is just in raw form and hasn't undergone feature engineering.

When graphed, that data looks like the following.

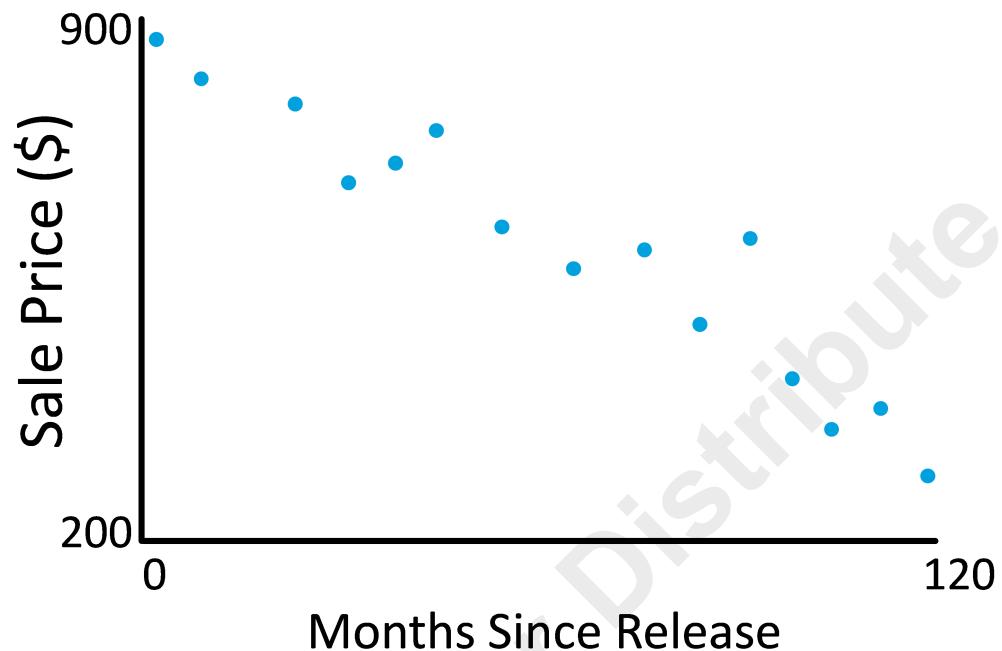


Figure 6–2: Mapping a TV's sale price as it correlates to the product's lifespan on the market.

Straight Line Fit to Example Data

Using the previous data, the calculation of the slope (m) is -4.144 and the intercept is 864.3 . The -4.144 value is the reduction in price every month the TV has been on the market. So, plugged into the linear equation, this is:

$$y = -4.144x + 864.3$$

This provides you with the line of best fit for the data which, when graphed, looks something like the following.

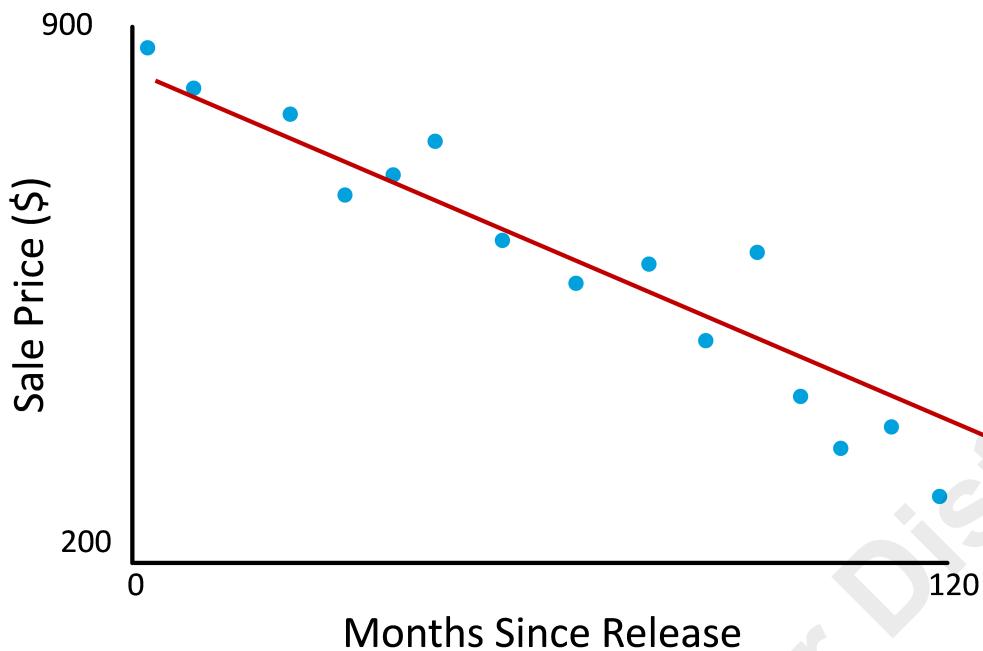


Figure 6-3: Fitting a straight line to the TV dataset.

Now, let's say you want to predict the price of a TV after it has been on the market for exactly 5 years (60 months). You'd simply plug that value in for x like so:

$$y = (-4.144)60 + 864.3 = 615.66$$

So, this very simple linear model has predicted that the TV will sell for \$615.66. When graphed, this might look like the following.

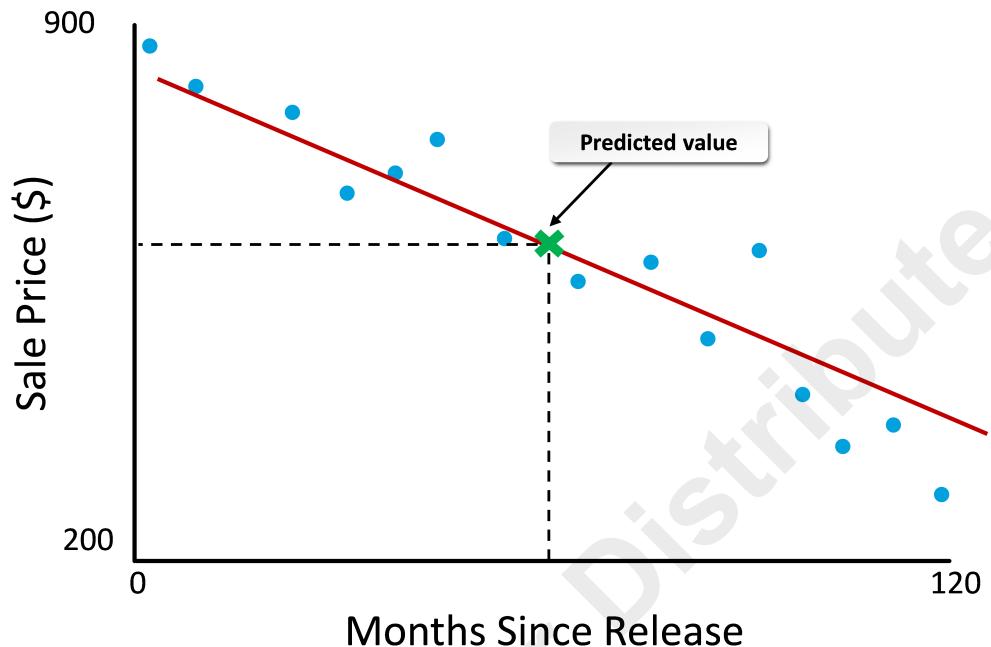


Figure 6–4: Making a prediction based on the line of best fit.

Linear Equation Shortcomings

The linear equation is a simple example of making estimations based on a dataset, but there are more powerful ways of applying linear regression to machine learning. The linear equation may not work well or as expected with data that cannot be fit linearly. It also fails to account for multiple predictors—after all, market lifespan is not the only factor that influences how much a TV sells for. You have other factors like the display resolution/number of pixels (1080p, 4K, etc.), the size of the screen, how many units have been sold, the cost of research and development, the cost of manufacturing, and many more. All of these potentially contribute to the value of the TV.

Machine learning tasks often necessitate somewhat more complex approaches to linear regression.

Linear Regression in Machine Learning

In a machine learning problem, the linear regression algorithm's objective is to find the difference between the input training data and the estimative line fit that the model generates. This difference is called the error or cost. Each feature of the dataset, as well as any permutations generated by a human practitioner during the feature engineering process, will have a corresponding parameter θ_i that the model must solve for. Recall that these parameters are what the model "learns" as part of logistic regression—that's also true of linear regression.

A basic linear model in machine learning can be expressed as:

$$\hat{y} = \theta_0 + \theta_1 \cdot x$$

Where:

- \hat{y} is the variable you're trying to estimate (the dependent variable).
- θ_0 is the intercept (equivalent to b in the linear equation).

- θ_1 is a model parameter (equivalent to m in the linear equation).
- x is the independent variable of interest—the features you'd extract and pass into the model.



Note: Remember, \hat{y} (pronounced "y-hat") is often used to mean an estimation from a model.

Linear Regression in Machine Learning Example

Using the TV price example from before, you train the model on historical data with multiple features. You can construct a linear model based on one or more of these features. For simplicity's sake, you want to start by mapping a single feature—the TV's time on the market. This can be plugged into the formula as:

$$\text{TV_price} = \theta_0 + \theta_1 \cdot \text{time_on_market}$$

When you map this linear function to more features, you can compare how well those functions fit to a straight line. If the line fits relatively straight through the data, that particular feature has a strong correlation with the prediction variable.

Linear regression is commonly used in supervised learning to estimate numerical values (the dependent variables) that increase or decrease based on multiple features (the independent variables).

Matrices in Linear Regression

Because a linear model represents n number of examples in a training set, there would be n number of linear equations for each relevant value of x and y . To calculate all of these instances, linear models represent the data in matrices. The single-parameter linear model equation can be restated as a vector of y values being equal to a matrix of x values multiplied by the model parameters. As an equation, this is:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$



Note: The column of all 1s in the \mathbf{x} matrix is necessary because these 1s are being multiplied by the constant θ_0 intercept value, whereas the column of variable x values is being multiplied by θ_1 .

For simplicity's sake, consider how these matrices would be filled in using just the last two points of the TV price dataset: (112, 405.99) and (120, 299.99). This would give you:

$$\begin{bmatrix} 405.99 \\ 299.99 \end{bmatrix} = \begin{bmatrix} 1 & 112 \\ 1 & 120 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

You could use this to find the model parameters θ_0 and θ_1 , but it's better to pre-multiply each side of the equation by the inverse of the \mathbf{x} matrix. Any matrix multiplied by its inverse is an identity matrix. An **identity matrix** is a matrix of all zeros except for the main diagonal, which consists of all 1s. This identity matrix can be removed from the equation, which leaves us with:

$$\begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

When the values are plugged in:

$$\begin{bmatrix} 15 & -14 \\ -0.125 & 0.125 \end{bmatrix} \begin{bmatrix} 405.99 \\ 299.99 \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} 1889.99 \\ -13.25 \end{bmatrix}$$

Now that you have the model parameters, you can create a straight line fit of the data using a linear equation. If only using the last two data points, this would be $y = -13.25x + 1889.99$. Of course, in a real scenario, you'd use the entire dataset as the matrix values rather than just two instances, so your model parameters would change to account for this.

Normal Equation

One issue with these matrices is that you cannot take the inverse of a non-square matrix. So, when you plug in all 15 data points for the TV price dataset, you end up with a 15×2 matrix for the x values. The solution for this is to take the pseudoinverse (specifically, the Moore–Penrose inverse) of the matrix. This involves *transposing* the matrix of x values. The equation that takes this pseudoinverse is called the **normal equation**. The normal equation is a closed-form solution, meaning that it will directly provide you with the model parameters that lead to the best possible fit to the training data.

To get to this normal equation, consider how the matrix math discussed previously can be rewritten:

$$\mathbf{y} = \mathbf{X} \cdot \boldsymbol{\theta}$$

Where:

- $\boldsymbol{\theta}$ is a matrix of the model parameters (e.g., m and b for slope and intercept).
- \mathbf{x} is a matrix of the x values.
- \mathbf{y} is the vector of y values.

Then, you can multiply each side of the equation by the transpose of \mathbf{x} (i.e., \mathbf{x}^T):

$$\mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X} \cdot \boldsymbol{\theta}$$

Note that $\mathbf{x}^T \mathbf{x}$ is now a square matrix. You can then apply the inverse of $\mathbf{x}^T \mathbf{x}$ to both sides of the equation:

$$(\mathbf{X}^T \mathbf{X})^{-1} \cdot \mathbf{X}^T \mathbf{y} = (\mathbf{X}^T \mathbf{X})^{-1} \cdot \mathbf{X}^T \mathbf{X} \cdot \boldsymbol{\theta}$$

Now, the right side of the equation (other than $\boldsymbol{\theta}$) can be converted into an identity matrix because a matrix is being multiplied by its inverse. So, this can simply be removed.

The normal equation can finally be expressed as:

$$\hat{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \cdot \mathbf{X}^T \mathbf{y}$$

By doing all of these calculations on the TV dataset, you'd come up with $\theta_0 = 864.3$ and $\theta_1 = -4.144$ —the same as the intercept and slope mentioned before. This is a more robust way of solving linear regression problems than just a simple linear equation.

Transposition and Inversion

If you're curious, $\mathbf{x}^T \mathbf{x}$ (using just the last two values in the dataset) looks like the following:

$$\mathbf{x}^T \mathbf{x} = \begin{bmatrix} 1 & 1 \\ 112 & 120 \end{bmatrix} \begin{bmatrix} 1 & 112 \\ 1 & 120 \end{bmatrix} = \begin{bmatrix} 2 & 232 \\ 232 & 26944 \end{bmatrix}$$

The inversion of that is:

$$(\mathbf{x}^T \mathbf{x})^{-1} = \begin{bmatrix} 2 & 232 \\ 232 & 26944 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{26944}{64} & -\frac{232}{64} \\ -\frac{232}{64} & \frac{2}{64} \end{bmatrix}$$

Guidelines for Training Linear Regression Models



Note: All Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Follow these guidelines when training linear regression models.

Train Linear Regression Models

When training linear regression models:

- Use linear regression in supervised learning to predict the change in value of a numeric dependent variable in relation to one or more independent variables.
- Consider how linear regression can lead to more simplified and interpretable model results as compared to other forms of regression.
- Consider that not all data can easily be fit to a straight line.
- Keep in mind the shortcomings of the closed-form normal equation: that large datasets will lead to memory issues due to the computational cost of inverse matrix calculations, and that it is prone to overfitting.

ACTIVITY 6-1

Training a Linear Regression Model

Data Files

/home/student/CDSP/Regression/Developing Regression Models.ipynb
/home/student/CDSP/Regression/data/users_data_final.pickle

Scenario

Now that you're done developing classification models, you want to turn your attention to developing models that can estimate continuous numeric values. The `total_amount_usd` that you created in the ETL phase by consolidating transactional data is of particular interest. It represents the total balance of each user's transactions with the bank over the given period of time. So, the `total_amount_usd` reveals either the debit that the user owes to the bank (negative values), or the credit that the bank extends to the user (positive values). You want to build models that can estimate a new customer's debit/credit based on various features.

As with your classification tasks, you won't just build one model, but several. You'll compare these regressors to determine which one best meets your needs. To start with, you'll train a very simple linear model to see how it handles the data.

1. Open the notebook.
 - a) In the Jupyter Notebook web client, navigate to the **CDSP/Regression** directory.
 - b) Select **Developing Regression Models.ipynb** to open it.
2. Import the relevant software libraries.
 - a) View the cell titled **Import software libraries**, and examine the code listing below it.
 - b) Select the cell that contains the code listing, then select **Run**.
 - c) Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.
3. Load and preview the data.
 - a) Scroll down and view the cell titled **Load and preview the data**, then select the code cell below it.
 - b) In the code cell, type the following:

```
1 users_data = pd.read_pickle('data/users_data_final.pickle')
2
3 users_data.head(n = 5)
```

- c) Run the code cell.

- d) Examine the output.

	user_id	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_se...
0	9231c446-cb16-4b2b-a7f7-ddfc8b25af6	3.0	2143.00	1	0	0	0	0	0	0
1	bb92765a-08de-4963-b432-496524b39157	0.0	1369.42	0	1	0	0	0	0	0
2	573de577-49ef-42b9-83da-d3cfb817b5c1	2.0	2.00	0	0	1	0	0	0	0
3	d6b66b9d-7c8f-4257-a682-e136f640b7e3	0.0	1369.42	0	0	0	1	0	0	0
4	fade0b20-7594-4d9a-84cd-c0279b1b526	1.0	1.00	0	0	0	0	0	0	0

5 rows × 33 columns

This is the same dataset you used in your classification tasks. As a refresher, you'll retrieve some fundamental information about the dataset.

4. Check the shape of data.

- a) Scroll down and view the cell titled **Check the shape of data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_data.shape
```

- c) Run the code cell.
 d) Examine the output.

```
(45179, 33)
```

There are 45,179 records and 33 columns.

5. Check the data types.

- a) Scroll down and view the cell titled **Check the data types**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_data.info()
```

- c) Run the code cell.

- d) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45179 entries, 0 to 45215
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          45179 non-null   object  
 1   number_transactions 45179 non-null   float64 
 2   total_amount_usd   45179 non-null   float64 
 3   job_management    45179 non-null   int64  
 4   job_technician    45179 non-null   int64  
 5   job_entrepreneur  45179 non-null   int64  
 6   job_blue-collar   45179 non-null   int64  
 7   job_retired       45179 non-null   int64  
 8   job_admin.        45179 non-null   int64  
 9   job_services      45179 non-null   int64  
 10  job_self-employed 45179 non-null   int64  
 11  job_unemployed   45179 non-null   int64  
 12  job_housemaid    45179 non-null   int64
```

All of the columns are in the proper data formats for machine learning, except for `user_id`, which you'll remove.

6. Explore the distribution of the target variable.

- Scroll down and view the cell titled **Explore the distribution of the target variable**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data.total_amount_usd.describe()
```

The target variable you're interested in studying is `total_amount_usd`.

- Run the code cell.
- Examine the output.

```
count    45179.000000
mean     1369.751283
std      2704.291321
min     -8019.000000
25%      160.000000
50%      862.000000
75%      1369.420000
max     102127.000000
Name: total_amount_usd, dtype: float64
```

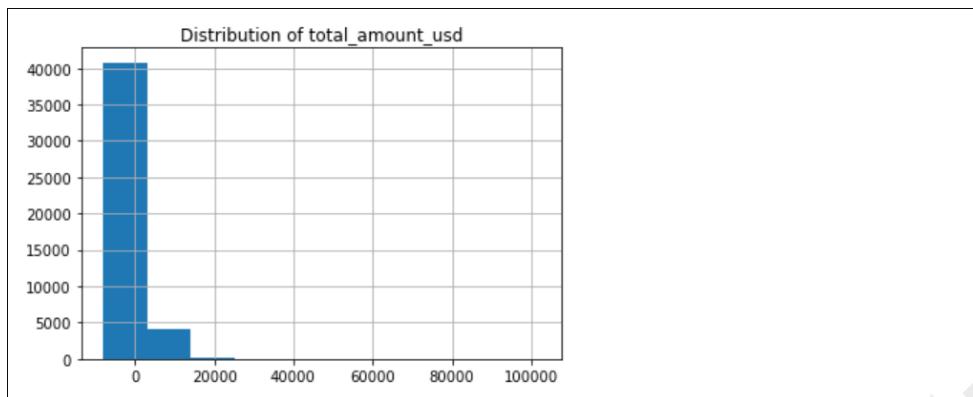
The largest debit is \$-8,019, and the largest credit is \$102,217.

- Select the next code cell, then type the following:

```
1 users_data.total_amount_usd.hist()
2 plt.title('Distribution of total_amount_usd');
```

- Run the code cell.

- g) Examine the output.

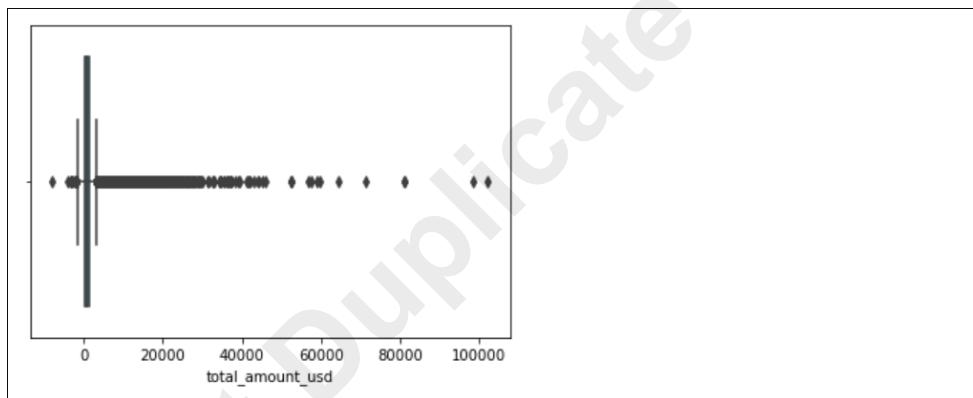


The data has a very heavy positive skew. Most users' total balances are around 0, whereas very few are above \$20,000.

- h) Select the next code cell, then type the following:

```
1 sns.boxplot(users_data.total_amount_usd);
```

- i) Run the code cell.
j) Examine the output.



This box plot confirms that there are several outliers, particularly at the upper bounds of the dataset. You'll remove these outliers so they don't interfere with the model's training.

7. Identify and remove the outliers.

- a) Scroll down and view the cell titled **Identify and remove the outliers**, then select the code cell below it.

- b) In the code cell, type the following:

```

1 q1 = np.percentile(users_data.total_amount_usd, 25)
2 q3 = np.percentile(users_data.total_amount_usd, 75)
3 iqr = q3 - q1
4
5 lb = q1 - 1.5 * iqr
6 ub = q3 + 1.5 * iqr
7
8 print('Lower bound:', round(lb, 2))
9 print('Upper bound:', round(ub, 2))

```

This code will use the percentiles to determine the lower and upper bounds of the dataset.

- c) Run the code cell.
d) Examine the output.

```

Lower bound: -1654.13
Upper bound: 3183.55

```

Any figure lower than \$-1,654.13 is an outlier, as is anything above \$3,183.55.

- e) Select the next code cell, then type the following:

```

1 print('Number of users with total_amount_usd greater than UB:',
2         users_data[(users_data.total_amount_usd >= ub)].shape[0])
3 print('Number of users with total_amount_usd lower than LB: ',
4         users_data[(users_data.total_amount_usd <= lb)].shape[0])

```

- f) Run the code cell.
g) Examine the output.

```

Number of users with total_amount_usd greater than UB: 4110
Number of users with total_amount_usd lower than LB: 26

```

4,110 customers have total balances that are high outliers, whereas 26 have total balances that are low outliers.

- h) Select the next code cell, then type the following:

```

1 users_data_wout_outliers = \
2 users_data[(users_data.total_amount_usd < ub) \
3             & (users_data.total_amount_usd > lb)]
4
5 users_data_wout_outliers.shape

```

This code will remove the identified outliers from the dataset.

- i) Run the code cell.
j) Examine the output.

```
(41043, 33)
```

There are now 41,043 rows.

8. Explore the dataset with the outliers removed.

- a) Scroll down and view the cell titled **Explore the dataset with the outliers removed**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 users_data_wout_outliers.describe()
```

- c) Run the code cell.
d) Examine the output.

	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services
count	41043.000000	41043.000000	41043.000000	41043.000000	41043.000000	41043.000000	41043.000000	41043.000000	41043.000000
mean	3.131179	793.578253	0.202032	0.169237	0.033087	0.220257	0.047657	0.116488	0.094316
std	3.916997	761.407759	0.401521	0.374966	0.178867	0.414425	0.213043	0.320813	0.292271
min	0.000000	-1636.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	123.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	2.000000	659.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	3.000000	1369.420000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	20.000000	3181.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

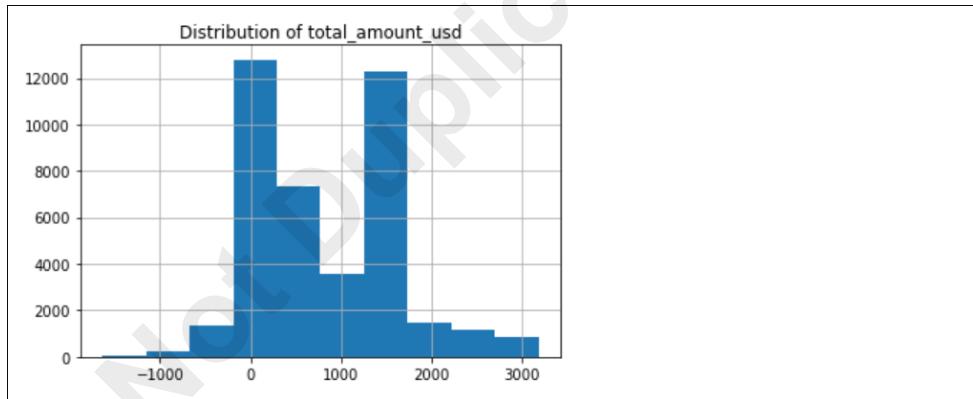
8 rows × 28 columns

As you can from the `total_amount_usd` column, the minimum and maximum are now at the lower and upper bounds, respectively. Other values, like the mean, have also changed to reflect the removal of outliers.

- e) Select the next code cell, then type the following:

```
1 plt.title('Distribution of total_amount_usd')
2 users_data_wout_outliers.total_amount_usd.hist();
```

- f) Run the code cell.
g) Examine the output.



The distribution of the dataset now exhibits much less skew than before.

9. Split the data into target and features.

- a) Scroll down and view the cell titled **Split the data into target and features**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 target_data = users_data_wout_outliers.total_amount_usd
2 features = users_data_wout_outliers.drop(['user_id', 'total_amount_usd'],
3 axis = 1)
```

Since you're still performing supervised learning, you'll need to split the dataset into the features and the target variable (label).

- c) Run the code cell.

10. Split the data into train and test sets.

- a) Scroll down and view the cell titled **Split the data into train and test sets**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 X_train, X_test, y_train, y_test = train_test_split(features,
2 target_data,
3 test_size = 0.3)
```

You'll perform the holdout method for this dataset in much the same way you did for the classification models.

- c) Run the code cell.
- d) Select the next code cell, then type the following:

```
1 print('Training data features: ', X_train.shape)
2 print('Training data target: ', y_train.shape)
```

- e) Run the code cell.
- f) Examine the output.

```
Training data features: (28730, 31)
Training data target: (28730,)
```

The training set has 70% of the dataset's examples, or 28,730 rows.

11. Check the distribution of the test data.

- a) Scroll down and view the cell titled **Check the distribution of the test data**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 print('Test data features: ', X_test.shape)
2 print('Test data target: ', y_test.shape)
```

- c) Run the code cell.
- d) Examine the output.

```
Test data features: (12313, 31)
Test data target: (12313,)
```

The test data has 30% of the data examples, or 12,313 rows.

- e) Select the next code cell, then type the following:

```
1 y_test.describe()
```

- f) Run the code cell.
g) Examine the output.

```
count    12313.000000
mean     794.124498
std      765.840040
min     -1629.000000
25%      121.000000
50%      655.000000
75%     1369.420000
max     3181.000000
Name: total_amount_usd, dtype: float64
```

The summary statistics for the test set are as expected. Note that, for these models, you won't be scaling the data. In a production environment, you should consider doing so for certain models that benefit from this scaling. However, since most of the regression models you'll be training will either use simple linear algorithms or tree-based algorithms, scaling isn't as important. Scaling can also have an impact on the interpretability of the results for a numeric target variable, so that's another reason to forego it this time.

12. Train a linear regression model.

- a) Scroll down and view the cell titled **Train a linear regression model**, then select the code cell below it.
b) In the code cell, type the following:

```
1 linreg = LinearRegression()
2 linreg.fit(X_train, y_train)
```

This code creates a linear regression object that uses the normal equation to determine the optimal model parameters. As with any other supervised algorithm, the model is fit with the input data.

- c) Run the code cell.

13. Make predictions using the linear regression model.

- a) Scroll down and view the cell titled **Make predictions using the linear regression model**, then select the code cell below it.
b) In the code cell, type the following:

```
1 linreg_y_pred = linreg.predict(X_test)
```

The model will generate `total_amount_usd` predictions for each customer.

- c) Run the code cell.
d) Select the next code cell, then type the following:

```
1 results = pd.concat([y_test.iloc[:5], X_test.iloc[:5]], axis = 1)
2 results.insert(1, 'total_pred', linreg_y_pred[:5].round(2))
3 results
```

This code will show the first five records of the test set and compare the actual `total_amount_usd` labels to the model's predictions.

- e) Run the code cell.

- f) Examine the output.

	total_amount_usd	total_pred	number_transactions	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services
33025	104.00	856.80	3.0	0	0	0	1	0	0	0
1893	1369.42	1129.94	0.0	0	1	0	0	0	0	0
26956	219.00	994.59	2.0	0	0	0	1	0	0	0
38050	170.00	829.65	2.0	0	0	0	1	0	0	0
19663	53.00	816.78	3.0	1	0	0	0	0	0	0

5 rows × 33 columns

In several cases, the model was far off in its predictions. For example, the model predicted a balance of \$856.80 for the first customer, but they actually had a much smaller balance of \$104.00. In other cases, like the second customer, the prediction was fairly close.

14. Obtain the linear regression model's score.

- a) Scroll down and view the cell titled **Obtain the linear regression model's score**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 r2_score(y_test, linreg_y_pred)
```

Whereas accuracy is the default metric for classifiers in scikit-learn, R^2 (coefficient of determination) is the default metric for regressors. You'll learn more about regression metrics later, but for now, just know that the ideal R^2 value is 1.

- c) Run the code cell.
- d) Examine the output.

```
0.2332152712408806
```

The simple linear model's R^2 score is ~ 0.23 . This isn't great when considered in isolation, but it's most useful when compared to other models.

15. Keep this notebook open.

Regression Using Decision Trees and Ensemble Models

CART, if you recall, stands for classification and *regression* trees. So, as you'd expect, it can be used for regression tasks in order to make estimations about continuous numerical variables. The general structure of a decision tree regressor is similar to its classification counterpart, but one of the key differences is that regression trees use different splitting metrics. Reducing Gini impurity is not really suitable for continuous variables, so CART regressors must instead attempt to reduce the error at each decision node. There are different ways to define "error" based on the metric used. Common error metrics include the mean squared error (MSE) and mean absolute error (MAE).

The other key difference between decision tree classifiers and regressors is that regressors output a continuous numeric estimation rather than a class value. In the figure, you can see a CART regressor modeled from the TV price dataset.



Figure 6–5: A simplified decision tree regressor.

Unlike linear regression, regression using CART does not make assumptions about the relationship between dependent and independent variables. So, CART regressors are the more common choice when linear regression is too restrictive to the problem at hand. As a result, they usually require much more training data to learn from in order to be effective.

As far as random forests go, the main difference with having regressors in the forest has to do with how the output is chosen. Since the target variable is continuous, there are no "votes" to tally up. Instead, the forest will select the mean of the estimations between all trees in the forest as the output value. Gradient boosting can also be used for regression tasks, the main difference being that it does not calculate residuals based on probabilities, but rather calculates them from direct numeric estimations.

Guidelines for Training Regression Trees and Ensemble Models

Follow these guidelines when training regression trees and ensemble models.

Train Regression Trees and Ensemble Models

When training regression trees and ensemble models:

- Consider adding a bagging or other ensemble learner for better results and resistance to overfitting.
- Encode categorical data into numerical values.
- Use a bagged random forest to increase training speed with large datasets.
- Reduce the number of dimensions to increase training speed.

ACTIVITY 6–2

Training Regression Trees and Ensemble Models

Before You Begin

Developing Regression Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Train a decision tree model**. Select **Cell**→**Run All Above**.

Scenario

The simpler linear regression model was able to make predictions on the `total_amount_usd` target variable, but there's definitely room for improvement. Decision trees and ensemble models are not just useful for classification, but regression as well. So you'll train several of these models to see if you can get some better results.

1. Train a decision tree model.

- Scroll down and view the cell titled **Train a decision tree model**, then select the code cell below it.
- In the code cell, type the following:

```
1 reg_tree = DecisionTreeRegressor()  
2 reg_tree.fit(X_train, y_train)
```

This code builds a decision tree object for regression tasks, then fits the training data.

- Run the code cell.

2. Make predictions using the decision tree model.

- Scroll down and view the cell titled **Make predictions using the decision tree model**, then select the code cell below it.
- In the code cell, type the following:

```
1 reg_tree_y_pred = reg_tree.predict(X_test)
```

The decision tree will make predictions on the test set.

- Run the code cell.
- Select the next code cell, then type the following:

```
1 results['total_pred'] = reg_tree_y_pred[:5]  
2 results
```

- Run the code cell.

- f) Examine the output.

	total_amount_usd	total_pred	number_transactions	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services
33025	104.00	2958.00	3.0	0	0	0	1	0	0	0
1893	1369.42	1369.42	0.0	0	1	0	0	0	0	0
26956	219.00	2787.00	2.0	0	0	0	1	0	0	0
38050	170.00	59.00	2.0	0	0	0	1	0	0	0
19663	53.00	862.00	3.0	1	0	0	0	0	0	0

5 rows × 33 columns

The predictions for the first five records are different than the linear regression model. Some of them seem to be considerably worse, like the prediction for the first user. The prediction for the second user was exactly on target, however. Keep in mind that this value was the mean you imputed for missing values, so the model may be overfitting to those examples.

3. Obtain the decision tree model's score.

- Scroll down and view the cell titled **Obtain the decision tree model's score**, then select the code cell below it.
- In the code cell, type the following:

```
1 | r2_score(y_test, reg_tree_y_pred)
```

- Run the code cell.
- Examine the output.

```
-0.40313871451252936
```

The decision tree not only scored significantly worse than the linear regression model, but scored in the negative. This is very likely due to overfitting, which decision trees are prone to. You'll see if you can improve this score during the tuning process.

4. Visualize the decision tree.

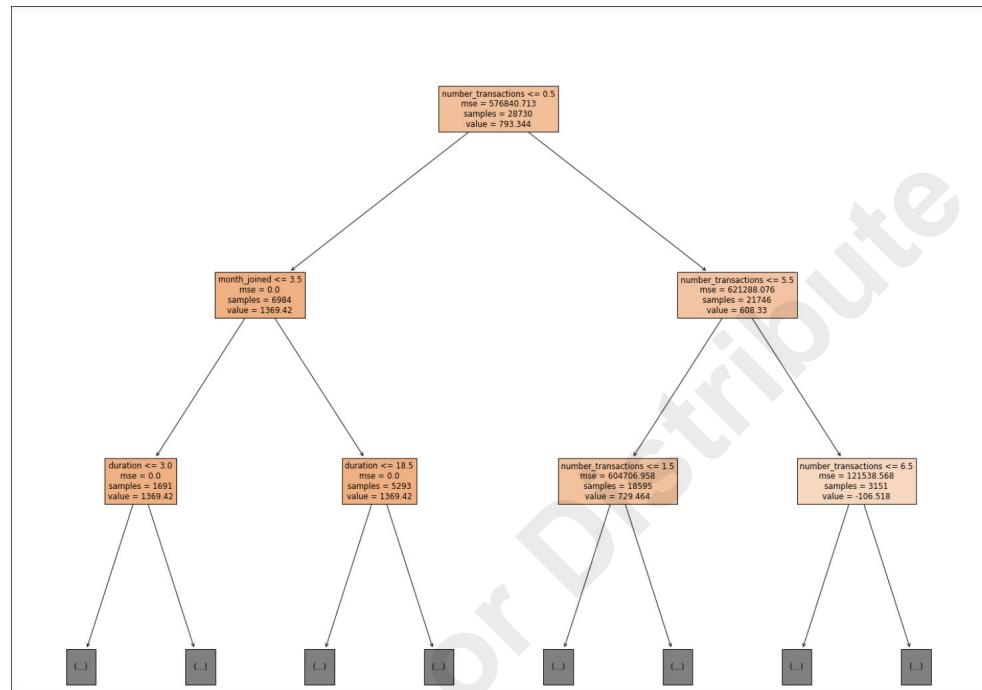
- Scroll down and view the cell titled **Visualize the decision tree**, then select the code cell below it.
- In the code cell, type the following:

```
1 | fig = plt.figure(figsize = (25, 20))
2 | _ = tree.plot_tree(reg_tree,
3 |                     feature_names = list(X_train.columns),
4 |                     max_depth = 2,
5 |                     filled = True)
```

As with classification, it helps to visualize the results of the tree to see how it's making splitting decisions.

- Run the code cell.

- d) Examine the output.



The root node of the tree is using `number_transactions` as the feature to start splitting from. In this case, it's checking to see if there actually *are* any transactions (i.e., less than 0.5 transactions—in other words, 0). If there are 0 transactions, the tree splits to a node that determines whether or not the user joined within the first three months of a year (`month_joined <= 3.5`). If there are more than 0 transactions, the tree splits to a node that considers 5 or fewer transactions—and so on.

5. Train a random forest model.

- Scroll down and view the cell titled **Train a random forest model**, then select the code cell below it.
- In the code cell, type the following:

```

1 rf = RandomForestRegressor()
2 rf.fit(X_train, y_train)
  
```

Now you'll aggregate multiple trees into a random forest model to see if the results improve.

- Run the code cell.

6. Make predictions using the random forest model.

- Scroll down and view the cell titled **Make predictions using the random forest model**, then select the code cell below it.
- In the code cell, type the following:

```

1 rf_y_pred = rf.predict(X_test)
  
```

- Run the code cell.

- d) Select the next code cell, then type the following:

```
1 results['total_pred'] = rf_y_pred[:5]
2 results
```

- e) Run the code cell.
f) Examine the output.

	total_amount_usd	total_pred	number_transactions	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services
33025	104.00	1187.61	3.0	0	0	0	1	0	0	0
1893	1369.42	1369.42	0.0	0	1	0	0	0	0	0
26956	219.00	1134.88	2.0	0	0	0	1	0	0	0
38050	170.00	514.93	2.0	0	0	0	1	0	0	0
19663	53.00	969.48	3.0	1	0	0	0	0	0	0

5 rows × 33 columns

Once again, the predicted values are different from the other models.

7. Obtain the random forest model's score.

- a) Scroll down and view the cell titled **Obtain the random forest model's score**, then select the code cell below it.
b) In the code cell, type the following:

```
1 r2_score(y_test, rf_y_pred)
```

- c) Run the code cell.
d) Examine the output.

```
0.2955761465520649
```

The random forest's score is much improved over the lone decision tree, and is slightly improved over the linear regression model (~0.30 vs. ~0.23).

8. Train a gradient boosting model.

- a) Scroll down and view the cell titled **Train a gradient boosting model**, then select the code cell below it.
b) In the code cell, type the following:

```
1 xgb = XGBRegressor(objective = 'reg:squarederror')
2 xgb.fit(X_train, y_train)
```

This code will build a gradient boosting model using the XGBoost algorithm. The `objective` argument refers to the learning objective, or how the model will attempt to minimize error.

- c) Run the code cell.

9. Make predictions using the gradient boosting model.

- a) Scroll down and view the cell titled **Make predictions using the gradient boosting model**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 xgb_y_pred = xgb.predict(X_test)
```

- c) Run the code cell.
d) Select the next code cell, then type the following:

```
1 results['total_pred'] = xgb_y_pred[:5]
2 results
```

- e) Run the code cell.
f) Examine the output.

	total_amount_usd	total_pred	number_transactions	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_servic
33025	104.00	803.491333	3.0	0	0	0	1	0	0	0
1893	1369.42	1357.178345	0.0	0	1	0	0	0	0	0
26956	219.00	939.850159	2.0	0	0	0	1	0	0	0
38050	170.00	425.883087	2.0	0	0	0	1	0	0	0
19663	53.00	735.929199	3.0	1	0	0	0	0	0	0

5 rows × 33 columns

This model also made a different set of predictions for the target variable.

10. Obtain the gradient boosting model's score.

- a) Scroll down and view the cell titled **Obtain the gradient boosting model's score**, then select the code cell below it.
b) In the code cell, type the following:

```
1 r2_score(y_test, xgb_y_pred)
```

- c) Run the code cell.
d) Examine the output.

```
0.31022081348267105
```

The XGBoost model's score is ~0.31, which is only very slightly better than the random forest. Both are far from the ideal. Later, you'll use other metrics to compare the different models.

11. Keep this notebook open.

Forecasting

Forecasting is a task that involves making predictions about future events based on the analysis of relevant past events. In machine learning, the term "forecasting" usually refers to a type of forecasting called **time series** forecasting. Time series forecasting is actually a subset of regression analysis because it attempts to identify the relationships between continuous variables. What makes time series forecasting distinct from typical regression tasks is that each successive value of an independent variable is directly correlated with the previous value. This means that the training data must be in a *time series* format.

Consider a task in which you need to predict the air temperature in the immediate vicinity of a warehouse. You have various Internet of Things (IoT) sensors positioned all around the area so that

you can efficiently record and access weather data. If that data were in a time series format, it might look something like this:

Time	Relative Humidity (%)	Dew Point (°F)	Wind Speed (mph)	Temperature (°F)
02:00:00	88	25	10	28
03:00:00	89	24	9	27
04:00:00	89	24	8	27
05:00:00	84	22	9	26
06:00:00	84	22	12	26

As you can see, each row is a specific time (in this case, weather information was recorded each hour). The data is therefore sequential, and values for each feature are directly dependent on those values in the past. So any predictions about future temperatures must consider each historical row of data as a member of a sequence, rather than as an independent observation. This also means that row order in a time series dataset is important, and new records must follow this order if the dataset is to grow.



Note: Time series can follow any time interval, not just hours like in the example.

Forecasting is very popular in fields like sales, business planning, risk assessment, and any other task that requires future predictions based on past data. There are many algorithms that perform time series forecasting, including complex neural networks used in deep learning. In this course, you'll focus on one widely used forecasting algorithm.

Autoregressive Integrated Moving Average (ARIMA)

Autoregressive integrated moving average (ARIMA) is one of the most common algorithms used in univariate time series forecasting. In other words, the model only considers a single variable in its prediction. Using the weather example, your time series dataset would only include the time index column and the temperature column. ARIMA would be able to forecast the temperature based on how it changed over time in the past.

Each term in the name ARIMA actually helps to describe what the algorithm does:

- **Autoregressive** describes the dependent relationship between one observation and one or more other observations. The gap between each observation in a time series is referred to as the *lag*, and observations are said to be *lagged* if they are some relative degree apart from each other. In the example table, the lag between the record at 03:00:00 and 05:00:00 is 2—not because they happen to be two hours apart, but because they are two records apart.
- **Integrated** means that an observation is subtracted from the previous observation ("differenced") to make the time series *stationary*. A stationary time series is one whose statistical properties (e.g., mean and variance) are constant over time. This makes prediction easy, since these values don't change. Later, the process of making the time series stationary is reversed in order to reveal the predictions on the original, non-stationary series.
- **Moving average** describes the dependent relationship between one observation and the residual errors from a moving average applied to lagged observations.

These three concepts are actually implemented as three different hyperparameters when training an ARIMA model. Those hyperparameters are:

- p —the number of lags to include in the model. So, a value of 5 would cause autoregression to consider an observation in context with the five observations before it.

- d —the number of times that the observations are differenced. When there is a clear trend in the data, it's common to start with a value of 1 so that the minimum number of differences is performed in order to make the time series stationary.
- q —the number of lagged forecast errors to include in the model. This value is used to remove any remaining autocorrelation—the correlation between lagged observations as a function of that lag—from the stationary series. A good starting value is 1.

You can set 0 for any of these hyperparameters in order to remove that functionality from the model. In other words, it's possible to have an AR model, an I model, an MA model, and so on.

There are various analysis methods you can use to determine the best hyperparameter values. However, libraries like `auto.arima` for R and `pmdarima` for Python perform these analyses for you to determine the optimal p , d , and q to use in your forecast.

ARIMA Variations

There are several variations of ARIMA that enable you to train forecasting models from multivariate data—data where other features like humidity, dew point, and wind speed would be considered as part of the prediction. One such example is vector autoregression moving average (VARMA), which incorporates some of the same concepts and hyperparameters as ARIMA.

Also, ARIMA can incorporate the concept of *seasonality*, in which a pattern repeats over some number of time periods s . A seasonal time series exhibits this repeating pattern. For example, if you took temperature data monthly, your s value might be 12, since the highest temperatures tend to occur in the same months every year, and the lowest temperatures do likewise. If you wanted to predict a temperature in February, seasonal ARIMA models would consider past years' February temperatures when making the prediction. "Seasonal," however, should not be taken literally—any time series data can be seasonal if it exhibits a repeating pattern over some period of time.

ARIMA Temperature Example

The following example shows a univariate dataset of temperature values over a 24-hour period, where the temperature was recorded at each hour.

Time	Temperature (°F)
2021-01-27 00:00:00	29
2021-01-27 01:00:00	28
2021-01-27 02:00:00	28
2021-01-27 03:00:00	27
2021-01-27 04:00:00	27
2021-01-27 05:00:00	26
2021-01-27 06:00:00	26
2021-01-27 07:00:00	26
2021-01-27 08:00:00	25
2021-01-27 09:00:00	26
2021-01-27 10:00:00	26
2021-01-27 11:00:00	27
2021-01-27 12:00:00	27
2021-01-27 13:00:00	27

Time	Temperature (°F)
2021-01-27 14:00:00	27
2021-01-27 15:00:00	26
2021-01-27 16:00:00	27
2021-01-27 17:00:00	25
2021-01-27 18:00:00	24
2021-01-27 19:00:00	23
2021-01-27 20:00:00	22
2021-01-27 21:00:00	23
2021-01-27 22:00:00	22
2021-01-27 23:00:00	22

When this data is plotted on a line graph, it clearly shows a downward trend.

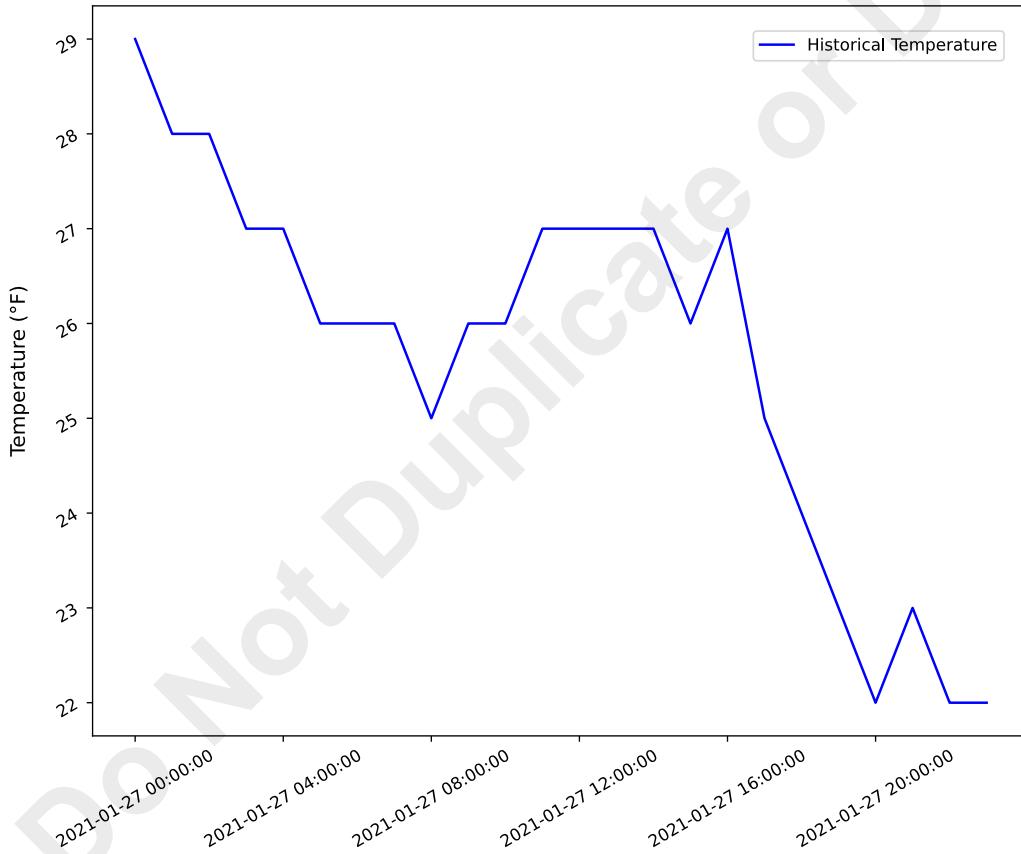


Figure 6–6: Example: historical data.

The statsmodels tool in Python® has a method called `ARIMA()` that can forecast this temperature data. In this case, the method is being called with the argument `order = (3, 1, 1)`. This is equivalent to:

- $p = 3$

- $d = 1$
- $q = 1$

So, the autoregressive lag is 3, whereas both the differencing count and the number of lagged forecast errors are both 1. After fitting the data to the model, you can use the `predict()` method to specify a start and end time to forecast. In this case, the model will forecast the next six hours (i.e., hourly starting at midnight on January 28th, 2021). The results are as follows.

Time	Forecasted Temperature (°F)
2021-01-28 00:00:00	21.615124
2021-01-28 01:00:00	21.281705
2021-01-28 02:00:00	21.389586
2021-01-28 03:00:00	20.921223
2021-01-28 04:00:00	21.124714
2021-01-28 05:00:00	20.840282

These forecasted values can be plotted alongside the historical values, as in the following figure.

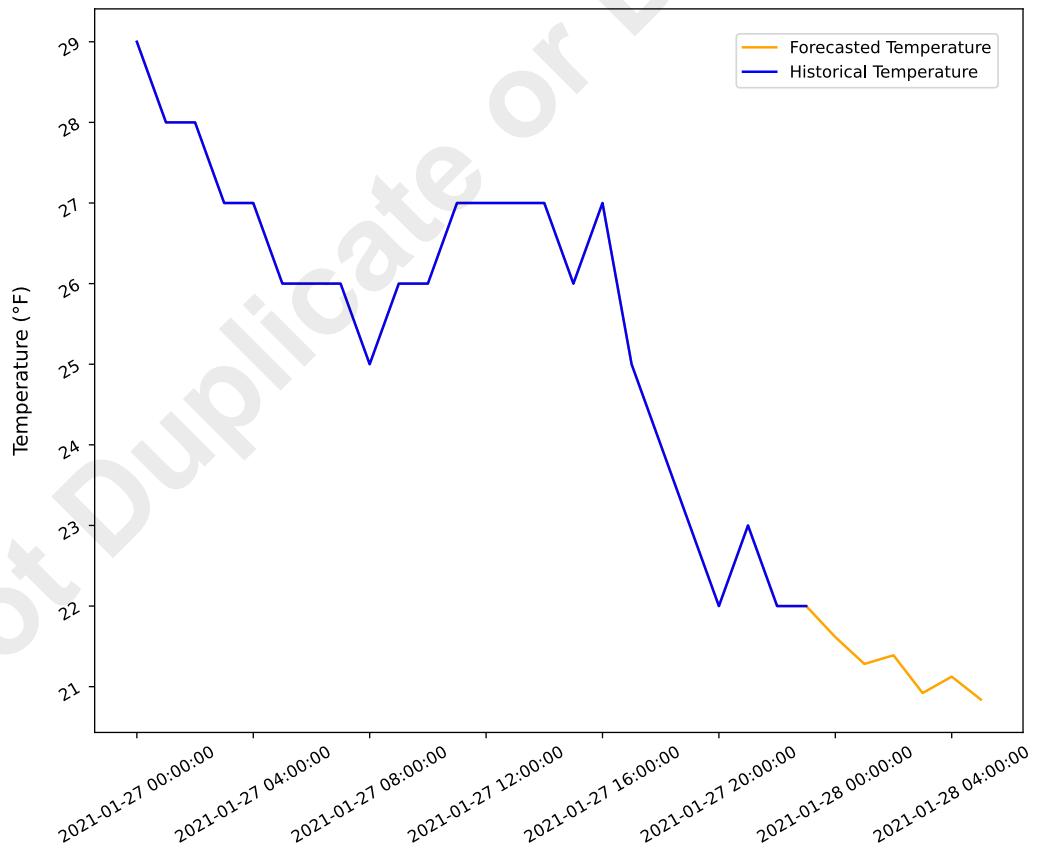


Figure 6-7: A graph of the temperature values forecasted by the ARIMA model.

Obviously, more training data will lead to a more effective model, as 24 observations are not a particularly robust sample for the model to learn from. Also, in practice, you'd want to split the data into separate training and test sets to evaluate the model's performance, as with any other supervised algorithm.

Guidelines for Training Forecasting Models

Follow these guidelines when training forecasting models.

Train Forecasting Models

When training forecasting models:

- Consider splitting your data into 60% training and 40% testing data.
- Split time series data into segments instead of selecting values at random, with training data always earlier than test data.
- Reduce training time by using fewer categories or applying dimensionality reduction to the data.
- Consider using differences rather than raw values when forecasting time series data.
- Use multiple regions for testing time series data to evaluate overfitting of the training data.

Cost Function

To find the best possible fit for a regression model, you need a way to determine the values for the model parameters (θ_i) that will lead to this best fit. Doing so involves evaluating the performance of the model on the training data. Rather than evaluating how *well* the model makes estimations, in regression, it's more common to evaluate how *poorly* it estimates—in other words, its cost.

A **cost function** attempts to quantify the error between the estimated values and the actual labeled training values. It does this by calculating the difference between the two. In other words, the machine learning model identifies how bad it is at estimating the relationship between the independent and dependent variables (x and y). A significant part of the learning process is the act of minimizing this cost function by determining the optimal model parameters. The normal equation, for example, is a method of minimizing the cost function. In addition, some of the forthcoming tuning techniques rely on minimizing this cost function to improve performance.

Regularization

One of the methods you can use to tune a regression model is regularization. **Regularization** is the technique of simplifying a model by constraining the model parameters, which helps the model avoid overfitting to the training data. This typically involves forcing one or more model parameters to only include values within a small range, or forcing parameters to 0. This helps to minimize the effect of outliers on the model.

In a machine learning model, you can control the amount of regularization by setting the λ (lambda) hyperparameter. As you increase the value of λ , the model will become less likely to overfit to the training data. This is because you are decreasing variance in the model. But decreasing variance increases bias, so you must be careful not to make λ too large. This would underfit the model, preventing it from making useful estimations on the training data.

One common method for selecting a λ value is to use cross-validation to randomly sample the data several times for one λ value, then repeat the process for different λ values. You can then choose the λ value that best minimizes the total error.



Note: λ is the Greek letter lambda.

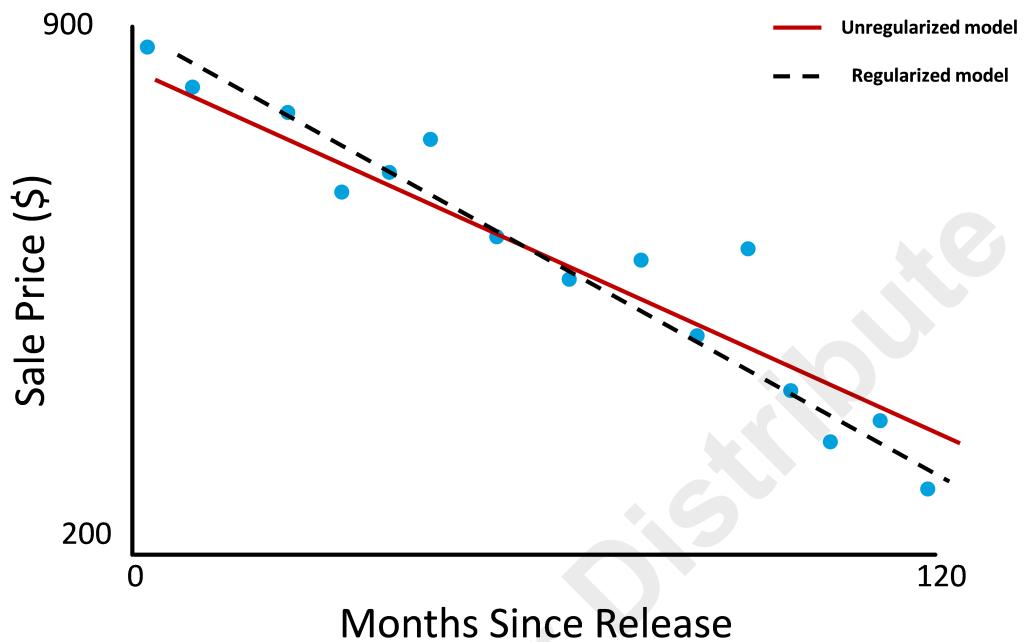


Figure 6–8: A linear model without regularization (solid red line) vs. with regularization (dashed black line). The former fits the training data better, but the latter generalizes to new data better.

Regularization Techniques

Regularization simplifies a model by applying the λ constraint hyperparameter. More specifically, this hyperparameter is part of a term that is added to the cost function. This term penalizes the model if its parameter values are too high, and therefore keeps the parameter values small. There are actually several techniques for applying regularization to a model, each of which uses a different regularization term. The three regularization techniques that have found the most success are as follows.

Regularization Technique	Description
<i>Ridge regression</i>	Uses a mathematical function called an ℓ_2 norm to implement its regularization term. The ℓ_2 norm is the sum of square coefficients, and the objective is to minimize it. This helps to keep the model parameter weights small, reducing overfitting. Ridge regression is suitable in datasets featuring a large number of features, each having at least some estimative power. This is because ridge regression helps to reduce overfitting without actually removing any of the features entirely.

Regularization Technique	Description
Lasso regression	<p>Uses the ℓ_1 norm to implement its regularization term. The ℓ_1 norm forces the coefficients of the least relevant features to 0—in other words, removing them from the model. Like with ridge regression, this helps the model avoid overfitting to the training data.</p>
Elastic net regression	<p>As opposed to ridge regression, lasso regression is suitable in datasets that have only a small or moderate number of features that have moderate estimative power. Lasso regression is able to eliminate the rest of the features that have no significant effect on the data, which may lead to better model performance than keeping and reducing them, as in ridge regression.</p> <p>Uses a weighted average of both ridge and lasso regression as part of its regularization term. It is therefore an attempt to leverage the best of both ℓ_1 and ℓ_2 norms. Along with λ, elastic net regression also uses the α ratio hyperparameter. The α value specifies which regression technique exerts more influence over the result, where values closer to 0 favor ridge regression and values closer to 1 favor lasso regression.</p> <p>Like lasso regression, elastic net regression tends to work well when there are only a small to moderate number of features that are actually relevant (though, depending on α, it can also equal ridge regression). Elastic net regression is typically preferable, as lasso regression may not perform optimally when the number of features far outnumber the training examples. Likewise, elastic net regression tends to perform better in situations where multiple features exhibit high correlation. Pure ridge regression may still be ideal if removing even a small number of features could impair the model's estimative skill.</p>



In most cases, whichever specific technique you choose, it's a good idea to apply at least some form of regularization while training a linear regression model. However, you should only perform regularization during training; when you evaluate the model's performance after training, you must not use the regularization hyperparameter.

Collinearity

Ridge regression addresses the issue of **collinearity**, in which two features exhibit a linear relationship—in other words, the features are either exactly or very closely related in terms of how they influence the dependent variable. Collinearity therefore makes it difficult to determine how each feature has an effect on the output independently. **Multicollinearity** refers to the same concept, but can extend to more than two features.

Gradient Descent

In machine learning, the most common iterative method for minimizing the cost function in a regression model is the gradient descent algorithm. In **gradient descent**, the model parameters are tuned over several iterations by taking gradual "steps" down a slope, toward a minimum error value.

Think of a cost function as a valley. The top edges of the valley are the values of θ that do not minimize the cost function. The bottom of the valley is the value of θ that best minimizes the cost function—also called the minimum. Gradient descent selects a value for θ by starting at a random location. It then calculates the partial derivative of each model parameter with respect to the cost function, then updates each value of θ (i.e., it *steps* in a direction). This new θ value is one in which

the model parameters give a cost that is lower than the previous trial. This process repeats, and as it does, it takes more and more steps down the valley until finally converging at the minimum (represented as $\hat{\theta}$).

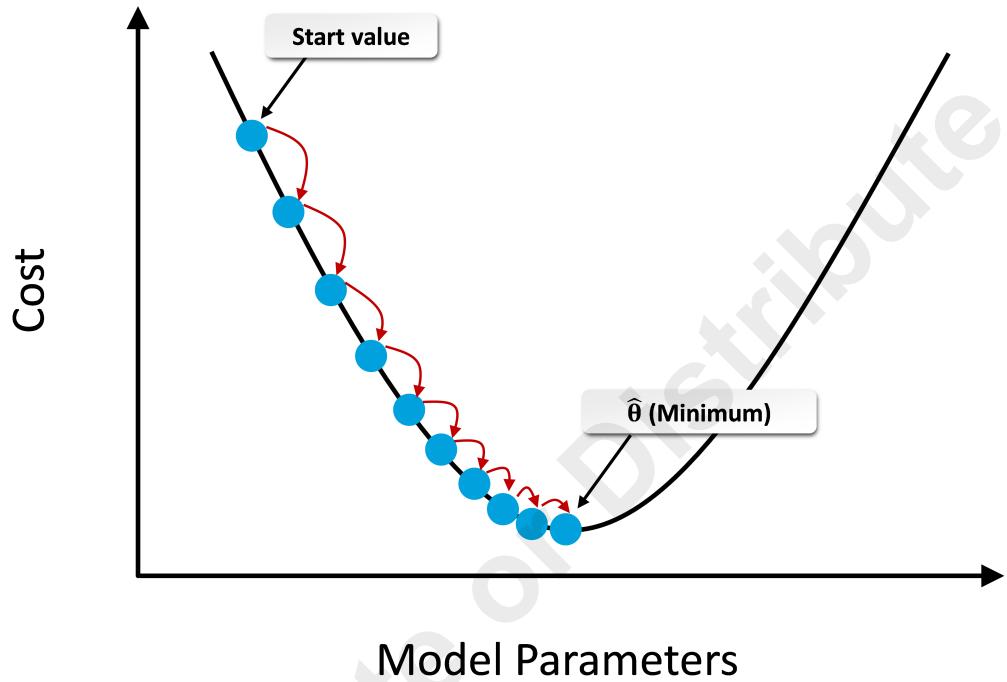


Figure 6–9: The gradient descent algorithm converges at the value that minimizes the cost function.

Gradient descent performs better than the normal equation in datasets that have a large number of features or examples, because calculating the inverse of a very large matrix of values takes a great deal of memory. Gradient descent avoids these memory issues and, depending on the type of gradient descent, may be able to work with virtually any size dataset. According to noted AI researcher Andrew Ng, a good rule of thumb is that a dataset with 10,000 or more examples and/or features will usually train too slowly for the normal equation and will likely be a better candidate for gradient descent. In any given scenario, if it's feasible to train a model with the normal equation, then do so—otherwise, consider an iterative approach.



Note: Gradient descent can be applied to many types of problems, not just regression.

Learning Rate

In gradient descent, the size of each "step" down the valley is called the **learning rate**. The learning rate is a hyperparameter (α) of the model that you must tune to get the best results. If the learning rate is small, more "steps" will be required to converge at the minimum. Too small, and the descent could end up taking a very long time.

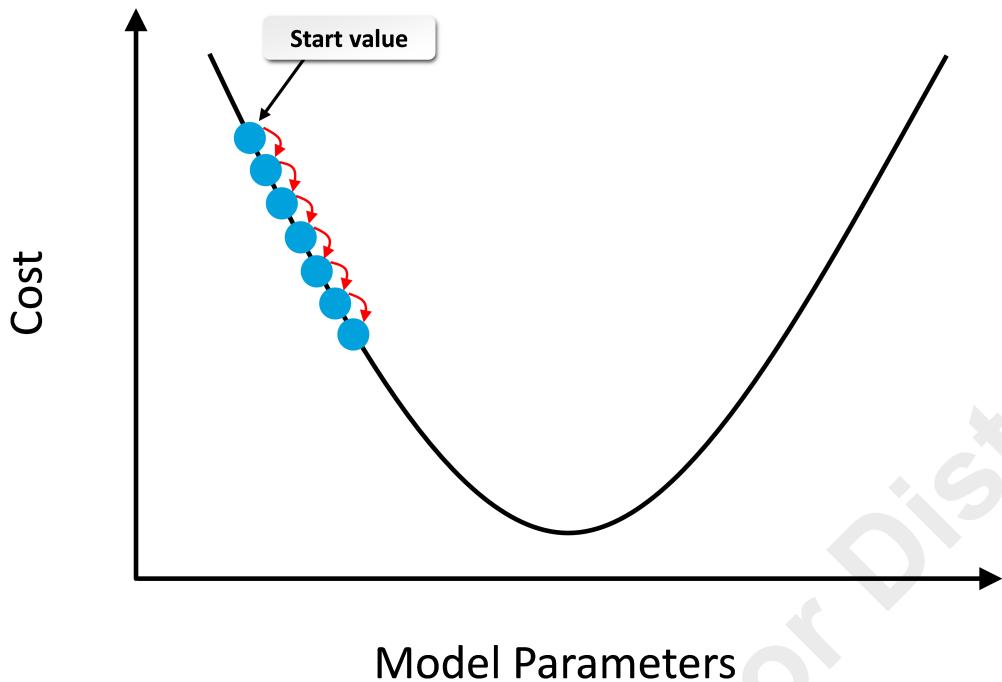


Figure 6-10: In this example, it will take a long time to reach the minimum because the learning rate is too small.

However, if the learning rate is too large, each "step" may jump to the opposite curve and actually end up higher than it was before. The model will therefore diverge instead of converging at a minimum.

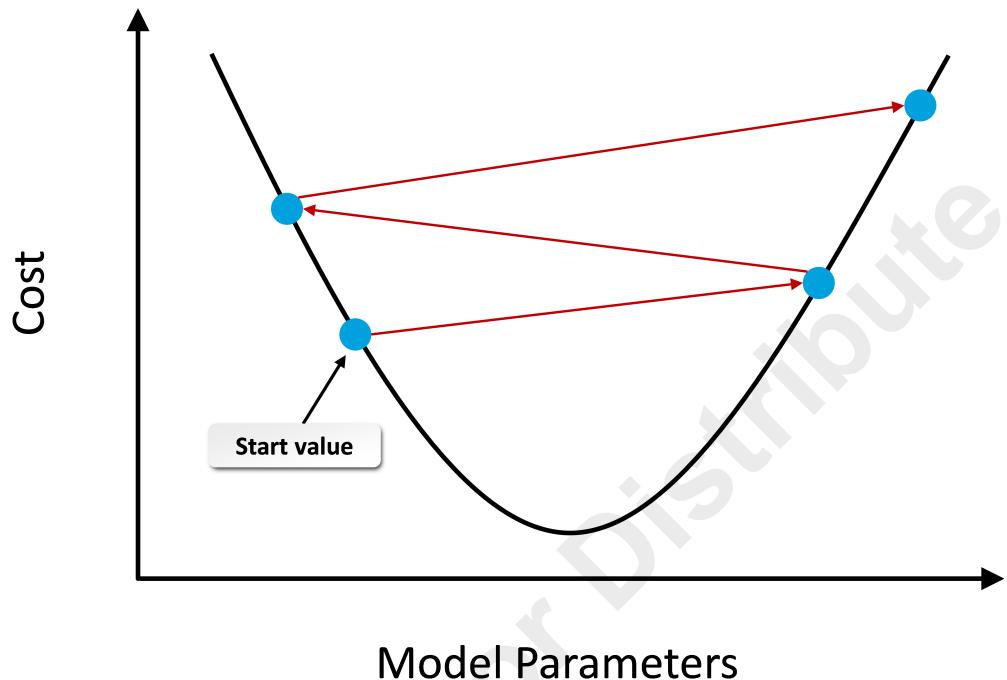


Figure 6–11: In this example, the model diverges because the learning rate is too large.

To avoid these issues, consider tuning the learning rate hyperparameter after several epochs. An epoch is one instance of the algorithm training over the entire dataset. If the error value decreases after an iteration, you can increase the learning rate slightly (e.g., around 5%). If the error value increases after an iteration, you can decrease the learning rate more significantly (e.g., around 50%). However, in most cases, you'll usually just start with a large learning rate and then gradually decrease it over time. This is one advantage the normal equation has over gradient descent—there's no need to tune a learning rate.



Note: In addition to tuning the learning rate, you also can speed up convergence by scaling your features.

Grid/Randomized Search for Regression

Just like with a classification task, you can use searching functions to find the optimal hyperparameters for regression tasks. The mechanics are essentially the same: define a parameter grid or distribution that you want to search, select an evaluation metric you want to optimize for, and then run the search. Of course, you must know which hyperparameters apply to your model, and which are actually worth your time trying to optimize. For example, basic linear regression is simple enough that there isn't much of a parameter space to search. However, once you incorporate techniques like regularization and gradient descent, you start to increase the number of hyperparameters that will have an impact on your model's performance. Likewise, algorithms like CART will have unique hyperparameters that other regression algorithms do not (e.g., max tree depth). The important thing is that you're fully aware of the different ways in which your regression model can be tuned.

The following is an example of a grid search used on a linear regression model that uses stochastic gradient descent (SGD). In scikit-learn, the `SGDRegressor()` class is used to build this model.

```
grid = [{}{'penalty': ['l1', 'l2'],
           'alpha': [0.0001, 0.001, 0.01, 0.1],
           'learning_rate': ['constant'],
           'eta0': [0.01, 0.05, 0.1]}]

search = GridSearchCV(model, param_grid = grid,
                      scoring = 'mean_squared_error',
                      cv = 5)
```

In this case, the model will be trained to use both ridge and lasso regularization, with several different alpha values that indicate the strength of regularization. In addition, the model will always be trained to use a constant number of steps in its descent (learning_rate), though the number of steps (eta0) will change for each training pass.

The combination of these hyperparameters that best reduces the mean squared error (MSE) cost function will be returned as the optimal model.

Guidelines for Tuning Regression Models

Follow these guidelines when tuning regression models.

Tune Regression Models

When tuning regression models:

- Use lower-order polynomials for linear regression to avoid overfitting the training data.
- Increase leaf size or tree size in tree-based regression models if overfitting is occurring.
- Tune the number of trees in a forest as well as leaf size, measuring overfitting with each change.
- Keep in mind that increasing leaf size for a decision tree can lead to diminished skill.
- Apply grid and/or randomized searches to evaluate a model across a range of hyperparameter values.

ACTIVITY 6–3

Tuning Regression Models

Before You Begin

Developing Regression Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Define the parameter grid used to tune the linear regression model**. Select **Cell→Run All Above**.

Scenario

You've trained some basic regression models, and now you can begin to tune them. You want to tune the linear regression model to see if you can improve its performance. Having a well-performing simple linear model would be beneficial since it's easier to explain and isn't as prone to overfitting as some others. You also want to see if you can reduce the high levels of overfitting in your lone decision tree. Again, having a well-performing simple model has its advantages.

You could also try to improve the XGBoost model, but in the interest of time, you'll just focus on the linear model and the decision tree for now.

1. Define the parameter grid used to tune the linear regression model.

- Scroll down and view the cell titled **Define the parameter grid used to tune the linear regression model**, then select the code cell below it.
- In the code cell, type the following:

```

1 param_grid = {'l1_ratio': [0.1, 0.5, 0.9],
2                 'alpha': [0.0001, 0.01, 0.1],
3                 'max_iter': [100, 1000, 10000]}
4
5 print(param_grid)

```

This will be the parameter grid used in a grid search.

- Run the code cell.
- Examine the output.

```
{'l1_ratio': [0.1, 0.5, 0.9], 'alpha': [0.0001, 0.01, 0.1], 'max_iter': [100, 1000, 10000]}
```

The parameter grid will be input into a linear model that uses elastic net regression to avoid overfitting. The hyperparameters you've defined in the grid are:

- l1_ratio**, which determines the weight of the ℓ_1 norm (lasso) or ℓ_2 norm (ridge). In this case, the lower the value, the more the regularization penalty will shift toward ridge regression. The higher the value, the more it will shift toward lasso regression.
- alpha**, which is the constant that determines the strength of regularization to apply. This is also called the λ (lambda) hyperparameter. Larger values lead to more regularization.
- max_iter**, which indicates the maximum number of iterations the model will perform using gradient descent to minimize the model's cost.

2. Perform a grid search for optimal elastic net hyperparameters.

- Scroll down and view the cell titled **Perform a grid search for optimal elastic net hyperparameters**, then select the code cell below it.
- In the code cell, type the following:

```

1 model = ElasticNet()
2 gs = GridSearchCV(estimator = model,
3                     param_grid = param_grid,
4                     n_jobs = -1,
5                     verbose = 2)
6
7 gs.fit(X_train, y_train)

```

In the case of scikit-learn, you don't apply elastic net regression as a hyperparameter to `LinearRegression()`, but as a separate function called `ElasticNet()`. The grid search will use the hyperparameter grid you just defined with the elastic net model.

- Run the code cell.



Note: It can take up to 5 minutes for the search to complete.

- Select the next code cell, then type the following:

```

1 print('Best R2 score: ', round(gs.best_score_, 4))
2 print('Best parameters: ', gs.best_params_)

```

- Run the code cell.
- Examine the output.

```

Best R2 score:  0.2343
Best parameters: {'alpha': 0.01, 'l1_ratio': 0.9, 'max_iter': 100}

```

The linear model improved, but only barely. It still has a score of ~0.23. Recall that you removed outliers manually, so that might have something to do with it. If you had more time, you could expand the hyperparameter grid to look for more values, but it's possible that linear regression is just not well-suited for this dataset. For now, the optimal hyperparameters are:

- An `alpha` regularization strength of `0.01`, which was in the middle of the grid values.
- An `l1_ratio` of `0.9`, which was the highest value in the grid.
- A `max_iter` of `100`, which was the lowest value in the grid.

3. Define the parameter grid used to tune the decision tree model.

- Scroll down and view the cell titled **Define the parameter grid used to tune the decision tree model**, then select the code cell below it.
- In the code cell, type the following:

```

1 param_grid = {'max_depth': [5, 10, 20],
2                 'min_samples_split': [10, 100, 1000],
3                 'min_samples_leaf': [10, 100, 1000]}
4
5 print(param_grid)

```

You'll run a new grid search, this time in an attempt to improve the poor performance of the single decision tree.

- Run the code cell.

- d) Examine the output.

```
{'max_depth': [5, 10, 20], 'min_samples_split': [10, 100, 1000], 'min_samples_leaf': [10, 100, 1000]}
```

This parameter grid will be input into the single decision tree algorithm. The hyperparameters you've defined in the grid are:

- `max_depth`, which determines the maximum depth of the tree before it stops splitting.
- `min_samples_split`, which defines how many samples are required in order to split a decision node.
- `min_samples_leaf`, which specifies how many samples are required to be at a leaf node.

4. Perform a grid search for optimal decision tree hyperparameters.

- Scroll down and view the cell titled **Perform a grid search for optimal decision tree hyperparameters**, then select the code cell below it.
- In the code cell, type the following:

```
1 model = DecisionTreeRegressor()
2 gs = GridSearchCV(estimator = model,
3                     param_grid = param_grid,
4                     n_jobs = -1,
5                     verbose = 2)
6
7 gs.fit(X_train, y_train)
```

The grid search is being conducted in the same way as before, but using `DecisionTreeRegressor()` this time.

- c) Run the code cell.



Note: This search should be much quicker than the previous one.

- d) Select the next code cell, then type the following:

```
1 print('Best R2 score: ', round(gs.best_score_, 4))
2 print('Best parameters: ', gs.best_params_)
```

- e) Run the code cell.
f) Examine the output.

```
Best R2 score:  0.3357
Best parameters: {'max_depth': 10, 'min_samples_leaf': 100, 'min_samples_split': 1000}
```

The decision tree's score is much improved at ~0.34, and the model appears to be better at learning patterns in the training data. The optimal hyperparameters are:

- A `max_depth` of 10, which was in the middle of the grid values.
- A `min_samples_leaf` of 100, which was also in the middle of the values.
- A `min_samples_split` of 1000, which was the highest value in the grid.

5. Keep this notebook open.

TOPIC B

Evaluate Regression Models

There are a few ways to evaluate regression models, so in this topic you'll apply the relevant metrics to see how well your models are performing.

Cost Functions Used to Evaluate Regression Models

There are several cost functions that you can use to evaluate the performance of regression and forecasting models. Some of the most common include:

- Mean squared error (MSE)
- Mean absolute error (MAE)
- Coefficient of determination (R^2)

Each of these functions outputs a "score" that you can use to guide your tuning efforts. For example, you may look for a lower or higher score (depending on the function) among several related models to see which one performs best according to that metric. However, like classification metrics, there is no objectively "good" score output by any of these regression metrics. The scores are highly dependent on the nature of the data, including the range of values in the label. So, it's important to consider cost scores as a method of comparison, not as a way to determine absolute performance of regression models.

Mean Squared Error (MSE)

The simplest and often most effective cost function in regression is the **mean squared error (MSE)**. Once the error (difference) between the estimated value and the actual value is calculated, that error is squared. Then, the average of these squared errors is taken. The reason for squaring the error is because errors can either be positive or negative; by simply taking the average of these values, the average will trend toward 0, which would not be very useful. Squaring the errors before taking the average makes them all positive.

This can be expressed as:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Where:

- n is the size of the training set, i.e., the total number of examples.
- y_i is the actual value of a variable.
- \hat{y}_i is the estimated value of the variable.
- $J(\boldsymbol{\theta})$ is the cost function itself.

You can also take the square root of MSE to get the **root mean squared error (RMSE)**. RMSE is to MSE as standard deviation is to variance—it makes the results more interpretable by putting them in the same scale as the label value the model is estimating.

Mean Absolute Error (MAE)

An alternative for calculating the cost function is the **mean absolute error (MAE)**, which calculates the average difference in values (the absolute value between y and \hat{y}) without considering the sign (i.e., positive or negative) of those values. One advantage of MAE over MSE is that MSE tends to minimize errors that are much less than 1, while also exaggerating errors that are much greater than 1; MAE is not as susceptible to this. However, MSE is often preferred because it is differentiable (a derivative exists for all values in the function) and it matches what the linear equation is minimizing.

MAE can be expressed as:

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Relative Error

Both of the mean error functions mentioned previously are most useful when comparing models whose estimated values are in the same scale. If you're tuning a model based on the same dataset, then this is likely to be the case. However, there may be circumstances where you want to compare the performance of regression models that estimate values on different scales. This is where the relative equivalents of each cost function come in handy. The *relative squared error* (RSE) and *relative absolute error* (RAE) both output scores that you can use for this type of comparison.

Coefficient of Determination

The **coefficient of determination**, also referred to as R^2 , is a statistical measure that indicates how much of a dependent variable's variance is explainable by a statistical model. In other words, it is the ratio of explained variation to total variation.

R^2 is commonly used as a scoring method for evaluating the performance of a regression model. R^2 is usually between 0 and 1. If the regression line were able to perfectly pass through every data point, the R^2 would be 1; on the other hand, as the line fails to pass through more points, its R^2 decreases, as it becomes less and less able to explain the variance. For example, an R^2 of 0.76 indicates that 76% of the variance in the target variable (i.e., the label you're trying to estimate) is explainable by the model. The other 24% cannot be explained by the model.

Although R^2 is typically positive, it can also be negative. This happens when the model actually performs *worse* than the baseline (i.e., random chance), which can be the result of the model learning the wrong patterns in the data. Despite using squaring operations in its calculations, the most common definition of R^2 subtracts those operations from 1, which is what can lead to a negative output.

It is important to note that a high R^2 value does not always imply a more skillful model, and vice versa. A model with a low R^2 may have better estimative power than a model with a high R^2 value. Even if the R^2 changes dramatically from model to model, the estimation error may still be exactly the same. This is because a strong connection between variables does not always imply that one variable has strong causality with another. Therefore, it's usually preferable to minimize a cost function like MSE rather than attempt to optimize R^2 .



Note: The coefficient of determination is not to be confused with the correlation coefficient. The latter is R , while the former is R^2 . As you might expect, the former takes the latter and multiplies it by itself. Whereas R^2 measures explained variance, R measures linear relationships between variables.

Guidelines for Evaluating Regression Models

Follow these guidelines when evaluating regression models.

Evaluate Regression Models

When evaluating regression models:

- Consider that regression models are usually not evaluated based on how well they fit, but on how poorly they fit (the cost).
- Prefer using a cost function like mean squared error (MSE) over R^2 when evaluating the skill of a regression model.
- Prefer using MSE over mean absolute error (MAE).
- Consider using *root* MSE to put the results on the same scale as the data, making them more interpretable.
- Consider using *relative* squared error/absolute error to compare models that estimate values on different scales.

ACTIVITY 6-4

Evaluating Regression Models

Before You Begin

Developing Regression Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Compare evaluation metrics for each model**. Select **Cell→Run All Above**.

Scenario

Your preliminary training would suggest that some of your regression models are better than others. But, there are more ways to evaluate these models, and you want to be sure you have a more comprehensive understanding of their performance. So, you'll compare each model using multiple metrics, then take a deeper look at one of the models you think is best.

1. Compare evaluation metrics for each model.

- Scroll down and view the cell titled **Compare evaluation metrics for each model**, then select the code cell below it.
- In the code cell, examine the following:

```

1 models = ['Linear Regression', 'Decision Tree',
2            'Random Forest', 'XGBoost', 'Dummy Regressor']
3
4 metrics = ['R2', 'MAE', 'MSE']
5
6 pred_list = ['linreg_y_pred', 'reg_tree_y_pred',
7               'rf_y_pred', 'xgb_y_pred', 'dummy_y_pred']
8
9 # Baseline algorithm.
10 dummy = DummyRegressor()
11 dummy.fit(X_train, y_train)
12 dummy_y_pred = dummy.predict(X_test)
13
14 scores = np.empty((0, 3))
15
16 for i in pred_list:
17     scores = np.append(scores,
18                         np.array([[r2_score(y_test, globals()[i]),
19                                    mean_absolute_error(y_test, globals()[i]),
20                                    mean_squared_error(y_test, globals()[i])]]),
21                         axis = 0)
22
23 scores = np.around(scores, 4)
24
25 scoring_df = pd.DataFrame(scores, index = models, columns = metrics)
26 scoring_df.sort_values(by = 'MSE', ascending = True)

```

This code should look familiar. It'll output a table that has each trained regression model as a row, as well as a dummy regressor, and each evaluation metric as a column.

- Run the code cell.

- d) Examine the output.

	R2	MAE	MSE
XGBoost	0.3102	431.7623	404530.2009
Random Forest	0.2956	424.5235	413118.7611
Linear Regression	0.2332	531.7977	449691.1279
Dummy Regressor	-0.0000	656.6227	586463.9422
Decision Tree	-0.4031	561.2292	822889.4076

The table shows each model's scores for R^2 , the mean absolute error (MAE), and the mean squared error (MSE). Lower values of MSE and MAE are best. The table is sorted by the lowest MSE, which is often the preferred regression metric, though MAE may be similarly useful. Both MSE and MAE tend to be more useful than R^2 , which identifies how much of the model's variance can be explained. MSE and MAE don't tell you much intuitively on their own, like a classification metric would, so they're best used as a point of comparison between models.

Some conclusions you can draw include:

- The XGBoost model has the best MSE and R^2 , and second best MAE.
- The random forest actually has a better MAE than XGBoost. Either of these models could be good candidates for "best" model, but for now, you'll go with XGBoost again.
- The unoptimized linear regression model is behind the ensemble methods, but is at least better than the dummy regressor baseline.
- The unoptimized decision tree has a better MAE than the dummy model, but worse R^2 and MSE.

2. Plot the residuals.

- a) Scroll down and view the cell titled **Plot the residuals**, then select the code cell below it.
 b) In the code cell, examine the following:

```

1 # Set up DataFrame for plotting.
2
3 resid_df = pd.DataFrame()
4 resid_df['total_amount_usd'] = y_test
5 resid_df['total_pred'] = xgb_y_pred
6 resid_df['residuals'] = resid_df['total_amount_usd'] - resid_df['total_pred']
7 resid_df = resid_df.sort_values('total_amount_usd')[:20]
8 resid_df['record_num'] = np.arange(len(resid_df))

```

You're going to build a residual plot from the XGBoost model, but first, you need to set up a DataFrame that has the residual values as well as the ground truth values and the predictions. A residual is just the magnitude of difference between the ground truth and the prediction. The closer the residual is to 0, the better. So, this kind of plot can help you visualize how far off the model is as compared to the prediction itself.

The users are being sorted by `total_amount_usd` so that you can more easily track how the increase in user balance does or does not affect the residuals. The sorted users are assigned a `record_num` that starts at 0 (the user with the lowest balance) and increments by one for each user with the next highest balance



Note: Only every 20 users are being plotted so that it'll be easier to read.

- c) Run the code cell.

- d) Select the next code cell, then type the following:

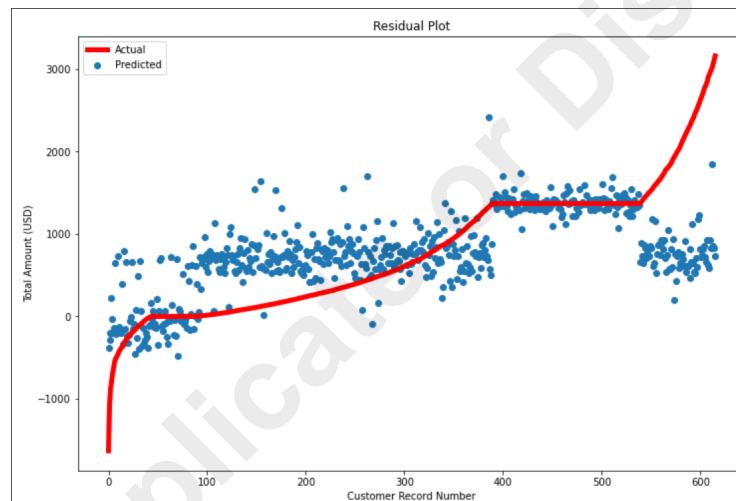
```

1 plt.figure(figsize = (12, 8))
2
3 plt.plot(resid_df['record_num'], resid_df['total_amount_usd'],
4           color = 'red', linewidth = 5)
5 plt.scatter(resid_df['record_num'], resid_df['total_pred'])
6
7 plt.legend(['Actual', 'Predicted'])
8 plt.title('Residual Plot')
9 plt.ylabel('Total Amount (USD)')
10 plt.xlabel('Customer Record Number')
11 plt.show();

```

This code plots the predictions as a scatter plot, and the ground truth values as a line on that same plot.

- e) Run the code cell.
f) Examine the output.



The users are plotted along the x-axis, where an increase in user record number indicates an increase in the ground truth of `total_amount_usd`. The y-axis shows the user's total balance, either predicted or ground truth. The line indicates the ground truth, whereas the dots indicate the model's predictions. The closer a dot is to the line where they both appear on the x-axis, the smaller the residual, and therefore the less error in the prediction.

For user record numbers around 150, the differences between the prediction dot and the ground truth line appears to be largest. In other words, the residuals here are relatively large. The actual balance for these users is around \$0, whereas the predictions could go as high as \$1,500+. Compare this to record numbers between 400 and 500, where the dots are very close to the flat portion of the line, indicating small residuals. The ground truth line is flat here because the values are the imputed mean of \$1,369.42.

Ultimately, you can use a plot like this to detect patterns in your model's effectiveness. The model might be better at predicting certain ranges of the target variable compared to other ranges.

3. Generate a feature importance plot.

- a) Scroll down and view the cell titled **Generate a feature importance plot**, then select the code cell below it.

- b) In the code cell, examine the following:

```

1 def feature_importance_plot(model, X_train, n):
2     """Plots feature importance. Only works for ensemble learning."""
3     plt.figure(figsize = (8, 5))
4     feat_importances = pd.Series(model.feature_importances_,
5                                   index = X_train.columns)
6     feat_importances.nlargest(n).plot(kind = 'barh')
7     plt.title(f'Top {n} Features')
8     plt.show()

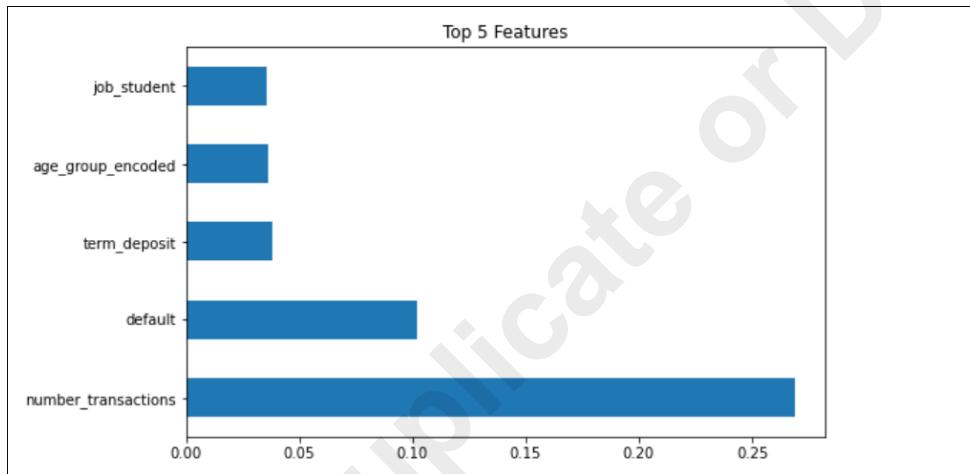
```

This is code you've seen before: it will generate a feature importance plot.

- c) Run the code cell.
d) Select the next code cell, then type the following:

```
1 feature_importance_plot(xgb, X_train, 5)
```

- e) Run the code cell.
f) Examine the output.



The most important feature to the XGBoost model was the number of transactions. This makes sense, as the number of transactions would likely impact the total balance a customer has. The second most important feature was whether or not the customer defaulted on a loan. Other features tend to be much less important in predicting the target variable.

4. Plot learning curves.

- a) Scroll down and view the cell titled **Plot learning curves**, then select the code cell below it.

- b) In the code cell, examine the following:

```

1 def plot_learning_curves(model, X_train, y_train):
2     """Plots learning curves for model validation."""
3     plt.figure(figsize = (5, 5))
4     train_sizes, train_scores, test_scores = \
5         learning_curve(model, X_train, y_train, cv = 5,
6                         scoring = 'neg_mean_squared_error',
7                         n_jobs = -1,
8                         shuffle = True,
9                         train_sizes = np.linspace(0.01, 1.0, 5))
10
11    # Means of training and test set scores.
12    train_mean = np.mean(train_scores, axis = 1)
13    test_mean = np.mean(test_scores, axis = 1)
14
15    # Draw lines.
16    plt.plot(train_sizes, train_mean, '--',
17              color = '#1f77b4', label = 'Training score')
18    plt.plot(train_sizes, test_mean,
19              color = '#1f77b4', label = 'Cross-validation score')
20
21    # Create plot.
22    plt.title('Learning Curves')
23    plt.xlabel('Training Set Size')
24    plt.ylabel('Negative MSE')
25    plt.legend(loc = 'best')
26    plt.tight_layout()
27
28    plt.show()

```

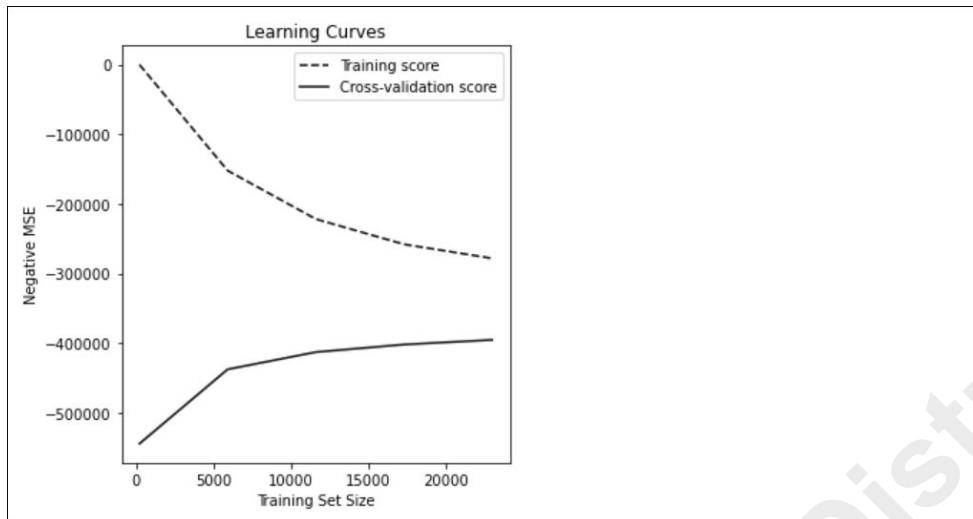
This is another block of code that you saw in the classification activities. One important thing to point out is that the plot is using the negative MSE instead of regular MSE. Making MSE negative simply means that higher values are ideal, rather than lower ones. This is useful when plotting, because it makes plots look consistent across other metrics, and the plot will generally be easier to read.

- c) Run the code cell.
d) Select the next code cell, then type the following:

```
1 plot_learning_curves(xgb, X_train, y_train)
```

- e) Run the code cell.

- f) Examine the output.



The training and cross-validation scores haven't yet converged, so the model will probably improve by adding more examples. Looking at the gap and positioning of the curves, the model does appear to be exhibiting some bias and variance issues. This could be improved by tuning the hyperparameters further, so in a production environment, you'd want to keep working on the model. For now, you'll take the model as is.

5. Save the best model.

- Scroll down and view the cell titled **Save the best model**, then select the code cell below it.
- In the code cell, type the following:

```
1 | pickle.dump(xgb, open('xgboost_regressor.pickle', 'wb'))
```

The XGBoost regressor will be saved as a binary pickle file.

- Run the code cell.

6. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Developing Regression Models** tab in Firefox, but keep a tab open to **CDSP/Regression/** in the file hierarchy.

Summary

In this lesson, you went through the process of training, tuning, and then evaluating models that are used in regression analysis. Depending on the industry, regression is not always as common as classification, but it's a huge aspect of machine learning nonetheless. By developing regression models, you can better estimate numeric values that may be of interest to the business.

What type of data in your organization do you think might be conducive to regression analysis?

What types of evaluation metrics do you think you'll find most useful when tuning a regression model?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

7

Developing Clustering Models

Lesson Time: 3 hours, 10 minutes

Lesson Introduction

You've built supervised learning models using both classification and regression. But now it's time to work with unsupervised learning, where labeled data is not readily available. In this lesson, you'll implement unsupervised learning in the form of clustering models, which can group observations that share common traits. Just like before, you'll develop these models as a process of training, tuning, and evaluation.

Lesson Objectives

In this lesson, you will:

- Train models using algorithms that solve clustering problems, then tune them to improve performance.
- Evaluate the performance of clustering models to inform the tuning process.

TOPIC A

Train and Tune Clustering Models

Even though clustering is an unsupervised learning task, you can still go through the familiar process of training and tuning models. There are some differences in that process, however, which you'll see in the following topic.

k-Means Clustering

k-means clustering is an algorithm for unsupervised machine learning that groups like data examples together for the purpose of revealing patterns in the data. It does this by defining a set of k groups (clusters). Each data example is placed within the cluster whose center (called a **centroid**) is the closest to that data example. Closeness can be defined by a distance metric that is chosen during training. So, you end up with clusters of data that exhibit statistical similarity, as visualized in the following figure.

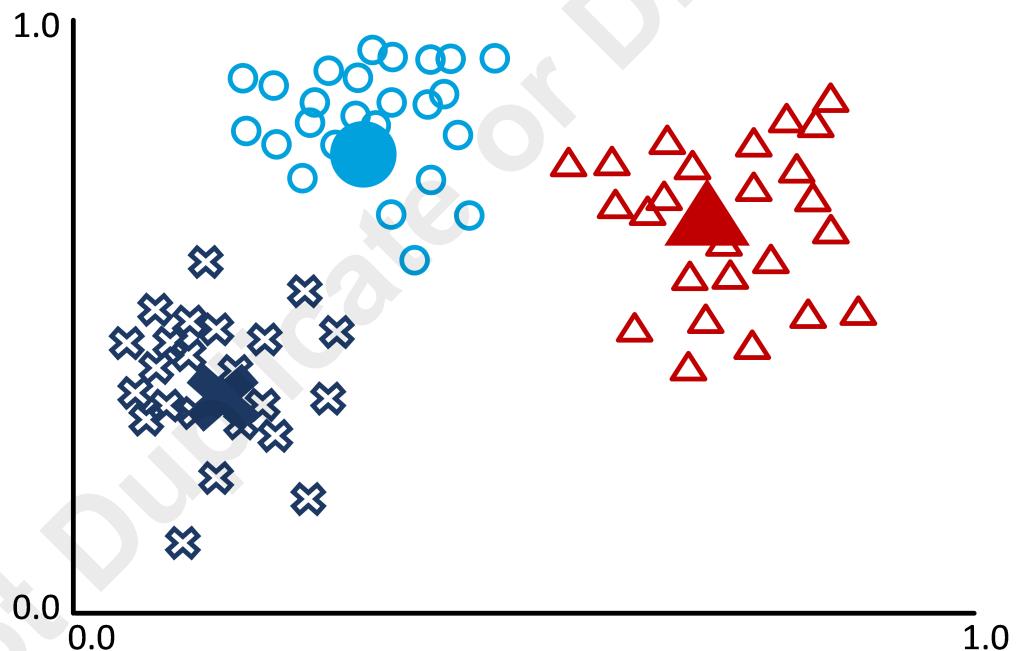


Figure 7-1: Three clusters of data, in which the large, filled-in shapes represent the centroids.

Once the algorithm assigns data examples to the initial clusters, it recomputes each centroid by calculating the mean of all data examples in the centroid's cluster. The centroids then move to this new mean value, and each data example is reassigned to whatever centroid is now closest. This process repeats until the data examples no longer change clusters (i.e., the k -means converge), or until a specified number of iterations is met.

k -means clustering is often used in the field of customer segmentation to group customers based on similar characteristics. It has also found use in categorizing sensor data from IoT devices, like images and video.

Global vs. Local Optimization

The ultimate goal of k -means clustering is the minimization of cost, much like other machine learning algorithms. The global cost function $J(\theta)$ can be written as follows:

$$J(\theta) = \sum_{i=1}^n \min_j (\text{dist}(x_i, c_j))$$

Where:

- n is the total number of examples.
- x_i is the i^{th} data example.
- c_j is the j^{th} centroid.

So, going from right to left, the distance between a data example and a centroid is taken (**dist**). Then, the centroid with the minimum (nearest) distance to the example is taken (**min**). Lastly, the sum of all the nearest distances is calculated. Ultimately, this helps to find the centroids that minimize this total distance. However, it is usually not feasible to apply this optimization globally. If you tried every possible assignment of n data examples to every cluster, you'd end up with many, many possible combinations—even if your sample size and number of clusters is low. For example, with an n of 25 and 4 as the number of clusters, there are roughly 47 *trillion* possible assignments.

This is why k -means clustering is an iterative algorithm that requires local optimization. The process is as follows:

1. It begins by taking the number of desired clusters, then it randomly assigns centroids for each cluster.
2. Next, it assigns each data example to whatever centroid is currently closest. This is effectively the same as minimizing the cost of these assignments.
3. Then, the algorithm moves each centroid so that it is in the center of the data examples that were assigned to it. This is effectively the same as minimizing the cost of the centroids.
4. The process repeats until convergence or until an iteration maximum is met.

So, this iterative cost minimizing scheme is a more efficient approach to optimization. However, it doesn't always achieve the perfect global optimization, especially if the initial randomly selected centroids were placed in sub-optimal locations. You can re-initialize the k -means algorithm with different randomly chosen centroids to overcome this.

k Determination

In k -means clustering, the primary challenge is determining k (number of clusters). There are several evaluation metrics you can use to help you determine the optimal k . Before you use these metrics, however, you should start by assessing the problem you're trying to solve. Based on your knowledge of the domain, you may be able to place useful constraints on the data. For example, assume you have an unlabeled dataset of thousands of photographs that depict cityscapes. You want to organize these images by the cities in which they were taken. The images don't have location-based metadata, so you'll need to rely on the actual contents of each image to identify its location. In this case, you know that each image was taken in one of four possible cities. So, because of your domain knowledge, you know that you will only accept four clusters—no more, no less. You therefore won't need to estimate a "correct" number of clusters.

Guidelines for Training k -Means Clustering Models



Note: All Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Follow these guidelines when training k -means clustering models.

Train k -Means Clustering Models

When training k -means clustering models:

- Use k -means clustering for unsupervised learning to find groups of data points that share similarities.
 - Examples: customer segmentation, categorizing images and video, categorizing alert data, etc.
- Apply k -means clustering to data that is spherical or overlapping in nature.
- Consider applying domain knowledge in determining k —you may know exactly or approximately how many clusters you need or is best for the problem.

ACTIVITY 7-1

Training a k -Means Clustering Model

Data Files

/home/student/CDSP/Clustering/Developing Clustering Models.ipynb
 /home/student/CDSP/Clustering/data/users_data_final.pickle

Scenario

Although supervised learning has been very useful to the project so far, there are other ways you can learn more from GCNB's customer data that don't involve labels. The marketing department might shape their campaigns to maximize term deposit subscriptions, but that's just one of many factors that define customers and their behavior. The business would benefit greatly from being able to target specific segments of customers for their marketing campaigns. Targeted marketing can lead to greater customer engagement with many different facets of the business. After all, the bank offers more services than just term deposits.

So, you decide to see how you can segment the bank's customers using cluster analysis. There are a few different methods you can try, but you'll start by building a model that leverages the k -means algorithm for generating clusters.

1. Open the notebook.

- In the Jupyter Notebook web client, navigate to the **CDSP/Clustering** directory.
- Select **Developing Clustering Models.ipynb** to open it.

2. Import the relevant software libraries.

- View the cell titled **Import software libraries**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.

3. Load and preview the data.

- Scroll down and view the cell titled **Load and preview the data**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data = pd.read_pickle('data/users_data_final.pickle')
2
3 users_data.head(n = 5)
```

- Run the code cell.

- d) Examine the output.

	user_id	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_se
0	9231c446-cb16-4b2b-a7f7-ddfb8b25aa6	3.0	2143.00	1	0	0	0	0	0	0
1	bb92765a-08de-4963-b432-496524d39157	0.0	1369.42	0	1	0	0	0	0	0
2	573de577-49ef-42b9-83da-d3cfb817b5c1	2.0	2.00	0	0	1	0	0	0	0
3	d6b66994-7c8f-4257-a682-e136f640b7e3	0.0	1369.42	0	0	0	1	0	0	0
4	fade0b20-7594-4d9a-84cd-c02f79b1b526	1.0	1.00	0	0	0	0	0	0	0

5 rows × 33 columns

This is the same dataset of GCNB customer information that you've been using. You'll want to make sure everything's ready to be used in unsupervised learning.

4. Check the shape of the data.

- a) Scroll down and view the cell titled **Check the shape of the data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_data.shape
```

- c) Run the code cell.
 d) Examine the output.

```
(45179, 33)
```

There are 45,179 records and 33 columns.

5. Check the data types.

- a) Scroll down and view the cell titled **Check the data types**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_data.info()
```

- c) Run the code cell.

- d) Examine the output.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45179 entries, 0 to 45215
Data columns (total 33 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   user_id          45179 non-null   object  
 1   number_transactions 45179 non-null   float64 
 2   total_amount_usd   45179 non-null   float64 
 3   job_management     45179 non-null   int64  
 4   job_technician     45179 non-null   int64  
 5   job_entrepreneur   45179 non-null   int64  
 6   job_blue-collar    45179 non-null   int64  
 7   job_retired        45179 non-null   int64  
 8   job_admin.         45179 non-null   int64  
 9   job_services       45179 non-null   int64  
 10  job_self-employed  45179 non-null   int64  
 11  job_unemployed    45179 non-null   int64  
 12  job_housemaid     45179 non-null   int64  
 13  job_student        45179 non-null   int64  
 14  education_tertiary 45179 non-null   int64
```

All of the columns are in the proper data formats for machine learning, except for `user_id`, which you'll remove.

6. Filter by demographics data.

- a) Scroll down and view the cell titled **Filter by demographics data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_data_demographics = \
2 users_data.filter(regex = 'education|job|age|single')
3
4 users_data_demographics.head(n = 3)
```

Since you're interested in clustering customers by their characteristics, rather than their direct interactions with the bank, you'll filter the data to only include demographic features.

- c) Run the code cell.
 d) Examine the output.

	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_housemaid	job_stude
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0

As expected, the resulting `DataFrame` only includes demographic information.

7. Scale the data.

- a) Scroll down and view the cell titled **Scale the data**, then select the code cell below it.
 b) In the code cell, type the following:

```
1 users_data_demographics.describe()
```

- c) Run the code cell.

- d) Examine the output.

	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_hous
count	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000	45179.000000
mean	0.209234	0.168043	0.032869	0.215255	0.050068	0.114389	0.091901	0.034906	0.028797	0.0
std	0.406767	0.373908	0.178296	0.411004	0.218087	0.318287	0.288889	0.183543	0.167236	0.1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

Because k -means is a distance-based algorithm, it's a good idea to scale all of these features.

- e) Select the next code cell, then type the following:

```

1 # Standardize the data.
2
3 scaler = StandardScaler()
4
5 scaler.fit(users_data_demographics)
6 users_data_scaled = scaler.transform(users_data_demographics)
7
8 print('New standard deviation: ', users_data_scaled.std())
9 print('New mean: ', round(users_data_scaled.mean()))

```

This code standardizes the data so that the mean of the distribution is 0 and the standard deviation is 1.

- f) Run the code cell.
g) Examine the output.

```

New standard deviation: 1.0
New mean: 0

```

As expected, the data has been scaled using the z -score.

8. Train a k -means clustering model.

- a) Scroll down and view the cell titled **Train a k -means clustering model**, then select the code cell below it.
b) In the code cell, type the following:

```

1 # Specify initial number clusters.
2
3 n_clusters = 5

```

Unfortunately, the problem at hand doesn't really suggest a "good" number of clusters to divide the customers into. Without domain knowledge to guide you, you'll need to start clustering using an arbitrary value for k . You'll be able to tune and evaluate this value later to determine a better number of clusters.

- c) Run the code cell.

- d) Select the next code cell, then type the following:

```

1 # Build k-means model.
2
3 kmeans = KMeans(n_clusters = n_clusters, random_state = 10)
4
5 # Fit scaled data to model.
6
7 kmeans.fit(users_data_scaled)

```

This code constructs the *k*-means model using the scaled data. It'll divide the data into the number of clusters you just specified (5).

- e) Run the code cell.

9. Generate the clusters.

- a) Scroll down and view the cell titled **Generate the clusters**, then select the code cell below it.
 b) In the code cell, type the following:

```

1 y_kmeans = kmeans.predict(users_data_scaled)
2
3 results = pd.DataFrame(users_data_demographics)
4 results.insert(0, 'cluster', y_kmeans)
5 results.head()

```

The `predict()` function is used to assign cluster labels to each data example. In this case, you'll assign labels to the existing training set rather than a new set.

- c) Run the code cell.
 d) Examine the output.

	cluster	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_housemaid	job_other
0	0	1	0	0	0	0	0	0	0	0	0	0
1	2	0	1	0	0	0	0	0	0	0	0	0
2	2	0	0	1	0	0	0	0	0	0	0	0
3	4	0	0	0	1	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	0	0

The new `cluster` column indicates what cluster each data example is in. The clusters are simply numbers, as it's up to you to determine what each cluster means qualitatively. The first customer is in cluster 0, the second, third, and fifth customers are in cluster 2, and the fourth customer is in cluster 4. There isn't necessarily a common feature shared among all three examples in cluster 2, though they appear to be people that are relatively young and have less than a tertiary education. However, focusing on more than just three examples could reveal more obvious patterns.

10. Visualize the number of users in each cluster.

- a) Scroll down and view the cell titled **Visualize the number of users in each cluster**, then select the code cell below it.

- b) In the code cell, examine the following:

```

1 def cluster_bar(cluster_labels):
2     """Create a bar chart to show number of users in each cluster."""
3     pd.DataFrame(Counter(cluster_labels).most_common()). \
4         set_index(0).plot.bar(legend = None)
5
6     plt.title('Distribution of Clusters')
7     plt.xlabel('Cluster ID')
8     plt.xticks(rotation = 0)
9     plt.ylabel('Number of users in cluster');

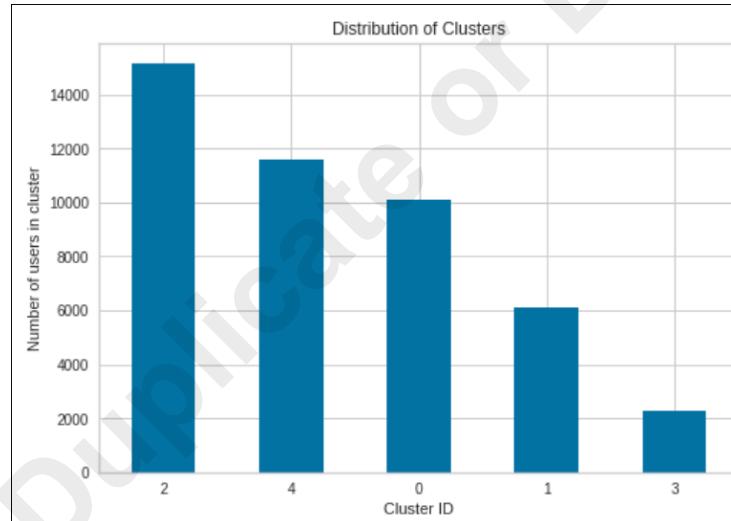
```

This function will plot the number of customers in each cluster.

- c) Run the code cell.
d) Select the next code cell, then type the following:

```
1 cluster_bar(y_kmeans)
```

- e) Run the code cell.
f) Examine the output.



In this model, cluster 2 includes the most customers. Cluster 3 has the least. The decreasing number of customers in each cluster seems to follow a relatively stable pattern; i.e., no one cluster dominates all the others, nor is one cluster dominated by the others.

11. Keep this notebook open.

k-Means Clustering Shortcomings

The k -means clustering algorithm is insensitive to data that is overlapping and works well with data that appears spherical in nature. However, consider the following graph:

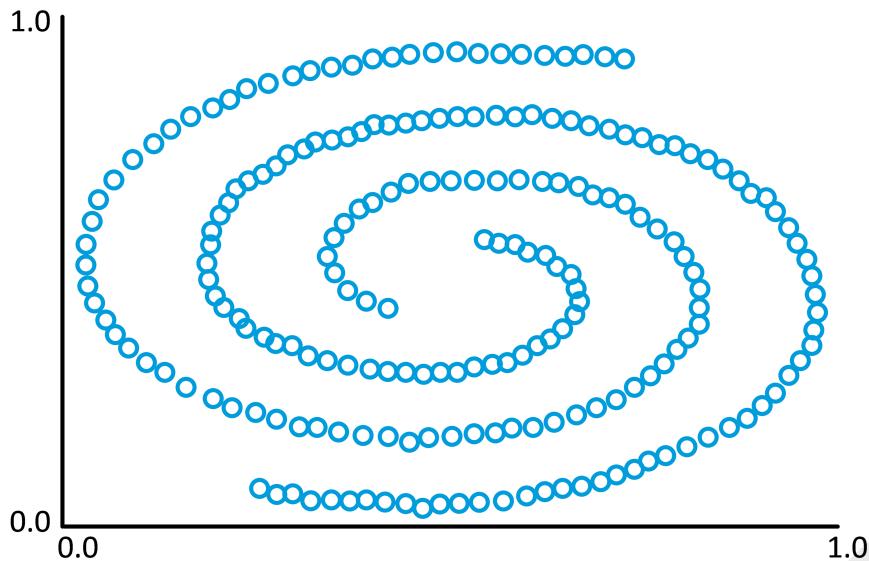


Figure 7-2: A spiral-shaped dataset.

This type of data distribution will present problems when trying to compute centroids until the means converge. For example, in the following figure, the graph on the left shows the initial centroid values, and the graph on the right shows the centroids after several iterations.

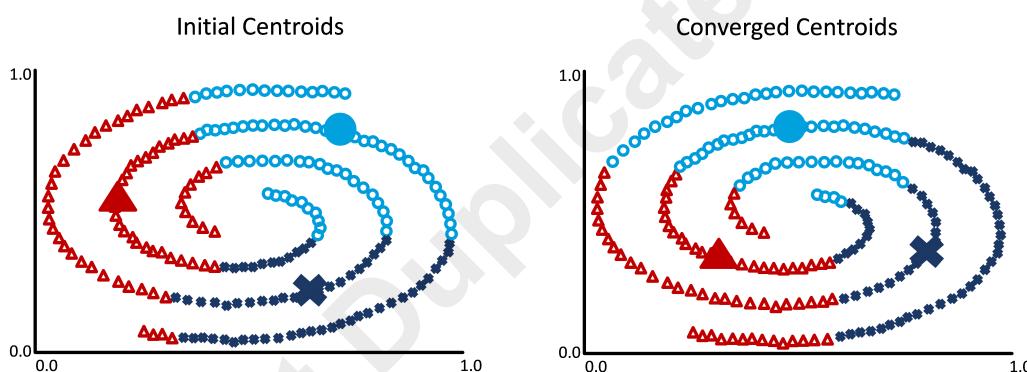


Figure 7-3: The clusters in this spiral-shaped dataset aren't particularly useful.

As you can see, the clusters don't do a good job of logically grouping similar data examples, even when the means start to converge. So, to summarize, *k*-means clustering is not very useful at clustering circular or spiral data—in other words, data that is well separated. This is where hierarchical clustering comes into play.

Hierarchical Clustering

Hierarchical clustering is a clustering method that, as the name implies, builds a hierarchy of groups in which similar data is placed. There are actually two approaches to hierarchical clustering: hierarchical agglomerative clustering (HAC) and hierarchical divisive clustering (HDC).

In **hierarchical agglomerative clustering (HAC)**, each data example starts out as its own cluster, then the two closest points are clustered together, then the next closest two points are clustered, and so on, until some stopping criterion is reached. HAC is referred to as a "bottom-up" approach to

constructing a hierarchy of clusters. The following figure demonstrates the first four iterations of this algorithm on a small sample, using two clusters.

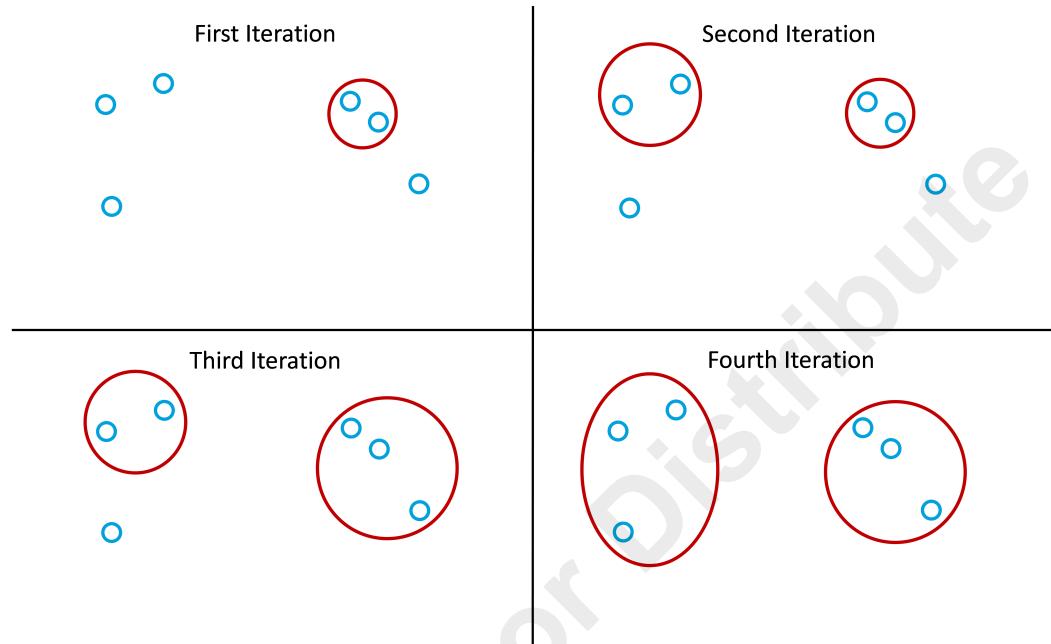


Figure 7-4: HAC starts merging the closest data points into clusters. A fifth iteration (not shown) would merge all of the points into one cluster.

Likewise, the following figure demonstrates HAC with three clusters.

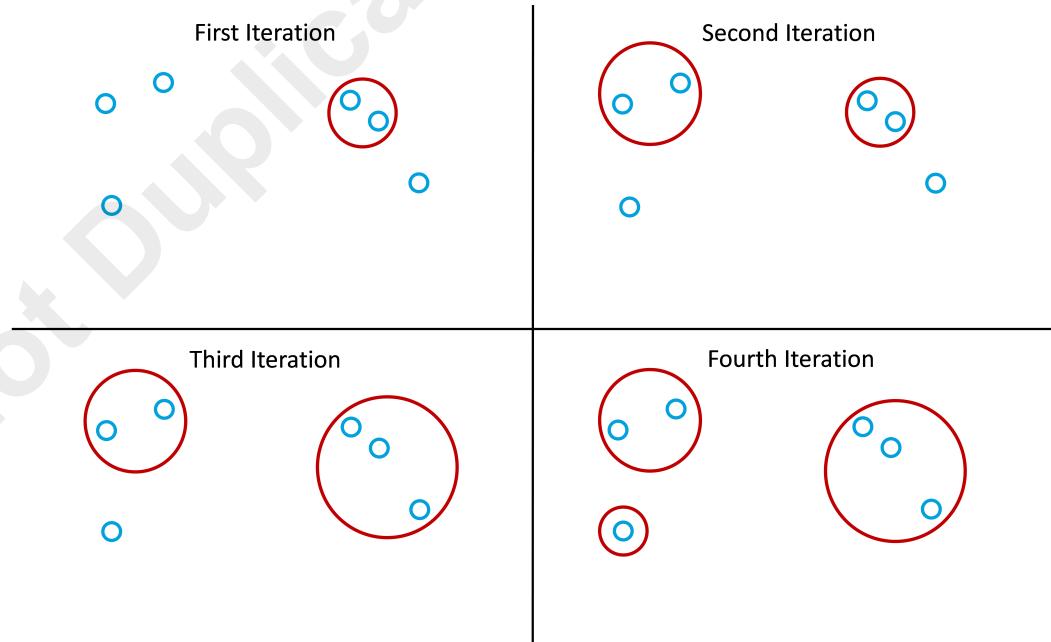


Figure 7-5: HAC applied to a three-cluster problem.

Hierarchical divisive clustering (HDC) is the opposite of HAC—all examples start out in the same single cluster, then this cluster is split using a particular method. The process continues until

some stopping criterion is reached. The splitting method used in HDC can vary, and is more complex than simply taking the distance between two points like in HAC. This is because there are many different ways to split a cluster. HDC is referred to as a "top-down" approach to constructing a hierarchy of clusters.

While HAC has the advantage of its simplicity, HDC tends to be more efficient if the complete hierarchy is not created all the way to the bottom level. Also, HDC can be more skillful in its segmentation than HAC due to the higher level of complexity involved in its splitting decisions.

Hierarchical Clustering Applied to a Spiral Dataset

When applied to the spiral dataset shown previously, a hierarchical clustering algorithm (whether agglomerative or divisive) will likely produce the graph in the following figure.

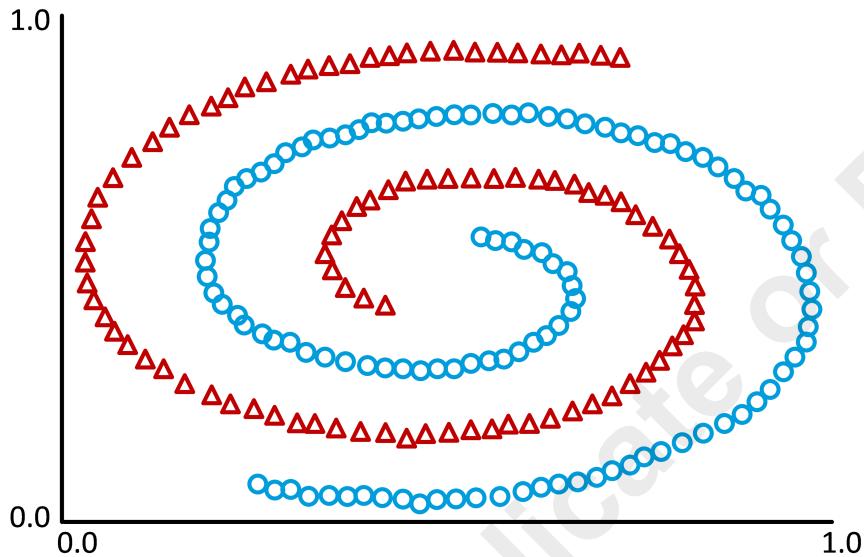


Figure 7-6: Hierarchical clustering applied to spiral data.

As you can see, the two clusters this hierarchical algorithm produced are segmented more logically than the ones produced by the k -means clustering algorithm.



Note: Hierarchical clustering can be applied to more than just datasets that make this exact shape. Any data that is well separated may be applicable to hierarchical clustering. One real-world example of data that might make this shape is topographical data placed on a map.

Hierarchical Clustering Applied to Spherical Data

Just like k -means clustering isn't all that useful on well-separated/spiral data, hierarchical clustering doesn't perform all that well on overlapping/spherical data. It will often lead to the majority of examples being placed into a single cluster or a few overly large clusters, with the outliers being placed into their own individual clusters.

Guidelines for Training Hierarchical Clustering Models

Follow these guidelines when training hierarchical clustering models.

Train Hierarchical Clustering Models

When training hierarchical clustering models:

- Like k -means clustering, use hierarchical clustering to find groups of data points that share similarities.
- Use hierarchical clustering on data that is well separated in shape, and does not overlap.
- Consider using hierarchical agglomerative clustering (HAC) over hierarchical divisive clustering (HDC) if simplicity is desired.
- Consider using HDC over HAC to maximize clustering skill.

ACTIVITY 7-2

Training a Hierarchical Clustering Model

Before You Begin

Developing Clustering Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Filter the data by education and scale it**. Select **Cell**→**Run All Above**.

Scenario

The *k*-means approach is the most common method for cluster analysis in unsupervised learning, and it seems to be a good choice for the GCNB dataset. Hierarchical clustering is less common and has more limited applications, and may not be as suitable for this dataset. Still, like any other type of machine learning, it's always a good idea to try out multiple algorithms when time permits. So, you'll generate a hierarchical clustering model to see how it groups customers.

1. Filter the data by education and scale it.

- a) Scroll down and view the cell titled **Filter the data by education and scale it**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 users_data_reduced = users_data.filter(regex = 'education')
2 users_data_reduced.head(n = 3)
```

Hierarchical clustering might benefit from a smaller feature space, so you'll filter the dataset to only include education level for this module.

- c) Run the code cell.
- d) Examine the output.

	education_tertiary	education_secondary	education_Unknown	education_primary
0	1	0	0	0
1	0	1	0	0
2	0	1	0	0

All four education categories are listed.

- e) Select the next code cell, then type the following:

```

1 # Standardize the data.
2
3 scaler = StandardScaler()
4
5 scaler.fit(users_data_reduced)
6 users_data_reduced_scaled = scaler.transform(users_data_reduced)

```

Since all of the data is in binary format, it's not strictly necessary to standardize it, but you'll do it anyway for consistency.

- f) Run the code cell.

2. Train a hierarchical clustering model.

- a) Scroll down and view the cell titled **Train a hierarchical clustering model**, then select the code cell below it.
 b) In the code cell, type the following:

```

1 agglom = AgglomerativeClustering(n_clusters = n_clusters,
2                                   affinity = 'euclidean',
3                                   linkage = 'single')
4
5 agglom.fit(users_data_reduced_scaled)
6 y_agglom = agglom.fit_predict(users_data_reduced_scaled)
7
8 results['cluster'] = y_agglom
9 results.head()

```

This code builds a hierarchical agglomerative clustering (HAC) model, which employs a bottom-up approach to generating clusters. Aside from the number of clusters, which is the same as the k -means model (5), the defined hyperparameters are:

- **linkage**, which refers to the linkage criterion. The linkage criterion specifies the distance metric to use when comparing two sets of data examples. In this case, the model is using the '`'single'`' method, which uses the minimum of the distances between all data examples in the two sets. This enables the model to determine which clusters to merge. Although not as effective as other linkage criteria, it tends to be more efficient on larger datasets.
- **affinity**, which is the method used to compute the linkage. The '`'euclidean'`' method refers to Euclidean distance, which is the distance of a straight line drawn between two points in Euclidean space. This is the most intuitive measurement of distance, though there are many others.

- c) Run the code cell.



Note: It will take a couple minutes for training to complete.

- d) Examine the output.

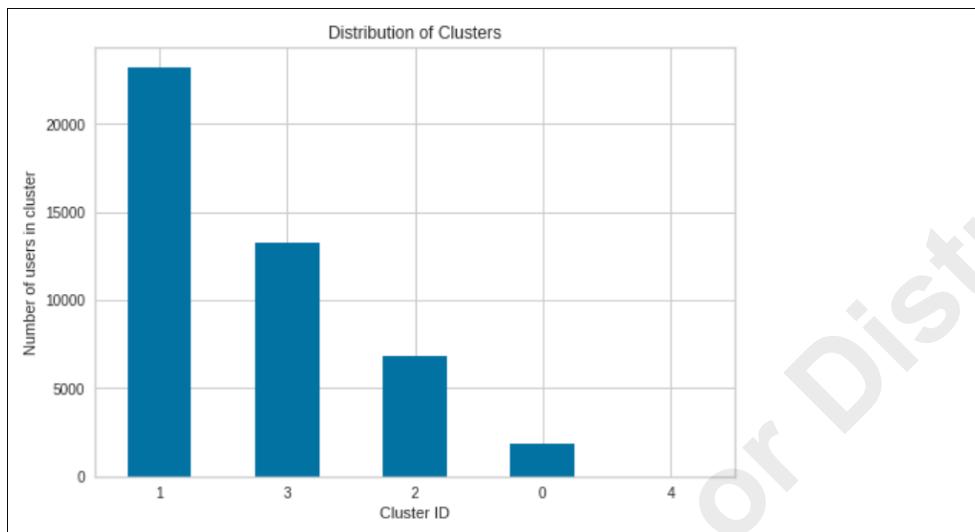
	cluster	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_housemaid	job_farmhand
0	3	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
2	1	0	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0

The agglomerative model produced different results for the first five customers as compared to the k -means model. The first customer is in cluster 3, the second and third customers are in cluster 1, and the fourth and fifth customers are in cluster 0.

- e) Select the next code cell, then type the following:

```
1 cluster_bar(y_agglom)
```

- f) Run the code cell.
g) Examine the output.



Cluster 1 has the most customers in it, whereas cluster 4 seems to have hardly any. While it's interesting to see the differences between the two models, ultimately, the k -means model is probably the better choice for this data. Later on, you'll focus mostly on tuning and evaluating the k -means model.

3. Keep this notebook open.

Latent Class Analysis

Latent class analysis (LCA) is a form of unsupervised learning that groups data examples together into groups called latent classes. A latent class refers to the concept of a *latent* variable—a variable that is not directly observed, but is only inferred through the presence and interaction of other observable variables. These latent variables might be inherently infeasible to measure, such as in the case of abstract concepts, or they may be very difficult to measure (i.e., they are "hidden"). For example, if you wanted to classify psychiatric patients in terms of their mental state, abstract descriptions like "sad" or "happy" are not really measurable. Or, perhaps you want to measure each patient's age, but you don't actually have this data available. Both of these variables can be considered latent—they may not be directly observable, but they're both still useful to you.



Note: Observable variables whose relationships might reveal a latent mental state variable could be: the number of visitors the patient receives; whether or not the patient was admitted to the hospital voluntarily; a score assigned by a psychiatrist that assesses the patient's level of improvement over time; the types of medication the patient is receiving; and so on.

So, a latent *class* is an unobservable group or cluster within which data examples are placed. The classes are inferred from the observable features in the dataset. Using the psychiatric example, if you had a dataset of patient attributes and wanted to cluster them according to mental state, LCA would be able to do so by measuring similarities between these attributes. It won't necessarily tell you which cluster is "sad" or "happy," but it will be able to create the clusters for you to analyze.

In terms of its mechanism of action, LCA is actually very similar to k -means clustering. After initializing the clusters, it recomputes centroids using a descriptive statistic like mean. However, one difference is that LCA also measures the size of each cluster. If a data example is equally similar to two clusters, it will likely be placed into the larger of the two clusters. Also, when computing the distance between each example and its centroid, LCA will also incorporate the cluster size in order to generate a probability for membership in the cluster. When it comes time to recompute the centroids, LCA uses this probability as a weight. In k -means clustering, the "weight" is either a 1 or 0. Otherwise, the process repeats in the same basic way until convergence or an iteration maximum is met.

LCA has several advantages over k -means clustering, including:

- Assigning a data example to the larger of two clusters when the example is equally distant between the two clusters is an effective way of reducing uncertainty.
- Incorporating weighted calculations typically results in more useful clusters.
- It tends to do better at handling heterogeneous, missing, and varied data.

However, LCA's biggest disadvantage is that it is much slower, particularly when it comes to large datasets. Consider using LCA over k -means clustering when your sample size is small and/or time is not a factor. Also note that there is currently no support for LCA in any of the major Python® data science libraries, though there is LCA support in R.

Clustering Hyperparameters and Tuning

Compared to some supervised learning algorithms, clustering algorithms tend to have fewer hyperparameters for you to tune. Perhaps the most important hyperparameter is the number of clusters, or k . Unless you're limited to a specific k value based on the problem you're trying to solve, you may want to train multiple models with different k values to observe how data is clustered differently when the number of clusters changes. This can reveal new patterns in the data that you wouldn't have otherwise considered. Or, it might find a better fit for certain data examples.

Other than k , some hyperparameters common to clustering models include:

- **Random centroid initialization:** Defines how the centroids are initialized. They can be initialized pseudorandomly without user input, or you can specify a seed value to the pseudorandom number generator (PRNG) to make the results of the random number generator deterministic. In other words, if you provide the same seed to the same data and parameters, the function call will produce the same model every time. You might want to provide a seed value if being able to exactly replicate results is important to you.
- **Max iterations:** The maximum number of times the centroids will be recalculated. It may take a long time for clusters to converge when working with larger datasets, so setting a limit to the number of iterations will ensure training stops at a specific point, even if the centroids aren't fully optimized.
- **Linkage:** The distance metric to use in hierarchical clustering. For example, 'ward' minimizes the variance of the merging clusters, whereas 'average' uses the average distance of data examples between two sets. This hyperparameter can affect the ultimate shape and location of clusters.

Unlike with supervised learning algorithms, you typically wouldn't use a grid or randomized search to find the clustering model with the optimal hyperparameters. There are some scoring metrics you can use, but it's usually just a matter of generating a few models with different k values and comparing their scores. There's also not much point in cross-validation without labels.

Guidelines for Tuning Clustering Models

Follow these guidelines when tuning clustering models.

Tune Clustering Models

When tuning clustering models:

- Encode categorical data to integers before training a clustering model.
- Try different numbers of clusters, avoiding sizes that generate clusters with too many or too few data points.
- Plot values in different colors based on each point's cluster to visually determine the usefulness of clustering.
- Consider that centroids that are close to each other indicate the data may not be suitable for clustering.
- Evaluate a dataset with multiple clustering models to choose the optimal one.
- Use features with the lowest positive/negative correlation (closer to zero) for best clustering performance.

ACTIVITY 7–3

Tuning Clustering Models

Data File

/home/student/CDSP/Clustering/data/users_data_demo_pca.pickle

Before You Begin

Developing Clustering Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel→Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Adjust the number of clusters in the k-means clustering model**. Select **Cell→Run All Above**.

Scenario

You developed an initial k -means clustering model based on an arbitrary choice of 5 clusters. However, you want to adjust the model to see how it changes in response to a different number of clusters. Also, earlier in the preprocessing phase, you reduced the dimensionality of the user demographics data. So, you'll load this reduced dataset to make the clusters easier to visualize on a chart.

1. Adjust the number of clusters in the k -means clustering model.

- Scroll down and view the cell titled **Adjust the number of clusters in the k -means clustering model**, then select the code cell below it.
- In the code cell, type the following:

```
1 n_clusters = 10
2
3 kmeans = KMeans(n_clusters = n_clusters, random_state = 10)
4
5 kmeans.fit(users_data_scaled)
```

This time, the number of clusters the model will output is 10. Experimenting with different numbers of clusters can lead to interesting results.

- Run the code cell.

2. Examine the results.

- Scroll down and view the cell titled **Examine the results**, then select the code cell below it.
- In the code cell, type the following:

```
1 y_kmeans = kmeans.predict(users_data_scaled)
2
3 results['cluster'] = y_kmeans
4 results.head()
```

- Run the code cell.

- d) Examine the output.

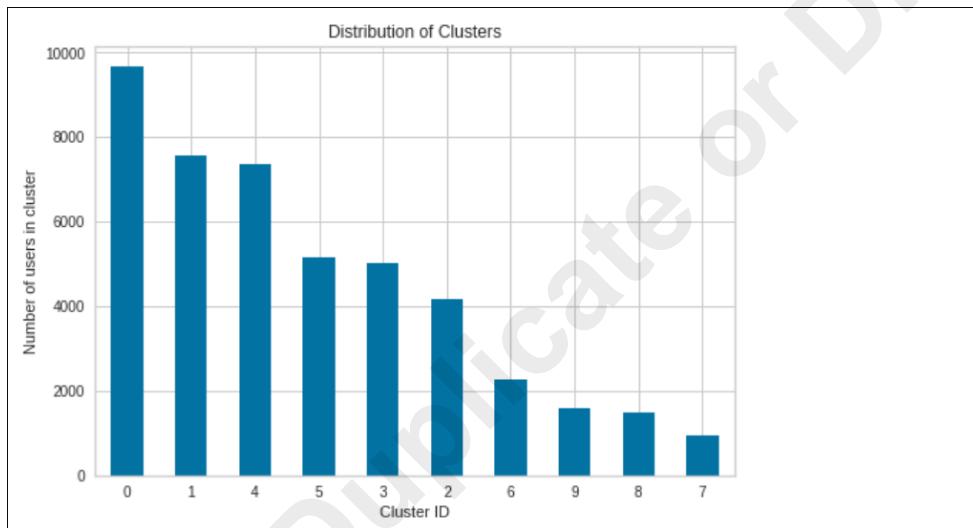
	cluster	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_housemaid	job_farm
0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
2	8	0	0	1	0	0	0	0	0	0	0	0
3	4	0	0	0	1	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0	0	0	0

Once again, the results of the first five customers have changed. The first customer is still in cluster 0, but the second customer is in cluster 1, the third in cluster 8, and the fourth and fifth in cluster 4.

- e) Select the next code cell, then type the following:

```
1 cluster_bar(y_kmeans)
```

- f) Run the code cell.
g) Examine the output.



Cluster 0 seems to have the most representation among the customers, whereas cluster 7 has the least. As before, the spread of each cluster seems to decrease in a relatively stable way.

3. Load the PCA-reduced demographics data.

- a) Scroll down and view the cell titled **Load the PCA-reduced demographics data**, then select the code cell below it.
b) In the code cell, type the following:

```
1 pca_df = pd.read_pickle('data/users_data_demo_pca.pickle')
2
3 pca_df = pd.concat([pca_df, pd.DataFrame(y_kmeans)], axis = 1). \
4 rename(columns = {0: 'cluster'})
5
6 pca_df
```

This code loads the demographics data you reduced to two features using principal component analysis (PCA). The reduced data will make it easier to visualize the clusters on a scatter plot.

- c) Run the code cell.
- d) Examine the output.

	PCA1	PCA2	cluster
0	2.557545	1.079613	0
1	-0.820505	-1.750581	1
2	-0.576607	-0.551404	8
3	-0.541647	1.593626	4
4	0.385598	-0.614768	4
...
45174	1.160485	0.073978	1
45175	-0.660738	4.371075	6
45176	-1.327676	2.367745	6
45177	-1.575612	1.040206	4
45178	-0.632467	-0.134990	8
45179 rows × 3 columns			

As before, the reduced demographics data is represented as two numeric columns. The cluster that each user belongs to is added as a third column.

4. Visualize the clusters on the reduced data.

- a) Scroll down and view the cell titled **Visualize the clusters on the reduced data**, then select the code cell below it.
- b) In the code cell, type the following:

```

1 cmap = sns.color_palette('tab10', n_colors = n_clusters, desat = .5)
2
3 sns.scatterplot(x = 'PCA1', y = 'PCA2',
4                  hue = 'cluster', data = pca_df[:25],
5                  palette = cmap, legend = True)
6
7 plt.title('K-means Clustering with 2 Dimensions');

```

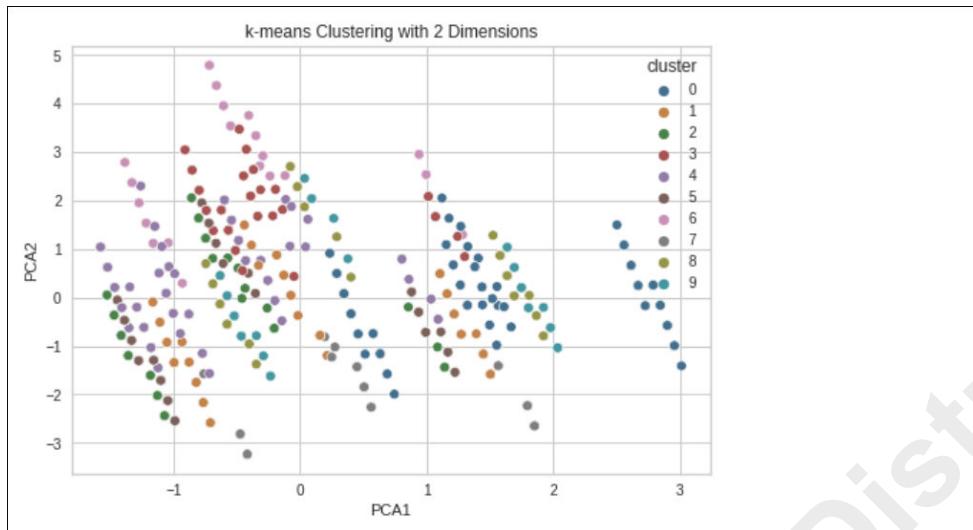
This code will create a scatter plot of the features while also showing the clusters as different colors.



Note: Only every 25 users will be plotted so that it's easier to read.

- c) Run the code cell.

d) Examine the output.



Scatterplots like this can help you visualize how clusters appear within the feature space. For example:

- Cluster 7, represented by grey dots, appears to include negative values of the PCA2 feature, but a wide range of PCA1 values.
- Cluster 0, represented by dark blue dots, seems to be the only cluster at the highest end of PCA1 values (from 2.5 to 3), though it also appears at lower (but still positive) values.
- Cluster 6, represented by pink dots, appears to have all positive values for PCA2, and also has the highest PCA2 values.

5. Keep this notebook open.

TOPIC B

Evaluate Clustering Models

Just as tuning an unsupervised model is different than tuning a supervised model, there are also differences in how you evaluate those models. You'll explore those differences in the following topic.

Evaluation Metrics for Clustering

Thus far you've used metrics to evaluate supervised learning models that perform classification and regression. Evaluation metrics used for unsupervised learning, like clustering tasks, don't have the benefit of measuring a model's performance on validation and test sets, due to the absence of a label. The data that is being clustered is used to evaluate that clustering, rather than some external, unseen dataset. Therefore, such evaluation metrics must measure the internal characteristics of a model, particularly the number of the clusters. This guides how existing data points are clustered, and will have an effect on how new data is clustered when it is introduced to the model. In other words, you can use these metrics to help you tune the model so that it best fulfills your goals, but you can't assess the objective validity of that model.

This also underscores why domain knowledge is so helpful in clustering tasks, and should be leveraged whenever possible. With adequate domain knowledge, you can more effectively use clustering metrics to help shape a clustering model to achieve the desired results, rather than working from a blank slate.

As you might expect, there is no one metric that is best for every situation. You can use multiple metrics in tandem to get multiple perspectives, if desired.

Elbow Point

One method of determining k (number of clusters) is to calculate the mean distance between each data example and its associated centroid. As k increases, the mean distance necessarily decreases. However, at some point, increasing k becomes pointless and doesn't reduce the mean distance in any significant way. The point at which the mean distance no longer decreases in a significant way is called the **elbow point**, and it's usually a good indicator of what k should be. Consider the following figure, in which k is plotted against the mean distance.

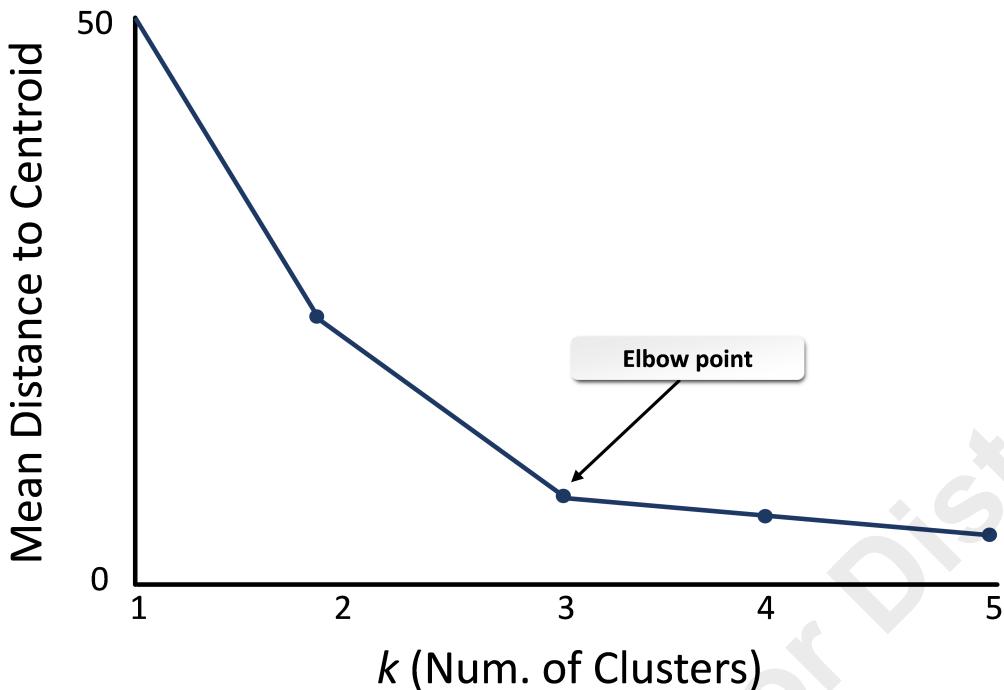


Figure 7-7: The elbow point indicates that k should be 3.

Cluster Sum of Squares

There are two ideal properties of a clustering model: the compactness of the cluster, and how well separated it is from other clusters. The better these properties are for a given problem, the better the outcome. To evaluate these properties, you can plug your data values into two separate equations.

The [within-cluster sum of squares \(WCSS\)](#) measures the compactness of clusters. It does this by measuring the distance between a given data point and its cluster centroid, then repeating this process for all other points in the cluster and calculating the mean distance. Smaller values of WCSS indicate that data points within a cluster are closer together, meaning that smaller values are ideal.

The [between-cluster sum of squares \(BCSS\)](#) measures the separation between clusters. It does this by measuring the distance between a given centroid and all other centroids, then repeating this process for all other centroids and calculating the sum. Larger BCSS values indicate clusters that are farther apart, meaning that larger values are ideal.

Silhouette Analysis

A [silhouette analysis](#) helps you evaluate the ideals of compactness and good separation in one mathematical equation. This equation calculates how well a particular data example fits within a cluster as compared to its neighboring clusters. Each example is assigned a value, called a silhouette coefficient or silhouette score, between -1 and 1 . This value indicates the following:

- A high coefficient means that the example is far away from neighboring clusters, and therefore fits well within its own cluster.
- A coefficient close to 0 means the example is near the decision boundary between clusters.
- A coefficient in the negative means the example is closer to a neighboring cluster than its own, and is therefore likely to have been placed in the wrong cluster.

The ideal is to have your data examples as close to 1 as possible. Low coefficients indicate that you may need to adjust your k value. Typically, you'd calculate the coefficient for each data example and then group each into its respective cluster. You'd also calculate the average coefficient of each cluster, as well as the average of the entire model given k . Then, you'd do the same calculations for different k values and compare the average coefficients of the different k values. Ultimately, you'd select the k value that leads to the highest silhouette coefficient.

The values derived from a silhouette analysis are typically plotted on a graph. For each cluster, the data example with the highest coefficient is on top, with the lowest coefficient at the bottom. This forms a silhouette-like shape for each cluster. The cluster with the highest average coefficient is often placed on top, and the rest of the clusters are placed in descending order.

In the following figure, the silhouette analyses of three different k values are plotted. For $k = 2$, the average silhouette coefficient is around .578; for $k = 3$, the average coefficient is around .732. For $k = 4$, the average silhouette coefficient is around .492. Since the average coefficient of $k = 3$ is highest, this suggests a better fit. Also note that the thickness of each plotted group in $k = 3$ is more evenly distributed than $k = 2$, as the model is no longer fitting many examples into a large group. This even distribution does not always lead to a better coefficient, however.

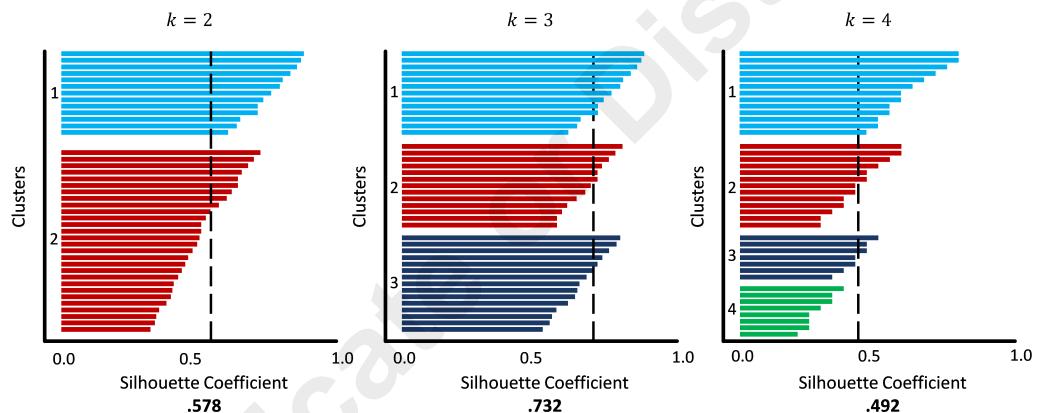


Figure 7–8: A silhouette analysis of three different k values. The dashed line indicates the average coefficient for that model.

When to Stop Hierarchical Clustering

Without any stopping criteria, hierarchical agglomerative clustering (HAC) will eventually merge all data examples into a single cluster. Likewise, hierarchical divisive clustering (HDC) will keep splitting clusters until each data point is by itself. Neither is ideal. So, you need to choose a point at which to stop the hierarchy from continuing in either case. As with other clustering methods, it helps to have domain knowledge of the problem you're trying to solve, as this might dictate how many clusters you need.

You won't always be so lucky, however. In cases where you don't have this level of domain knowledge, you can leverage much of the same techniques to determine when to stop. For example, you can use silhouette analysis to try to push the silhouette coefficient as close to 1 as possible while trying different numbers of clusters. This can help ensure that data examples fit well within their clusters and are ideally far away from neighboring clusters.

Dendrogram

One type of analysis unique to hierarchical clustering is the use of a dendrogram. A **dendrogram** is a diagram that represents a tree-like hierarchy. In the context of hierarchical clustering, a dendrogram visually demonstrates which particular data points and clusters are either being merged (HAC) or split (HDC). The following figure shows an HAC dendrogram.

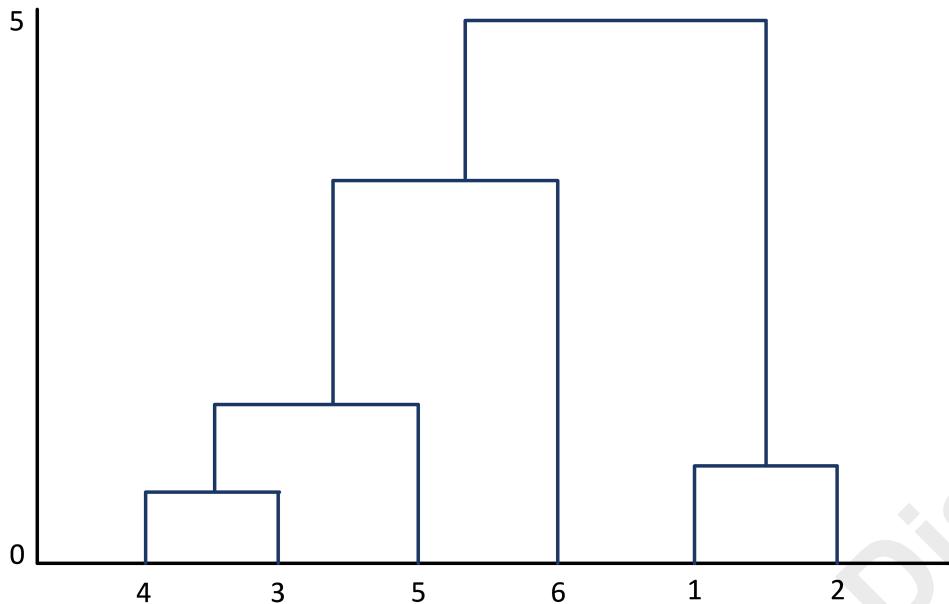


Figure 7–9: A dendrogram of hierarchical clustering data.

You start on the bottom (the x-axis) where the data examples are. As you go up the y-axis, each data example is placed into a cluster, and those clusters begin merging with other clusters. The y-axis is a measurement of how distant the data examples and clusters are from one another.

You can use a dendrogram to visually analyze how specific data points are being clustered. Typically you'd plot just a few data examples at a time to determine how they are being merged. If you were to plot all data examples at once, the dendrogram would become very difficult to read in all but the smallest datasets.



Note: One advantage that dendrograms have over other analysis methods is that you don't need to re-run the training each time you want to experiment with a different number of clusters.

Guidelines for Evaluating Clustering Models

Follow these guidelines when evaluating clustering models.

Evaluate Clustering Models

When evaluating clustering models:

- For k -means clustering and latent class analysis (LCA):
 - When domain knowledge isn't enough, use statistical analysis methods to help determine k .
 - Use an elbow point graph to identify the value of k when the mean distance between a point and its centroid is no longer decreasing significantly.
 - Use within- and between-cluster sum of squares (WCSS and BCSS) to evaluate the compactness of a cluster and how well separated it is from other clusters, respectively.
 - Use a silhouette analysis to evaluate both compactness and separation at once.
 - In silhouette analysis, consider choosing the k with the highest silhouette coefficient.
- For hierarchical clustering:
 - Use some of the same techniques as in k -means clustering to analyze clusters and determine the optimal number.
 - Consider applying domain knowledge in determining the number of clusters to stop at.

- Create a dendrogram to analyze how data points are being clustered.
- Overall:
 - Consider that there is no objectively "correct" number of clusters for many unsupervised problems.
 - Consider that different analysis methods may give you different results for an optimal number of clusters, and that one is not necessarily more "correct" than another.

Do Not Duplicate or Distribute

ACTIVITY 7-4

Evaluating Clustering Models

Before You Begin

Developing Clustering Models.ipynb is open in Jupyter Notebook.



Note: If you have shut down Jupyter Notebook since you completed the previous activity, then you need to restart Jupyter Notebook and reopen the file. To ensure all Python objects and output are in the correct state to begin this activity, select **Kernel**→**Restart & Clear Output**, and select **Restart and Clear All Outputs**. Scroll down and select the cell labeled **Use the elbow method to determine the optimal number of clusters**. Select **Cell**→**Run All Above**.

Scenario

As with supervised learning, you'll need some way to evaluate your clustering models. This is especially important for choosing the optimal number of clusters, as any change to this number will have a significant impact on how GCNB's customers are segmented.

1. Use the elbow method to determine the optimal number of clusters.

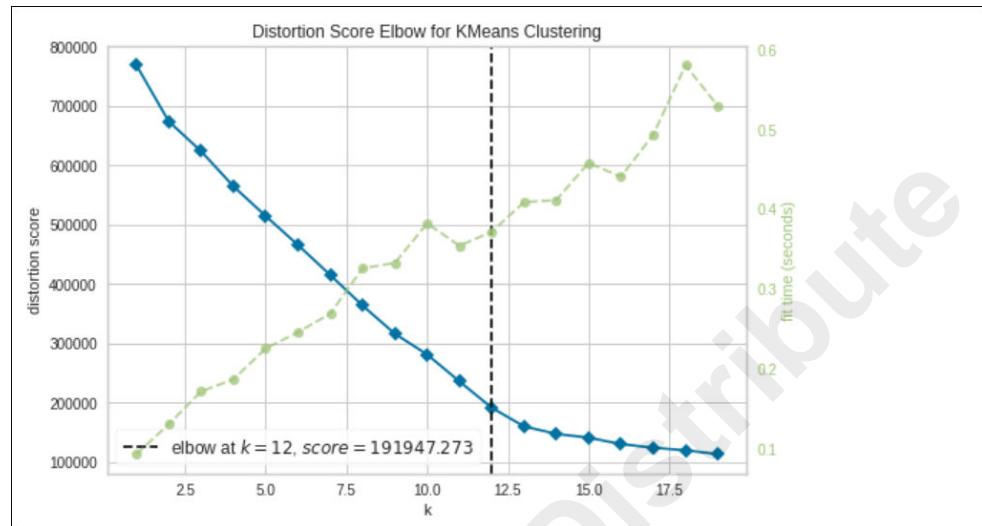
- a) Scroll down and view the cell titled **Use the elbow method to determine the optimal number of clusters**, then select the code cell below it.
- b) In the code cell, type the following:

```
1 elbow = KElbowVisualizer(kmeans, k = (1, 20))
2 elbow.fit(users_data_scaled)
3 elbow.poof();
```

This code uses the Yellowbrick library to automatically generate an elbow point graph of the (non-PCA) k -means model you trained earlier. In this case, the model will be trained on a range of clusters from 1 to 20.

- c) Run the code cell.

- d) Examine the output.



The elbow point indicates the point of diminishing returns at around a k of 12. This does not objectively mean that 12 is the optimal number of clusters, only that a model with 12 clusters is worth exploring.

2. Use silhouette analysis to determine the optimal number of clusters.

- Scroll down and view the cell titled **Use silhouette analysis to determine the optimal number of clusters**, then select the code cell below it.
- In the code cell, type the following:

```
1 silhouette = SilhouetteVisualizer(KMeans(12, random_state = 10))
2 silhouette.fit(users_data_scaled)
3 silhouette.poof();
```

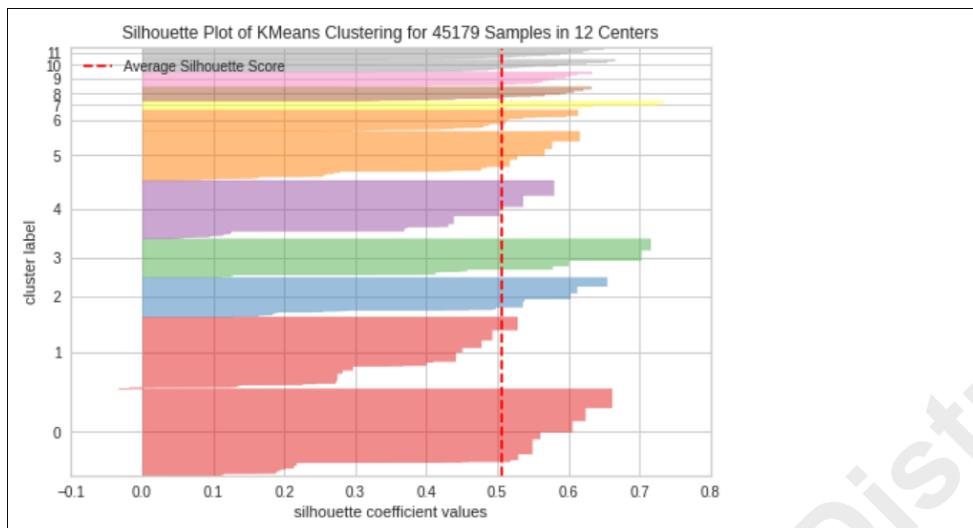
An alternative to the elbow point method is the silhouette score, which this code will visualize. Based on the elbow point's conclusions, you'll generate the silhouettes with a k of 12 first.

- Run the code cell.



Note: It will take a few minutes for the silhouette analysis to finish.

- d) Examine the output.



Each of the 12 clusters is plotted as its own silhouette, and the average silhouette score for the clusters is drawn as a vertical red line.



Note: You can see that at least one silhouette score for a data example in cluster 1 is negative (i.e., its silhouette line is plotted to the left of 0.0). The data example is therefore closer to a neighboring cluster than its own, and was probably placed in the wrong cluster.

- e) Select the next code cell, then type the following:

```
1 print('Number of clusters: ', silhouette.n_clusters_)
2 print('Silhouette score: ', silhouette.silhouette_score_)
```

- f) Run the code cell.
g) Examine the output.

```
Number of clusters: 12
Silhouette score: 0.5051362488084534
```

The silhouette score for 12 clusters is ~0.51. Scores closer to 1 indicate a greater deal of cluster compactness and separation, which are the two main goals of effective clustering. However, a silhouette score for one number of clusters is most useful when it's compared to a different number of clusters.

3. Try a different number of clusters and compare the silhouette scores.

- a) Scroll down and view the cell titled **Try a different number of clusters and compare the silhouette scores**, then select the code cell below it.
b) In the code cell, type the following:

```
1 silhouette = SilhouetteVisualizer(KMeans(15, random_state = 10))
2 silhouette.fit(users_data_scaled)
3 silhouette.poof();
```

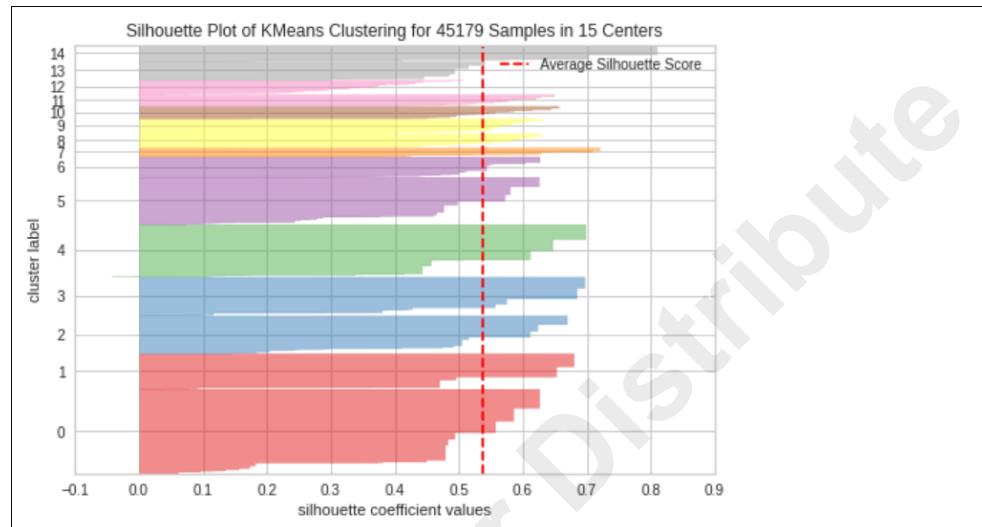
This time, the silhouettes will be calculated for 15 clusters using k -means.

- c) Run the code cell.



Note: It will take a few minutes for the silhouette analysis to finish.

- d) Examine the output.



The silhouette score appears to have increased somewhat, but you'll retrieve the exact score to make sure.

- e) Select the next code cell, then type the following:

```
1 print('Number of clusters: ', silhouette.n_clusters_)
2 print('Silhouette score: ', silhouette.silhouette_score_)
```

- f) Run the code cell.
g) Examine the output.

```
Number of clusters: 15
Silhouette score: 0.5370783078641602
```

As expected, the silhouette score did increase somewhat. This suggests that 15 clusters may be better than 12. If you had time, you'd probably want to generate silhouette scores for many more values of k , as there could be some that score even better.

However, the score didn't improve that drastically from the model with 12 clusters. Also, you need to consider how the number of clusters will influence the way in which you apply those customers to the real world. There might be an upper limit to the number of clusters that's imposed by the problem domain. For example, will it be feasible for the marketing department to create and track 15 different market segments? For now, you'll just consider 12 to be the "optimal" number of clusters since it also aligns with what the elbow point concluded.

4. Retrain the k -means model on the optimal number of clusters.

- a) Scroll down and view the cell titled **Retrain the k -means model on the optimal number of clusters**, then select the code cell below it.

- b) In the code cell, type the following:

```
1 n_clusters = 12
2
3 kmeans = KMeans(n_clusters = n_clusters, random_state = 10)
4
5 kmeans.fit(users_data_scaled)
```

The model is being retrained to generate 12 clusters.

- c) Run the code cell.

5. Generate the clusters and the number of users in each cluster.

- a) Scroll down and view the cell titled **Generate the clusters and the number of users in each cluster**, then select the code cell below it.
b) In the code cell, type the following:

```
1 y_kmeans = kmeans.predict(users_data_scaled)
2
3 results['cluster'] = y_kmeans
4 results.head()
```

- c) Run the code cell.
d) Examine the output.

	cluster	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_housemaid	job_farmhand
0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
2	8	0	0	1	0	0	0	0	0	0	0	0
3	4	0	0	0	1	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0	0	0	0

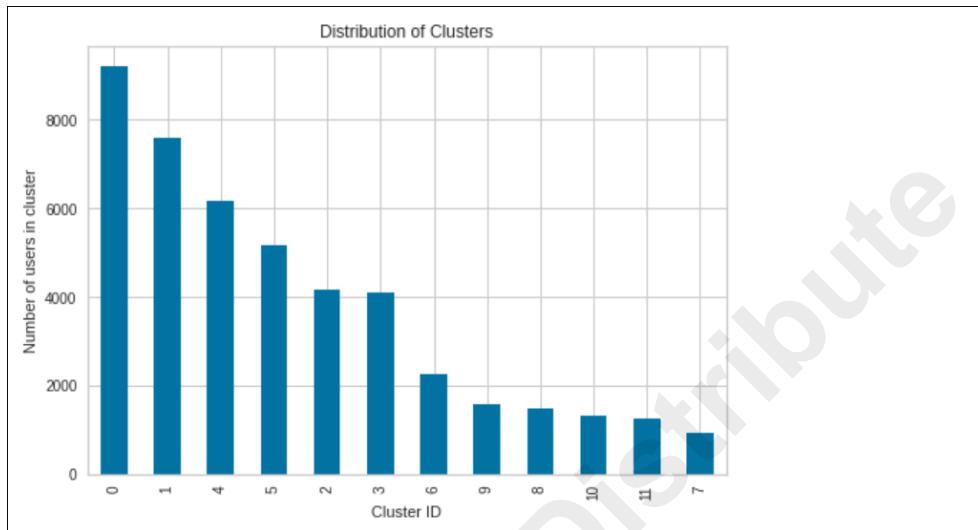
The clusters for the first five customers are the same as they were when you trained the *k*-means model on 10 clusters.

- e) Select the next code cell, then type the following:

```
1 cluster_bar(y_kmeans)
```

- f) Run the code cell.

- g) Examine the output.



Cluster 0 still has the most users, and cluster 7 still has the fewest. Once again the decrease in customer count per cluster seems to be gradual.

6. Compare two different clusters.

- Scroll down and view the cell titled **Compare two different clusters**, then select the code cell below it.
- In the code cell, type the following:

```
1 cluster_6 = results[results.cluster == 6]
2 cluster_6.describe()
```

If you're going to apply these clusters to a practical situation, you need to learn more about them by understanding how the clusters differ. This code will show summary statistics for cluster 6.

- Run the code cell.

d) Examine the output.

cluster	job_retired	job_admin.	job_services	job_self-employed	job_unemployed	job_housemaid	job_student	education_tertiary	education_secondary	education_Unknown
.0	2262.0	2262.0	2262.0	2262.0	2262.0	2262.0	2262.0	2262.000000	2262.000000	2262.000000
.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.161362	0.434571	0.052605
.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.367946	0.495810	0.223300
.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000
.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000
.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000
.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	1.000000	0.000000
.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.000000	1.000000	1.000000

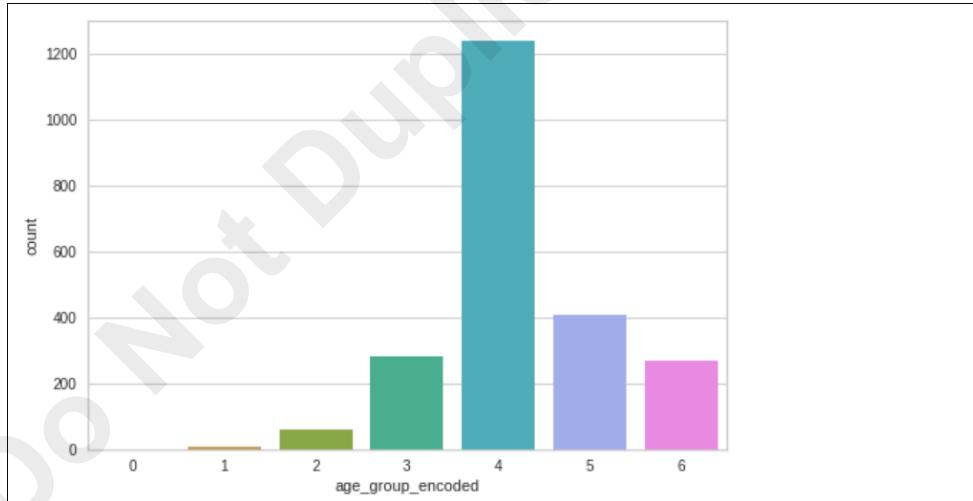
Looking at these statistics, here are a few interesting conclusions you could draw:

- Every member of this cluster appears to be retired. The `job_retired` feature, like most of the other features, is a one-hot encoded binary value. Its minimum is 1.0, which means that every customer in this cluster has a 1 ("yes") for this value.
- Members of this cluster are most likely to have a secondary level of education. The `education_secondary` feature has the highest mean value of the education features by far.
- Members of this cluster are very likely to have been married at some point (or still are). The mean `single` value is very close to 0; since this is a binary feature, a value of 0 means that the user is *not* single.
- Members of this cluster appear to be relatively older. The `age_group_encoded` feature shows a mean value of ~4.22 out of a total of 6. Recall that bin 4 is the age group 55–64. You'll explore these age values further using a chart.

e) Select the next code cell, then type the following:

```
1 | sns.countplot(cluster_6['age_group_encoded']);
```

- f) Run the code cell.
g) Examine the output.



Age bin 4 (55–64) has the most representation in this cluster by far, with bins 3 (45–54), 5 (65–74), and 6 (75+) each including about a third of that. Bins 0 (18–24), 1 (25–34), and 2 (35–44) have very little representation.

- h) Select the next code cell, then type the following:

```
1 cluster_7 = results[results.cluster == 7]
2 cluster_7.describe()
```

This code will print the same summary statistics, but for cluster 7.

- i) Run the code cell.
- j) Examine the output.

b_unemployed	job_housemaid	job_student	education_tertiary	education_secondary	education_Unknown	education_primary	single	age_group_encoded
937.0	937.0	937.0	937.000000	937.000000	937.000000	937.000000	937.000000	937.000000
0.0	0.0	1.0	0.237994	0.542156	0.172892	0.046958	0.935966	0.721451
0.0	0.0	0.0	0.426083	0.498486	0.378356	0.211663	0.244945	0.592239
0.0	0.0	1.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.0	0.0	1.0	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000
0.0	0.0	1.0	0.000000	1.000000	0.000000	0.000000	1.000000	1.000000
0.0	0.0	1.0	0.000000	1.000000	0.000000	0.000000	1.000000	1.000000
0.0	0.0	1.0	1.000000	1.000000	1.000000	1.000000	1.000000	3.000000

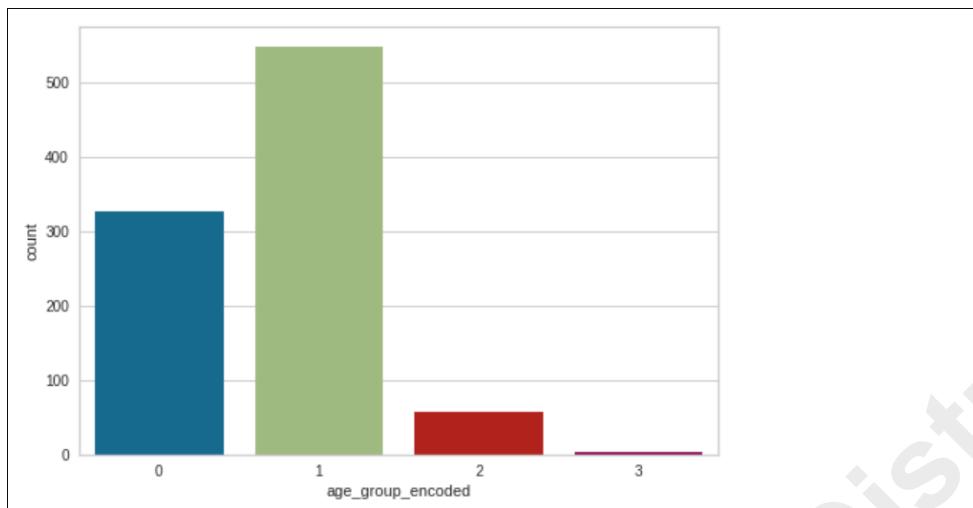
Some conclusions you can draw from this include:

- Every member of this cluster is a student, for the same reason that every customer in cluster 6 is retired.
 - Secondary education also seems to be the most common level of education among customers in this cluster.
 - Most of the users in this cluster are single. The mean `single` value is very close to 1, meaning a "yes" for single.
 - The average age of customers in this cluster is much lower than cluster 6.
- k) Select the next code cell, then type the following:

```
1 sns.countplot(cluster_7['age_group_encoded']);
```

- l) Run the code cell.

- m) Examine the output.



Bin 1 has the most customers in this cluster, whereas bin 0 has a decent amount of representation. Bins 2 and 3 have little representation, and no one 55 or above is included in this cluster at all.

So, in comparing these two clusters, you might conclude that cluster 6 represents a segment of the market that can be aimed at older retirees with a secondary education who have been or still are married, whereas cluster 7 represents a segment that can be aimed at young students with a secondary education who have never been married.

7. Save the best model.

- Scroll down and view the cell titled **Save the best model**, then select the code cell below it.
- In the code cell, type the following:

```
1 | pickle.dump(kmeans, open('kmeans.pickle', 'wb'))
```

You'll save the *k*-means model with 12 clusters as the optimal model.

- Run the code cell.

8. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close the **Developing Clustering Models** tab in Firefox, but keep a tab open to **CDSP/Clustering/** in the file hierarchy.

Summary

In this lesson, you leveraged unlabeled data by training clustering models. While unsupervised learning does lead to some key differences in how you approach model development, you were still able to improve the models through tuning and evaluation. Like other forms of machine learning, clustering is a powerful way of extracting insights from the data and translating those insights into business actions.

What type of data in your organization do you think might be conducive to clustering analysis?

What types of clustering evaluation metrics do you think you'll rely on most to determine the optimal number of clusters? Why?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

8

Finalizing a Data Science Project

Lesson Time: 2 hours, 45 minutes

Lesson Introduction

You've put your data through the extract, transform, and load (ETL) process, conducted an analysis of the data, and developed statistical models from the data that cover the three major disciplines of machine learning: classification, regression, and clustering. But you're not done yet. Now it's time to gather your results and present them to stakeholders. After all, you undertook the data science project to achieve business goals, so you need to demonstrate that you were actually successful in doing so.

Lesson Objectives

In this lesson, you will:

- Communicate the results of a model to stakeholders so that these results can be translated into business actions.
- Build a basic web app to demonstrate machine learning models to stakeholders.
- Implement and test pipelines that automate the model training, tuning, and deployment processes.

TOPIC A

Communicate Results to Stakeholders

An important step in finalizing a data science project is being able to report to stakeholders. In this topic, you'll go over some of the key factors that go into these reports.

Know Your Audience

Unless you were creating a model just to answer your own questions, once you have trained and tuned a good model, you'll have an audience you need to report back to. Your audience may be a single person, a group of stakeholders, or perhaps an entire organization. They may work for the same company you work for, or they may be external clients.

In many cases, the audience for your report may consist of many different people, with different knowledge, needs, and expectations. To produce a good report, then, you'll need to know your audience so you can identify what information you must provide, and how you must provide it to communicate it effectively.

Derive Insights from Findings

The results produced by the data science process can certainly be interesting to a practitioner; perhaps you were able to squeeze an extra bit of performance out of a model, or maybe you were able to consolidate data so that it was easy for you to work with. But these things won't be valuable to a wider audience on their own. You need to be able to translate these results into business insights in order to prove the value of the data science project.

It helps to begin by reviewing the overall data science process and the results that came from it. As you do so, you can ask yourself (and your colleagues) questions like:

- What did I know about our business issues before the project started?
- What do I know now about our business issues?
- How does the analysis I performed supplement my knowledge of our business issues?
- How do the models I built address or not address these business issues?
- How do the results align or not align with pre-defined key performance indicators (KPIs)?
- What business actions can be taken now or later based on the results of the project?
- What have I learned about the data science process that can be used to improve it in the future?

When it comes to communicating insights, you need to make sure they are both relevant and placed in the proper context. Although it may seem like your insight is inherently worthwhile, it may not mean much to certain stakeholders or when communicated at certain times. For example, customers will care less about your insight into how to increase business profits, and more about insights into how their user experience can be improved.

Insights should also be clear and precise. Saying that your image classifier is "highly accurate" doesn't really communicate how it can help the business. Instead, you could say something to the effect of, "The image classifier will determine the correct type of image 95% of the time and save 20 work hours per week as compared to the existing process of manual review."

Explainability

The actual conclusions you draw and then communicate to stakeholders will be driven by multiple factors. Not only will these factors help you identify key insights, but they will also help you explain the value of those insights to anyone who was not directly involved in the technical phases of the process.

One example of a major factor that drives data science outcomes is *explainability*, also referred to as *interpretability*. An explainable process is one whose inner workings can be identified and communicated to a relevant audience. It's usually not enough that a machine learning model produces a result—e.g., a banking system says that Customer X is a good candidate for a loan. Ideally, you'd be able to explain *why* the model made that decision. Not only does this help you prove the model's skill and value to stakeholders, but its decisions are more defensible to any people affected by them as well. Knowing the model's decision-making process can allay concerns people naturally have for automation.

Unfortunately, some algorithms and their resulting models are what are known as "black boxes," or systems that make it difficult or even impossible to explain its decisions. This is particularly true of models built on neural network architectures. However, plenty of models are still explainable to at least some degree, including most of the models you built in this course. There are several ways to explain them, and several tools for the job, but two common methods are global interpretability and local interpretability.

Global vs. Local Interpretability

Global interpretability measures the model's decision-making processes as a whole. **Local interpretability** measures the decision-making processes for specific data examples. Both of these are often used to determine variable (i.e., feature) importance. Global interpretability can be used to determine how much each feature contributed to a model overall, whereas local interpretability can be used to determine how much each feature contributed to the estimation of a specific data example. Your choice to reduce or outright remove variables you deem unimportant or actively harmful to a model will need to be justified. Even if you don't take such a direct approach, the relative importance of each feature will almost always have a significant effect on the resulting model and the decisions it makes.

In the following figure, on the left, a tool called Shapley Additive Explanations (SHAP) measures variable importance from a global perspective. On the right, a tool called ELI5 measures variable importance for a single data example that is run through a classifier—an example of local interpretability.

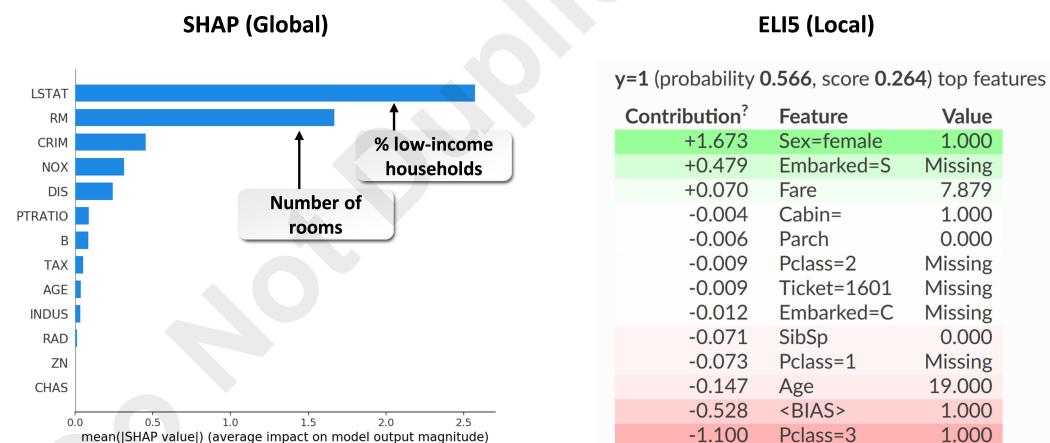


Figure 8-1: Global (left) and local (right) interpretability demonstrated using variable importance plots.

The model represented by the plot on the left estimates the value of houses based on several features. According to this plot, you can conclude that LSTAT, or the percentage of low-income households in the area, has the greatest impact on the model's estimations, with number of rooms (RM) a distant second. The model represented by the plot on the right classifies passengers of the ill-fated RMS *Titanic* as having survived the disaster or perished. So, the model concludes that this

particular passenger will have survived ($y = 1$). The feature that was most important in this specific decision is the fact that the passenger was female. The passenger not embarking from Southampton (Embarked=S is missing) was also somewhat important to the model's decision.

Additional Factors That Drive Outcomes

In addition to explainability/interpretability, other key factors that can influence data science outcomes include:

- **Accountability.** Assigning responsibility for the actions taken by a model or human personnel can change how outcomes are received. Certain outcomes may be accepted or rejected based on whether or not they uphold accountability principles.
- **Safety and security.** Outcomes can bring risks to user privacy, organizational security, and personnel safety. This may prompt the organization to implement risk management processes to determine if the outcomes are worth preserving, and how to address their risks.
- **Fairness and non-discrimination.** Unintentional bias can appear in the results of any data science project, particularly those that deal with human-centered data. Outcomes may be rejected or require change if they exhibit discriminatory or unfair behavior.
- **Time and resources.** Ideal outcomes are sometimes achievable, but only if enough time and resources are dedicated to achieving them. The quality of outcomes may need to be sacrificed due to scheduling and budgetary constraints.

There may be other factors at play as well, so you should take time to review the unique needs and circumstances of your own data science project. By identifying factors that drive outcomes, you'll be better prepared to communicate those outcomes to stakeholders.

Present Model Results

A typical business audience will need to know the business issue you set out to address, and your proposed solution. At its core, a data science solution will involve data and a model that operates on that data, so you'll need to explain how you leveraged those components, sticking to information that is relevant for the audience, and providing the kinds of details they need to know. Primarily, they will need to understand how the results of your project impact the business. The following components, then, should make up the essential parts of your presentation.

- **An executive summary or abstract of the project.** A report on a data science project may contain many details and complex findings. To help your audience prepare to understand what you will present in detail, start with a brief summary of the high points—what you set out to do, and what you found as the answer or solution.
- **The business issue you set out to address.** Set the stage by describing the problem and identifying how a solution might be measured in terms of business metrics, such as a percentage of increased sales, reduced maintenance costs, increased production rates, and so forth.
- **The data.** Describe, in terms your audience can relate to, the data that was used, where it came from, and any relevant explanations of how you had to modify the data to make it useful. If appropriate, identify additional data it would be helpful to collect in the future to support ongoing analysis.
- **The hypothesis.** What you initially surmised from analyzing the available data, and how your solution evolved, sticking with details that would be relevant to your audience. Avoid diving into technical topics like the algorithms, programming tools, and statistical techniques you used unless they are relevant to your audience, and you can present them in a way your audience will understand.
- **The solution.** Identify what the final model revealed, clearly identifying its business implications. Use examples and visualizations to help your audience understand. Be transparent, describing any assumptions you made, identifying where you might make improvements, and any risks the solution might introduce, including risks and possible ethical challenges posed to the company, staff, users, and customers.

Use Visuals in Presentations

Early in the project, you may have used visualizations to explore the dataset and come up with a hypothesis for the model. You may also have used visualizations as you trained and tested the model, to evaluate its performance and verify that it is working as you intended. Visualizations provide a powerful means of conveying patterns in numbers. Just as they are helpful to you during the development of a model, they can be very helpful to assist your audience in understanding the results produced by your machine learning model.

Some chart types are more conducive to a non-technical audience than others. Bar charts and line graphs are easy to understand, whereas heatmaps and pair plots can be difficult for a layperson to interpret. The information on a chart can also make it more difficult to interpret, even when you're using a simpler chart type. A grid of multiple bar charts that presents four or more dimensions of data can quickly become overwhelming to your audience. You want to keep your visualizations simple enough to be understood, but still capable of communicating some important idea or telling a worthwhile story. So, consider how placing each visualization within an overall context can make it easier to understand.

When communicating your results in a visual report or presentation, you should also:

- Label axes and chart titles with meaningful names.
- Label ticks as needed to identify general values, but don't crowd the chart with too many ticks.
- Use legends as needed to identify what values are shown in a chart.
- Not rely solely on color coding, but when you must use it, use color palettes that can be detected by people with color blindness.
- Avoid adding unnecessary details or decoration that hinder readability, such as backgrounds or themes.

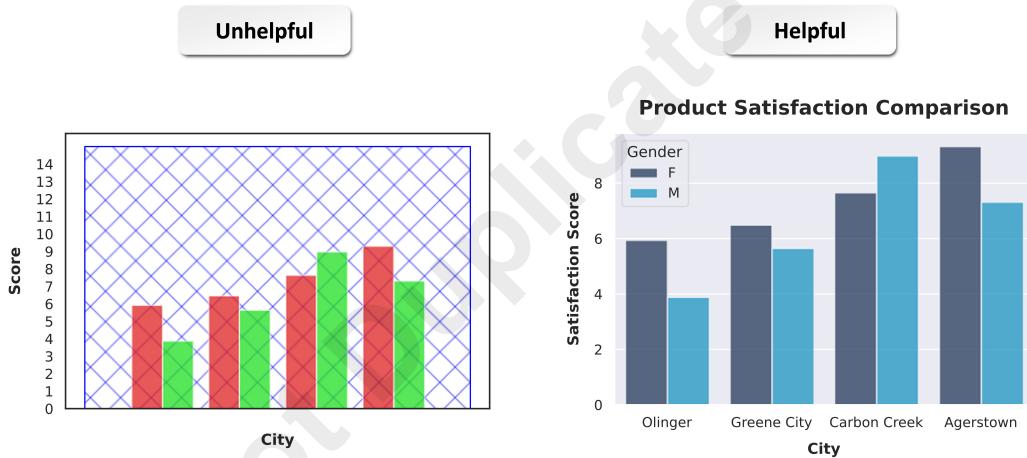


Figure 8-2: An unhelpful chart (left) vs. a more helpful one (right).

Dashboards

A **dashboard** is a high-level visualization tool that summarizes the status of an entire project or one or more parts of a project. They are typically used by project managers and other high-level stakeholders to view the progress of a project at a quick glance. Practitioners can also use them to get a quick overview of the performance of a system, and any errors that have occurred.

Dashboards give the viewer the most pertinent information up front, without having to wade through unnecessary details. This enables the viewer to quickly identify and raise any concerns they might have, and/or ask the right questions. For high-level stakeholders, this means providing them with the big picture of how the project is or is not meeting business goals. Dashboards can be viewed at an end-of-project presentation as a way for the practitioner to convey the most important

details to their audience, but it's more common for dashboards to be viewed by stakeholders on a rolling basis, throughout the project's lifecycle.

The structure of a dashboard will change based on numerous factors, including the scope and business goals of the project. However, most dashboards are designed to track multiple KPIs associated with the project. In a data science project, such KPIs might include:

- The size of the dataset.
- How long a model has been training on a dataset.
- The price of computation, if hosted on a cloud provider.
- Metrics for model performance like accuracy and cost functions.
- The complexity of model parameters.

KPIs like these and many more are often visualized using charts, graphs, data tables, and so on. The visuals are arranged on a single screen so there's no need to flip pages to look for something. The visual elements are also typically arranged in such a way that similar elements are grouped together —this makes it easier for the viewer to get what they need.

The following figure shows a simple dashboard for an image classification model. Tables on the left give an overview of some KPIs, including dataset size, training time (in the form of epochs, or rounds of training), and cost generated by cloud services. There's also a table that keeps track of key classification metrics like accuracy, recall, and precision. On the right, the dashboard shows a line graph that visualizes how accuracy is changing as more and more training epochs are run. So someone viewing this dashboard can see that accuracy appears to be increasing, though it does so less dramatically toward the fifth epoch.

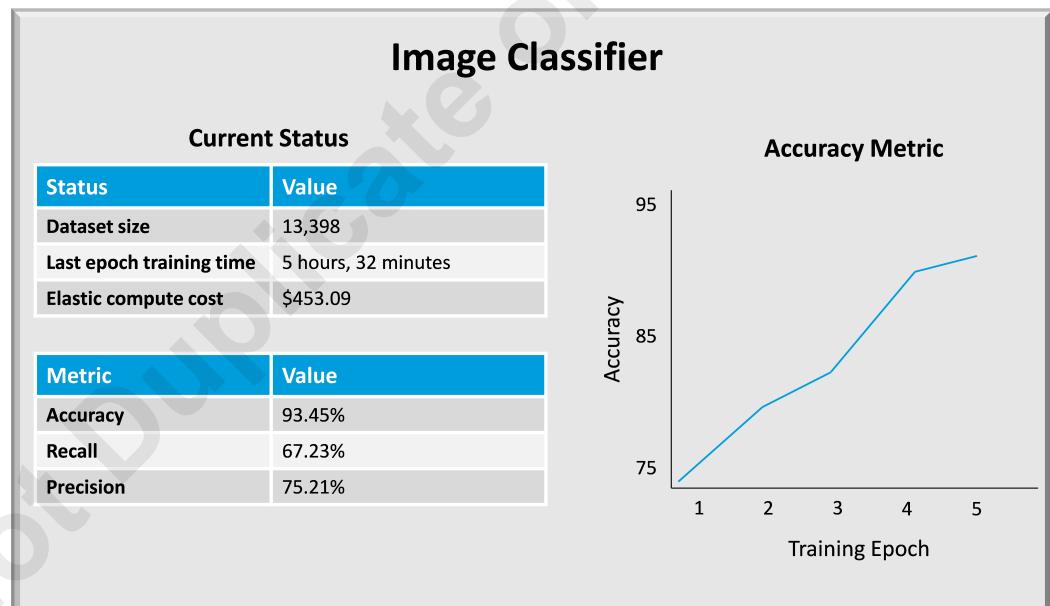


Figure 8-3: An example dashboard.

Cumulative Gains Charts

A *cumulative gains chart* is a method of plotting the percentage of some target number of examples in a given class against a percentage of the total number of examples. Consider an example of a customer retention model that classifies a customer as likely to return ("Yes") or not likely to return ("No"). You want to prioritize your marketing efforts so that customers more likely to make a repeat purchase are given a small credit on their next purchase, whereas customers unlikely to return are not. There are 10,000 total customers in the training dataset, and of those, 3,000 are labeled as returning.

An example of a cumulative gains chart for this scenario is plotted in the following figure.

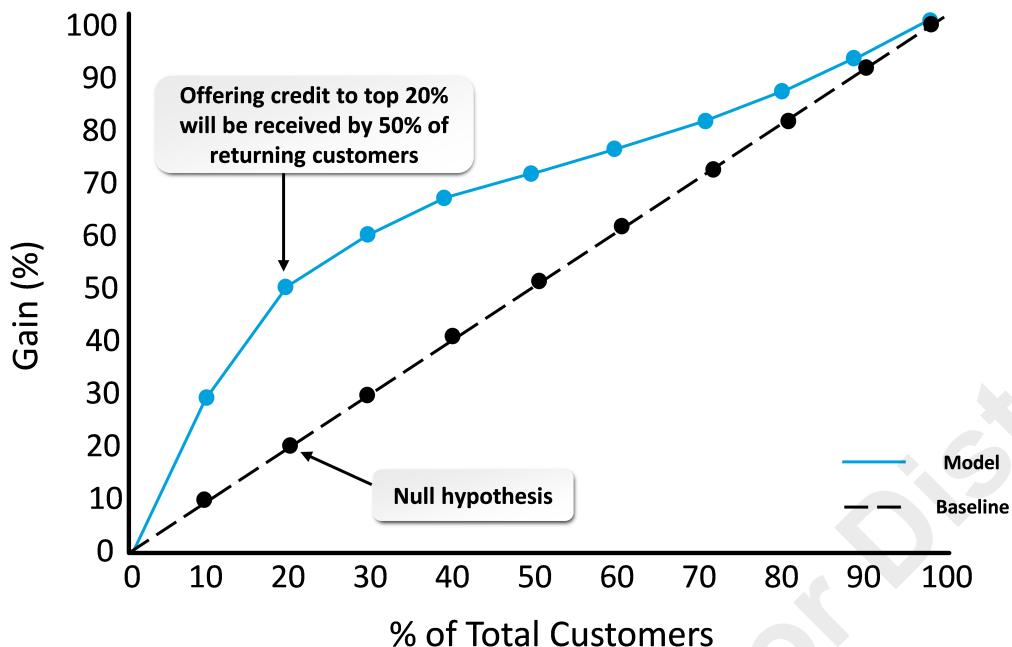


Figure 8-4: A cumulative gains chart of the customer retention model.

The y-axis shows the percentage of customers labeled "Yes" (i.e., they will return to make another purchase), also called the *gain*. So, that's a percentage of 3,000 customers. The x-axis is the percentage of customers you offer the purchase credit to out of the entire sample. In this case, that's 10,000. Consider the first data point at the bottom left, for the blue solid line: (10, 30). What this means is that, if you sorted the model's estimations by the "Yes" class, and you offered the purchase credit to the top 10% of those sorted customers, you can expect the credit to be received by 30% of the total number of returning customers. If you offered the credit to the top 20% of customers, you can expect it to be received by 50% of returning customers, and so on.

The dashed black line is called a baseline, and acts just like it does in a ROC curve: it is a random sampling—the null hypothesis, in other words. If you offered the credit to a random 10% of customers, it would be received by 10% of returning customers. So, the model—the blue solid line, in this case—must be above and to the left of this random sampling line in order for it to be effective. Since it is, you can consider the model better than a random guess.

Unlike ROC curves and other charts you've seen thus far, cumulative gains charts tend to have a more direct application to business decisions. A high-level stakeholder might decide what constitutes an acceptable level of gain for the model when compared to the cost of offering the credit. They might choose to offer the credit to only the top 40% of customers, which will be received by 65% of all returning customers. The cost of offering the credit to any more customers might outweigh its benefit, which is why cumulative gains charts are an effective tool for presenting model results to a business audience.



Note: You can also configure a cumulative gains chart to plot all class values (e.g., both "Yes" and "No").

Lift Charts

A **lift chart** is very similar to a cumulative gains chart, except that the gain on the y-axis has been replaced by a *lift* value, which is just the ratio of the model's gain to the baseline at each point. So, in the same customer retention example, the first lift value is $30 / 10 = 3.0$. Repeat this for the rest of the percentage values, and you'll get something like the following figure.

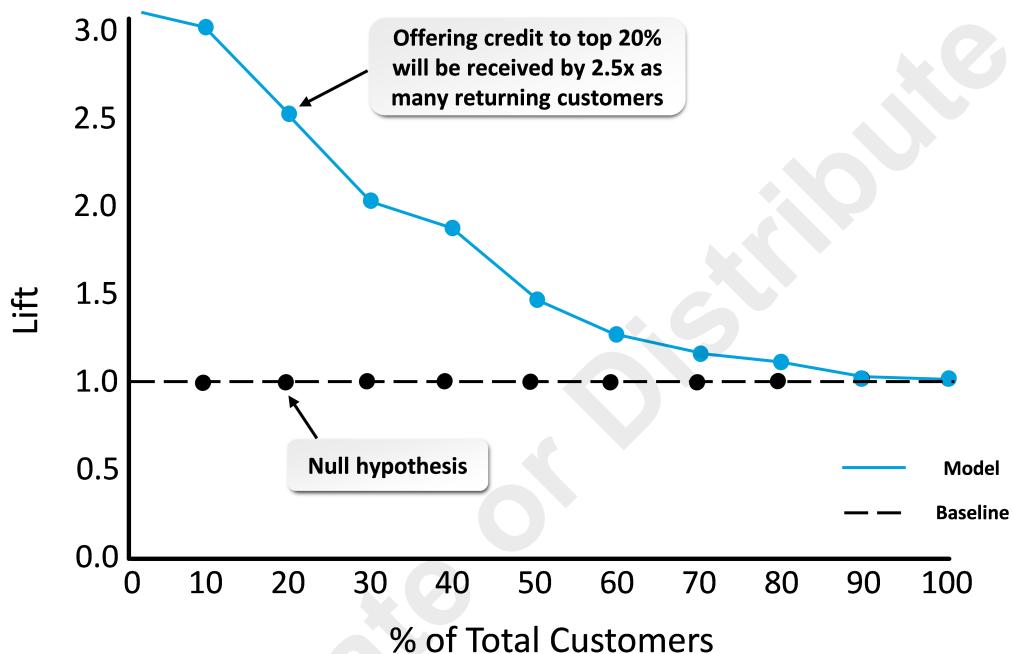


Figure 8-5: A lift chart of the customer retention model.

In this case, the baseline is just a flat horizontal line that's always at a lift value of 1. Your model must be above this line to be more effective than a random guess.

Lift charts are just another way to analyze the same basic information of a cumulative gains chart. It tends to focus more on the reach disparity, as in, how many more returning customers will receive the purchase credit if you offered some percentage of the population that credit, as compared to the baseline? In this case, if you offered the credit to the top 10% of customers identified by the model, three times as many returning customers will receive the credit than if you had just chosen them randomly. Like with a cumulative gains chart, stakeholders can use lift charts to make business decisions that incorporate cost–benefit analyses.

Guidelines for Communicating Results to Stakeholders



Note: All Guidelines for this lesson are available as checklists from the **Checklist** tile on the CHOICE Course screen.

Follow these guidelines when communicating results to stakeholders.

Communicate Results to Stakeholders

When communicating results to stakeholders:

- **Don't drown your audience in a sea of numbers.** Adapt the presentation for different audiences you must address, including only information relevant to that audience, avoiding jargon, and explaining things in a way the audience will understand.

- **Use visuals strategically.** Use charts or diagrams when they more clearly illustrate patterns in the data than words or numbers, particularly where they support important points you have to make about your findings. Experiment with various chart types and formatting options as needed until you come up with a configuration that clearly communicates the point you're trying to make with little or no explanation.
- **Use formatting, labels, and legends.** When displaying numbers, use commas and decimal points to help viewers read the numbers. If values are really close in size, use more decimal places to show differences in values. Use currency symbols for money values. Include labels, legends, and other captions where they will help the reader understand what they're looking at. Align columns of numbers on the right, or at the decimal point so they are more easily compared.
- **Avoid suspense.** A good presentation flows well and tells a story, but not in the way a mystery novel tells a story. You will likely provide your audience with a lot of information to process, and they will have questions and concerns. Help them understand and buy in to your conclusions by anticipating the types of questions you think they will have, and sharing important points early on.
- **Provide context for details.** If you must tunnel down into details showing numbers, tables, and charts, be sure to connect them back to the main idea or point you're trying to communicate.
- **Be honest and transparent.** Be clear regarding how you obtained results. Don't hide significant data or results—even if they don't fully support your hypothesis, objectives, or proposed solution.
- **Identify key assumptions you have made.** This will provide context and a rationale for the results you've produced.
- **Check your work.** Although your machine learning models may be impeccable, the quality of your findings may be called into question if your presentations contain errors.
- **Invite feedback.** Make sure your audience has a chance to ask questions so you can provide clarification and promote buy-in.
- **Provide supplementary communication channels for those with different communication styles.** Some people may be hesitant to ask questions during a presentation, and may be more comfortable asking you follow-up questions one on one or through online messaging. Consider inviting them to share feedback through such side channels.
- **Provide supplementary documentation for those with different informational needs.** Some people may require more details or technical depth than others. While it may not be appropriate to bog down your presentation slides or report with details that are only of interest to a small portion of your audience, consider directing those who require more information to view supplemental documents that you've posted on the company intranet, a network directory, or some other secure location where they can view the additional details.
- **Test it before you deliver it.** Depending on the importance of the outcome of your presentation, consider testing it before you deliver it. Pass it by reviewers who can represent your target audience and will give you honest feedback. Ask them questions to assess how well the presentation gets important points across. Do a dry run of your delivery. Make revisions as needed.

ACTIVITY 8-1

Communicating Results to Stakeholders

Scenario

You and your team at Rudison Technologies have finished developing machine learning models that address several of Greene City National Bank's concerns. You're satisfied with the results of these models and are confident that they can help GCNB improve its business processes. Now, it's time for you to present these results to the project's stakeholders. Your goal is to ensure that the stakeholders understand your conclusions and agree that those conclusions can bring value to the business.

1. Early on in the project, you identified key external stakeholders like GCNB's high-level management team and GCNB's customers.

How might you approach these audiences differently when presenting your results?

2. Recall that you built models that classified whether or not someone would sign up for a term deposit based on the bank's marketing campaign. Also recall that the overall goal of the project is to improve GCNB's marketing campaigns so that they can better retain customers, attract new ones, and encourage customers to sign up for more of the bank's services.

How do these classification models address GCNB's business goals?

3. You also built models that estimated the total balance of a customer's withdrawals and deposits.

How do these regression models address GCNB's business goals?

4. You also built models that clustered groups of customers based on their demographic data.

How do these clustering models address GCNB's business goals?

5. At this point, you've obtained several useful insights into the project that you want to share with GCNB's business leaders. Given these insights, you want to prepare a written report to this audience.

What are some key elements that you should include in this written report?

6. The written report is a good start, but it's not always the best way to illustrate your findings to a general audience.

How might you incorporate visuals into your presentation?

7. As you present your findings to GCNB's business leaders, you want to offer some suggestions.

Based on the conclusions you've drawn from the project, what business actions would you suggest that GCNB take?

Do Not Duplicate or Distribute

TOPIC B

Demonstrate Models in a Web App

One way to create a robust and interesting presentation is to show off your results in a web app. In this topic, you'll focus on some of the major technologies that go into creating a web app that you might want to use in a demonstration.

Web App

A [web app](#) is an application that runs on a server using a web-based protocol, particularly Hypertext Transfer Protocol (HTTP) and its secure equivalent, HTTPS. Web apps run in a client-server architecture, where client web browsers make requests to a web server, which receives those requests and responds. There are many uses for web apps, but in the world of data science, they are often used to demonstrate the results of a project. This can mean everything from building a proof of concept (POC) at the beginning of the project lifecycle to demonstrate a product's feasibility, or it could mean building a robust presentation to stakeholders at the project's conclusion.

In either case, a web app can go above and beyond a traditional presentation medium like a slide deck because of its interactive nature. Like any web app you've ever used, a web app that demonstrates a data science project can dynamically accept input and change as a result of that input.

For example, let's say you've built a sales forecasting model and want to show off its predictions to stakeholders. Those stakeholders might be interested in seeing the projected sales from week to week, month to month, quarter to quarter, or year to year. Or maybe they only want to see certain years, months, etc., and not others. If this were a slide show, you would probably have to create one slide for every potential configuration, then flip back and forth between slides during your presentation. Obviously, this can get tedious, and you can't always anticipate what your audience wants to see. Instead, you could build a web app that anyone in your audience can connect to through a browser. The app will have various controls like menus, buttons, and text fields that they can use to configure the visuals however they want.

Making it easy for your audience to access the exact information they're looking for is vital to any demonstration. So, as you're planning your POC or your end-of-project presentation, consider how much more effective it will be if developed as a web app. Of course, developing a web app is more complicated and requires more time and effort than a traditional presentation. It also requires specialized expertise that might prompt involvement from web developers or other members of the IT department. So, you must also take those factors into account.

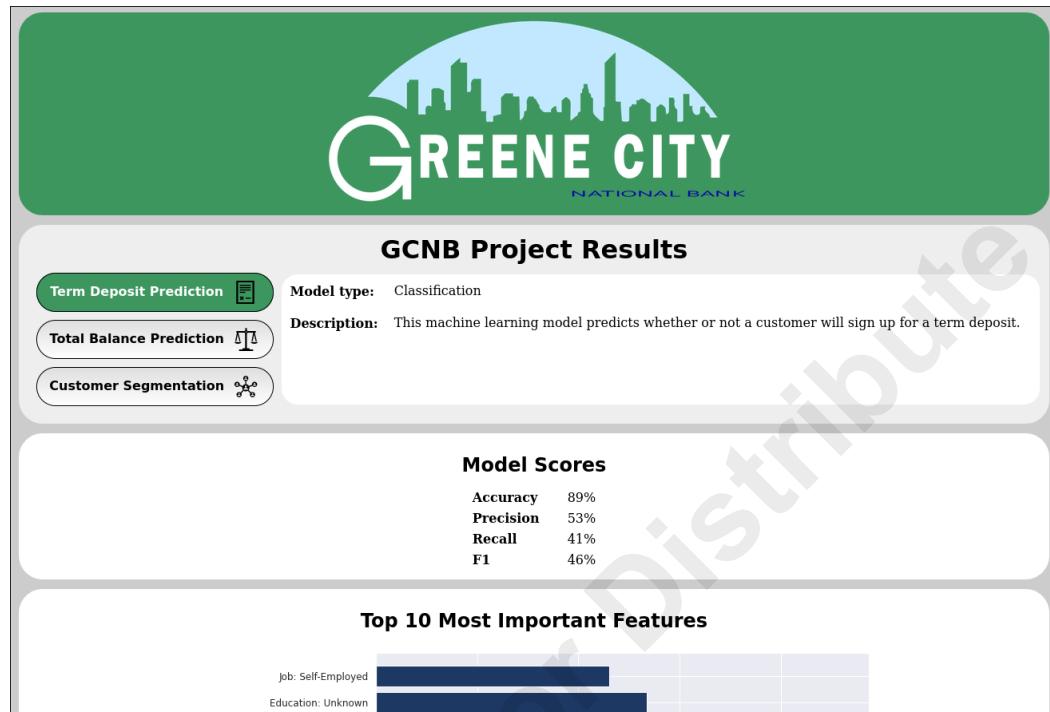


Figure 8–6: An example of a web app used to demonstrate model results.

HTML

Hypertext Markup Language (HTML) is the most common language used to develop web pages consisting of text, graphics, links, and interactive features. A web page is composed of individual HTML elements that provide these features. In addition to controlling how content is presented, HTML also handles the structure of that content as it is rendered on the page. Compared to other markup languages like XML, HTML tends to be lenient with regard to errors like incomplete or mismatched markup.

Most modern web apps are not built on HTML alone, but HTML is still an important component. If you plan on demonstrating a project in a web app, you need to at least be familiar with HTML syntax and some common elements and attributes, particularly if you plan on keeping your web app simple and don't want to leverage a web framework.

For the most part, HTML syntax involves placing content within opening and closing tags that define page elements, and nesting those elements within other elements to create a structure. For example, an unordered list might be written as:

```
<ul>
  <li>List item 1</li>
  <li>List item 2</li>
  <li id='item3'>List item 3</li>
</ul>
```

The overall `` element defines a list, and is closed by the corresponding `` tag. Nested within these tags are individual `` elements that define each item in the list. Likewise, they all have closing `` tags. The last list item also defines an attribute that gives the element an ID that can be referenced later (`id='item3'`). In the context of the overall web page, the `` element will itself probably be nested within a `<body>` element, which will be nested within an `<html>` element. That's essentially how HTML is written.

CSS

When it comes to developing a standard web app, HTML is just one part of a trio of languages. The next is ***Cascading Style Sheets (CSS)***, which is a set of rules that define the appearance of web content. This can include everything from how certain page elements are colored, to how text is aligned and indented, and so much more. The main purpose of CSS is to separate the web page content (defined through HTML) from its presentation style (defined through CSS). So, most web pages pair HTML with CSS to keep these functions distinct and manageable. Otherwise, trying to define a page's appearance solely through HTML is going to be tedious and will leave you with a mess of code.

As the name implies, CSS is implemented in style sheets. Style sheets are text files that specify how various HTML elements are displayed on the page. An HTML file will reference one or more style sheets in its markup in order to use it. Just like HTML, the style rules in CSS have a syntax that you should be familiar with. The basic syntax involves selecting the HTML element you want to apply the style to, then specifying the style property, and finally providing the actual value for the property. For example, if you wanted to make the text in all list items the color blue, you could write the following:

```
li { color: blue; }
```

So, rather than define this color as an attribute of each individual HTML `` element, you've made all of the list items blue with a single line. Of course, CSS allows for more complex rules than this, but this is fundamentally how it's used. Like in any web app, using CSS will make your demonstration more visually appealing and enhance its ability to communicate your intended message.

JavaScript

JavaScript is the last part of the web language trio. It is a scripting language that provides interactive functionality to a web app. JavaScript essentially tells the browser how to change the page in

response to some event, like a user clicking on a button, or submitting text in a form. The dynamic nature of the web is in large part due to JavaScript.

Like CSS, JavaScript can be written as a separate file and referenced in HTML where it is needed. And, of course, it has its own syntax. Because it's a scripting language, the syntax should be recognizable as it supports the object-oriented paradigm of other modern scripting languages like Python, as well as other paradigms such as procedural programming and imperative programming. The following example references the unordered list defined through HTML earlier, and creates a button through JavaScript that, when pressed, changes the text of the last list item:

```
<button onclick='document.getElementById("item3").innerHTML =
"List item changed!"'>Update List</button>
```

The `<button>` tag is just an HTML element; the value of the `onclick` attribute is where the actual JavaScript appears. In this case, it gets the list item that has an ID of `item3` using the `getElementById()` method, then updates the list item content through the `innerHTML` property when the button is pressed. The result changes the list item from saying "List item 3" to "List item changed!" There are of course many more methods and properties for dynamically changing a page, but this is basically how it's done. JavaScript can add that extra level of interactivity to your data science demonstrations, giving your audience a more interesting and effective look into the project.

Web Frameworks

A **web framework** is a programming library that supports the development of web apps and other web-based services through a standardized interface. The purpose of a web framework is to alleviate the effort involved in writing certain features and functionality from scratch using basic tools like HTML. The more dynamic a web app becomes, the more complicated and tedious it is to program using these basic tools. Web frameworks provide functions that the developer can use to automate and streamline the process. So, the app requires less overall effort to build, while still being highly interactive. In fact, because web frameworks standardize web development procedures, they have become very popular among developers who are tasked with building highly dynamic apps.

Like any other programming library, a web framework is implemented within a specific programming language. Ruby on Rails supports the Ruby language, for instance, and can't be used with a language like Python®. Still, many frameworks provide some of the same basic features to a developer programming in the appropriate environment, including: HTML template generation, input validation, defense against web-based attacks like cross-site scripting (CSS) and SQL injection, database connection and configuration, session management, and so on.

Common Web Frameworks

There are many frameworks available for each programming language. Here's a sampling of some of the most popular:

Language	Web Frameworks
Python	<ul style="list-style-type: none"> • Flask • Django • web2py
Ruby	<ul style="list-style-type: none"> • Ruby on Rails • Sinatra • Padrino

<i>Language</i>	<i>Web Frameworks</i>
PHP	<ul style="list-style-type: none"> • Laravel • Symfony • CodeIgniter
Java	<ul style="list-style-type: none"> • Spring Framework • Apache Struts • Grails
JavaScript	<ul style="list-style-type: none"> • AngularJS • Express • Meteor

Consider researching these in more depth, particularly ones that support the language(s) you already work with.

Flask

[Flask](#) is a lightweight web framework—also called a microframework—for Python. It does not provide many common web framework features on its own, but rather relies on the developer to leverage these features through extensions. Most of these extensions are created by third-party developers and released publicly on the Python Package Index (PyPI), the official repository for third-party Python packages. So, you can add whatever functionality you desire to your Flask environment on an ad hoc basis. You might add an extension for connecting to a database in order to access your project's dataset. You might add another extension to make graphing and charting easier. There are also extensions for providing RESTful web services, forms, tests, and many more. You can even build your own extensions if existing ones don't suit your needs.

Flask is meant to be simple to set up and easy to use. It's very common for developers new to web app development to choose it as their first framework. Here's an example of a "Hello, World!" app developed in Flask:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, World!'
```

The `route()` decorator tells Flask what URL will trigger the function that follows. In this case, the root URL will trigger the function that returns "Hello, World!" to the page. Now the `flask` server just needs to run this script, and you'll be able to connect to the page via a browser.

Even experienced web developers can benefit from using Flask to build single-purpose web apps. Each project in Flask builds a single application, so smaller, more focused projects tend to be the best fit. Also, Flask's flexibility means that it supports apps that need to change on a frequent basis. For example, your model demonstration might need to change based on the stakeholder(s) you're presenting it to, or if the project requirements change after you build the first prototype.



Note: Flask's official website is located at: <https://flask.palletsprojects.com/>

Django

Django is a fully featured web framework for Python. It is often compared to Flask. Although both are web frameworks, they have drastically different philosophies. Whereas Flask places emphasis on simplicity and modularity, Django has a robust toolkit with most features already built in.

Compared to Flask, Django is much more involved and time consuming when it comes to setup. Even just writing a simple "Hello, World!" program requires a lot of preliminary work to get the environment running properly. However, the strength of Django is that development work can proceed much more quickly once setup is complete, because most of the functionality that you'll need for a web app is already included. Django is also versatile in that it can be used for just about any purpose. In addition, a Django project can be split into multiple apps that work together in an overall environment. It's for these reasons that Django is better suited to larger, more complex projects that need to include a lot of different features.

So, new developers who start out in Flask might eventually migrate to Django when their projects start becoming more complex. For a data science project, a simple demonstrative web app written in Flask might suffice. However, the organization may want to build a highly interactive app that will be used by many stakeholders. For example, if you work in the government sector or for a large corporation, you might want to (or be required to) make your findings public on the Internet. Django might be the right tool for creating such an app.

Meet Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

Ridiculously fast.
Django was designed to help developers take applications from concept to completion as quickly as possible.

Reassuringly secure.
Django takes security seriously and helps developers avoid many common security mistakes.

Exceedingly scalable.
Some of the busiest sites on the Web leverage Django's ability to quickly and flexibly scale.

[Download latest release: 3.2](#)

[DJANGO DOCUMENTATION >](#)

Support Django!

Messense Lv donated to the Django Software Foundation to support Django development. Donate today!

Latest news

Django 3.2 released
Django 3.2 has been released!
Posted by Carlton Gibson on April 6, 2021

Django security releases issued: 3.1.8, 3.0.14, and 2.2.20
Django 3.1.8, 3.0.14, and 2.2.20 fix a security issue.
Posted by Mariusz Felisiak on April 6, 2021

[MORE NEWS >](#)

Figure 8-7: The Django home page.



Note: Django's official website is located at: <https://www.djangoproject.com/>.

Guidelines for Demonstrating Models in a Web App

Follow these guidelines when demonstrating models in a web app.

Demonstrate Models in a Web App

When demonstrating models in a web app:

- Start small and simple, using an iterative approach to refine a web app into a more complex solution.
- Be aware that web apps can be complicated to create and maintain, especially those that have a mobile component.
- Consider engaging an IT or development organization to assist with web app design for an application projected to have a longer lifespan.
- Use a built-in utility or application to display results, conclusions, and charts, if provided with the data science tool.
- Consider prototyping a solution with a small group before it is enabled for everyone.
- Remember to tell a story with the data.
- Focus more on the information being presented than fancy features. Sometimes an interactive Jupyter Notebook will be sufficient for initial presentations.
- Consider where a web app will be deployed based on the availability required.
- Be aware of security issues for your web app. Consider any proprietary information that may be revealed, and the security of your web app platform and system. These are especially important if your web app is being exposed to the public.

ACTIVITY 8-2

Demonstrating Models in a Web App

Data Files

All files in /home/student/CDSP/Web App/

Scenario

You and your team have time to spare before presenting your project results to stakeholders, so you decide to develop a web app that can demonstrate your machine learning models to GCNB managers. Although you'll use the app during a live presentation, the app should be usable by managers who want to retrieve results themselves, based on whatever it is they're interested in. The results should focus on the main business tasks that the project set out to accomplish, as well as the performance of the models that you developed to support those tasks.

You'll build a proof of concept (POC) for this web app so that you and your team can determine whether or not it's worth developing something more fully featured. You've decided to use Flask as your web app framework for its simplicity and ease of implementation.

1. Open the notebook and start the web app.

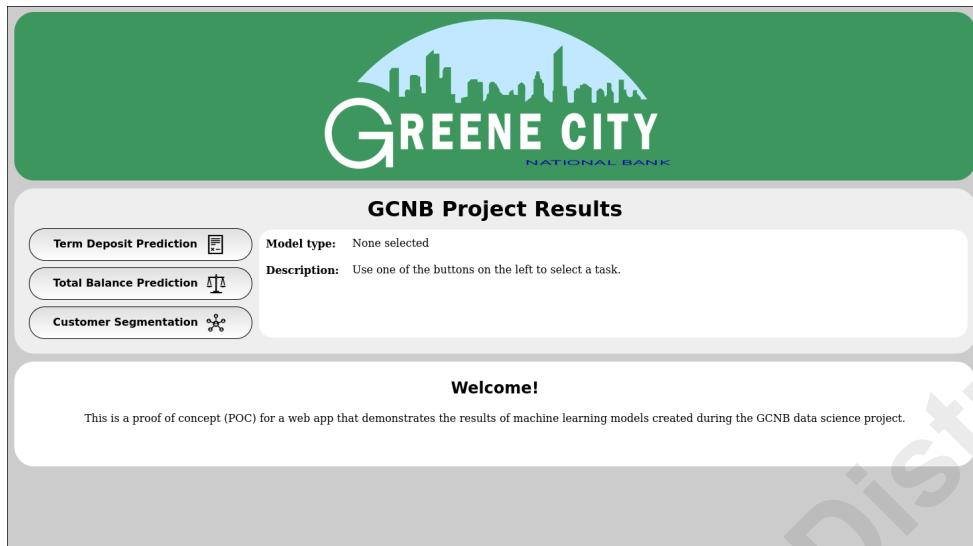
- a) In the Jupyter Notebook web client, navigate to the **CDSP/Web App** directory.
- b) Select **Demonstrating Models in a Web App.ipynb** to open it.
- c) From the menu, select **Kernel→Restart & Run All**.
- d) Select **Restart and Run All Cells**.
- e) Scroll down to the bottom of the notebook and verify that the web server is running.

```
In [*]: 1 if __name__ == "__main__":
2     app.run()
* Serving Flask app "__main__" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- f) Select the link to <http://127.0.0.1:5000/> to open it in a new tab.

2. Take a tour of the web app.

- a) Examine the web app's welcome page.



The page has a welcome message, as well as three buttons on the left that each correspond to tasks that map to the models you built earlier.

- b) Select the **Term Deposit Prediction** button.
 c) Verify that the **Model type** and **Description** changed to match the selected task.
 d) Scroll down and observe the results of the model.

The first section lists evaluation metrics for the classification model, as well as how the model scored on each. Below that is a feature importance plot, and below that is a lift chart.

- e) Scroll back up and select the **Total Balance Prediction** button.
 f) Scroll down and observe the results of the model.

This page also shows a feature importance plot. Below that is a residual plot for the total balance predictions.

- g) Scroll back up and select the **Customer Segmentation** button.
 h) Scroll down and observe the results of the model.

The first plot is a bar chart that shows the percentage of customers that appear in each cluster. Below that is a grid of bar charts that shows the frequency of customer ages for each of the 12 clusters.

- i) When you're done navigating the web app, switch back to the **CDSP/Web App** tab.

3. Examine the web app's HTML templates.

- a) From the directory listing, select **templates** to open that directory.
 b) Check the check box next to **demo.html**, then, from the menu, select **Edit**.

- c) Verify that the HTML file was opened in a new tab.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <link rel='stylesheet' href='{{ url_for("static", filename="main.css") }}'>
5 </head>
6
7 <body>
8   <div class='logo_area'><img src='static/logo.png' alt='Greene City National Bank' class='gcnb_logo'></div>
9   <div class='controlgrid'>
10    <div class='header'>
11      <h1>GCNB Project Results</h1>
12    </div>
13    <div class='controls'>
14      <a href='/?model=clf' class='{{ img_clf }}'>Term Deposit Prediction<img src='static/clf.svg' class='icon' /></a>
15      <a href='/?model=reg' class='{{ img_reg }}'>Total Balance Prediction<img src='static/reg.svg' class='icon' /></a>
16      <a href='/?model=clust' class='{{ img_clust }}'>Customer Segmentation<img src='static/clust.svg' class='icon' /></a>
17    </div>
18    <div class='desc'>
19      <table>
20        <tr><td><b>Model type:</b></td><td>{{ model_type }}</td></tr>
21        <tr><td><b>Description:</b></td><td>{{ model_desc }}</td></tr>

```

- d) Examine the HTML code.

- Lines 1 through 5 set up the HTML file so that browsers know how to parse it. The `<html>` tag is closed at the bottom of the file.
- Line 4 links the HTML file to a CSS file that controls the page's style. You'll examine this file shortly.
- Lines 7 through 24 set up the navigation menu and description section. The task buttons are really just hyperlinks.
- Note that some values are enclosed in double curly braces (`{{ }}`). These denote variables that are used by Flask to change the web app based on conditions programmed in Python. For example, `{{ model_desc }}` changes the model's description based on what model the user selected.
- Lines 26 through 38 define the area where the charts are plotted, depending on what model is selected.

- e) Switch back to the **CDSP/Web App/templates** tab and examine the other files in this folder.

There's a separate `welcome.html` page for the initial welcome page, as well as a `demo_w_scores.html` file. This template is used for the page that demonstrates the classification model, since it includes an extra element that the other two don't: a table of the model's scores. Both of these templates are similar in construction to `demo.html`.

- f) If desired, open these other templates in **Edit** view and examine their contents.
- g) When you're ready, switch back to the **CDSP/Web App/templates** tab.

4. Examine the web app's CSS file.

- a) Navigate to **CDSP/Web App/static**.
- b) Check the check box next to `main.css`, then, from the menu, select **Edit**.

- c) Examine the CSS code.

```

1 body {
2   background: #cccccc;
3 }
4
5 /* Bank logo image */
6 .gcnb_logo {
7   max-height: 250px;
8   height: auto;
9 }
10
11 /* Bank logo area */
12 .logo_area {
13   text-align: center;
14   background: #3e965f;
15   border-radius: 30px;
16 }
17
18 /* Format for any first-level headings */
19 h1 {
20   text-align: center;
21   font-family: Sans-Serif;
22   padding-top: 0px;
23   padding-bottom: 0px;
24   margin-top: 0px;
25   margin-bottom: 0px;

```

This style sheet formats multiple sections and contexts. They all relate to the HTML templates you just looked at. For example, lines 5 through 9 are setting the style for the GCNB logo at the top of the page. In the HTML template, that logo was given a class called `gcnb_logo`, which is what line 6 is referencing.

- d) Briefly examine the rest of the CSS file as desired to see how the appearance of the web app is being defined.
 e) When you're ready, switch back to the **Demonstrating Models in a Web App** tab.

5. Examine the Python code that prepares the models for demonstration.

- a) View the cell titled **Import software libraries**, and examine the code listing below it.

As with the previous notebooks you've worked with, this one needs to import several necessary libraries.

- b) Scroll down and view the cell titled **Load the users dataset**, then examine the code listing below it.

This is the same users dataset from which you've created your models. Although you'll load each model, you need to load the dataset as well to generate predictions and scores from those models.

- c) Scroll down and view the cell titled **Load and prepare the classification model**, then examine the code listings below it.

The pickle file of the classification model you saved earlier is being loaded. The code block below that is splitting the dataset, generating predictions from the model, calculating test scores, and determining feature importances. All of these are used in the results that appear on the web app.

- d) Scroll down and view the cell titled **Load and prepare the regression model**, then examine the code listings below it.

As you might have guessed, these code blocks do something similar, except this time for the regression model and its results.

- e) Scroll down and view the cell titled **Load and prepare the clustering model**, then examine the code listings below it.

These code blocks prepare the clustering model as needed.

- f) Scroll down and view the cell titled **Set model descriptions**, then examine the code listing below it.

These variables set the description for each model type, and will be passed to the HTML template so that the app knows which one to display based on the user's selection.

- g) Scroll down and view the cell titled **Set plot style**, then examine the code listing below it.

This code block sets a consistent style for the plots using Seaborn.

6. Examine the Python and Flask code that constructs the web app.

- a) View the cell titled **Construct the web app**, and examine the code listings below it.

```
In [11]: 1 app = Flask(__name__)

In [12]: 1 @app.route('/', methods = ['GET'])
2 def demo():
3
4     # Defaults before task selection.
5     which_layout = 'welcome.html'
6     model_type = 'None selected'
7     model_desc = 'Use one of the buttons on the left to select a task.'
8     table_title = ''
9     img_1_title = ''
10    img_2_title = ''
11    output_table = {}
12    img_1_pngB64 = ''
13    img_2_pngB64 = ''
14    img_1_fig = Figure()
15    img_2_fig = Figure()
16
17    # Normal button states are shown by default.
18    img_clf = 'clf_normal'
19    img_reg = 'reg_normal'
20    img_clust = 'clust_normal'
21
22    # Get selected task from the URL query string.
23    selected_model = request.args.get('model')
24
25    if selected_model == 'clf':
26        img_clf = 'clf_selected'
27        which_layout = 'demo_w_scores.html'
28        model_type = 'Classification'
29        model_desc = clf_desc
```

The first block is just a single line that defines `app` as a Flask object. The second block is where the web app code appears.

- Line 1 sets a route for the Flask app. Although there are essentially four "pages," (one for each model, plus the welcome page) only one route is defined. It takes the root page as its path (' / ') and will use GET as its HTTP request method.
- Lines 4 through 20 set default values for the variables that will be used to change the app's state.
- Line 23 uses a GET request to obtain the state of the URL query string; e.g., <http://127.0.0.1:5000/?model=clf> would get `clf` as the query string. This is done so that the app knows what model/task is currently selected.
- Lines 25 through 136 use that `selected_model` state to determine what to do on the page through a series of `if...else` conditions. These conditions determine the model's description, the content of the HTML headings, what plots to generate, etc.
- Each condition plots two graphs. For example, lines 36 through 45 generate the feature importance plot for the classification model. Note that these plots are deliberately generated on the fly, rather than referenced as static images. One approach is not necessarily better than the other. Static images require less code in the web app, and will load quicker than complex plots that are generated each time. However, generating plots dynamically ensures that the plots always show the current results in case the model changes.
- Lines 138 through 150 convert the plots into PNG byte stream, then into Base64, so that they can be plotted on a web page without that page referencing a static image file.
- Lines 152 through 164 tell Flask how to render the HTML templates using the variables that were defined through the `if...else` conditions. These variables are referenced in the template as `{{ variable_name }}`

7. Examine the Python code that runs the app.

- a) View the cell titled **Run the web app**, and examine the code listing below it.

This code runs the Flask `app` object when the main module (the notebook, in this case) is executed. The execution state is continuous, since it needs to act as a server that receives and processes incoming requests. In fact, those requests and responses are logged and sent to the output block. Every time you select a button in the web app, a new GET request is sent for that particular model's query string. The 200 code indicates the request succeeded.

8. Shut down this Jupyter Notebook kernel.

- a) From the menu, select **Kernel->Shutdown**.
 - b) In the **Shutdown kernel?** dialog box, select **Shutdown**.
 - c) Close all open tabs except for the **CDSP/Web App/static** tab.
-

Do Not Duplicate or Distribute

TOPIC C

Implement and Test Production Pipelines

Much of what you've done throughout the data science project can be automated in some way. The goal is to spend less time performing some of the more repetitive tasks, and more time on tasks that require your own judgment. This is where pipeline automation comes into play.

Put a Model into Production

Once project development has completed and the method for addressing the business issue has been identified and accepted by stakeholders, it is time to deploy the model to production. Some machine learning models you develop may not simply result in a one-time recommendation or answer. They may instead represent the prototype for a new capability that you must roll into existing products or processes, perhaps a new feature or function in software running on the organization's website, mobile app, or enterprise application. Or, perhaps the solution the model will be a part of is still under development.

When you put a machine learning model into production, you'll need to consider how you'll deploy the model on a software platform, perhaps involving different code than you used to develop the initial model. You may hand your prototype off to software developers to be incorporated into a larger solution. A machine learning model put into production will require further development, testing, and maintenance over time.

At this point in the project, it becomes necessary to answer questions such as:

- Should a data pipeline be constructed to support the model, and if it should, then how?
- How should the solution be developed and deployed to perform adequately at scale and allow for growth and future improvements?
- How should the system be maintained to keep it running optimally?

Data Pipelines

In the worlds of data science and machine learning, data ***pipelines*** automate various phases of the process from the initial accumulation and cleanup of data from multiple sources, to the data's incorporation into machine learning models, to deployment where results such as predictions and classifications are provided, to the ultimate retirement and safe disposal of data. In other words, the data science process can follow a repeatable pattern where the project can move from phase to phase without a practitioner needing to manually perform every task or subtask. This does not mean that you can be completely hands off. It just means that your job is made easier so that you can focus more on the aspects of the project that require human judgment, rather than getting bogged down in tasks that a machine should handle. For example, removing duplicate data can be automated in a pipeline, but drawing conclusions and formulating hypotheses based on an exploratory analysis of that data is still yours to do.

Automated pipelines enable you to optimize a project for simplicity, speed, portability, and reusability. Consider some examples of how automation might apply to each step of the process:

- **Problem formulation**—This phase is mostly conceptual and involves human assessment, but you can actually use machine learning models to identify problems that the business may be interested in solving. Perhaps a model detects some sort of shortcoming in the manufacturing process that results in delays, and you therefore set out to improve manufacturing speeds through a new data science project.
- **Data collection**—Your endpoints can be configured to constantly ingest data from services like web APIs. For example, if your project involves analyzing weather patterns, you can retrieve today's weather data from an existing resource.

- **Data processing**—Many issues with data can be identified and corrected by automated processes, including duplicates, missing values, etc. For example, you might configure a programming environment to automatically remove missing values from weather data as it comes in.
- **Data analysis**—You can automate the creation of charts, graphs, and other visuals based on the characteristics of the dataset. For example, bar charts can be automatically created from the frequency of categorical variables.
- **Model training**—Training basic models on prepared data can follow a repeatable process. For example, the pipeline might train the model on one algorithm, then another algorithm, and compare their scores to determine how to proceed.
- **Project finalization**—If a presentation environment is set up at the beginning of the project lifecycle, the pipeline may be able to automatically load it with relevant data as the project progresses.

Also, when you set up a pipeline to automate deployment processes, you should be aware of how that model will be active within the production environment. You can think of models as being *online* or *offline*. An offline model trains on a corpus of historical data and can make estimations on new data. This means the model is static and doesn't change. An *online* model is one that continuously trains on new data and is therefore dynamic. Because its learning parameters are updated on a schedule, the estimations it makes will change over time, even on the same data.

scikit-learn Pipeline

The scikit-learn `Pipeline` module is an example of a simple automation tool. It enables a developer to define a list of transformation steps that will be executed sequentially. A transformer is any data preparation function available in scikit-learn, like `KBinsDiscretizer()` to bin a continuous variable, or `OneHotEncoder()` to one-hot encode a categorical variable. These transformations are added to a `Pipeline` object. The final step is to specify an algorithm to train the model from. When you call `fit()` on the `Pipeline` object, it will perform all of the transformations on the data and train a model. So, all of these tasks can be repeated on other datasets by calling a single function.

Model Drift

Over time, offline machine learning models may become less effective in making estimations on new data due to **concept drift**, also called **model drift**. Model drift occurs when the domain being studied has variables that naturally change over time. This accounts for a great deal of domains, but some change much more rapidly than others, so model drift may be more of a concern in certain domains than others.

For example, a model may predict the behavior of online shoppers, such as how they respond to advertising, coupons, or sales posted on a specific website or mobile app. However, over time, the general behavior of shoppers may change. Websites or apps that used to generate a lot of traffic may decline in popularity, so advertisements on those sites don't produce the same results they used to. The effectiveness of the model may decay as the concept it is trying to estimate changes with the passage of time.

You may not always be able to predict when a model will be subject to concept drift, but the possibility should always be considered when planning how models will be maintained when they are integrated into long-term software solutions. Online models, for instance, can help stem the tide of model drift. Since the model's knowledge of some domain is constantly kept up to date, the assumptions it expresses about that domain can also be considered up to date. In the example of online shoppers' behavior changing over time, you may deploy the solution so that this behavior is continuously tracked and then fed into a pipeline where it can produce a new model. Assuming the model maintains a level of skill, it can help you make advertising decisions that are based on current trends rather than obsolete data.

However, keep in mind that online models are usually more difficult to implement because they require a constant ingest of new and useful data, which may not always be available. It also takes a lot of time and resources to constantly train, tune, test, and deploy updated models.

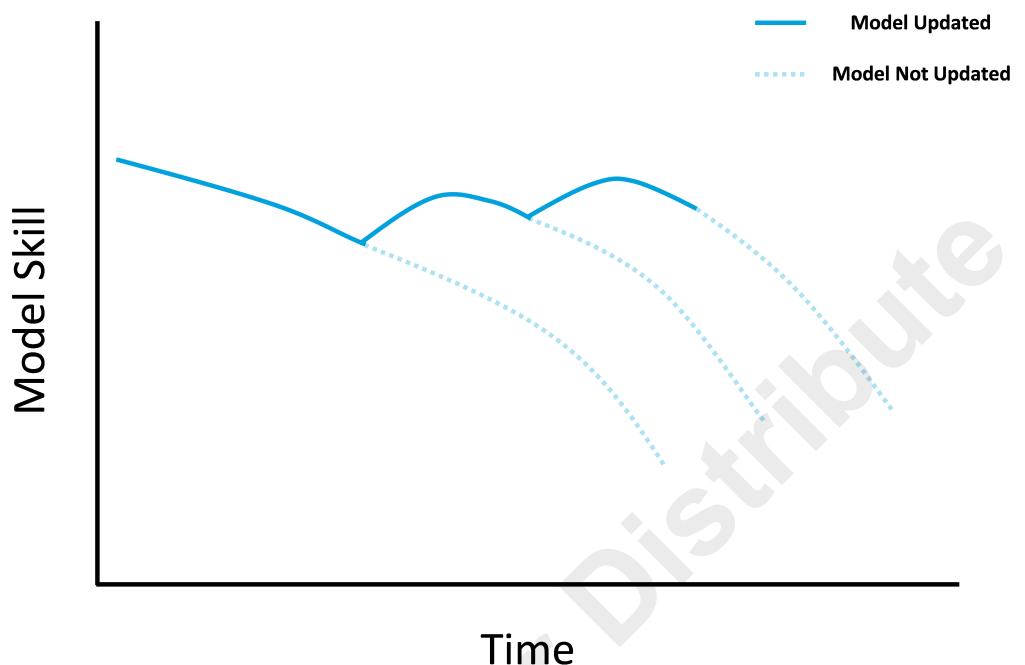


Figure 8–8: Demonstrating model drift over time when models are not updated.

Docker

Docker is an open source platform for building and maintaining virtual containers. A virtual container provides a virtual environment in which an application can run without disrupting the host system. While a form of virtualization, containers differ from virtual machines (VMs) in that they do not virtualize an entire operating system. Instead, an application in the container shares resources provisioned by the host operating system's kernel while also being isolated from that OS or other containers. Because they only access resources when needed and don't contain an entire OS, containers are faster, more lightweight, and more portable than a traditional VM. It's common to run multiple containers side by side on a single host, where each container includes its own standalone application. However, the major downside is that a container built for one operating system (e.g., Windows) is not necessarily compatible with a completely different operating system (e.g., Linux). So, portability can become an issue for organizations that have mixed operating environments.

Docker is the most popular containerization software for both business and personal use. Docker Desktop is the main program that provides a user interface for managing containers, though you can also manage containers through a command-line interface (CLI). Docker Hub is the repository service that enables you to retrieve pre-built Docker container images and upload your own container images.

While not specifically a data pipelining platform, Docker can facilitate pipelining for a data science project due to its ability to run code in an isolated environment. So, you could create a separate container for each step in the pipeline and have their outputs flow from one container to the next. This will ensure that each major task can be abstracted from both the underlying OS and the other containers, making it easier to configure that task without disrupting the others. Docker also helps make a data pipeline portable, since you can package each container and share them with your colleagues as needed.

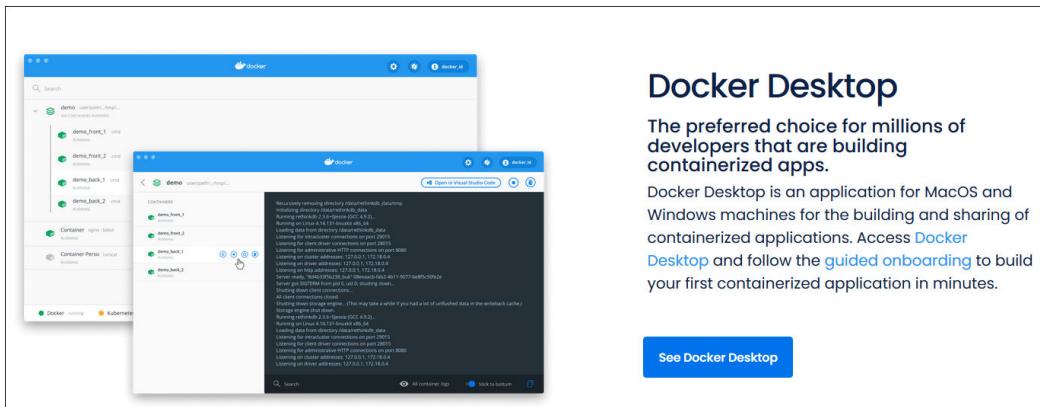
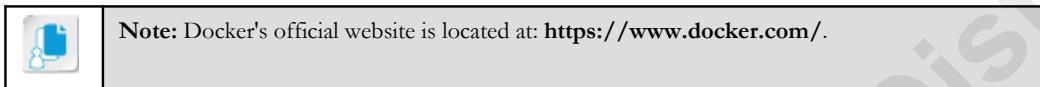


Figure 8-9: The Docker products page demonstrating Docker Desktop.



Kubernetes

Kubernetes is an open source platform for orchestrating the deployment and management of virtual containers. Kubernetes automates container-related tasks and helps organizations scale their container environments as needed. It does this through the use of computer clusters. A cluster is a group of hosts that work together as if they were one system. So, you could run an entire infrastructure of containers across multiple physical hosts or VMs to simulate a deployment at certain scales, or to leverage more hardware for resource-intensive projects. Kubernetes also supports container orchestration on cloud platforms, and clusters can span multiple locations, both on premises and off.

Kubernetes supports multiple containerization platforms, but primarily Docker. You might consider using Kubernetes if you find that setting up and maintaining the Docker containers in your pipeline becomes tedious. Or, perhaps your project deals with large-scale data or requires a large-scale deployment; you can use Kubernetes to strengthen your data pipeline and ensure that it has access to enough resources.

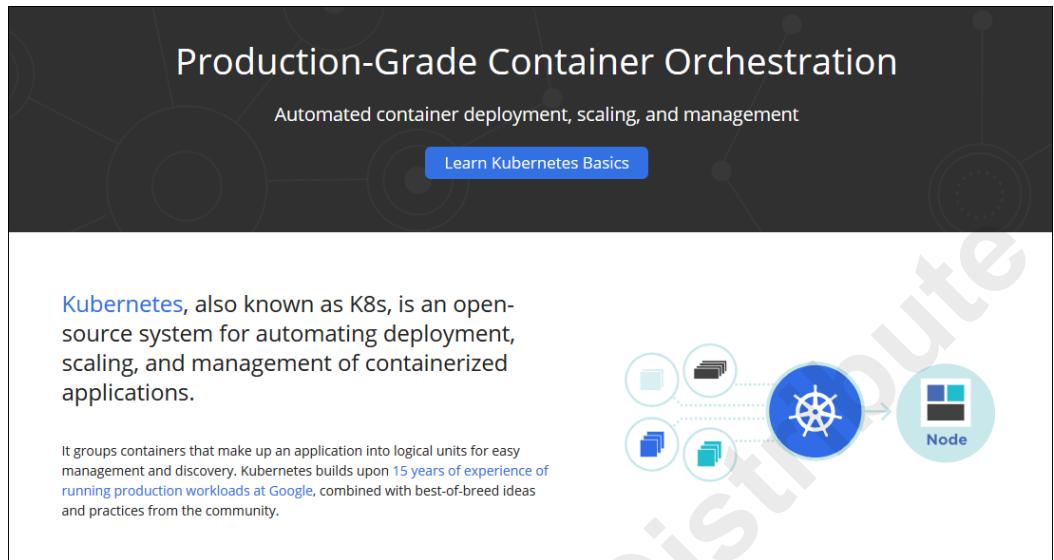


Figure 8-10: The Kubernetes home page.



Note: Kubernetes' official website is located at: <https://kubernetes.io/>.

Amazon SageMaker

Amazon SageMaker is a managed machine learning service that is part of the Amazon Web Services (AWS) cloud platform. It facilitates each step of the data science process, from collecting and preparing data to building and deploying machine learning models. Amazon SageMaker has many individual services to help data science practitioners perform tasks as part of an overall automated pipeline. For example, SageMaker Ground Truth can be used to automatically label training data; SageMaker Clarify can automatically detect bias in data and models; and so on. These services can integrate with multiple different languages and frameworks common to the industry, like R, Python, Jupyter, TensorFlow, and more.

Aside from automation, several Amazon SageMaker services also include pre-built templates and other solutions that you might be able to leverage in your project. For example, SageMaker JumpStart provides access to pre-trained and pre-tuned models that can save you time on training and tuning your own. In addition, Amazon SageMaker also supports container orchestration in the cloud using Kubernetes, and even has its own custom-built orchestration and workflow automation service called SageMaker Pipelines.

Organizations that deal with large and complex data science projects often rely on cloud-based pipelining services like Amazon SageMaker rather than on-premises solutions because of their scalability. Providers like Amazon have access to massive amounts of resources that most organizations don't, so it can be both practical and cost-effective to move data science operations to the cloud.

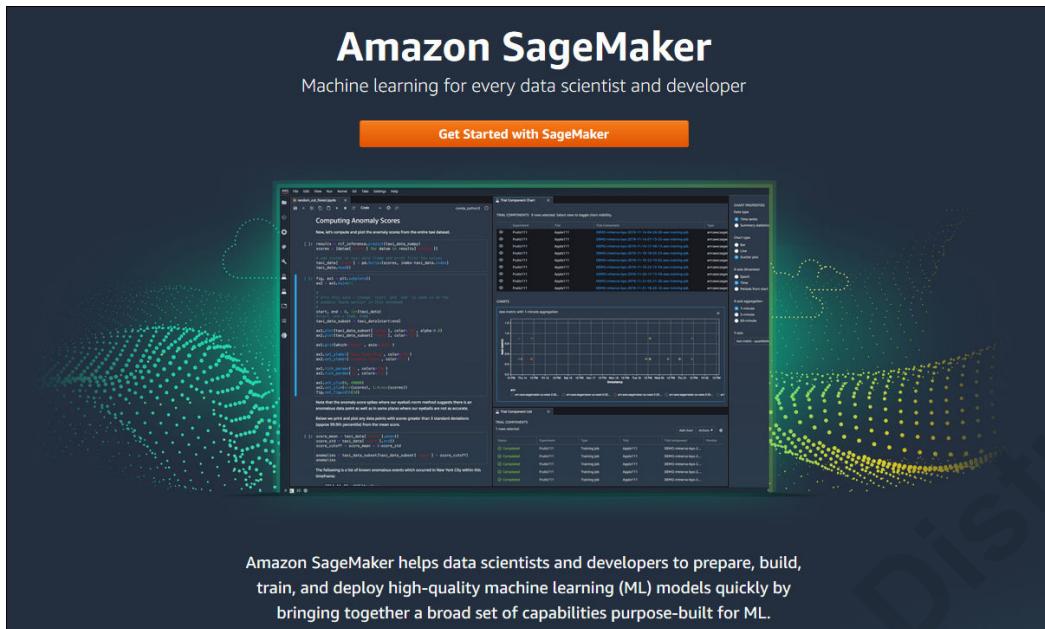


Figure 8-11: The Amazon SageMaker home page.



Note: Amazon SageMaker's official website is located at: <https://aws.amazon.com/sagemaker/>.

Azure Machine Learning

Azure Machine Learning is a cloud-based data science and machine learning automation service offered as part of the Microsoft® Azure® cloud platform. Like Amazon SageMaker, Azure Machine Learning includes many tools and solutions for automating the data science process as part of an overall production pipeline. You can streamline the data preparation process, train models using pre-configured algorithms, deploy models to endpoints, and so on. These services also support much of the same data science languages and frameworks that are common within the industry.

Azure Machine Learning also emphasizes the idea of MLOps, or the integration of machine learning practices and traditional IT operational practices—similar to the marriage of software development and operations in DevOps. In addition to automating tasks like data preparation, model training, and model deployment, an MLOps approach focuses on tracking many different types of data, including data used in training, data about models, and process metadata. It also supports data governance principles so that projects align with business objectives and stakeholder requirements.

Like Amazon SageMaker, Azure Machine Learning is another viable option for moving data science operations to the cloud. One is not necessarily better than the other, and there are certainly other options available. You and your team should do your own research and compare your business needs to what each service offers in order to guide the organization's decision makers.

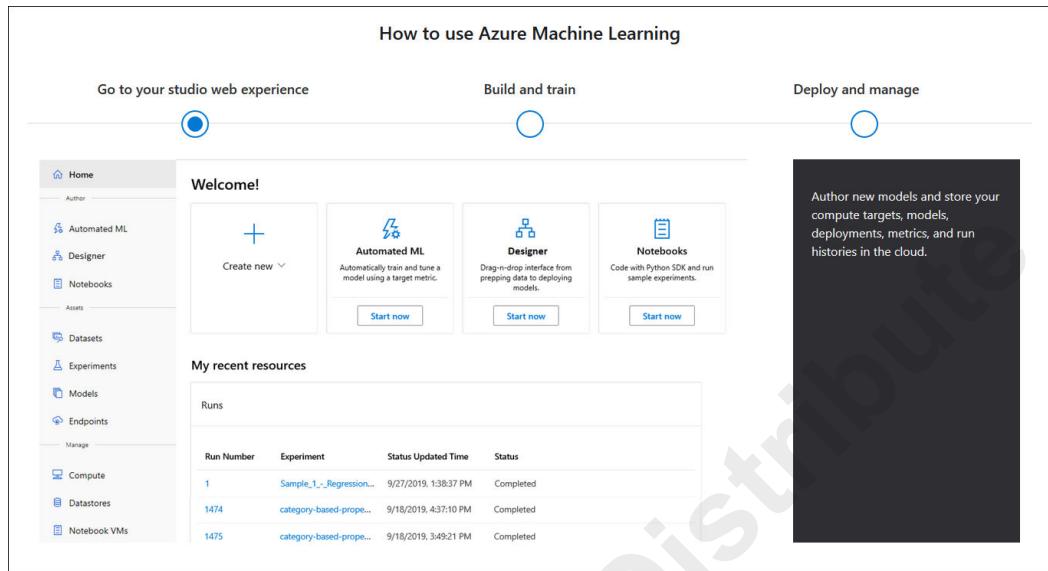


Figure 8-12: A demonstration of the Azure Machine Learning interface from the home page.



Note: Azure Machine Learning's official website is located at: <https://azure.microsoft.com/en-us/services/machine-learning/>.

Monitor Models in Production

One of the major benefits of a production pipeline is that every intermediary task, as well as its output, is contained within a manageable environment. This should make it easier for anyone operating or administrating the pipeline to monitor what exactly is going on. You don't want to be in a situation where you set up the pipeline, pull the data through, and end up with a result you weren't expecting. Your model might outright fail at some point due to a misconfiguration, or it could experience more subtle effects like skill degradation. Another possibility is that the process has unwanted side effects, like private user data leaking outside of the pipeline. Whatever the issue may be, if you catch them early enough, you can avoid wasting a lot of time and effort. This is why it's important to be able to monitor your environment.

What's actually involved in monitoring will depend on the platform you're using and the needs of the business. For example, you might log each epoch of a model in training to ensure it is progressing along an acceptable schedule. If the model's training speed appears to decrease beyond some threshold, you can be notified and then take action. Or, perhaps you have a dashboard that continually updates the evaluation metrics for a model as it undergoes hyperparameter tuning. You can quickly verify whether your model's skill is improving or degrading over time. Essentially, you can monitor your models at any level of granularity you desire.

Pipeline Monitoring Solutions

The following table lists some common solutions for monitoring data pipelines.

Solution	Description
Kubeflow	Whereas Kubernetes is a general platform for container orchestration, Kubeflow specifically targets the orchestration of data science and machine learning pipelines that run within a Kubernetes environment. Kubeflow therefore automates much of the tasks discussed previously. As far as monitoring goes, Kubeflow provides several features for keeping track of your projects, including a central dashboard that gives an overview of clusters and recent tasks; logging of model metrics; and support for a third-party monitoring tool called Prometheus. Official website: https://www.kubeflow.org/
MLflow	MLflow is another open source platform for managing and automating data science and machine learning projects. It was incorporated into the Linux Foundation in 2020. One of its components is MLflow Tracking, which can log various information about a project including parameter configurations, model metrics, file outputs, and more. MLflow Tracking has APIs for Python, R, Java, and RESTful querying. It also has a GUI program. Official website: https://mlflow.org/
Datadog	Datadog is not specifically related to data science or machine learning. Rather, it is meant as a monitoring platform for a number of different cloud services. It can be integrated with services from Microsoft Azure, AWS, Google Cloud Platform™, and many more. Datadog is able to track application data and logs from these services and consolidate that information into a single monitoring platform. The platform provides interactive dashboards and alert functionality. So, you might use Datadog to monitor information from one or more cloud-based data pipelines. Official website: https://www.datadoghq.com/

Guidelines for Implementing and Testing Production Pipelines

Follow these guidelines when implementing and testing production pipelines.

Implement and Test Production Pipelines

When implementing and testing production pipelines:

- **Create software unit tests that validate the model every time new code is pushed out to the application.** Changes in code or data may adversely affect the model. Modular software design with unit tests that address performance of the model can help to mitigate this problem.
- **Protect the model from upstream changes to the data pipeline.** If possible, work with outside data providers to ensure that they will not make changes that will adversely affect your model's data pipeline. Avoid depending on inputs that are likely to change over time. Provide a configuration layer at the front end of the data pipeline so you can easily adapt to changes in the input data over time.
- **Protect downstream consumers of the model.** When you are unaware of other processes that consume outputs produced by the model, changes to your model (such as retraining or changes to inputs and output) may adversely affect them. By implementing access controls, you will be aware of all consumers and will be able to provide them with proper notifications when changes are anticipated. Require that outside components request access to use the model and/or its outputs so you can track consumer processes.

- **Design for periodic retraining of the model.** Model performance typically decays over time as inputs evolve. It is generally a good idea to periodically retrain a model with recent data.
- **Provide a versioning system for various components used in the application.** Consider providing version numbers for components that may change over time, including model configuration and parameters, specific features used in the data pipeline, and training and validation datasets.
- **Put appropriate protections in place for the data pipeline and all outputs.** The organization must ensure that data is protected for security (the company's confidential data, for example) and privacy (user data protected by laws and regulations).

Do Not Duplicate or Distribute

ACTIVITY 8-3

Building an ML Pipeline

Data Files

/home/student/CDSP/Finalizing/Building an ML Pipeline.ipynb
 /home/student/CDSP/Finalizing/data/users_data_final.pickle
 /home/student/CDSP/Finalizing/data/new_users_data.csv

Scenario

The GCNB marketing project is coming to a close, but that doesn't mean your work is finished. It was good for you to go through the entire data science process, but your approach wasn't necessarily the most efficient. It would help you save time and effort if you were able to automate the process, making it repeatable for future projects. What you need is to set up a data pipeline. Eventually, the team wants to look into cloud-based options for setting up a pipeline that can automate the entire data science process—from data collection all the way to presentation.

For now, however, you want to test out scikit-learn's `Pipeline` module, which focuses on automating the machine learning model training process. The tasks you performed earlier to train, tune, and evaluate models can be condensed and streamlined using this pipeline functionality. This pipeline will be able to take any new set of training data and build a working model from it. However, because the pipeline only focuses on the model training process, it will assume that any new training data fed into it will already have been cleaned.

In this case, you'll implement the pipeline using the familiar GCNB customer dataset, with classification as the goal. After you create the pipeline, you'll test the results on a new set of unlabeled customer data so that you can predict whether or not these new customers will sign up for a term deposit.

1. Open the notebook.

- In the Jupyter Notebook web client, navigate to the **CDSP/Finalizing** directory.
- Select **Building an ML Pipeline.ipynb** to open it.

2. Import the relevant software libraries.

- View the cell titled **Import software libraries**, and examine the code listing below it.
- Select the cell that contains the code listing, then select **Run**.
- Verify that the version of Python is displayed, as are the versions of the other libraries that were imported.

3. Load and preview the data.

- Scroll down and view the cell titled **Load and preview the data**, then select the code cell below it.
- In the code cell, type the following:

```
1 users_data = pd.read_pickle('data/users_data_final.pickle')
2
3 users_data.head(n = 3)
```

- Run the code cell.

- d) Examine the output.

	user_id	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_se...
0	9231c446-cb16-4b2b-a7f7-ddfcfb25aa6	3.0	2143.00	1	0	0	0	0	0	0
1	bb92765a-09de-4963-b432-496524b39157	0.0	1369.42	0	1	0	0	0	0	0
2	573de577-49ef-42b9-83da-d3cf817b5c1	2.0	2.00	0	0	1	0	0	0	0

3 rows × 33 columns

This is the same customer data you've been working with all throughout the course, so it should look familiar.

4. Split the data into train and test sets.

- a) Scroll down and view the cell titled **Split the data into train and test sets**, then select the code cell below it.
b) In the code cell, type the following:

```
1 target = users_data.term_deposit
2 features = users_data.drop(['user_id', 'term_deposit'], axis = 1)
3
4 X_train, X_test, y_train, y_test = train_test_split(features, target)
```

Some tasks, like splitting the target variable from the features, and performing the holdout method, still need to be done manually.

- c) Run the code cell.

5. Define an initial pipeline.

- a) Scroll down and view the cell titled **Define an initial pipeline**, then select the code cell below it.
b) In the code cell, type the following:

```
1 pipe = Pipeline([('scaler', MinMaxScaler()),
2                  ('reduce_dim', PCA(n_components = 2)),
3                  ('model', DecisionTreeClassifier())])
```

This code builds a pipeline using scikit-learn's `Pipeline` module. Each tuple represents a step in the pipeline process, where each step is executed sequentially. So, the `MinMaxScaler()` function will be applied to the data first, then the `PCA()` function, and finally, the last step is always the algorithm used to train the model (in this case, a decision tree model).

- c) Run the code cell.

6. Evaluate the initial pipeline.

- a) Scroll down and view the cell titled **Evaluate the initial pipeline**, then select the code cell below it.
b) In the code cell, type the following:

```
1 pipe = pipe.fit(X_train, y_train)
2 print('Model accuracy on test data:', pipe.score(X_test, y_test))
```

Whenever you call `fit()` on the `Pipeline` object, it will perform all of the transformation steps in the pipeline. The `score()` function will take the default score on the test data. In the case of classification, the default metric is accuracy.

- c) Run the code cell.

- d) Examine the output.

```
Model accuracy on test data: 0.8110668437361664
```

The decision tree model created by the pipeline has an accuracy of ~0.81.

- e) Select the next code cell, then type the following:

```
1 y_pred = pipe.predict(X_test)
2 print(Counter(y_pred))
```

- f) Run the code cell.

- g) Examine the output.

```
Counter({False: 9935, True: 1360})
```

The model predicted 9,935 False values and 1,360 True values in the test set.

- h) Select the next code cell, then type the following:

```
1 results = pd.concat([y_test.iloc[:5], X_test.iloc[:5]], axis = 1)
2 results.insert(1, 'term_deposit_pred', y_pred[:5])
3 results
```

- i) Run the code cell.

- j) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_services
43308	False	True	0.0	1369.42	1	0	0	0	0	0
32770	False	False	2.0	246.00	0	0	0	1	0	0
17440	False	False	0.0	1369.42	0	0	0	0	0	0
36164	False	False	0.0	1369.42	0	0	0	1	0	0
29218	False	False	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

The model incorrectly predicted the first record, but correctly predicted the four after that. Like you've seen before, hyperparameter optimization can improve a model. You can do the same thing with pipelines.

7. Tune the pipeline.

- a) Scroll down and view the cell titled **Tune the pipeline**, then select the code cell below it.

- b) In the code cell, type the following:

```

1 scalers = [None, StandardScaler(), MinMaxScaler()]
2 pca_dims = [None, PCA(n_components = 2), PCA(n_components = 5)]
3 models = [DecisionTreeClassifier(),
4            RandomForestClassifier(random_state = 1)]
5
6 params = {'scaler': scalers,
7            'reduce_dim': pca_dims,
8            'model': models}

```

You'll be performing a grid search, only this time, rather than searching for one model's optimal hyperparameters, you'll be searching for the optimal steps to perform in the pipeline. So, your parameter grid will instruct the search to try out two different scalers (or no scaler, as indicated by None), two different dimensions for PCA (or no PCA), and two different algorithms (decision tree and random forest).

- c) Run the code cell.
d) Select the next code cell, then type the following:

```

1 gs = GridSearchCV(pipe, params, n_jobs = -1, verbose = 2,
2                    cv = StratifiedKFold(5, shuffle = False)). \
3 fit(X_train, y_train)

```

Rather than passing in a model to `GridSearchCV()`, you're passing in the `pipe` object. Otherwise, the process is essentially the same.



Note: Remember that `n_jobs = -1` is telling scikit-learn to use all available CPU threads.

- e) Run the code cell.



Note: It can take up to 10 minutes for the search to complete.

8. Evaluate the tuned pipeline.

- a) Scroll down and view the cell titled **Evaluate the tuned pipeline**, then select the code cell below it.
b) In the code cell, type the following:

```

1 print('Best accuracy score:', gs.score(X_test, y_test))
2 print('Best parameters: ', gs.best_params_)

```

- c) Run the code cell.
d) Examine the output.

```

Best accuracy score: 0.888092076139885
Best parameters:  {'model': RandomForestClassifier(random_state=1), 'reduce_dim': None, 'scaler': MinMaxScaler()}

```

The best score for the tuned pipeline is ~0.89, an improvement over the initial pipeline. The grid search retrieved the following optimal steps:

- The features are standardized.
- No PCA is performed.
- The model is based on the random forest algorithm.

If you had a different set of training data, it might retrieve a different set of pipeline steps. Also, if you had more time, you'd probably want to expand the parameter grid to include more types of scaling and dimensionality reduction options, as well as more types of machine learning algorithms.

- e) Select the next code cell, then type the following:

```
1 y_pred = gs.predict(X_test)
2 print(Counter(y_pred))
```

- f) Run the code cell.
g) Examine the output.

```
Counter({False: 10599, True: 696})
```

The random forest model created from the pipeline predicted 10,599 False values and 696 True values.

- h) Select the next code cell, then type the following:

```
1 results['term_deposit_pred'] = y_pred[:5]
2 results
```

- i) Run the code cell.
j) Examine the output.

	term_deposit	term_deposit_pred	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_services
43308	False	False	0.0	1369.42	1	0	0	0	0	0
32770	False	False	2.0	246.00	0	0	0	1	0	0
17440	False	False	0.0	1369.42	0	0	0	0	0	0
36164	False	False	0.0	1369.42	0	0	0	1	0	0
29218	False	False	0.0	1369.42	0	0	0	0	0	0

5 rows × 33 columns

The model correctly predicted the first five records.

9. Test the model generated by the pipeline on new data.

- a) Scroll down and view the cell titled **Test the model generated by the pipeline on new data**, then select the code cell below it.
b) In the code cell, type the following:

```
1 new_data = pd.read_csv('data/new_users_data.csv')
2
3 new_data
```

This is a new set of data that the model has not yet seen.

- c) Run the code cell.

- d) Examine the output.

	number_transactions	total_amount_usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	job_self-employed	...
0	4	237.10	0	0	0	0	1	0	0	0	0 ...
1	2	-43.12	0	0	1	0	0	0	0	0	0 ...
2	1	789.45	0	1	0	0	0	0	0	0	0 ...
3	4	3291.41	0	1	0	0	0	0	0	0	0 ...

4 rows × 31 columns

This new data has four total customer records. It's also not labeled, so it's closer to approximating the real-world application of your classification tasks: predicting whether or not a customer will sign up for a term deposit.

- e) Select the next code cell, then type the following:

```
1 y_pred = gs.predict(new_data)
2
3 new_data.insert(0, 'term_deposit_pred', y_pred)
4 new_data
```

- f) Run the code cell.
g) Examine the output.

	term_deposit_pred	number_transactions	total_amount.usd	job_management	job_technician	job_entrepreneur	job_blue-collar	job_retired	job_admin.	job_services	...
0	False	4	237.10	0	0	0	0	1	0	0	0 ...
1	False	2	-43.12	0	0	1	0	0	0	0	0 ...
2	False	1	789.45	0	1	0	0	0	0	0	0 ...
3	False	4	3291.41	0	1	0	0	0	0	0	0 ...

4 rows × 32 columns

The model generated from the pipeline predicts that all four of these new customers won't sign up for a term deposit.

10. Shut down this Jupyter Notebook kernel.

- From the menu, select **Kernel→Shutdown**.
- In the **Shutdown kernel?** dialog box, select **Shutdown**.
- Close all tabs in Firefox, then close the terminal window that is running the program.

Summary

In this lesson, you concluded your data science project by gathering your findings and presenting results to the project's stakeholders. You also prepared for future projects by automating key aspects of the data science process through pipelining. All of this is done in service of generating a data-driven solution that stakeholders agree adds value to the business.

In your data science projects, to whom will you typically have to present your findings?

What form will your presentations most likely take?



Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Do Not Duplicate Or Distribute

Course Follow-Up

Congratulations! You have completed the *Certified Data Science Practitioner (CDSP) (Exam DSP-110)* course. You have gained the practical skills and information you will need to plan data science projects, prepare data for effective analysis and training, and build machine learning models that can provide useful predictions and other data-driven business needs. All of these skills combined will help you bring value to the organization by efficiently and intelligently addressing key business issues.

You've also gained the knowledge you will need to prepare for the Certified Data Science Practitioner (CDSP) (Exam DSP-110) certification examination. If you combine this class experience with review, private study, and hands-on experience, you will be well prepared to demonstrate your data science expertise both through professional certification and with solid technical competence on the job.

What's Next?

Your next step after completing this course will probably be to prepare for and obtain your Certified Data Science Practitioner (CDSP) certification. If you're interested in a deeper dive into artificial intelligence and machine learning, including neural networks used in deep learning, consider taking the CertNexus *Certified Artificial Intelligence (AI) Practitioner (Exam AIP-110)* course. If you wish to learn more about specific data visualization tools, consider taking the Logical Operations course *Tableau® Desktop: Part 1 (Second Edition)* and/or *Microsoft® Power BI®: Data Analysis Professional (Second Edition)*.

You may also wish to explore the ethical and security implications of emerging technologies like AI and data science. The Logical Operations course *Certified Ethical Emerging Technologist™ (CEET): Exam CET-110* teaches you how to identify, assess, and mitigate these risks.

Since presenting results to a wider audience is crucial to a data science practitioner, consider taking the Logical Operations course *Effective Presentations (Second Edition)* if you want to further hone that skillset. Also, you may want to pursue the CertNexus *Certified Internet of Things (IoT) Practitioner (Exam ITP-110)* course if you're interested in the application of data science principles to the world of IoT.

You are encouraged to explore data science further by actively participating in any of the social media forums set up by your instructor or training administrator through the **Social Media** tile on the CHOICE Course screen.

Do Not Duplicate Or Distribute

A | Mapping Course Content to CertNexus® Certified Data Science Practitioner (CDSP) (Exam DSP-110)

Obtaining the Certified Data Science Practitioner (CDSP) certification requires candidates to pass CertNexus® Certified Data Science Practitioner (CDSP) (Exam DSP-110).

To assist you in your preparation for the exam, CertNexus has provided a reference document that indicates where the exam objectives are covered in the CertNexus *Certified Data Science Practitioner (CDSP) (Exam DSP-110)* courseware.

The exam-mapping document is available from the **Course** page on CHOICE. Log on to your CHOICE account, select the tile for this course, select the **Files** tile, and download and unzip the course files. The mapping reference will be in a subfolder named **Mappings**.

Best of luck in your exam preparation!

Do Not Duplicate Or Distribute

Solutions

ACTIVITY 1-1: Initiating a Data Science Project

1. So GCNB is clear about what they're asking for, what is data science?

A: Answers may vary, but generally, data science is the process of accumulating data, processing data, and extracting insights from that data. There are many individual tasks that go into the overall process.

2. What stakeholders might be involved in the project, and how might each type of stakeholder contribute?

A: Answers may vary. Some likely stakeholders could include: the data science team members, who perform the day-to-day data science tasks on a technical level; project managers, who ensure that team members have enough resources to do their data science tasks and that these tasks are on time and within scope; business partners, particularly the high-level managers of GCNB who ensure the project is generating value for their business; GCNB's customers, who may provide feedback on user experience or concerns about how they will be affected by any changes to business operations as a result of the project; and potentially many more.

3. What are some additional questions that you could ask of GCNB to help generate ideas for the project?

A: Answers will vary as design thinking tends to be open ended. Some example questions might be: "Why do you believe a data-driven approach will improve your ability to market your services, as opposed to what you're doing now?" "How might decisions made during the project affect customers' attitudes toward the bank?" "How would an increase in service adoption provide direct benefit to customers?" "How might you solicit input from your customers to ensure they participate in the project?" "How could the project be transformative, i.e., fundamentally change the way the bank does business?"

4. What else do both organizations need to discuss with regards to the project's scope before proceeding?

A: Answers may vary. Even though Rudison will provide the personnel and resources, they still need to know how much of each is actually needed. Those decisions will likely be influenced by the proposed timelines and requested deliverables. So the organizations will need to discuss these ideas and come to some conclusions. For example, GCNB might want the project to finish in three months, but Rudison might argue that this is not feasible given the current personnel and resources that they have available, and what type of deliverables GCNB is asking for. The organizations also need to set parameters or limits on the project to keep it focused and prevent scope creep.

5. What might be some key performance indicators (KPIs) for either of these objectives?

A: Answers may vary. KPIs for the high-level objective might be: the overall trend of the service adoption rate before and after implementation of the new system, and the change in revenue before and after implementation. KPIs for the sub-objective might be: how many of the customers targeted by marketing campaigns actually sign up for new services, and the change in the rate of customer churn (dropout).

6. What are some potential security concerns that could impact the project?

A: Answers will vary. Banking information is highly sensitive, and can be used to identify individuals. Neither GCNB nor their customers wants data to be leaked to unauthorized users. GCNB could run afoul of various laws and regulations, whereas users will have their privacy violated and lose trust in GCNB. Therefore, the security of data as it is used in the project is a significant concern. To mitigate these concerns, certain restrictions to data access might be put in place by either GCNB or Rudison. Likewise, data governance policies by either organization may limit how the team is allowed to handle data. For example, GCNB might require that the data is made anonymous if it isn't already.

ACTIVITY 1–2: Formulating a Data Science Problem

1. How might you frame the problem that the project is trying to solve?

A: Answers will vary, as there are many ways to frame the problem. In any case, it should be a clear statement of what the problem actually is. For example: "GCNB has collected a great deal of data about its customers but currently has no way to extract insights from that data that can be used to improve how it conducts its marketing campaigns."

2. What are the components that go into identifying the "why" of the problem, and how might you address them for this project?

A: Answers may vary, but the components generally are: the rationale for solving the problem, the benefits that a solution to the problem could bring, and who will use the solution and for how long. An example of addressing these components could be: "If GCNB does not adopt a data-driven approach to extracting business insights, it will be unable to grow its business and target the appropriate customers in its marketing. The benefits of this project will be an improvement in the accuracy of targeting campaigns, a better sense of what content should go in those campaigns, and a chance to increase customer adoption of services like loans, which can generate more revenue from interest. Internal employees, particularly marketing personnel and managers, will be the likeliest users of the solution. They will be able to generate intelligent decisions about the best ways to approach their campaigns. The solution will be used for as long as GCNB finds it useful."

3. What background information does GCNB need to provide, or that you need to research yourself, to help you develop the solution?

A: Answers may vary. GCNB holds all of the data, so they could help the team greatly if they provided some information about the sources of that data, how it was collected, how it's stored, what format it takes, what it's been used for in the past, and so on. The team at Rudison may need to do their own investigation if GCNB's answers aren't satisfactory. Also, Rudison should research similar case studies of banks and other financial organizations that have used customer data to optimize their marketing campaigns. This could provide some valuable insight into the process before it begins, including any pitfalls.

4. Why not just leverage data science anyway, even if the problem can be solved without it?

A: Answers may vary. Data science is extremely powerful and can provide a great deal of benefit to an organization, but it has risks. The process can be quite complex, especially when trying to collect, transform, analyze, and model large volumes of data. Even though there's a general process to data science, each data science project is different, so there's not necessarily a one-size-fits-all approach to developing a solution. A related problem is that the complexity of data science projects means they often cost a lot of time and money. So if the organization can find a simpler way to achieve their goals, they may save quite a bit on expenses. Ultimately, the organization must be able to justify the use of data science.

5. Given what you know so far, what are some potential ways that this data might be used to achieve outcomes in classification?

A: Answers will vary. Examples of classification outcomes include: whether or not someone should qualify for a loan, whether or not someone should be contacted by the next marketing campaign, and what devices customers are most likely to use.

6. What are some potential ways this data might be used to achieve outcomes in regression?

A: Answers will vary. Examples of regression outcomes include: determining how many transactions a customer is likely to make over a given period, and determining how much money will be deposited to or withdrawn from their account.

7. What are some potential ways this data might be used to achieve outcomes in clustering?

A: Answers will vary. Examples of clustering outcomes include: grouping customers with similar banking behavior together to market different services to different groups, and grouping customers with similar demographics together so that campaigns can be tweaked differently based on the cultural context of which group they are targeting.

8. Why is it important to recognize the role of randomness and uncertainty in data science projects?

A: Answers may vary, but all data science projects have some degree of randomness and uncertainty, as statistical models can only estimate some assumption about the world—they can't make perfectly factual claims. So data science does not provide promises or guarantees, and neither Rudison nor GCNB should be under the impression that it will.

ACTIVITY 2-11: Exploring Data Visualization Tools

2. What types of features do these tools offer? How do they compare to other data visualization tools?

A: Answers will vary based on whatever tool(s) students have chosen. Most of the tools likely offer a wide range of charting types, as well as the ability to read data from multiple input formats. Some may have cloud capabilities, and some may offer rudimentary machine learning functionality.

ACTIVITY 3-1: Examining Data

8. At this point in the process, which of these variables do you think might make good candidates for target features?

A: Answers will vary, as there can be many potential target features, and the dataset may have new features added to it later. Still, numeric features like `total_amount_usd` and `number_transactions` could be good candidates for a prediction model, whereas categorical features like `default` (whether the user has defaulted on a loan) and `term_deposit` (whether or not the user signed up for a term deposit) could be good candidates for a classification model.

ACTIVITY 3-8: Comparing Visual Analysis Methods

1. When it comes to visualizing variable distributions, how would you compare histograms to box plots and violin plots?

A: Answers will vary. Histograms are a great way to show the overall frequency or probability of continuous variable distributions, and for determining the overall shape and skew of a variable's distribution. Box plots are better at showing the general spread of values, particularly when comparing quantiles of the data. They are also good at indicating the presence of outliers. Violin plots leverage some of the same advantages of a box plot, but they can also illustrate distribution probabilities through their shapes.

2. When it comes to visualizing the intersection between two or more variables, how would you compare scatter plots to line and area plots?

A: Answers will vary. Scatter plots are good for comparing two or more continuous variables to see if they are correlated, and to what degree they are correlated. Line plots, on the other hand, are typically best when they indicate the trend of some continuous variable over time. Area plots are similar to line plots, but they can also do a better job of emphasizing the overall trend by filling in the area of the chart below the line.

3. How would you compare bar charts to histograms?

A: Answers will vary. Bar charts are much better suited to plotting the frequency of categorical variables. They can also be used to plot a categorical variable against a summary statistic like mean for a continuous variable. Histograms, on the other hand, deal with the distributions of continuous variables. So, each one shows something completely different.

4. How do heatmaps compare to other types of visualizations?

A: Answers will vary. Heatmaps are often used to visualize variable correlations in a matrix format since they show gradations of "heat" (i.e., color) to indicate the strength of those correlations. They are also used to show the changes in magnitude of some continuous variable as it relates to a physical location, like a building or a device. It is therefore the "heat" component that distinguishes them from other forms of visualization, which don't typically show changes in magnitude using color gradients.

ACTIVITY 3-15: Comparing Data Preprocessing Techniques

- When it comes to handling missing values, how do imputation techniques compare to dropping the data entirely or filling the data in with an arbitrary value?

A: Answers will vary. Imputation offers a more mathematically refined approach to filling in missing values, but it's not always feasible based on the type or volume of missing data. Dropping missing data may be the more desired approach in certain circumstances, especially if dropping that data won't have much of an impact on the project's goals. Filling missing data with an arbitrary value, like zeroing all missing continuous variables, might be preferable if the other pieces of data for that record are worth keeping.

- How do feature scaling techniques like normalization, standardization, log, cube root, and Box-Cox compare?

A: Answers will vary. Normalization and standardization are some of the most common approaches as they can help focus the data on the spread of values irrespective of the magnitude of those values. The former is a little more useful when trying to minimize the effect of outliers, whereas the latter is a little more useful when you want to be insensitive to outliers. Log, cube root, and Box-Cox can all reduce the skewness of a variable, though they do so in slightly different ways.

- How do encoding schemes like label encoding, one-hot encoding, and binary encoding compare?

A: Answers will vary. Label encoding is best for encoding ordinal data, as it assigns categories to integers that start at 0 and increase. On the other hand, one-hot encoding is a common choice for encoding non-ordinal categorical data so that it can be used with machine learning algorithms. Binary encoding is used for a similar purpose, but helps to reduce the new feature space when the number of categories is high.

- When it comes to discretizing variables, how does equal-width binning compare to equal-frequency binning?

A: Answers will vary. Equal-width binning ensures that each bin has the same range of values, which makes it sensitive to outliers and can lead to information loss. Equal-frequency binning ensures the same number of values are in each bin, which helps reduce information loss and is insensitive to outliers.

ACTIVITY 4-1: Identifying Machine Learning Concepts

- How might machine learning go above and beyond the ETL and analysis you've done so far to better fulfill the project's goals?

A: Answers may vary, as machine learning can have many potential applications to this particular project. For example, it can be used to make classification decisions about whether or not a user is likely to sign up for a term deposit given many different features, some of which have higher correlation than others. It can also be used to make predictions, like predicting a user's total transaction amount given many different features. Machine learning might also help GCNB identify clusters of similar users that they can target in future marketing campaigns. Essentially, machine learning can estimate many different things about GCNB's business, as well as its users.

2. What factors do you need to consider when selecting machine learning algorithms to fulfill the project's goals?

A: Answers may vary, as there are many factors that go into the decision to choose one or more algorithms. Those factors include things like the speed of training, how effective the resulting model will be, what kind of additional work is required to prepare the data, how complex the model will be and whether or not that will limit its interpretability, and so on.

3. How might you ensure that your model is "skillful"; i.e., effective at performing its given task?

A: Answers may vary, but one of the most common techniques for improving model skill is iterative tuning. By continually adjusting the model and evaluating those adjustments, it's a good bet that the model will at least become more skillful over time.

4. How might this influence the model training process?

A: Answers may vary. Machine learning tasks use up a lot of processing cycles and other hardware resources. So, it may be necessary to compromise on a model's skill if Rudison's resources aren't adequate enough to push the model further. Time is also a related factor, as limited resources can significantly slow down the training process. So, repeatedly training a model to optimize its performance might be out of the question if your deadlines are tight.

5. Why is a model's ability to generalize to new data so important?

A: A model that cannot generalize well to new data only learns patterns that apply to the specific data it was trained on, and therefore cannot make useful estimations on new data that it hasn't yet seen. Since the goal of most models is to estimate some aspect of the world that cannot easily be explained by humans, models that fail to generalize also fail at accomplishing their goals. On the other hand, models that excel at generalizing to new data are able to effectively answer questions when given some previously unknown input.

6. How does cross-validation help improve a model's ability to generalize?

A: Answers may vary, as it depends on the specific technique used. But, more broadly, all cross-validation techniques divide the input data into two or three different sets, where one set is used to train the model and the others are used to validate and test the model. The model rotates these different sets so that, eventually, all data points have a chance to be used to both train and validate the model. This lowers the model's variance, which minimizes overfitting, and therefore improves generalization.

ACTIVITY 4–2: Testing a Hypothesis

1. In the example given, what is the specific null hypothesis? What does this null hypothesis suggest?

A: The null hypothesis for this example would be that the model's decisions result in less than 10% of all bank customers signing up for a term deposit. The null hypothesis is suggesting that the alternative hypothesis (the model) is no better at accomplishing this objective than random chance.

2. What factors might be involved in this A/B test, and how does the test relate to machine learning?

A: Answers may vary, but an A/B test for this scenario would likely involve sending out two different emails to two different randomly chosen samples of GCNB's contact list. Emails sent to Group A will be slightly different than emails sent to Group B. Perhaps the colors and text formatting are different, or maybe each email will use a different image or put it in a different spot, etc. Whichever email leads to the highest level of engagement will be chosen. But, because there are many factors that could potentially influence users' reaction to an email—both internal to the email itself and external, like the users' demographics—then evaluating what "engagement" looks like might require training a machine learning model that can make that decision.

3. What does this suggest about the null hypothesis and your alternative hypothesis?

A: This suggests that you have a level of confidence with which to reject the null hypothesis. In other words, you have confidence that the model performs better than random chance.

4. What does this suggest about how the mean might appear within these confidence intervals?

A: It suggests that 95% of confidence intervals taken from random samplings of this population will contain the true mean value of the population, whereas 5% will not. It does *not* suggest that there is a 95% probability of the population mean being between 145.79 and 190.81 seconds, or between 143.12 and 189.90 seconds.

5. Why is testing a hypothesis through a design of experiments approach so important to the data science process?

A: Answers may vary, but as the name implies, DOE enables you to experiment with your models in order to effectively identify and produce the best model for your needs. Testing a hypothesis can increase the amount of confidence you have that a model is indeed achieving the project's goals and providing value to the business.

ACTIVITY 8-1: Communicating Results to Stakeholders

1. How might you approach these audiences differently when presenting your results?

A: Answers will vary. The GCNB managers will be most interested in how your findings will result in improvements to business processes, or how they directly impact aspects like revenue, customer retention, brand recognition, etc. Therefore your report should focus mostly on how the project has developed solutions that affect these business aspects. The style of the report should be relatively high level, as you don't want to bog this audience down with overly technical details. When presenting results to GCNB's customers, you'll probably want to work with GCNB management to determine the best way to disseminate your findings. It may be best to display your report on GCNB's website, where users can easily access it. You may also want to issue a press release to reach a wider audience. As far as the tone of the report, customers are most interested in knowing how things like data science and AI will affect their relationship with the bank as well as their personal financial information. The report should therefore focus on the benefits that the project provides to GCNB's user base, without getting too much into the business aspects. The report should also be generalized and not include too many technical details that users aren't interested in.

2. How do these classification models address GCNB's business goals?

A: Answers will vary. The classification models can help the business determine which customers are most likely to sign up for a key service that the bank offers, so that their marketing campaigns can focus on those customers. Likewise, these models also help the business understand which factors are most important for determining who will sign up for the service or who will not. The classification models can be applied to many other services so that the bank has a more comprehensive approach to generating customer engagement.

3. How do these regression models address GCNB's business goals?

A: Answers will vary. The regression models can help GCNB predict the balance of debit and credit that a customer will have with the bank given various factors. So, they might want to engage specific marketing tactics toward customers who meet certain thresholds, like customers who have a debit of \$100 or more.

4. How do these clustering models address GCNB's business goals?

A: Answers will vary. The clustering models are also useful for directing the bank's marketing campaigns because they can lead to efficient segmentation of the customer base. GCNB should be better able to identify and serve the major types of customers it has relationships with. It can also encourage the marketing campaigns to target demographic groups that are currently underrepresented in the overall customer base.

5. What are some key elements that you should include in this written report?

A: Answers may vary, but key components of the report will usually include: an executive summary of the project and its goals; the nature of the data you worked with; the initial hypotheses you generated as a result of analyzing the data; how your hypotheses evolved over the project's lifecycle and became working machine learning models; and the business implications of these models.

6. How might you incorporate visuals into your presentation?

A: Answers will vary. The ultimate purpose of incorporating visuals into a presentation is to make your findings easier for your audience to understand. It can also make the presentation more interesting and lively. So, you'll probably want to incorporate different kinds of charts. These charts should be simplified, yet still informative. For example, you might show a bar chart of how each customer segment generated by a clustering model differs based on their demographic information. Perhaps each cluster's average age will be plotted as a bar, so that the audience can easily identify what age groups are most associated with what clusters. You can also demonstrate charts that have a more direct impact on business decisions, like a cumulative gains and/or lift chart. Perhaps you can show the lift that your classification model has when it comes to publishing marketing materials for customers likely to sign up for a term deposit. So, the business leaders can determine for themselves what percentage of customers they want to target. Aside from individual charts, you might also want to give your audience access to a dashboard that can show key information, including visuals, on a single page.

7. Based on the conclusions you've drawn from the project, what business actions would you suggest that GCNB take?

A: Answers will vary. You could encourage GCNB to start developing solutions that incorporate these models, like a platform that can ingest new customer data and automatically place that customer into one of the pre-defined clusters (i.e., market segments). Therefore, the new customer will automatically be targeted by the appropriate campaign, without the marketing team needing to do so manually. Another suggestion might be to encourage GCNB to collect more data about their customers' banking habits, particularly when it comes to the many different types of services the bank offers. That way new models can be generated to classify customers based on these habits, or predict future monetary values.

Do Not Duplicate or Distribute

Do Not Duplicate Or Distribute

Glossary

A/B test

A type of hypothesis test that compares two different values of the same variable in order to determine which value is most effective.

accuracy

A measure of how frequently each classification is correctly deemed positive or negative.

AI

(artificial intelligence) The ability of machines to exhibit human-like intelligence, as well as the scientific discipline concerned with this idea.

algorithm

A set of rules that defines how a problem-solving operation is performed.

Amazon SageMaker

A cloud-based data science and machine learning automation service that is part of the Amazon Web Services (AWS) cloud platform.

ANOVA

(analysis of variance) A type of hypothesis test that compares the mean of three or more distributions.

API

(application programming interface) A service that facilitates interaction between multiple software environments.

area plot

A type of line plot in which the space below the line is filled in with some color or texture.

ARIMA

(autoregressive integrated moving average) A common algorithm for performing univariate time series forecasting.

arithmetic mean

The average of all numbers in a set.

attribute

See [feature](#).

attrition bias

A type of bias that occurs when the training data excludes participants that dropped out over time.

AUC

(area under curve) The total space that is under a learning model's ROC curve.

Azure Machine Learning

A cloud-based data science and machine learning automation service offered as part of the Microsoft Azure cloud platform.

bag-of-words

An approach to representing textual content as a list of individual words, irrespective of other language components like grammar and punctuation.

bagging

(bootstrap aggregating) An ensemble learning technique for data sampling with replacement.

bar chart

A type of plot that represents the proportion measurement of categorical variables by using either horizontal or vertical bars.

Bayesian optimization

A hyperparameter optimization method that determines the next optimal hyperparameter space to sample from by using past samples to influence where sampling is conducted in subsequent iterations.

BCSS

(between-cluster sum of squares) A clustering model evaluation metric that measures the separation between clusters.

Bessel's correction

A value of 1 subtracted from the number of items in a sample set in order to compensate for the lack of information that would be provided by the larger population.

bias

In machine learning, a type of error that occurs when a model's estimations are different than the ground truth.

big data

Collections of data that are so large and complex that they require advanced tools to process and analyze them.

bimodal

See [multimodal](#).

binary classification

A type of classification task that categorizes data as either a 1 or 0 (i.e., there are only two choices).

box plot

A type of plot that represents the distribution of a numeric variable using

summary statistics like quartiles and minimum and maximum.

Box-Cox

A transformation function that obtains a normal distribution of data using log and power transformations.

CART

(classification and regression tree) A machine learning decision tree algorithm that uses the Gini index for data splitting to solve classification or regression problems.

CCPA

(California Consumer Privacy Act) A law that protects the data privacy and access rights of California citizens.

central tendency

A measure such as the mean, median, and mode intended to identify the typical value in a dataset.

centroid

In a clustering model, the mean (average) of all of the data points that the cluster contains, across all features.

chi-squared test

A type of hypothesis test that compares the effect of categorical variables.

classification

A type of data science task in which a data example is placed into one or more categories.

clustering

A type of data science task that places data examples into groups based on their similarities.

coefficient of determination

A statistical measure that indicates how much of a dependent variable's variance is explainable by a statistical model.

collinearity

See [multicollinearity](#).

concept drift

See [model drift](#).

confidence interval

A measurement that returns a range of plausible values for some unknown variable, like the population mean.

confusion matrix

A method of visualizing the truth results of a classification problem.

continuous variable

A quantitative variable whose values are uncountable and can extend infinitely.

correlation

A mathematical association between two variables. When variables correlate, it suggests that each variable has some kind of effect on the other.

cost function

A function that attempts to quantify the error between the estimated values and the actual labeled training values.

cross-validation

A set of methods for partitioning data so that a model is able to generalize to new test data.

CSS

(Cascading Style Sheets) A set of rules that define the appearance of web content.

cumulative gains chart

A method of plotting the percentage of some target number of examples in a given class against a percentage of the total number of examples.

dashboard

A high-level visualization tool that summarizes the status of an entire project or one or more parts of a project.

data binning

The process of discretizing a continuous variable by placing its values within specific intervals.

data cleaning

The process of locating and addressing errors and inconsistencies in data.

data encoding

The process of converting data of a certain type into a coded value of a different type.

data governance

A concept in which stakeholders ensure that those who govern data are fulfilling objectives and strategies and creating value for the business.

data munging

See [data wrangling](#).

data parsing

The process of taking data as input and then representing that data in a certain structure or syntax.

data preparation

The process of altering data so that it more effectively supports tasks like data analysis and modeling.

data preprocessing

The task of applying various transformation and encoding techniques to data so that it can be interpreted and analyzed by a machine learning algorithm.

data science

The discipline that involves accumulating data, analyzing the data, extracting value from the data, and presenting the value of the data in a meaningful way.

data wrangling

The process of transforming data into a usable form.

Datadog

A general-purpose cloud service monitoring platform.

dataset

A collection of data that will be directly used to accomplish the business goals set forth in the project specifications.

decision boundary

The division line that separates negative classes and positive classes in a classification problem.

decision tree

An arrangement of conditional statements and their conclusions in a branch–leaf structure.

deduplication

The process of identifying and removing duplicate entries from a dataset.

deep learning

A type of machine learning that makes complex decisions using multiple layers of information.

deliverable

A tangible, measurable result or outcome required to complete a project or portion of a project.

dendrogram

A diagram that represents a tree-like hierarchy, commonly used to visualize hierarchical clustering tasks.

dependent variable

In an experiment, the variable that is being studied and that is affected by one or more independent variables.

descriptive statistics

A type of data analysis that quantitatively summarizes the patterns and relationships in a dataset using various mathematical calculations and visualizations. May also refer to an individual calculation that is part of this analysis.

design thinking

An approach to generating business ideas that focus on human needs and innovation.

DevOps

An IT approach in which software development practices help automate systems operations.

dimension

The number of features used in a model.

dimensionality reduction

A type of data science task that minimizes irrelevant or unnecessary elements from a dataset in order to improve the data science process.

discrete variable

A quantitative variable whose values are countable and limited, because there is a definite gap between each value in a range of values.

discretization

The process of converting a continuous variable into a discrete variable.

Django

A fully featured web framework for Python that offers robust tools and feature versatility out of the box.

Docker

An open source platform for building and maintaining virtual containers.

DOE

(design of experiments) An approach to identifying, analyzing, and controlling variables used in an experiment. Also referred to as experimental design or DOX.

EDA

(exploratory data analysis) A data science approach to closely examining data in order to reveal new information and insights.

elastic net regression

A regularization technique that uses a weighted average of both ridge regression and lasso regression when training a model.

elbow point

In clustering, the point at which the mean distance between each data example and its associated centroid no longer decreases in a significant way.

embedding

The process of condensing a language vocabulary into vectors of relatively small dimensions.

ensemble learning

An application of machine learning in which the estimations of multiple models are considered in combination.

error

Incorrect or missing values in data.

ETL

(extract, transform, load) The process of combining data from multiple sources, preparing the data, and loading the resulting data into a destination format.

evaluation metric

A method of assessing the skill, performance, and characteristics of a model based on a specific measurement.

experimental design

See [DOE](#).

F₁ score

The weighted average (harmonic mean) of both precision and recall.

feature

In the field of data science, a measurable property of an example in a training set.

feature engineering

The technique of generating and extracting features from data in order to improve the ability for a machine learning model to make estimations.

feature extraction

A type of dimensionality reduction in which you derive new features from the original features.

feature selection

A type of dimensionality reduction in which you select a subset of the original features.

Flask

A lightweight web framework for Python that emphasizes simplicity and modularity.

forecasting

A task that involves making predictions about future events based on the analysis of relevant past events.

FPR

(false positive rate) A measure of how frequently the learning model incorrectly classified positive values.

frequency distribution

A type of distribution that demonstrates the frequency of outcomes for a particular sample of a random variable.

full outer join

A method of merging two relational database tables in which all rows from a combination of both tables are returned.

Gaussian

Having the shape of a normal curve or a normal distribution.

GDPR

(General Data Protection Regulation) A European Union regulation that regulates the export of EU citizens' personal data for entities that collect or process this data, even if said entities are not based in the EU.

generalization

A model's ability to adapt properly to new, previously unseen data.

geographical map

A type of plot that visually represents data points as they relate to a location.

Gini index

A decision tree splitting metric that splits trees based on the "purity" of decision nodes by squaring each feature's class probability.

global interpretability

A method of measuring the overall decision-making processes of a model.

Goodhart's law

A principle that states: "When a measure becomes a target, it ceases to be a good measure." Used as a reminder not to rely too heavily on one or a small number of metrics when evaluating machine learning model performance.

gradient boosting

An iterative ensemble learning method that builds multiple decision trees in succession, where each tree attempts to reduce the errors of the previous tree.

gradient descent

A method of minimizing a cost function in which a model's parameters are tuned over several iterations by taking gradual "steps" down a slope, toward a minimum error value.

grid search

A hyperparameter optimization method that takes a set (or grid) of parameter combinations, trains a model using each of those combinations, and then returns the combination that best optimizes a specified evaluation metric.

HAC

(hierarchical agglomerative clustering) A type of clustering algorithm that initializes each data example in its own cluster, then gradually merges the closest examples and clusters.

hard-margin classification

A type of classification in SVMs where all data examples are outside of the margins, and each example is on the "correct" side of the margins.

HDC

(hierarchical derivative clustering) A type of clustering algorithm that initializes all data examples in a single cluster, then gradually splits the data into more and more clusters.

heatmap

A type of plot that shows different shades or intensities of color on a matrix based on data values in that location of the matrix.

HIPAA

(Health Insurance Portability and Accountability Act) A law enacted in 1996 to establish several rules and regulations regarding healthcare in the United States.

histogram

A type of plot that represents the probability distribution of a given variable using bins.

holdout

A cross-validation method in which the dataset is split into two: the training dataset and the test dataset.

HTML

(Hypertext Markup Language) The primary markup language used to develop web page content.

hyperparameter

A parameter that is external to a machine learning model; i.e., set on the algorithm itself and not the learning model.

hyperparameter optimization

The process of repeatedly altering the hyperparameters that an algorithm uses to train a model in order to determine the set of hyperparameters that lead to the best or the desired level of model performance.

hyperplane

In SVMs, a decision boundary that has parallel and equidistant lines or curves on either side of the boundary.

hypothesis

A candidate machine learning model that you create to test its performance, particularly whether it is able to produce the outcome that you require.

identity matrix

A matrix of all zeros except for the main diagonal, which consists of all 1s.

imputation

The process of filling in missing data values that consists of using statistical calculations to determine what the missing values should be.

independent variable

In an experiment, a variable that can have an effect on the dependent variable.

inner join

A method of merging two relational database tables in which only rows that match from both tables are returned.

IQR

(interquartile range) The middle half of data values in a distribution.

irreducible error

Errors that cannot be reduced any further when fitting a machine learning model, due to the way the problem was framed, and caused by factors such as unused or unknown features that would have an effect on the output had they been used.

JavaScript

A scripting language that provides interactive functionality to web apps.

k-fold cross-validation

A cross-validation method in which the dataset is split into k groups (folds). One group is the test set, and the remaining groups form the training set.

k-means clustering

A type of clustering algorithm that iteratively updates cluster centroids based on the mean value of each data example in the centroid's cluster.

k-NN

(k -nearest neighbor) An algorithm commonly used to classify data examples based on their similarities to other data examples within the feature space.

KPI

(key performance indicator) A metric used to evaluate the success of a project or its activities.

Kubeflow

A component of Kubernetes that orchestrates data science and machine learning pipelines.

Kubernetes

An open source platform for orchestrating the deployment and management of virtual containers.

kurtosis

A measure of the shape of the tails of a distribution, representing the combined weight of the tails relative to the center of the distribution.

label

In the field of machine learning, the variable in a training set that you are trying to predict for new samples of data.

lasso regression

A regularization technique that uses an ℓ_1 norm to reduce irrelevant features to 0 when training a model.

LCA

(latent class analysis) A form of unsupervised learning that groups data examples together into unobservable groups called latent classes.

learning curve

A method of visually comparing the change in a model's score or error to the number of data examples used as input.

learning rate

In gradient descent, the size of each "step" down the slope.

left outer join

A method of merging two relational database tables in which all rows from the first table are returned, as well as column data from the second table for any matching rows.

lemmatization

The process of using language morphology to determine the base dictionary form of an inflected word.

leptokurtic

Used to describe a distribution curve that is bunched toward the center, with heavy tails on the right and left sides.

lift chart

The ratio of a percentage of examples in a given class to a baseline, plotted against a percentage of the total number of examples.

line plot

A variant of a scatter plot in which a series of lines connects data points in order.

linear regression

A type of regression analysis in which there is a linear relationship between one independent variable and one dependent variable.

local interpretability

A measure of the decision-making processes in a model as applied to specific data examples.

logistic function

The value between 0 and 1 that a logistic regression algorithm outputs, taking an *S* shape.

logistic regression

A type of regression analysis in which the output is a classification probability between 0 and 1.

LOOCV

(leave-one-out cross-validation) A leave-*p*-out cross-validation method in which *p* is set to 1 to minimize performance issues.

LPOCV

(leave-*p*-out cross-validation) A *k*-fold cross-validation method in which *k* (folds) is equal to all data points in the dataset (*n*), with *n* – *p* being the training set and *p* being the test set.

machine learning

An AI discipline in which a machine is able to gradually improve its estimative capabilities without being given explicit instructions.

machine learning model

A specific implementation of an algorithm that is used to generate predictions and other decision-making outcomes based on some training data.

MAE

(mean absolute error) A cost function that calculates the average difference between estimated and actual values without considering the sign of those values.

mesokurtic

A distribution with average or normal shaped tails on the right and left.

milestone

An event during a project that triggers a reporting requirement or that requires approval from stakeholders before proceeding with the project.

MLflow

An open source data automation platform that is part of the Linux Foundation.

model drift

A process through which the patterns initially used to train a machine learning model change over time such that the model no longer performs well with new data.

model parameter

A parameter that is internal to a machine learning model; i.e., derived from the model as it undergoes the training process.

moment

A set of four statistical parameters commonly used to measure a distribution, including mean, variance, skewness, and kurtosis.

MSE

(mean squared error) A cost function that squares the error between estimated and actual values, then calculates the average of all squares.

multi-class classification

A classification problem in which a data example can be placed into one of three or more classes.

multi-label classification

A classification problem in which a data example can be given multiple labels.

multicollinearity

The property that describes multiple variables as exhibiting a linear relationship.

multimodal

A distribution with more than one peak, or mode.

multinomial logistic regression

An algorithm commonly used to solve multi-class classification problems.

MVP

(minimum viable product) A version of the product or service that is, at the bare minimum, usable by early adopters.

naïve Bayes

A type of classification algorithm that computes classification probabilities using Bayes' theorem.

noise

Irrelevant or irregular data values, examples, or features that make it difficult to "hear" patterns revealed by other data that is actually relevant.

non-parametric

A description of a machine learning algorithm that indicates the algorithm can generate a potentially infinite number of model parameters.

normal distribution

A function that represents the distribution of a random variable as a symmetrical bell-shaped graph.

normal equation

A closed-form solution to linear regression problems.

normalization

A technique in which features are scaled so that the lowest value is 0 and the highest value is 1.

NoSQL

Any database technology that does not represent data as relational tables.

null hypothesis

The assumption that there is no statistically significant difference between models under comparison.

ordinal data

Data that can be placed in an order.

outlier

A value outside the main distribution, deviating significantly from the rest of the values in the dataset.

overfitting

A problem in machine learning in which a model's estimations fit well to the training data, but fail to generalize well to other data. An overfit model exhibits high variance and low bias.

p-value

The probability of obtaining a result from the test given that the null hypothesis is true.

parametric

A description of a machine learning algorithm that indicates the algorithm generates a fixed number of model parameters.

PCC

(Pearson correlation coefficient) provides a measure of the linear correlation between two variables commonly called x and y . It produces a value between +1 and -1 that shows the strength of their dependence on each other.

PCI DSS

(Payment Card Industry Data Security Standard) A proprietary standard that specifies how organizations should handle information security for major card brands to increase controls on cardholder data and reduce fraudulent use of accounts.

PII

(personally identifiable information) Data that must be protected to ensure the privacy of the people described by that data.

pipeline

A sequential set of processes that automate the data science process by feeding the output of one process into the input of the next process.

platykurtic

Used to describe a distribution curve that is flat, with light tails on the right and left sides.

POC

(proof of concept) Evidence that supports the feasibility of the product or service that the project is meant to create.

precision

A measure of how often the positives identified by the learning model are true positives.

probability distribution

A type of distribution that demonstrates the probability of outcomes for a random variable.

problem formulation

The process of identifying an issue that should be addressed, and putting that issue in terms that are understandable and actionable.

pruning

The process of reducing the overall size of a decision tree by eliminating nodes, branches, and leaves that provide little value for the classification or regression problem at hand.

qualitative data

Data that holds categorical values.

quantitative data

Data that holds numerical values that represent magnitude.

 R^2

See [coefficient of determination](#).

random forest

An ensemble learning method that aggregates multiple decision tree models together and selects the optimal classifier or predictor.

randomized search

A hyperparameter optimization method that takes a distribution of parameter combinations, trains a model using a random sampling of those combinations, and then returns the combination that best optimizes a specified evaluation metric.

recall

A measure of the percentage of positive instances that are found by a machine learning model as compared to all relevant instances.

regression

A type of data science task that measures the relationship between variables and outputs an estimation for a numeric variable.

regular expression

A group of characters that describe how to execute a specific search pattern on a given text.

regularization

The technique of simplifying a machine learning model by constraining the model

parameters, which helps the model avoid overfitting to the training data.

reporting bias

A type of bias that occurs when the training data is missing observations that were not reported.

ridge regression

A regularization technique that uses an ℓ_2 norm to constrain features used to train a model.

right outer join

A method of merging two relational database tables in which all rows from the second table are returned, as well as column data from the first table for any matching rows.

RMSE

(root mean squared error) The square root of the [MSE](#).

ROC curve

(receiver operating characteristic curve) A method of plotting the relationship between estimated hits (true positive rate) versus false alarms (false positive rate).

sample set

A smaller group than the entire population.

scatter plot

A type of plot that represents the relationship between two variables through the use of points on a graph.

scope

In project management, an outline of all aspects of a project, including any constraints.

scope creep

The condition by which a project continues to grow beyond its ability to be sustained or meet expectations.

selection bias

A type of bias that occurs when the training dataset doesn't truly represent the population the model will ultimately be applied to.

semi-structured data

Data that is in a format that facilitates searching, filtering, or extracting some elements of that data, whereas other elements are not so easy to work with.

sensitivity

See [recall](#).

silhouette analysis

A method of calculating how well a particular data example fits within a cluster as compared to its neighboring clusters.

skewness

The property of a distribution that has a high density of values distributed toward the lower or higher end of the x-axis.

skillful

Used to describe a model that is useful for its intended task. There are degrees of skill; some models are more useful than others. Improving a model's skill is the ultimate goal of the iterative tuning process.

soft-margin classification

An approach to classification with SVMs that keeps the distance between the margins as large as possible while minimizing the number of examples that end up inside the margins.

specificity

A measure of how frequently a machine learning model correctly identifies all actual negative instances.

SQL

(Structured Query Language) A language for programmatically creating, retrieving, modifying, and deleting data in a relational database.

stakeholder

A person who has a vested interest in the outcome of a project or who is actively involved in its work.

standard deviation

A measure of variability; the square root of variance.

standardization

A technique in which features are scaled so that the mean value is 0 and the standard deviation is 1.

statistical model

A mathematical system that generates assumptions about data using statistical methods and probability.

stemming

The process of removing the affix of a word in order to retrieve the word stem.

stochastic

The property by which a randomly determined process cannot perfectly estimate individual events or data points, but can demonstrate a general pattern common to the entire set of data.

stop word

A word in a text document that is so common it is typically removed when the text is processed.

stratified k-fold cross-validation

A k -fold cross-validation method in which each fold has a representative sample of data in datasets that exhibit class imbalance.

stratified random sampling

A statistical sampling method in which the population is divided into groups (strata) according to chosen characteristics (e.g., demographics), then members of each group are randomly sampled, and those samples are combined to form the ultimate sample.

structured data

Data that is in a format that facilitates searching, filtering, or extracting that data.

summary statistics

See [descriptive statistics](#).

supervised learning

A type of machine learning in which known label values are provided as input so that a model can estimate these values in future datasets.

SVMs

(support-vector machines) Supervised learning algorithms that can be used to solve classification and regression problems by separating data values using a hyperplane.

t-test

A type of hypothesis test that compares the mean of two distributions in which the population standard deviation is not known.

target feature

A variable that the data science practitioner is interested in learning more about.

target function

A representation of the mapping between input variables and output variables that best approximates some desired outcome from a machine learning model.

threshold

A value used by a classification model to classify anything higher than the threshold as positive, and anything lower than the threshold as negative.

time series

A representation of data in which observations are ordered according to a sequential change in time.

TNR

(true negative rate) See *specificity*.

tokenization

The process of partitioning text into smaller units.

TPR

(true positive rate) See *recall*.

training

In the field of machine learning, the process by which a model learns from input data.

training sample

A dataset of examples used to generate a machine learning model.

underfitting

A problem in machine learning in which a model cannot make effective estimations due

to an inability to identify the underlying patterns in the data. An underfit model exhibits low variance and high bias.

unimodal

A distribution with one peak, or mode.

unstructured data

Data that is in a format that makes it difficult to search, filter, or extract that data.

unsupervised learning

A type of machine learning in which label values are not provided as input, so the model does not have an explicit variable that it is estimating.

variability

The property that indicates the extent to which data varies across all values in a dataset.

variance

A measurement of the spread between numbers in a dataset, or the variation of a model's estimations across datasets.

violin plot

A type of plot that shows the distribution of a numerical value through probability density.

WCSS

(within-cluster sum of squares) A clustering model evaluation metric that measures the compactness of clusters.

web app

An application that runs on a server using a web-based protocol, particularly Hypertext Transfer Protocol (HTTP) and its secure equivalent, HTTPS.

web framework

A programming library that supports the development of web apps and other web-based services through a standardized interface.

z-score

The number of standard deviations that a sample is above or below the mean of all values in the sample.

z-test

A type of hypothesis test that compares the mean of two distributions when the standard deviation of a population is known.

Do Not Duplicate or Distribute

Do Not Duplicate Or Distribute

Index

A

accuracy 307
AI 226
algorithms
 non-parametric 235
 parametric 235
Amazon SageMaker 444
analysis of variance test, *See* ANOVA test
ANOVA test 243
area plots 151
area under curve, *See* AUC
ARIMA 353
arithmetic mean 126
artificial intelligence, *See* AI
attributes 29
AUC 314
autoregressive integrated moving average,
See ARIMA
Azure Machine Learning 445

B

bagging 287
bar charts 155
Bayesian optimization 299
BCSS 401
Bessel's correction 128
between-cluster sum of squares, *See* BCSS
big data 30
bimodal 124
binary classification 252
Box-Cox 188
box plots 143

C

CARTs
 hyperparameters 285
 overview 281
cascading style sheets, *See* CSS
categorical data 58
Chi-squared test 243
classification and regression trees, *See*
CARTs
coefficient of determination 368
 See also R²
collinearity 359
concept drift 441
 See also model drift
confidence interval 244
confusion matrix 305
continuous variables 59
correlation 113
cost function 357
cross-validation
 defined 234
 k-fold 234
 leave-one-out 234
 leave-p-out 234
 stratified k-fold 234
CSS 429
cumulative gains chart 420

D

dashboard 419
data binning 201
data cleaning 57
data encoding 191
data join

inner join example 51
 types 50
 data munging 57
 data preparation 57
 data preprocessing 174
 data science
 overview 2
 dataset 28
 data sources 31
 data wrangling 57
 decision boundary 253
 decision trees 280
 deep learning 228
 dendrograms 402
 dependent variables 239
 descriptive statistical analysis 125
 design of experiments, *See* DOE
 DevOps 16
 dimensionality reduction 214
 dimensions 29
 discrete variables 59
 discretization 201
 Django 432
 docker 442
 DOE 239

E

EDA 108
 elbow point 400
 embedding 76
 ensemble learning 286
 errors 112
 ETL 35
 evaluation metrics 304
 experimental design 239
 exploratory data analysis, *See* EDA
 extract, transform, and load, *See* ETL

F

false positive rate, *See* FPR
 feature engineering 191
 feature extraction 214
 features 29
 feature selection 214
 flask 431
 forecasting 352
 FPR 311
 frequency distribution 120
 F1 score 310

G

Gaussian curve 123
 GDPR 9
 General Data Protection Regulation, *See* GDPR
 geographical maps 164
 Gini index 281, 282
 global interpretability 417
 Goodhart's Law 304
 gradient boosting 287
 gradient descent 359
 grid search 298

H

HAC 387, 402
 hard-margin classification 271
 HDC 388, 402
 Health Insurance Portability and Accountability Act, *See* HIPAA
 heatmaps 165
 hierarchical agglomerative clustering, *See* HAC
 hierarchical divisive cluster, *See* HDC
 HIPAA 9
 histograms 140
 holdout 233
 HTML 428
 hyperparameter optimization 297
 hyperparameters 235
 hyperplanes 268
 hypertext markup language, *See* HTML
 hypothesis
 A/B test 241
 testing 240
 testing methods 242

I

identity matrix 335
 imputation 175
 independent variables 239
 interquartile range, *See* IQR
 IQR 126
 irreducible error 232

J

JavaScript 429

K

k-means clustering
 shortcomings 386
k-nearest neighbor, *See* k-NN
k-NN 264
kubernetes 443
kurtosis
 calculation of 133
 types of 132

L

labels 20
latent class analysis, *See* LCA
LCA 393
learning curve 315
learning rate 360
leave-one-out cross-validation, *See* LOOCV
leave-p-out cross-validation, *See* LPOCV
leptokurtic 133
lift chart 422
linear equation
 shortcomings 334
linear regression
 in machine learning 334
 matrices in 335
line plots 150
local interpretability 417
logistic function 252
logistic regression 252
LOOCV 234
LPOCV 234

M

machine learning
 overview 226
machine learning algorithms 228
machine learning model
 defined 227
 training 227
machine learning models
 attrition bias 231
 generalization 231
 overfitting 231
 reporting bias 231
 selection bias 230
 skillful 229
 target function 230
 underfitting 231
MAE 368

mean absolute error, *See* MAE
mean squared error, *See* MSE
measures of central tendency 125
mesokurtic 132
model drift 441
model parameters 234
moments
 statistical 134
MSE 367
multi-class classification 255
multi-label classification 254
multimodal 124
multinomial logistic regression 255

N

naïve Bayes 275
noise 113
non-normal distribution 124
normal distribution 123
normal equation 336
normalization 187
NoSQL 43
null hypothesis 240
numerical data 58

O

ordinal data 58
outliers 112

P

parsing 59
Payment Card Industry Data Security Standard,
See PCI DSS
PCC 113
PCI DSS 9
Pearson correlation coefficient, *See* PCC
pipeline monitoring solutions 446
pipelines 440
platykurtic 133
precision 307
probability distribution 121
problem formulation 16
pruning 285
p-value 243

Q

qualitative data 58
 See also categorical data
quantitative data 58

See also numerical data

R

RAE 368
 random forest 286
 randomized search 298
 range 126
 recall 308
 receiver operating characteristic, *See* ROC
 regular expression 208
 regularization
 overview 357
 techniques 358
 relative absolute error, *See* RAE
 relative squared error, *See* RSE
 representative data
 stratified random sampling 111
 RMSE 367
 ROC 311
 root mean squared error, *See* RMSE
 RSE 368
 R² 368

S

samples 239
 sample set 128
 sampling techniques 111
 scatter plots 149
 scope 3
 scope creep 3
 semi-structured data 28
 sensitivity 312
 silhouette analysis 401
 skew 124
 skewness
 calculation of 132
 soft-margin classification 271
 specificity 310
 SQL 41
 standard deviation
 calculation of 129
 standardization 187
 standard score 187
 See also z-score
 statistical model 19
 stochastic modeling 21
 structured data 28
 Structured Query Language, *See* SQL
 summary statistics 125
 supervised learning 20

SVMs 268

T

target feature 110
 target function 239
 text data transformation 76
 thresholds 312
 time series 352
 TPR 311
 true positive rate, *See* TPR
 t-test 243

U

unimodal 123, 124
 unstructured data 28
 unsupervised learning 20

V

variability 126
 variance
 calculation of 128
 in a sample set 128
 violin plots 144
 visualizations 140

W

WCSS 401
 web app 427
 web framework 430
 within-cluster sum of squares, *See* WCSS

Z

z-score 187
 See also standard score
 z-test 242

Do Not Duplicate Or Distribute

Do Not Duplicate or Distribute

CNX0011S rev 1.0
ISBN-13 978-1-4246-4087-4
ISBN-10 1-4246-4087-3



9 781424 640874