

# PREFACE

Machines have come a long way since the onset of the Industrial Revolution. They continue to fill factory floors and manufacturing plants, but their capabilities extend beyond manual activities to cognitive tasks that, until recently, only humans were capable of performing. Judging song contests, driving automobiles, and detecting fraudulent transactions are three examples of the complex tasks machines are now capable of simulating.

But these remarkable feats trigger fear among some observers. Part of their fear nestles on the neck of survivalist insecurities and provokes the deep-seated question of *what if?* *What if* intelligent machines turn on us in a struggle of the fittest? *What if* intelligent machines produce offspring with capabilities that humans never intended to impart to machines? *What if* the legend of the *singularity* is true?

The other notable fear is the threat to job security, and if you're a taxi driver or an accountant, there's a valid reason to be worried. According to joint research from the Office for National Statistics and Deloitte UK published by the BBC in 2015, job professions including bar worker (77%), waiter (90%), chartered accountant (95%), receptionist (96%), and taxi driver (57%) have a high chance of being automated by the year 2035. <sup>[1]</sup> Nevertheless, research on planned job automation and crystal ball gazing concerning the future evolution of machines and artificial intelligence (AI) should be read with a pinch of skepticism. In *Superintelligence: Paths, Dangers, Strategies*, author Nick Bostrom discusses the continuous redeployment of AI goals and how "two decades is a sweet spot... near enough to be attention-grabbing and relevant, yet far enough to make it possible that a string of breakthroughs...might by then have occurred."<sup>[2]</sup> (<sup>[3]</sup>)

While AI is moving fast, broad adoption remains an unchartered path fraught with known and unforeseen challenges. Delays and other obstacles are inevitable. Nor is machine learning a simple case of flicking a switch and asking the machine to predict the outcome of the Super Bowl and serve you a delicious martini.

Far from a typical out-of-the-box analytics solution, machine learning relies on

statistical algorithms managed and overseen by skilled individuals called data scientists and machine learning engineers. This is one labor market where job opportunities are destined to grow but where supply is struggling to meet demand.

In fact, the current shortage of professionals with the necessary expertise and training is one of the primary obstacles delaying AI's progress. According to Charles Green, the Director of Thought Leadership at Belatrix Software:

*"It's a huge challenge to find data scientists, people with machine learning experience, or people with the skills to analyze and use the data, as well as those who can create the algorithms required for machine learning. Secondly, while the technology is still emerging, there are many ongoing developments. It's clear that AI is a long way from how we might imagine it."* [\[4\]](#)

Perhaps your own path to working in the field of machine learning starts here, or maybe a baseline understanding is sufficient to fulfill your curiosity for now.

This book focuses on the high-level fundamentals, including key terms, general workflow, and the statistical underpinnings of basic algorithms to set you on your path. To design and code intelligent machines, you'll first need to develop a strong grasp of classical statistics. Algorithms derived from classical statistics sit at the core of machine learning and constitute the metaphorical neurons and nerves that power artificial cognitive abilities. Coding is the other indispensable part of machine learning, which includes managing and manipulating large amounts of data. Unlike building a web 2.0 landing page with click-and-drag tools like Wix and WordPress, machine learning requires Python, C++, R or another programming language. If you haven't learned a relevant programming language, you will need to if you wish to make further progress in this field. But for the purpose of this compact starter's course, the following chapters can be completed without any programming experience.

While this book serves as an introductory course to machine learning, please note that it does not constitute an absolute beginner's introduction to mathematics, computer programming, and statistics. A cursory knowledge of these fields or convenient access to an Internet connection may be required to aid understanding in later chapters.

For those who wish to dive into the coding aspect of machine learning, Chapter 17 and Chapter 19 walk you through the entire process of setting up a machine learning model using Python. A gentle introduction to coding with Python has also been included in the Appendix and information regarding further learning resources can be found in the final section of this book.

Lastly, video tutorials and other online materials (included free with this book) can be found at <https://scatterplotpress.teachable.com/p/ml-code-exercises> .

# WHAT IS MACHINE LEARNING?

In 1959, IBM published a paper in the *IBM Journal of Research and Development* with an intriguing and obscure title. Authored by IBM's Arthur Samuel, the paper investigated the application of machine learning in the game of checkers "to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program." [\[5\]](#)

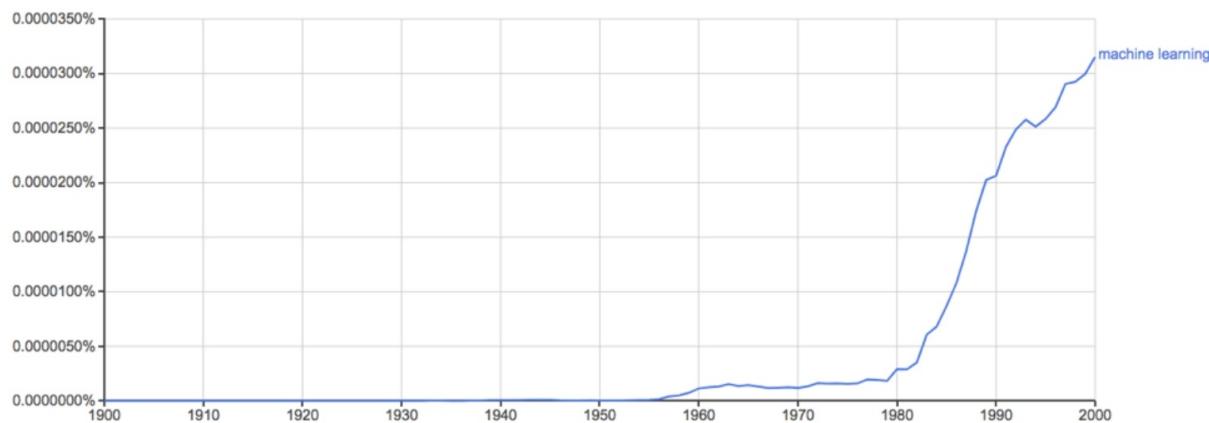


Figure 1: Historical mentions of "machine learning" in published books. Source: Google Ngram Viewer, 2017

Although it wasn't the first published paper to use the term "machine learning" per se, Arthur Samuel is regarded as the first person to coin and define machine learning as the concept and specialized field we know today. Samuel's landmark journal submission, *Some Studies in Machine Learning Using the Game of Checkers*, introduced machine learning as a subfield of computer science that gives computers the ability to learn without being explicitly programmed.

While not directly treated in Arthur Samuel's initial definition, a key characteristic of machine learning is the concept of *self-learning*. This refers to the application of statistical modeling to detect patterns and improve performance based on data and empirical information; all without direct programming commands. This is what Arthur Samuel described as the ability to

learn without being explicitly programmed. Samuel didn't infer that machines may formulate decisions with no upfront programming. On the contrary, machine learning is heavily dependent on code input. Instead, he observed machines can perform a set task using *input data* rather than relying on a direct *input command*.

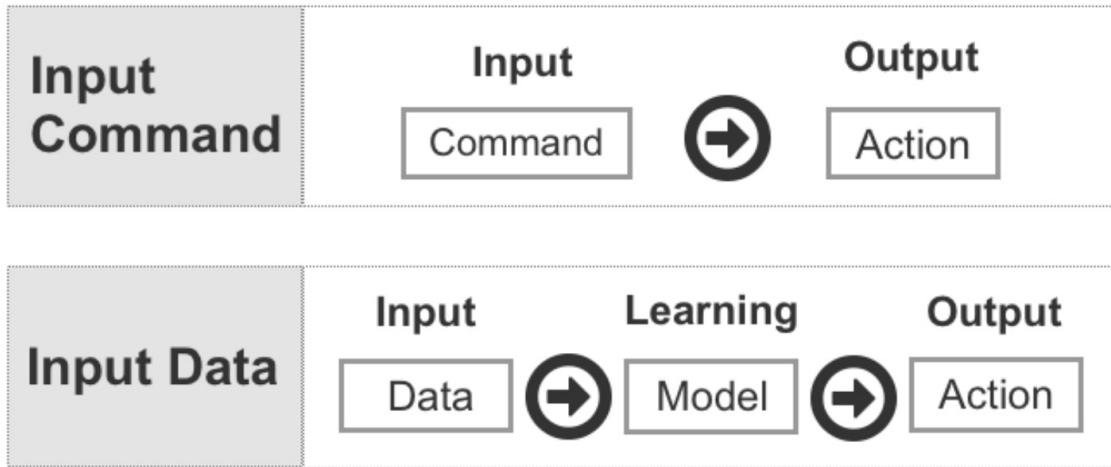


Figure 2: Comparison of Input Command vs Input Data

An example of an input command is entering “2+2” in a programming language such as Python and clicking “Run” or hitting “Enter” to view the output.

```
>>> 2+2
```

```
4
```

```
>>>
```

This represents a direct command with a pre-programmed answer, which is typical of most computer applications. Unlike traditional computer programming, though, where outputs or decisions are pre-defined by the programmer, machine learning uses data as input to build a decision model. Decisions are generated by deciphering relationships and patterns in the data using probabilistic reasoning, trial and error, and other computationally-intensive techniques. This means that the output of the decision model is determined by the contents of the input data rather than any pre-set rules defined by a human programmer. The human programmer is still responsible for feeding the data into the model, selecting an appropriate algorithm and tweaking its settings (called *hyperparameters*) in a bid to reduce prediction error, but ultimately the machine and developer operate a layer apart in contrast to traditional programming.

To draw an example, let's suppose that after analyzing YouTube viewing habits, the decision model identifies a significant relationship among data scientists who like watching cat videos. A separate model, meanwhile, identifies patterns

among the physical traits of baseball players and their likelihood of winning the season's Most Valuable Player (MVP) award.

In the first scenario, the machine analyzes which videos data scientists enjoy watching on YouTube based on user engagement; measured in likes, subscribes, and repeat viewing. In the second scenario, the machine assesses the physical attributes of previous baseball MVPs among other features such as age and education. However, at no stage was the decision model told or programmed to produce those two outcomes. By decoding complex patterns in the input data, the model uses machine learning to find connections without human help. This also means that a related dataset collected from another time period, with fewer or greater data points, might push the model to produce a slightly different output.

Another distinct feature of machine learning is the ability to improve predictions based on experience. Mimicking the way humans base decisions on experience and the success or failure of past attempts, machine learning utilizes exposure to data to improve its decision making. The socializing of data points provides experience and enables the model to familiarize itself with patterns in the data. Conversely, insufficient input data restricts the model's ability to deconstruct underlying patterns in the data and limits its capacity to respond to potential variance and random phenomena found in live data. Exposure to input data thereby deepens the model's understanding of patterns, including the significance of changes in the data, and to construct an effective self-learning model.

A common example of a self-learning model is a system for detecting spam email messages. Following an initial serving of input data, the model learns to flag emails with suspicious subject lines and body text containing keywords that correlate strongly with spam messages flagged by users in the past. Indications of spam email may include words like *dear friend*, *free*, *invoice*, *PayPal*, *Viagra*, *casino*, *payment*, *bankruptcy*, and *winner*. However, as more data is analyzed, the model might also find exceptions and incorrect assumptions that render the model susceptible to bad predictions. If there is limited data to reference its decision, the following email subject, for example, might be wrongly classified as spam: “**PayPal** has received your **payment** for **Casino** Royale purchased on eBay.”

As this is a genuine email sent from a PayPal auto-responder, the spam detection system is lured into producing a false-positive based on previous input data. Traditional programming is highly susceptible to this problem because the model is rigidly defined according to pre-set rules. Machine learning, on the other hand,

emphasizes exposure to data as a way to refine the model, adjust weak assumptions, and respond appropriately to unique data points such as the scenario just described.

While data is used to source the self-learning process, more data doesn't always equate to better decisions; the input data must be relevant. In *Data and Goliath: The Hidden Battles to Collect Your Data and Control Your World*, Bruce Schneir writes that, "When looking for the needle, the last thing you want to do is pile lots more hay on it." <sup>[6]</sup> This means that adding irrelevant data can be counter-productive to achieving a desired result. In addition, the amount of input data should be compatible with the processing resources and time that is available.

## Training & Test Data

In machine learning, the input data is typically split into *training data* and *test data*. The first split of data is the *training data*, which is the initial reserve of data used to develop the model. In the spam email detection example, false-positives similar to the PayPal auto-response message might be detected from the training data. Modifications must then be made to the model, e.g., email notifications issued from the sending address "payments@paypal.com" should be excluded from spam filtering. Using machine learning, the model can be trained to automatically detect these errors (by analyzing historical examples of spam messages and deciphering their patterns) without direct human interference.

After you have developed a model based on patterns extracted from the training data and you are satisfied with the accuracy of its predictions, you can test the model on the remaining data, known as the *test data*. If you are also satisfied with the model's performance using the test data, the model is ready to filter incoming emails in a live setting and generate decisions on how to categorize those messages. We will discuss training and test data further in Chapter 6.

## The Anatomy of Machine Learning

The final section of this chapter explains how machine learning fits into the broader landscape of data science and computer science. This includes understanding how machine learning connects with parent fields and sister disciplines. This is important, as you will encounter related terms in machine learning literature and courses. Relevant disciplines can also be difficult to tell apart, especially machine learning and data mining.

Let's start with a high-level introduction. Machine learning, data mining, artificial intelligence, and computer programming all fall under the umbrella of

computer science, which encompasses everything related to the design and use of computers. Within the all-encompassing space of computer science is the next broad field of data science. Narrower than computer science, data science comprises methods and systems to extract knowledge and insights from data with the aid of computers.

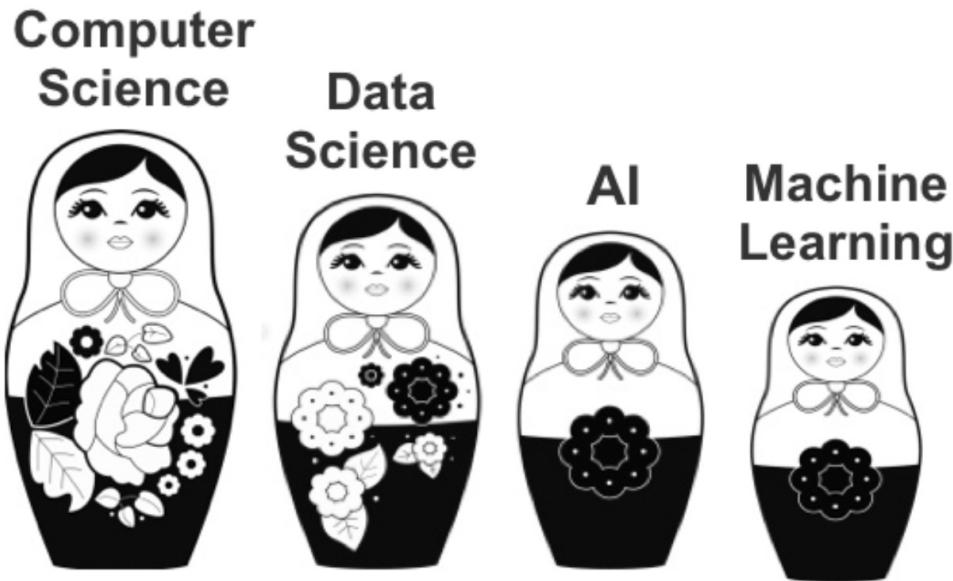


Figure 3: The lineage of machine learning represented by a row of Russian matryoshka dolls

Emerging from computer science and data science as the third matryoshka doll from the left in Figure 3 is artificial intelligence. Artificial intelligence, or AI, encompasses the ability of machines to perform intelligent and cognitive tasks. Comparable to how the Industrial Revolution gave birth to an era of machines simulating physical tasks, AI is driving the development of machines capable of simulating cognitive abilities.

While still broad but dramatically more honed than computer science and data science, AI spans numerous subfields that are popular and newsworthy today. These subfields include search and planning, reasoning and knowledge representation, perception, natural language processing (NLP), and of course, machine learning.

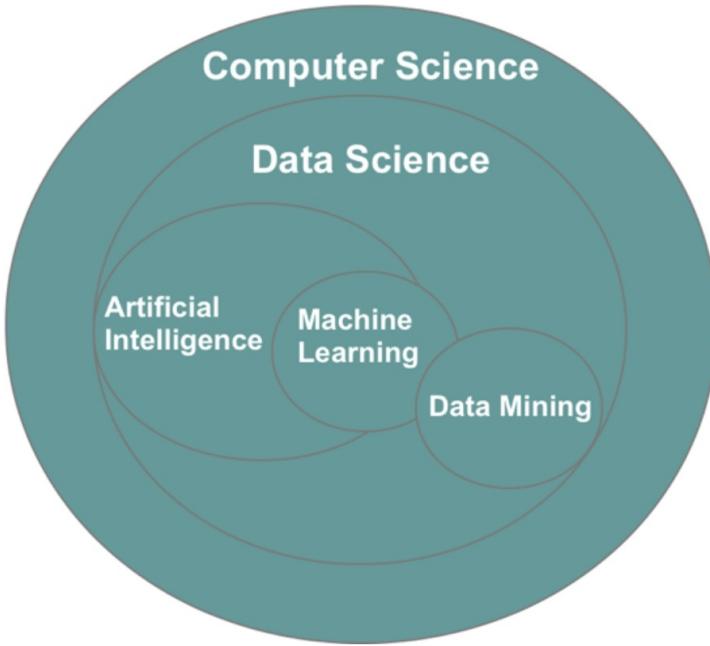


Figure 4: Visual representation of the relationship between data-related fields

For students interested in AI, machine learning provides an excellent starting point as it provides a narrower and more practical lens of study (in comparison to AI). Algorithms applied in machine learning can also be used in other disciplines, including perception and natural language processing. In addition, a Master's degree is adequate to develop a certain level of expertise in machine learning, but you may need a PhD to make genuine progress in the field of artificial intelligence.

As mentioned, machine learning overlaps with data mining—a sister discipline based on discovering and unearthing patterns in large datasets. Both techniques rely on inferential methods, i.e. predicting outcomes based on other outcomes and probabilistic reasoning, and draw from a similar assortment of algorithms including principal component analysis, regression analysis, decision trees, and clustering techniques. To add further confusion, the two techniques are commonly mistaken and misreported or even explicitly misused. The textbook *Data mining: Practical machine learning tools and techniques with Java* is said to have originally been titled *Practical machine learning*, but for marketing reasons “data mining” was later appended to the title. [\[7\]](#)

Lastly, because of their interdisciplinary nature, experts from a diverse spectrum of disciplines often define data mining and machine learning differently. This has led to confusion, in addition to a genuine overlap between the two disciplines. But whereas machine learning emphasizes the incremental process

of self-learning and automatically detecting patterns through experience derived from exposure to data, data mining is a less autonomous technique of extracting hidden insight.

Like randomly drilling a hole into the earth's crust, data mining doesn't begin with a clear hypothesis of what insight it will find. Instead, it seeks out patterns and relationships that are yet to be mined and is, thus, well-suited for understanding large datasets with complex patterns. As noted by the authors of *Data Mining: Concepts and Techniques*, data mining developed as a result of advances in data collection and database management beginning in the early 1980s <sup>[8]</sup> and an urgent need to make sense of progressively larger and complicated datasets. <sup>[9]</sup>

Whereas data mining focuses on **analyzing input variables to predict a new output**, machine learning extends to **analyzing both input and output variables**. This includes supervised learning techniques that compare known combinations of input and output variables to discern patterns and make predictions, and reinforcement learning which randomly trials a massive number of input variables to produce a desired output. Another machine learning technique, called unsupervised learning, generates predictions based on the analysis of input variables with no known target output. This technique is often used in combination or in preparation for supervised learning under the name of *semi-supervised learning*, and although it overlaps with data mining, unsupervised learning tends to deviate from standard data mining methods such as association and sequence analysis.

Technique	Input is Known	Output is Known	Methodology
Data Mining	✓		Analyzes inputs to generate an unknown output.
Supervised Learning	✓	✓	Analyzes combinations of known inputs and outputs to predict future outputs based on new input data.
Unsupervised Learning	✓		Analyzes inputs to generate an output—algorithms may differ from data mining.
Reinforcement Learning		✓	Randomly trials a high number of input variables to produce a desired output.

Table 1: Comparison of techniques based on the utility of input and output data/variables

To consolidate the difference between data mining and machine learning, let's consider an example of two teams of archaeologists. One team has little knowledge of their target excavation site and imparts domain knowledge to optimize their excavation tools to find patterns and remove debris to reveal hidden artifacts. The team's goal is to manually excavate the area, find new

valuable discoveries, and then pack up their equipment and move on. A day later, they fly to another exotic destination to start a new project with no relationship to the site they excavated the day before.

The second team is also in the business of excavating historical sites, but they pursue a different methodology. They refrain from excavating the main pit for several weeks. In this time, they visit other nearby archaeological sites and examine patterns regarding how each archaeological site is constructed. With exposure to each excavation site, they gain experience, thereby improving their ability to interpret patterns and reduce prediction error. When it comes time to excavate the final and most important pit, they execute their understanding and experience of the local terrain to interpret the target site and make predictions.

As is perhaps evident by now, the first team puts their faith in data mining whereas the second team relies on machine learning. While both teams make a living excavating historical sites to discover valuable insight, their goals and methodology are different. The machine learning team invests in self-learning to create a system that uses exposure to data to enhance its capacity to make predictions. The data mining team, meanwhile, concentrates on excavating the target area with a more direct and approximate approach that relies on human intuition rather than self-learning.

We will look more closely at self-learning specific to machine learning in the next chapter and how input and output variables are used to make predictions.

# MACHINE LEARNING CATEGORIES

Machine learning incorporates several hundred statistical-based algorithms and choosing the right algorithm(s) for the job is a constant challenge of working in this field. Before examining specific algorithms, it's important to consolidate one's understanding of the three overarching categories of machine learning and their treatment of input and output variables.

## Supervised Learning

Supervised learning imitates our own ability to extract patterns from known examples and use that extracted insight to engineer a repeatable outcome. This is how the car company Toyota designed their first car prototype. Rather than speculate or create a unique process for manufacturing cars, Toyota created its first vehicle prototype after taking apart a Chevrolet car in the corner of their family-run loom business. By observing the finished car (output) and then pulling apart its individual components (input), Toyota's engineers unlocked the design process kept secret by Chevrolet in America.

This process of understanding a known input-output combination is replicated in machine learning using supervised learning. The model analyzes and deciphers the relationship between input and output data to learn the underlying patterns. Input data is referred to as the independent variable (uppercase "X"), while the output data is called the dependent variable (lowercase "y"). An example of a dependent variable (y) might be the coordinates for a rectangle around a person in a digital photo (face recognition), the price of a house, or the class of an item (i.e. sports car, family car, sedan). Their independent variables—which supposedly impact the dependent variable—could be the pixel colors, the size and location of the house, and the specifications of the car respectively. After analyzing a sufficient number of examples, the machine creates a model: an algorithmic equation for producing an output based on patterns from previous input-output examples.

Using the model, the machine can then predict an output based exclusively on the input data. The market price of your used Lexus, for example, can be

estimated using the labeled examples of other cars recently sold on a used car website.

	<b>Input</b>	<b>Input</b>	<b>Input</b>	<b>Output</b>
	<b>Car Brand</b>	<b>Mileage (km)</b>	<b>Year of Make</b>	<b>Price (USD)</b>
<b>Car 1</b>	Lexus	51715	2012	15985
<b>Car 2</b>	Lexus	7980	2013	19600
<b>Car 3</b>	Lexus	82497	2012	14095
<b>Car 4</b>	Lexus	85199	2011	12490
<b>Car 5</b>	Audi	62948	2008	13985

Table 2: Extract of a used car dataset

With access to the selling price of other similar cars, the supervised learning model can work backward to determine the relationship between a car’s value (output) and its characteristics (input). The input features of your own car can then be inputted into the model to generate a price prediction.

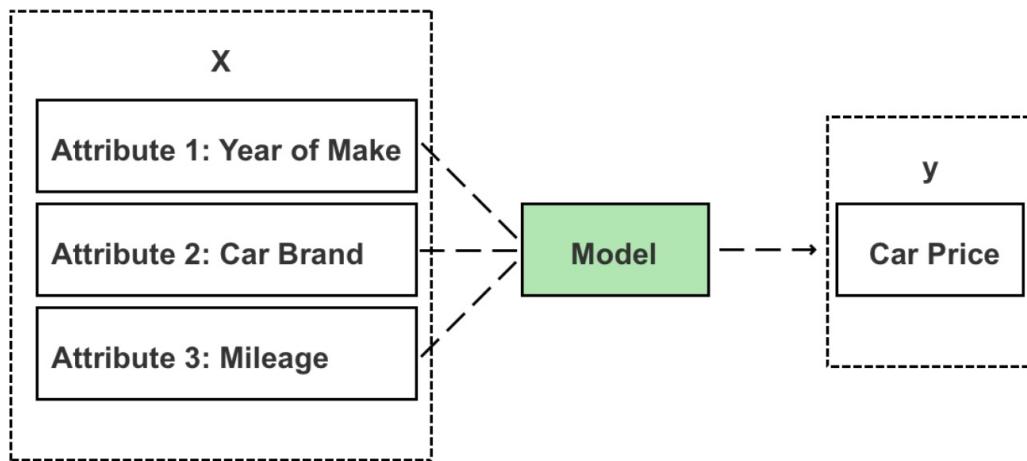


Figure 5: Inputs (X) are fed to the model to generate a new prediction (y)

While input data with an unknown output can be fed to the model to push out a prediction, unlabeled data cannot be used to build the model. When building a supervised learning model, each item (i.e. car, product, customer) must have labeled input and output values—known in data science as a “labeled dataset.” Examples of common algorithms used in supervised learning include regression analysis (i.e. linear regression, logistic regression, non-linear regression), decision trees,  $k$ -nearest neighbors, neural networks, and support vector machines, each of which are examined in later chapters.

## Unsupervised Learning

In the case of unsupervised learning, the output variables are unlabeled, and combinations of input and output variables aren't known. Unsupervised learning instead focuses on analyzing relationships between input variables and uncovering hidden patterns that can be extracted to create new labels regarding possible outputs.

If you group data points based on the purchasing behavior of SME (Small and Medium-sized Enterprises) and large enterprise customers, for example, you're likely to see two clusters of data points emerge. This is because SMEs and large enterprises tend to have different procurement needs. When it comes to purchasing cloud computing infrastructure, for example, essential cloud hosting products and a Content Delivery Network (CDN) should prove sufficient for most SME customers. Large enterprise customers, though, are likely to purchase a broader array of cloud products and complete solutions that include advanced security and networking products like WAF (Web Application Firewall), a dedicated private connection, and VPC (Virtual Private Cloud). By analyzing customer purchasing habits, unsupervised learning is capable of identifying these two groups of customers without specific labels that classify a given company as small/medium or large.

The advantage of unsupervised learning is that it enables you to discover patterns in the data that you were unaware of—such as the presence of two dominant customer types—and provides a springboard for conducting further analysis once new groups are identified. Unsupervised learning is especially compelling in the domain of fraud detection—where the most dangerous attacks are those yet to be classified. One interesting example is DataVisor; a company that has built its business model on unsupervised learning. Founded in 2013 in California, DataVisor protects customers from fraudulent online activities, including spam, fake reviews, fake app installs, and fraudulent transactions. Whereas traditional fraud protection services draw on supervised learning models and rule engines, DataVisor uses unsupervised learning to detect unclassified categories of attacks.

As DataVisor explains on their website, "to detect attacks, existing solutions rely on human experience to create rules or labeled training data to tune models. This means they are unable to detect new attacks that haven't already been identified by humans or labeled in training data." [\[10\]](#) Put another way, traditional solutions analyze chains of activity for a specific type of attack and then create rules to predict and detect repeat attacks. In this case, the dependent variable (output) is the event of an attack, and the independent variables (input) are the common

predictor variables of an attack. Examples of independent variables could be:

**a) A sudden large order from an unknown user.** I.E., established customers might generally spend less than \$100 per order, but a new user spends \$8,000 on one order immediately upon registering an account.

**b) A sudden surge of user ratings.** I.E., As with most technology books sold on Amazon.com, the first edition of this book rarely receives more than one reader review per day. In general, approximately 1 in 200 Amazon readers leave a review and most books go weeks or months without a review. However, I notice other authors in this category (data science) attract 50-100 reviews in a single day! (Unsurprisingly, I also see Amazon remove these suspicious reviews weeks or months later.)

**c) Identical or similar user reviews from different users.** Following the same Amazon analogy, I sometimes see positive reader reviews of my book appear with other books (even with reference to my name as the author still included in the review!). Again, Amazon eventually removes these fake reviews and suspends these accounts for breaking their terms of service.

**d) Suspicious shipping address.** I.E., For small businesses that routinely ship products to local customers, an order from a distant location (where their products aren't advertised) can, in rare cases, be an indicator of fraudulent or malicious activity.

Standalone activities such as a sudden large order or a remote shipping address might not provide sufficient information to detect sophisticated cybercrime and are probably more likely to lead to a series of false-positive results. But a model that monitors combinations of independent variables, such as a large purchasing order from the other side of the globe or a landslide number of book reviews that reuse existing user content generally leads to a better prediction.

In supervised learning, the model deconstructs and classifies what these common variables are and design a detection system to identify and prevent repeat offenses. Sophisticated cybercriminals, though, learn to evade these simple classification-based rule engines by modifying their tactics. Leading up to an attack, for example, the attackers often register and operate single or multiple accounts and incubate these accounts with activities that mimic legitimate users. They then utilize their established account history to evade detection systems, which closely monitor new users. As a result, solutions that use supervised learning often fail to detect sleeper cells until the damage has been inflicted and especially for new types of attacks.

DataVisor and other anti-fraud solution providers instead leverage unsupervised learning techniques to address these limitations. They analyze patterns across

hundreds of millions of accounts and identify suspicious connections between users (input)—without knowing the actual category of future attacks (output). By grouping and identifying malicious actors whose actions deviate from standard user behavior, companies can take actions to prevent new types of attacks (whose outcomes are still unknown and unlabeled).

Examples of suspicious actions may include the four cases listed earlier or new instances of unnormal behavior such as a pool of newly registered users with the same profile picture. By identifying these subtle correlations across users, fraud detection companies like DataVisor can locate sleeper cells in their incubation stage. A swarm of fake Facebook accounts, for example, might be linked as friends and like the same pages but aren't linked with genuine users. As this type of fraudulent behavior relies on fabricated interconnections between accounts, unsupervised learning thereby helps to uncover collaborators and expose criminal rings.

The drawback, though, of using unsupervised learning is that because the dataset is unlabeled, there aren't any known output observations to check and validate the model, and predictions are therefore more subjective than those coming from supervised learning.

We will cover unsupervised learning later in this book specific to  $k$ -means clustering. Other examples of unsupervised learning algorithms include social network analysis and descending dimension algorithms.

## Semi-supervised Learning

A hybrid form of unsupervised and supervised learning is also available in the form of semi-supervised learning, which is used for datasets that contain a mix of labeled and unlabeled cases. With the “more data the better” as a core motivator, the goal of semi-supervised learning is to leverage unlabeled cases to improve the reliability of the prediction model. One technique is to build the initial model using the labeled cases (supervised learning) and then use the same model to label the remaining cases (that are unlabeled) in the dataset. The model can then be retrained using a larger dataset (with less or no unlabeled cases). Alternatively, the model could be iteratively re-trained using newly labeled cases that meet a set threshold of confidence and adding the new cases to the training data after they meet the set threshold. There is, however, no guarantee that a semi-supervised model will outperform a model trained with less data (based exclusively on the original labeled cases).

## Reinforcement Learning

Reinforcement learning is the third and most advanced category of machine learning. Unlike supervised and unsupervised learning, reinforcement learning builds its prediction model by gaining feedback from random trial and error and leveraging insight from previous iterations.

The goal of reinforcement learning is to achieve a specific goal (output) by randomly trialing a vast number of possible input combinations and grading their performance.

Reinforcement learning can be complicated to understand and is probably best explained using a video game analogy. As a player progresses through the virtual space of a game, they learn the value of various actions under different conditions and grow more familiar with the field of play. Those learned values then inform and influence the player's subsequent behavior and their performance gradually improves based on learning and experience.

Reinforcement learning is similar, where algorithms are set to train the model based on continuous learning. A standard reinforcement learning model has measurable performance criteria where outputs are graded. In the case of self-driving vehicles, avoiding a crash earns a positive score, and in the case of chess, avoiding defeat likewise receives a positive assessment.

## **Q-learning**

A specific algorithmic example of reinforcement learning is Q-learning. In Q-learning, you start with a set environment of *states*, represented as “S.” In the game Pac-Man, states could be the challenges, obstacles or pathways that exist in the video game. There may exist a wall to the left, a ghost to the right, and a power pill above—each representing different states. The set of possible actions to respond to these states is referred to as “A.” In Pac-Man, actions are limited to left, right, up, and down movements, as well as multiple combinations thereof. The third important symbol is “Q,” which is the model’s starting value and has an initial value of “0.”

As Pac-Man explores the space inside the game, two main things happen:

- 1) Q drops as negative things occur after a given state/action.
- 2) Q increases as positive things occur after a given state/action.

In Q-learning, the machine learns to match the action for a given state that generates or preserves the highest level of Q. It learns initially through the process of random movements (actions) under different conditions (states). The model records its results (rewards and penalties) and how they impact its Q level and stores those values to inform and optimize its future actions.

While this sounds simple, implementation is computationally expensive and

beyond the scope of an absolute beginner's introduction to machine learning. Reinforcement learning algorithms aren't covered in this book, but, I'll leave you with a link to a more comprehensive explanation of reinforcement learning and Q-learning using the Pac-Man case study.

<https://inst.eecs.berkeley.edu/~cs188/sp12/projects/reinforcement/reinforcement.html>

# THE MACHINE LEARNING TOOLBOX

A handy way to learn a new skill is to visualize a toolbox of the essential tools and materials of that subject area. For instance, given the task of packing a dedicated toolbox to build a website, you would first need to add a selection of programming languages. This would include frontend languages such as HTML, CSS, and JavaScript, one or two backend programming languages based on personal preferences, and of course, a text editor. You might throw in a website builder such as WordPress and then pack another compartment with web hosting, DNS, and maybe a few domain names that you've purchased.

This is not an extensive inventory, but from this general list, you start to gain a better appreciation of what tools you need to master on the path to becoming a successful web developer.

Let's now unpack the basic toolbox for machine learning.

## Compartment 1: Data

Stored in the first compartment of the toolbox is your data. Data constitutes the input needed to train your model and generate predictions. Data comes in many forms, including structured and unstructured data. As a beginner, it's best to start with (analyzing) structured data. This means that the data is defined, organized, and labeled in a table, as shown in Table 3. Images, videos, email messages, and audio recordings are examples of unstructured data as they don't fit into the organized structure of rows and columns.

Date	Bitcoin Price	No. of Days Transpired
19-05-2015	234.31	1
14-01-2016	431.76	240
09-07-2016	652.14	417
15-01-2017	817.26	607
24-05-2017	2358.96	736

Table 3: Bitcoin Prices from 2015-2017

Before we proceed, I first want to explain the anatomy of a tabular dataset. A tabular (table-based) dataset contains data organized in rows and columns. Contained in each column is a *feature*. A feature is also known as a *variable*, a *dimension* or an *attribute*— but they all mean the same thing. Each row represents a single observation of a given feature/variable. Rows are sometimes referred to as a *case* or *value*, but in this book, we use the term “row.”

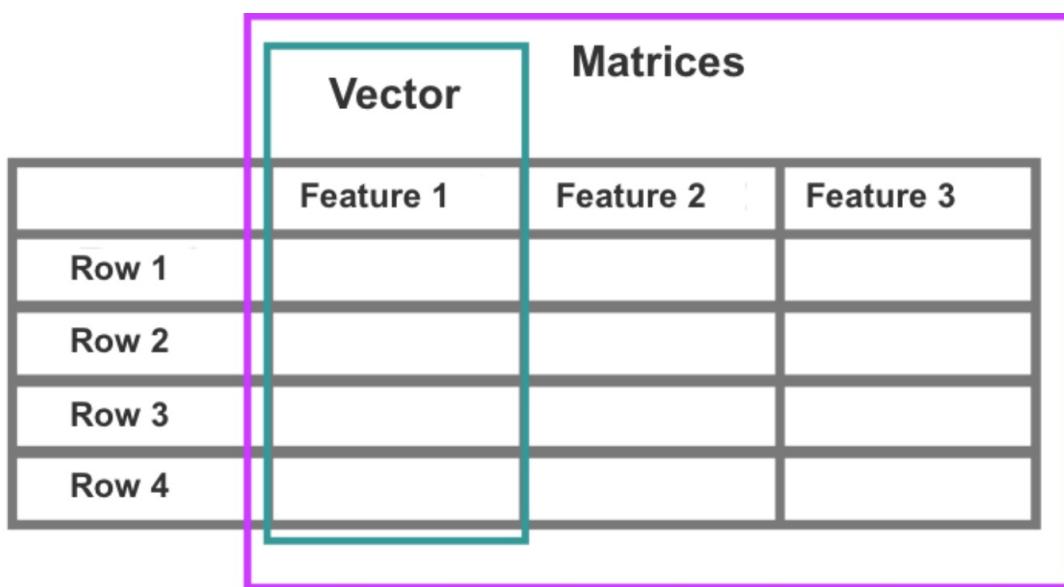


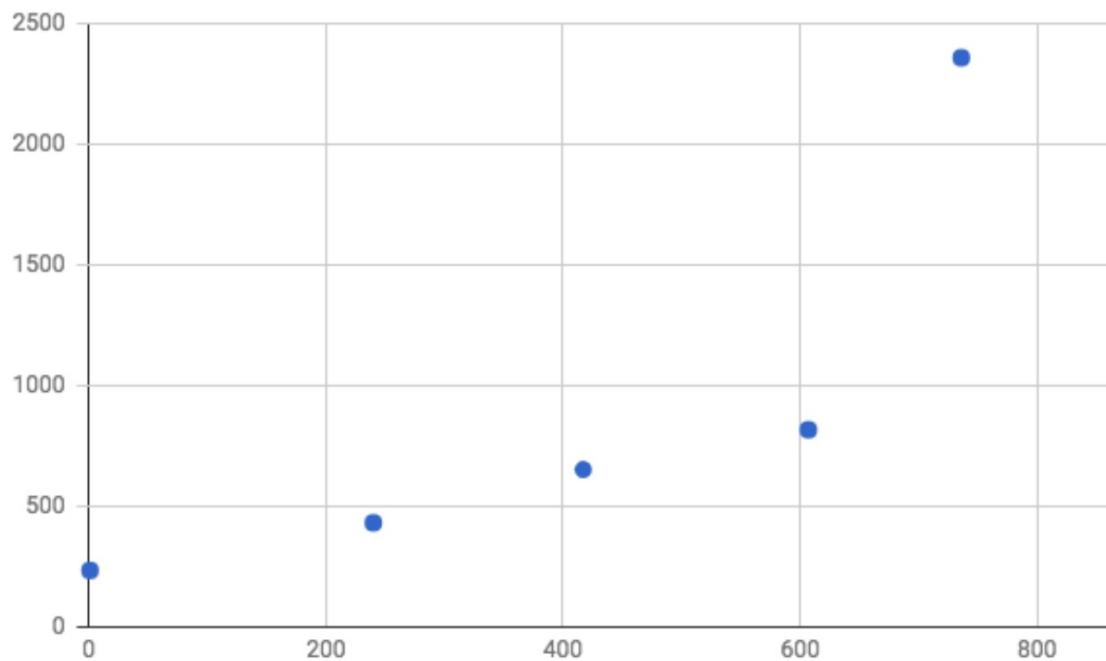
Figure 6: Example of a tabular dataset

Each column is known also as a *vector*. Vectors store your X and y values and multiple vectors (columns) are commonly referred to as *matrices*. In the case of supervised learning, y will already exist in your dataset and be used to identify patterns in relation to the independent variables (X). The y values are commonly expressed in the final vector, as shown in Figure 7.

Matrices			
Vector	Maker (X)	Year (X)	Model (X)
Row 1			
Row 2			
Row 3			
Row 4			

Figure 7: The y value is often but not always expressed in the far-right vector

Scatterplots, including 2-D, 3-D, and 4-D plots, are also packed into the first compartment of the toolbox with the data. A 2-D scatterplot consists of a vertical axis (known as the y-axis) and a horizontal axis (known as the x-axis) and provides the graphical canvas to plot variable combinations, known as data points. Each data point on the scatterplot represents an observation from the dataset with X values on the x-axis and y values on the y-axis.



	<b>Independent Variable (X)</b>	<b>Dependent Variable (y)</b>
<b>Row 1</b>	<b>1</b>	<b>243.31</b>
<b>Row 2</b>	<b>240</b>	<b>431.76</b>
<b>Row 3</b>	<b>417</b>	<b>653.14</b>
<b>Row 4</b>	<b>607</b>	<b>817.26</b>
<b>Row 5</b>	<b>736</b>	<b>2358.96</b>

Figure 8: Example of a 2-D scatterplot. X represents days passed and y is Bitcoin price.

## Compartment 2: Infrastructure

The second compartment of the toolbox contains your machine learning infrastructure, which consists of platforms and tools for processing data. As a beginner to machine learning, you are likely to be using a web application (such as Jupyter Notebook) and a programming language like Python. There are then a series of machine learning libraries, including NumPy, Pandas, and Scikit-learn, which are compatible with Python. Machine learning libraries are a collection of pre-compiled programming routines frequently used in machine learning that enable you to manipulate data and execute algorithms with minimal use of code. You will also need a machine to process your data, in the form of a physical computer or a virtual server. In addition, you may need specialized libraries for data visualization such as Seaborn and Matplotlib, or a standalone software program like Tableau, which supports a range of visualization techniques including charts, graphs, maps, and other visual options.

With your infrastructure sprayed across the table (hypothetically of course), you're now ready to build your first machine learning model. The first step is to crank up your computer. Standard desktop computers and laptops are both sufficient for working with smaller datasets that are stored in a central location, such as a CSV file. You then need to install a programming environment, such as Jupyter Notebook, and a programming language, which for most beginners is Python.

Python is the most widely used programming language for machine learning because:

- a) It's easy to learn and operate.
- b) It's compatible with a range of machine learning libraries.
- c) It can be used for related tasks, including data collection (web scraping) and data piping (Hadoop and Spark).

Other go-to languages for machine learning include C and C++. If you’re proficient with C and C++, then it makes sense to stick with what you know. C and C++ are the default programming languages for advanced machine learning because they can run directly on the GPU (Graphical Processing Unit). Python needs to be converted before it can run on the GPU, but we’ll get to this and what a GPU is later in the chapter.

Next, Python users will need to import the following libraries: NumPy, Pandas, and Scikit-learn. NumPy is a free and open-source library that allows you to efficiently load and work with large datasets, including merging datasets and managing matrices.

Scikit-learn provides access to a range of popular shallow algorithms, including linear regression, clustering techniques, decision trees, and support vector machines. Shallow learning algorithms refer to learning algorithms that predict outcomes directly from the input features. Non-shallow algorithms or deep learning, meanwhile, produce an output based on preceding layers in the model (discussed in Chapter 13 in reference to artificial neural networks) rather than directly from the input features. [\[11\]](#)

Finally, Pandas enables your data to be represented as a virtual spreadsheet that you can control and manipulate using code. It shares many of the same features as Microsoft Excel in that it allows you to edit data and perform calculations. The name Pandas derives from the term “panel data,” which refers to its ability to create a series of panels, similar to “sheets” in Excel. Pandas is also ideal for importing and extracting data from CSV files.

```
In [1]:  
1 # Import library  
2 import pandas as pd  
3  
4 # Read in data from CSV as a Pandas dataframe  
5 df = pd.read_csv('~/Downloads/Melbourne_housing_FULL.csv')  
6  
7 df.head()  
  
Out[1]:  
Suburb Address Rooms Type Price Method SellerG Date Distance Postcode ... Bathroom Car Landsize BuildingArea YearBuilt Co  
0 Abbotsford 68 Studley St 2 h NaN SS Jellis 3/09/2016 2.5 3067.0 ... 1.0 1.0 126.0 NaN NaN  
1 Abbotsford 85 Turner St 2 h 1480000.0 S Biggin 3/12/2016 2.5 3067.0 ... 1.0 1.0 202.0 NaN NaN  
2 Abbotsford 25 Bloomberg St 2 h 1035000.0 S Biggin 4/02/2016 2.5 3067.0 ... 1.0 0.0 156.0 79.0 1900.0  
3 Abbotsford 18/659 Victoria St 3 u NaN VB Rounds 4/02/2016 2.5 3067.0 ... 2.0 1.0 0.0 NaN NaN  
4 Abbotsford 5 Charles St 3 h 1465000.0 SP Biggin 4/03/2017 2.5 3067.0 ... 2.0 0.0 134.0 150.0 1900.0  
5 rows × 21 columns
```

Figure 9: Previewing a table in Jupyter Notebook using Pandas

For students seeking alternative programming options for machine learning

beyond Python, C, and C++, there is also R, MATLAB, and Octave.

R is a free and open-source programming language optimized for mathematical operations and useful for building matrices and performing statistical functions. Although more commonly used for data mining, R also supports machine learning.

The two direct competitors to R are MATLAB and Octave. MATLAB is a commercial and proprietary programming language that is strong at solving algebraic equations and is a quick programming language to learn. MATLAB is widely used in the fields of electrical engineering, chemical engineering, civil engineering, and aeronautical engineering. Computer scientists and computer engineers, however, tend not to use MATLAB and especially in recent years. MATLAB, though, is still widely used in academia for machine learning. Thus, while you may see MATLAB featured in online courses for machine learning, and especially Coursera, this is not to say that it's as commonly used in industry. If, however, you're coming from an engineering background, MATLAB is certainly a logical choice.

Lastly, there is Octave, which is essentially a free version of MATLAB developed in response to MATLAB by the open-source community.

### **Compartment 3: Algorithms**

Now that the development environment is set up and you've chosen your programming language and libraries, you can next import your data directly from a CSV file. You can find hundreds of interesting datasets in CSV format from kaggle.com. After registering as a Kaggle member, you can download a dataset of your choosing. Best of all, Kaggle datasets are free, and there's no cost to register as a user. The dataset will download directly to your computer as a CSV file, which means you can use Microsoft Excel to open and even perform basic algorithms such as linear regression on your dataset.

Next is the third and final compartment that stores the machine learning algorithms. Beginners typically start out using simple supervised learning algorithms such as linear regression, logistic regression, decision trees, and  $k$ -nearest neighbors. Beginners are also likely to apply unsupervised learning in the form of  $k$ -means clustering and descending dimension algorithms.

### **Visualization**

No matter how impactful and insightful your data discoveries are, you need a way to communicate the results to relevant decision-makers. This is where data visualization comes in handy to highlight and communicate findings from the

data to a general audience. The visual story conveyed through graphs, scatterplots, heatmaps, box plots, and the representation of numbers as shapes make for quick and easy storytelling.

In general, the less informed your audience is, the more important it is to visualize your findings. Conversely, if your audience is knowledgeable about the topic, additional details and technical terms can be used to supplement visual elements. To visualize your results, you can draw on a software program like Tableau or a Python library such as Seaborn, which are stored in the second compartment of the toolbox.

## **The Advanced Toolbox**

We have so far examined the starter toolbox for a beginner, but what about an advanced user? What does their toolbox look like? While it may take some time before you get to work with more advanced tools, it doesn't hurt to take a sneak peek.

The advanced toolbox comes with a broader spectrum of tools and, of course, data. One of the biggest differences between a beginner and an expert is the kind of data they manage and operate. Beginners work with small datasets that are easy to handle and downloaded directly to one's desktop as a simple CSV file. Advanced users, though, will be eager to tackle massive datasets, well in the vicinity of big data. This might mean that the data is stored across multiple locations, and its composition is streamed (imported and analyzed in real-time) rather than static, which makes the data itself a moving target.

### **Compartment 1: Big Data**

Big data is used to describe a dataset that, due to its variety, volume, and velocity, defies conventional methods of processing and would be impossible for a human to process without the assistance of advanced technology. Big data doesn't have an exact definition in terms of size or a minimum threshold of rows and columns. At the moment, petabytes qualify as big data, but datasets are becoming increasingly bigger as we find new ways to collect and store data at a lower cost.

Big data is also less likely to fit into standard rows and columns and may contain numerous data types, such as structured data and a range of unstructured data, i.e. images, videos, email messages, and audio files.

### **Compartment 2: Infrastructure**

Given that advanced learners are dealing with up to petabytes of data, robust

infrastructure is required. Instead of relying on the CPU of a personal computer, the experts typically turn to distributed computing and a cloud provider such as Amazon Web Services (AWS) or Google Cloud Platform to run their data processing on a virtual graphics processing unit (GPU). As a specialized parallel computing chip, GPU instances are able to perform many more floating-point operations per second than a CPU, allowing for much faster solutions with linear algebra and statistics than with a CPU.

GPU chips were originally added to PC motherboards and video consoles such as the PlayStation 2 and the Xbox for gaming purposes. They were developed to accelerate the rendering of images with millions of pixels whose frames needed to be continuously recalculated to display output in less than a second. By 2005, GPU chips were produced in such large quantities that prices dropped dramatically and they became almost a commodity. Although popular in the video game industry, their application in the space of machine learning wasn't fully understood or realized until quite recently. Kevin Kelly, in his novel *The Inevitable: Understanding the 12 Technological Forces That Will Shape Our Future*, explains that in 2009, Andrew Ng and a team at Stanford University made a discovery to link inexpensive GPU clusters to run neural networks consisting of hundreds of millions of connected nodes.

"Traditional processors required several weeks to calculate all the cascading possibilities in a neural net with one hundred million parameters. Ng found that a cluster of GPUs could accomplish the same thing in a day," explains Kelly. [\[12\]](#)

As mentioned, C and C++ are the preferred languages to directly edit and perform mathematical operations on the GPU. Python can also be used and converted into C in combination with a machine learning library such as TensorFlow from Google. Although it's possible to run TensorFlow on a CPU, you can gain up to about 1,000x in performance using the GPU. Unfortunately for Mac users, TensorFlow is only compatible with the Nvidia GPU card, which is no longer available with Mac OS X. Mac users can still run TensorFlow on their CPU but will need to run their workload on the cloud if they wish to use a GPU.

Amazon Web Services, Microsoft Azure, Alibaba Cloud, Google Cloud Platform, and other cloud providers offer pay-as-you-go GPU resources, which may also start off free using a free trial program. Google Cloud Platform is currently regarded as a leading choice for virtual GPU resources based on performance and pricing. Google also announced in 2016 that it would publicly release a Tensor Processing Unit designed specifically for running TensorFlow, which is already used internally at Google.

### **Compartment 3: Advanced Algorithms**

To round out this chapter, let's take a look at the third compartment of the advanced toolbox containing machine learning algorithms. To analyze large datasets and respond to complicated prediction tasks, advanced practitioners work with a plethora of algorithms including Markov models, support vector machines, and Q-learning, as well as combinations of algorithms to create a unified model, known as ensemble modeling (explored further in Chapter 15). However, the algorithm family they're most likely to work with is artificial neural networks (introduced in Chapter 13), which comes with its own selection of advanced machine learning libraries.

While Scikit-learn offers a range of popular shallow algorithms, TensorFlow is the machine learning library of choice for deep learning/neural networks. It supports numerous advanced techniques including automatic calculus for back-propagation/gradient descent. The depth of resources, documentation, and jobs available with TensorFlow also make it an obvious framework to learn. Popular alternative libraries for neural networks include Torch, Caffe, and the fast-growing Keras.

Written in Python, Keras is an open-source deep learning library that runs on top of TensorFlow, Theano, and other frameworks, which allows users to perform fast experimentation in fewer lines of code. Similar to a WordPress website theme, Keras is minimal, modular, and quick to get up and running. It is, however, less flexible in comparison to TensorFlow and other libraries. Developers, therefore, will sometimes utilize Keras to validate their decision model before switching to TensorFlow to build a more customized model.

Caffe is also open-source and is typically used to develop deep learning architectures for image classification and image segmentation. Caffe is written in C++ but has a Python interface that supports GPU-based acceleration using the Nvidia cuDNN chip.

Released in 2002, Torch is also well established in the deep learning community and is used at Facebook, Google, Twitter, NYU, IDIAP, Purdue University as well as other companies and research labs. [\[13\]](#) Based on the programming language Lua, Torch is open-source and offers a range of algorithms and functions used for deep learning.

Theano was another competitor to TensorFlow until recently, but as of late 2017, contributions to the framework have officially ceased. [\[14\]](#)

# DATA SCRUBBING

Like most varieties of fruit, datasets need upfront cleaning and human manipulation before they're ready for consumption. The "clean-up" process applies to machine learning and many other fields of data science and is known in the industry as *data scrubbing*. This is the technical process of refining your dataset to make it more workable. This might involve modifying and removing incomplete, incorrectly formatted, irrelevant or duplicated data. It might also entail converting text-based data to numeric values and the redesigning of features.

For data practitioners, data scrubbing typically demands the greatest application of time and effort.

## Feature Selection

To generate the best results from your data, it's essential to identify which variables are most relevant to your hypothesis or objective. In practice, this means being selective in choosing the variables you include in your model. Moreover, preserving features that don't correlate strongly with the output value can manipulate and derail the model's accuracy. Let's consider the following data excerpt downloaded from kaggle.com documenting dying languages.

Name in English	Name in Spanish	Countries	Country Code	Num. of Speakers
South Italian	Napolitano -calabres	Italy	ITA	7500000
Sicilian	Siciliano	Italy	ITA	5000000
Low Saxon	Bajo Sajón	Germany, Denmark, Netherlands, Poland, Russian Federation	DEU, DNK, NLD, POL, RUS	4800000
Belarusian	Bielorruso	Belarus, Latvia, Lithuania, Poland, Russian Federation, Ukraine	BRB, LVA, LTU, POL, RUS, UKR	4000000
Lombard	Lombardo	Italy, Switzerland	ITA, CHE	3500000
Romani	Romaní	Albania, Germany, Austria, Belarus, Bosnia and Herzegovina, Bulgaria, Croatia, Estonia, Finland, France, Greece, Hungary, Italy, Latvia, Lithuania, The former Yugoslav Republic of Macedonia, Netherlands, Poland, Romania, United Kingdom of Great Britain and Northern Ireland, Russian Federation, Slovakia, Slovenia, Switzerland, Czech Republic, Turkey, Ukraine, Serbia, Montenegro	ALB, DEU, AUT, BRB, BIH, BGR, HRV, EST, FIN, FRA, GRC, HUN, ITA, LVA, LTU, MKD, NLD, POL, ROU, GBR, RUS, SVK, SVN, CHE, CZE, TUR, UKR, SRB, MNE	3500000
Yiddish	Yiddish	Israel	ISR	3000000
Gondi	Gondi	India	IND	2713790

Table 4: Endangered languages, database: <https://www.kaggle.com/the-guardian/extinct-languages>

Let's say our goal is to identify variables that contribute to a language becoming endangered. Based on the purpose of our analysis, it's unlikely that a language's "Name in Spanish" will lead to any relevant insight. We can therefore delete this vector (column) from the dataset. This helps to prevent over-complication and potential inaccuracies as well as improve the overall processing speed of the model.

Secondly, the dataset contains duplicated information in the form of separate vectors for "Countries" and "Country Code." Analyzing both of these vectors doesn't provide any additional insight; hence, we can choose to delete one and retain the other.

Another method to reduce the number of features is to roll multiple features into one, as shown in the following example.

	Protein Shake	Nike Sneakers	Adidas Boots	Fitbit	Powerade	Protein Bar	Fitness Watch	Vitamins
Buyer 1	1	1	0	1	0	5	1	0
Buyer 2	0	0	0	0	0	0	0	1
Buyer 3	3	0	1	0	5	0	0	0
Buyer 4	1	1	0	0	10	1	0	0

Table 5: Sample product inventory

Contained in Table 5 is a list of products sold on an e-commerce platform. The dataset comprises four buyers and eight products. This is not a large sample size of buyers and products—due in part to the spatial limitations of the book format. A real-life e-commerce platform would have many more columns to work with but let's go ahead with this simplified example.

To analyze the data more efficiently, we can reduce the number of columns by merging similar features into fewer columns. For instance, we can remove individual product names and replace the eight product items with fewer categories or subtypes. As all product items fall under the category of “fitness,” we can sort by product subtype and compress the columns from eight to three. The three newly created product subtype columns are “Health Food,” “Apparel,” and “Digital.”

	<b>Health Food</b>	<b>Apparel</b>	<b>Digital</b>
<b>Buyer 1</b>	6	1	2
<b>Buyer 2</b>	1	0	0
<b>Buyer 3</b>	8	1	0
<b>Buyer 4</b>	12	1	0

Table 6: Synthesized product inventory

This enables us to transform the dataset in a way that preserves and captures information using fewer variables. The downside to this transformation is that we have less information about the relationships between specific products. Rather than recommending products to users according to other individual products, recommendations will instead be based on associations between product subtypes or recommendations of the same product subtype.

Nonetheless, this approach still upholds a high level of data relevancy. Buyers will be recommended health food when they buy other health food or when they buy apparel (depending on the degree of correlation), and obviously not machine learning textbooks—unless it turns out that there is a strong correlation there! But alas, such a variable/category is outside the frame of this dataset.

Remember that data reduction is also a business decision and business owners in counsel with their data science team must consider the trade-off between convenience and the overall precision of the model.

## Row Compression

In addition to feature selection, you may need to reduce the number of rows and thereby compress the total number of data points. This may involve merging two or more rows into one, as shown in the following dataset, with “Tiger” and “Lion” merged and renamed as “Carnivore.”

### **Before**

<b>Animal</b>	<b>Meat Eater</b>	<b>Legs</b>	<b>Tail</b>	<b>Race Time</b>
<b>Tiger</b>	Yes	4	Yes	2:01 mins
<b>Lion</b>	Yes	4	Yes	2:05 mins
<b>Tortoise</b>	No	4	No	55:02 mins

### **After**

<b>Animal</b>	<b>Meat Eater</b>	<b>Legs</b>	<b>Tail</b>	<b>Race Time</b>
<b>Carnivore</b>	Yes	4	Yes	2:03 mins
<b>Tortoise</b>	No	4	No	55:02 mins

Table 7: Example of row merge

By merging these two rows (Tiger & Lion), the feature values for both rows must also be aggregated and recorded in a single row. In this case, it’s possible to merge the two rows because they possess the same categorical values for all features except Race Time—which can be easily aggregated. The race time of the Tiger and the Lion can be added and divided by two.

Numeric values are normally easy to aggregate given they are not categorical. For instance, it would be impossible to aggregate an animal with four legs and an animal with two legs! We obviously can’t merge these two animals and set “three” as the aggregate number of legs.

Row compression can also be challenging to implement in cases where numeric values aren’t available. For example, the values “Japan” and “Argentina” are very difficult to merge. The values “Japan” and “South Korea” can be merged, as they can be categorized as countries from the same continent, “Asia” or “East Asia.” However, if we add “Pakistan” and “Indonesia” to the same group, we may begin to see skewed results, as there are significant cultural, religious, economic, and other dissimilarities between these four countries.

In summary, non-numeric and categorical row values can be problematic to

merge while preserving the true value of the original data. Also, row compression is usually less attainable than feature compression and especially for datasets with a high number of features.

## One-hot Encoding

After finalizing the features and rows to be included in your model, you next want to look for text-based values that can be converted into numbers. Aside from set text-based values such as True/False (that automatically convert to “1” and “0” respectively), most algorithms are not compatible with non-numeric data.

One method to convert text-based values into numeric values is *one-hot encoding*, which transforms values into binary form, represented as “1” or “0”—“True” or “False.” A “0,” representing False, means that the value does not belong to a given feature, whereas a “1”—True or “hot”—confirms that the value does belong to that feature.

Below is another excerpt from the dying languages dataset which we can use to observe one-hot encoding.

Name in English	Speakers	Degree of Endangerment
<b>South Italian</b>	7500000	Vulnerable
<b>Sicilian</b>	5000000	Vulnerable
<b>Low Saxon</b>	4800000	Vulnerable
<b>Belarusian</b>	4000000	Vulnerable
<b>Lombard</b>	3500000	Definitely endangered
<b>Romani</b>	3500000	Definitely endangered
<b>Yiddish</b>	3000000	Definitely endangered
<b>Gondi</b>	2713790	Vulnerable
<b>Picard</b>	700000	Severely endangered

Table 8: Endangered languages

Before we begin, note that the values contained in the “No. of Speakers” column do not contain commas or spaces, e.g., 7,500,000 and 7 500 000. Although formatting makes large numbers easier for human interpretation, programming languages don’t require such niceties. Formatting numbers can lead to an invalid syntax or trigger an unwanted result, depending on the programming language—so remember to keep numbers unformatted for programming purposes. Feel free, though, to add spacing or commas at the data visualization stage, as this will make it easier for your audience to interpret and especially when presenting large numbers.

On the right-hand side of the table is a vector categorizing the degree of endangerment of nine different languages. We can convert this column into numeric values by applying the one-hot encoding method, as demonstrated in the subsequent table.

Name in English	Speakers	Vulnerable	Definitely Endangered	Severely Endangered
<b>South Italian</b>	7500000	1	0	0
<b>Sicilian</b>	5000000	1	0	0
<b>Low Saxon</b>	4800000	1	0	0
<b>Belarusian</b>	4000000	1	0	0
<b>Lombard</b>	3500000	0	1	0
<b>Romani</b>	3500000	0	1	0
<b>Yiddish</b>	3000000	0	1	0
<b>Gondi</b>	2713790	1	0	0
<b>Picard</b>	700000	0	0	1

Table 9: Example of one-hot encoding

Using one-hot encoding, the dataset has expanded to five columns, and we have created three new features from the original feature (Degree of Endangerment). We have also set each column value to “1” or “0,” depending on the value of the original feature. This now makes it possible for us to input the data into our model and choose from a broader spectrum of machine learning algorithms. The downside is that we have more dataset features, which may slightly extend processing time. This is usually manageable but *can* be problematic for datasets

where the original features are split into a large number of new features. One hack to minimize the total number of features is to restrict binary cases to a single column. As an example, a speed dating dataset on kaggle.com lists “Gender” in a single column using one-hot encoding. Rather than create discrete columns for both “Male” and “Female,” they merged these two features into one. According to the dataset’s key, females are denoted as “0” and males as “1.” The creator of the dataset also used this technique for “Same Race” and “Match.”

<b>Subject Number ID</b>	<b>Gender</b>	<b>Same Race</b>	<b>Age</b>	<b>Match</b>
1	0	0	27	0
1	0	0	22	0
1	0	1	22	1
1	0	0	23	1
1	0	0	24	1
1	0	0	25	0
1	0	0	30	0

**Gender:**                   **Same Race:**                   **Match:**

Female = 0                   No = 0                   No = 0

Male = 1                   Yes = 1                   Yes = 1

Table 10: Speed dating results, database: <https://www.kaggle.com/annavictoria/speed-dating-experiment>

## Binning

Binning (also called bucketing) is another method of feature engineering but is used for converting continuous numeric values into multiple binary features called bins or buckets according to their range of values.

Whoa, hold on! Aren’t numeric values a good thing? Yes, in most cases continuous numeric values are preferred as they are compatible with a broader selection of algorithms. Where numeric values are not ideal, is in situations where they list variations irrelevant to the goals of your analysis.

Let's take house price evaluation as an example. The exact measurements of a tennis court might not matter much when evaluating house property prices; the relevant information is whether the property *has* a tennis court. This logic probably also applies to the garage and the swimming pool, where the existence or non-existence of the variable is generally more influential than their specific measurements.

The solution here is to replace the numeric measurements of the tennis court with a True/False feature or a categorical value such as “small,” “medium,” and “large.” Another alternative would be to apply one-hot encoding with “0” for homes that *do not* have a tennis court and “1” for homes that *do* have a tennis court.

## Normalization

While machine learning algorithms can run without using the next two techniques, normalization and standardization help to improve model accuracy when used with the right algorithm. The former (normalization) rescales the range of values for a given feature into a set range with a prescribed minimum and maximum, such as  $[0, 1]$  or  $[-1, 1]$ . By containing the range of the feature, this technique helps to normalize the variance among the dataset’s features which may otherwise be exaggerated by another factor. The variance of a feature measured in centimeters, for example, might distract the algorithm from another feature with a similar or higher degree of variance but that is measured in meters or another metric that downplays the actual variance of the feature.

Normalization, however, usually isn’t recommended for rescaling features with an extreme range as the normalized range is too narrow to emphasize extremely high or low feature values.

## Standardization

A better technique for emphasizing high or low feature values is standardization. This technique converts unit variance to a standard normal distribution with a mean of zero and a standard deviation ( $\sigma$ ) of one.<sup>[\[15\]](#)</sup> This means that an extremely high or low value would be expressed as three or more standard deviations from the mean.

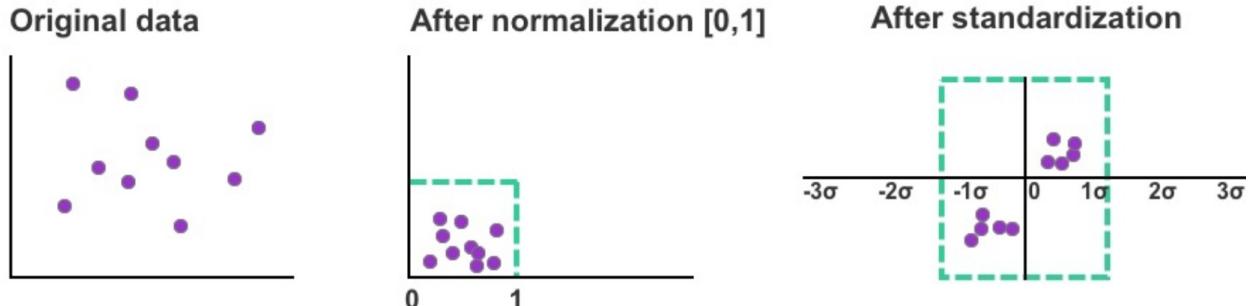


Figure 10: Examples of rescaled data using normalization and standardization

Standardization is generally more effective than normalization when the variability of the feature reflects a bell-curve shape of normal distribution and is often used in unsupervised learning. In other situations, normalization and standardization can be applied separately and compared for accuracy.

Standardization generally recommended when preparing data for support vector machines (SVM), principal component analysis (PCA), and  $k$ -nearest neighbors ( $k$ -NN).

## Missing Data

Dealing with missing data is never a desired situation. Imagine unpacking a jigsaw puzzle with five percent of the pieces missing. Missing values in your dataset can be equally frustrating and interfere with your analysis and the model's predictions. There are, however, strategies to minimize the negative impact of missing data.

One approach is to approximate missing values using the *mode* value. The mode represents the single most common variable value available in the dataset. This works best with categorical and binary variable types, such as one to five-star rating systems and positive/negative drug tests respectively.

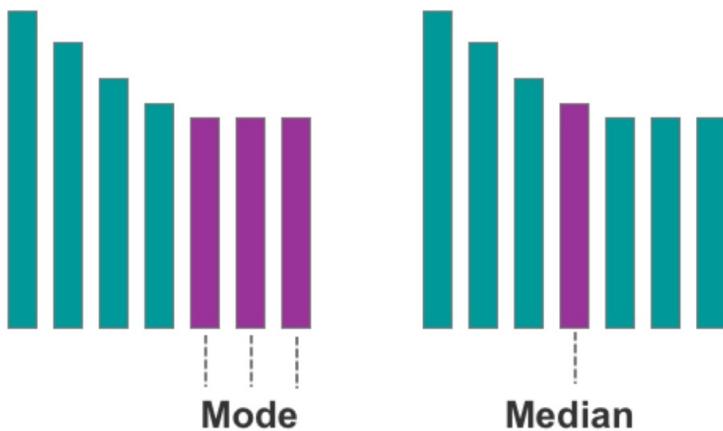


Figure 11: A visual example of the mode and median respectively

The second approach is to approximate missing values using the *median* value, which adopts the value(s) located in the middle of the dataset. This works best with continuous variables, which have an infinite number of possible values, such as house prices.

As a last resort, rows with missing values can be removed altogether. The obvious downside to this approach is having less data to analyze and potentially less comprehensive insight.

# SETTING UP YOUR DATA

After cleaning your dataset, the next job is to split the data into two segments for training and testing, also known as *split validation*. The ratio of the two splits is usually 70/30 or 80/20. This means, assuming that your variables are expressed horizontally and instances vertically (as shown in Figure 12), that your training data should account for 70 percent to 80 percent of the rows in your dataset, and the remaining 20 percent to 30 percent of rows are left for your test data.

	Variable 1	Variable 2	Variable 3
Row 1	Training Data		
Row 2			
Row 3			
Row 4		r	
Row 5			
Row 6			
Row 7			
Row 8	Test Data		
Row 9			
Row 10			

Figure 12: 70/30 partitioning of training and test data

While it's common to split the data 70/30 or 80/20, there is no set rule for preparing a training-test split. Given the growing size of modern datasets (with upwards of a million or more rows), it might be optimal to use a less even split such as 90/10 as this will give you more data to train your model while having

enough data left over to test your model.

Before you split your data, it's essential that you randomize the row order. This helps to avoid bias in your model, as your original dataset might be arranged alphabetically or sequentially according to when the data was collected. If you don't randomize the data, you may accidentally omit significant variance from the training data that can cause unwanted surprises when you apply the training model to your test data. Fortunately, Scikit-learn provides a built-in command to shuffle and randomize your data with just one line of code as demonstrated in Chapter 17.

After randomizing the data, you can begin to design your model and apply it to the training data. The remaining 30 percent or so of data is put to the side and reserved for testing the accuracy of the model later; it's imperative not to test your model with the same data you used for training. In the case of supervised learning, the model is developed by feeding the machine the training data and analyzing relationships between the features ( $X$ ) of the input data and the final output ( $y$ ).

The next step is to measure how well the model performed. There is a range of performance metrics and choosing the right method depends on the application of the model. Area under the curve (AUC) – Receiver Operating Characteristic (ROC) [\[16\]](#), confusion matrix, recall, and accuracy are four examples of performance metrics used with classification tasks such as an email spam detection system. Meanwhile, mean absolute error and root mean square error (RMSE) are commonly used to assess models that provide a numeric output such as a predicted house value.

In this book, we use mean absolute error (MAE), which measures the average of the errors in a set of predictions on a numeric/continuous scale, i.e. how far is the regression hyperplane to a given data point. Using Scikit-learn, mean absolute error is found by inputting the  $X$  values from the training data into the model and generating a prediction for each row in the dataset. Scikit-learn compares the predictions of the model to the correct output ( $y$ ) and measures the model's accuracy. You'll know that the model is accurate when the error rate for the training and test dataset is low, which means the model has learned the dataset's underlying trends and patterns. If the average recorded MAE or RMSE is much higher using the test data than the training data, this is usually an indication of overfitting (discussed in Chapter 11) in the model. Once the model can adequately predict the values of the test data, it's ready to use in the wild.

If the model fails to predict values from the test data accurately, check that the training and test data were randomized. Next, you may need to modify the

model's hyperparameters. Each algorithm has hyperparameters; these are your algorithm's learning settings( and not the settings of the actual model itself). In simple terms, hyperparameters control and impact how fast the model learns patterns and which patterns to identify and analyze. Discussion of algorithm hyperparameters and optimization is discussed in Chapter 11 and Chapter 18.

## Cross Validation

While split validation can be effective for developing models using existing data, question marks naturally arise over whether the model can remain accurate when used on new data. If your existing dataset is too small to construct a precise model, or if the training/test partition of data is not appropriate, this may later lead to poor predictions with live data.

Fortunately, there is a valid workaround for this problem. Rather than split the data into two segments (one for training and one for testing), you can implement what's called *cross validation* . Cross validation maximizes the availability of training data by splitting data into various combinations and testing each specific combination.

Cross validation can be performed using one of two primary methods. The first method is *exhaustive cross validation* , which involves finding and testing all possible combinations to divide the original sample into a training set and a test set. The alternative and more common method is non-exhaustive cross validation, known as *k-fold validation* . The  $k$ -fold validation technique involves splitting data into  $k$  assigned buckets and reserving one of those buckets for testing the training model at each round.

To perform  $k$ -fold validation, data are randomly assigned to  $k$  number of equal-sized buckets. One bucket is reserved as the test bucket and is used to measure and evaluate the performance of the remaining ( $k - 1$ ) buckets.

# Buckets

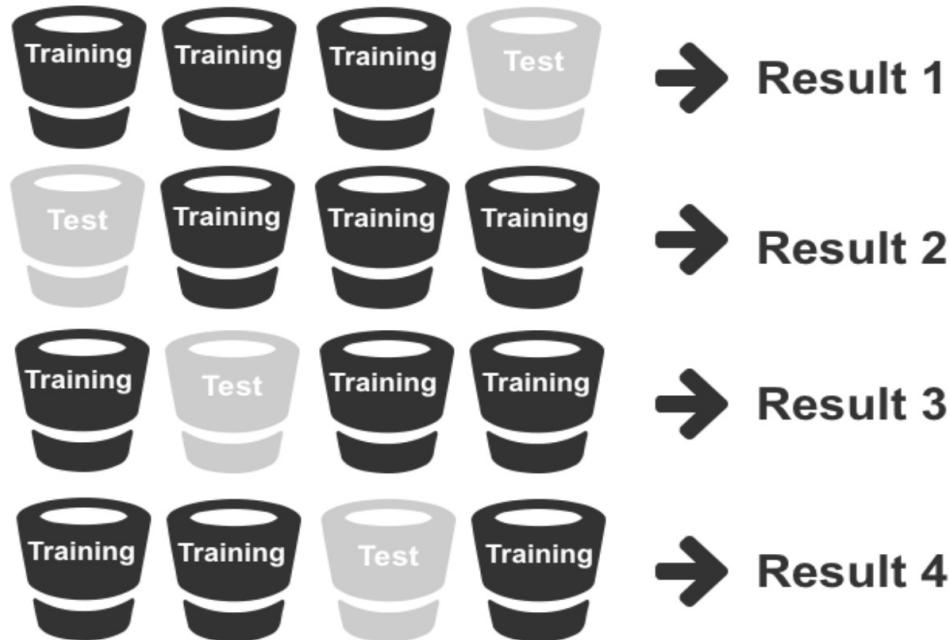


Figure 13:  $k$  -fold validation

The cross validation technique is repeated  $k$  number of times (“folds”). At each fold, one bucket is reserved to test the training model generated by the other buckets. The process is repeated until all buckets have been utilized as both a training and test set. The results are then aggregated and combined to formulate a single model.

By using all available data for both training and testing and averaging the model’s outputs, the  $k$  -fold validation technique minimizes the prediction error normally incurred by relying on a fixed training-test split. This method, though, is slower because the training process is multiplied by the number of validation sets.

## How Much Data Do I Need?

A common question for students starting out in machine learning is *how much data do I need to train my model?* In general, machine learning works best when your training dataset includes a full range of feature combinations.

What does a full range of feature combinations look like? Imagine you have a dataset about data scientists categorized into the following features:

- University degree (X)
- 5+ years of professional experience (X)

- Children (X)
- Salary (y)

To assess the relationship that the first three features (X) have to a data scientist's salary (y), we need a dataset that includes the y value for each combination of features. For instance, we need to know the salary for data scientists with a university degree and 5+ years of professional experience who don't have children, as well as data scientists with a university degree and 5+ years of professional experience that do have children.

The more available combinations in the dataset, the more effective the model is at capturing how each attribute affects y (the data scientist's salary). This ensures that when it comes to putting the model into practice on the test data or live data, it won't unravel at the sight of unseen combinations.

At an absolute minimum, a basic machine learning model should contain ten times as many data points as the total number of features. So, for a small dataset with 5 features, the training data should ideally have at least 50 rows. Datasets with a large number of features, though, require a higher number of data points as combinations grow exponentially with more variables.

Generally, the more relevant data you have available as training data, the more combinations you can incorporate into your prediction model, which can help to produce more accurate predictions. In some cases, it might not be possible or cost-effective to source data covering all possible combinations, and you may have to make do with what you have at your disposal. Conversely, there is a natural diminishing rate of return after an adequate volume of training data (that's widely representative of the problem) has been reached.

The last important consideration is matching your data to an algorithm. For datasets with less than 10,000 samples, clustering and dimensionality reduction algorithms can be highly effective, whereas regression analysis and classification algorithms are more suitable for datasets with less than 100,000 samples. Neural networks require even more samples to run effectively and are more cost-effective and time-efficient for working with massive quantities of data.

For more information, Scikit-learn has a cheat-sheet for matching algorithms to different datasets at [http://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/](http://scikit-learn.org/stable/tutorial/machine_learning_map/).

*The following chapters examine specific algorithms commonly used in machine learning. Please note that I include some equations out of necessity, and I have tried to keep them as simple as possible. Many of the machine learning techniques that are discussed in this book already have working implementations in your programming language of choice with no equation solving required.*

*You can also find video tutorials on how to code models in Python using algorithms mentioned in this book. You can find these free video tutorials at <https://scatterplotpress.teachable.com/p/ml-code-exercises> .*

# LINEAR REGRESSION

As the “Hello World” of supervised learning algorithms, regression analysis is a simple technique for predicting an unknown variable using the results you do know. The first regression technique we’ll examine is linear regression, which generates a straight line to describe linear relationships. We’ll start by examining the basic components of simple linear regression with one independent variable before discussing multiple regression with multiple independent variables.

Using the Seinfeld TV sitcom series as our data, let’s start by plotting the two following variables, with season number as the x coordinate and the number of viewers per season (in millions) as the y coordinate.

Season (x)	Viewers (y)
1	19.22
2	18.07
3	17.67
4	20.52
5	29.59
6	31.27
7	33.19
8	32.24
9	38.11

Table 11: Seinfeld dataset

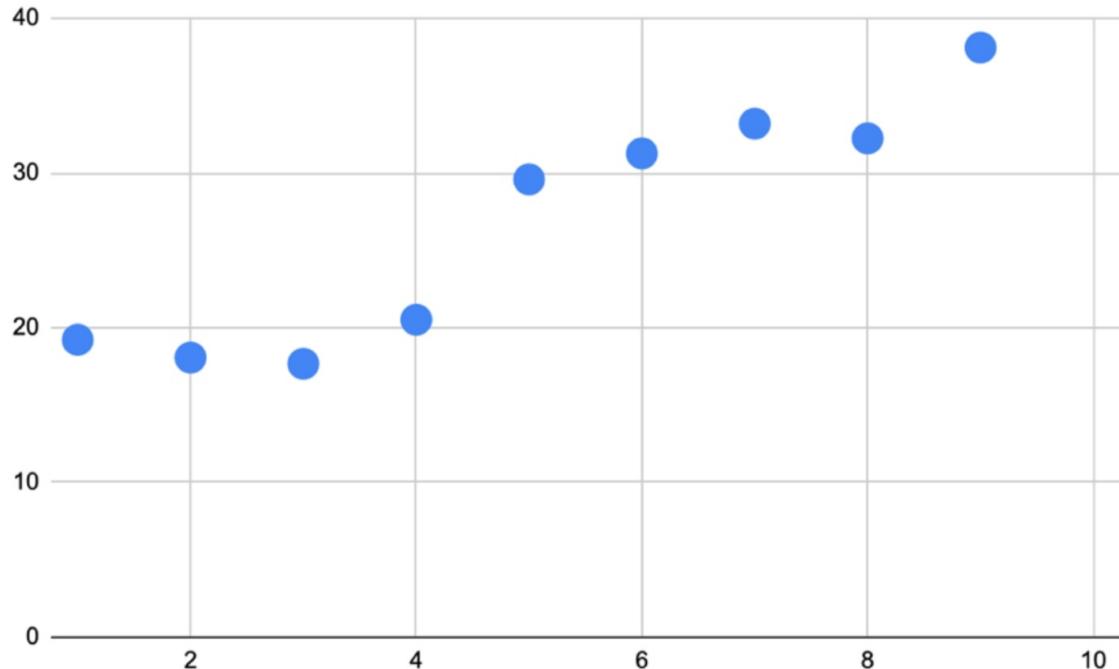


Figure 14: Seinfeld dataset plotted on a scatterplot

We can now see the dataset plotted on the scatterplot, with an upward trend in viewers starting at season 4 and the peak at season 9.

Let's next define the independent and dependent variables. For this example, we'll use the number of viewers per season as the dependent variable (what we want to predict) and the season number as the independent variable.

Using simple linear regression, let's now insert a straight line to describe the upward linear trend of our small dataset.

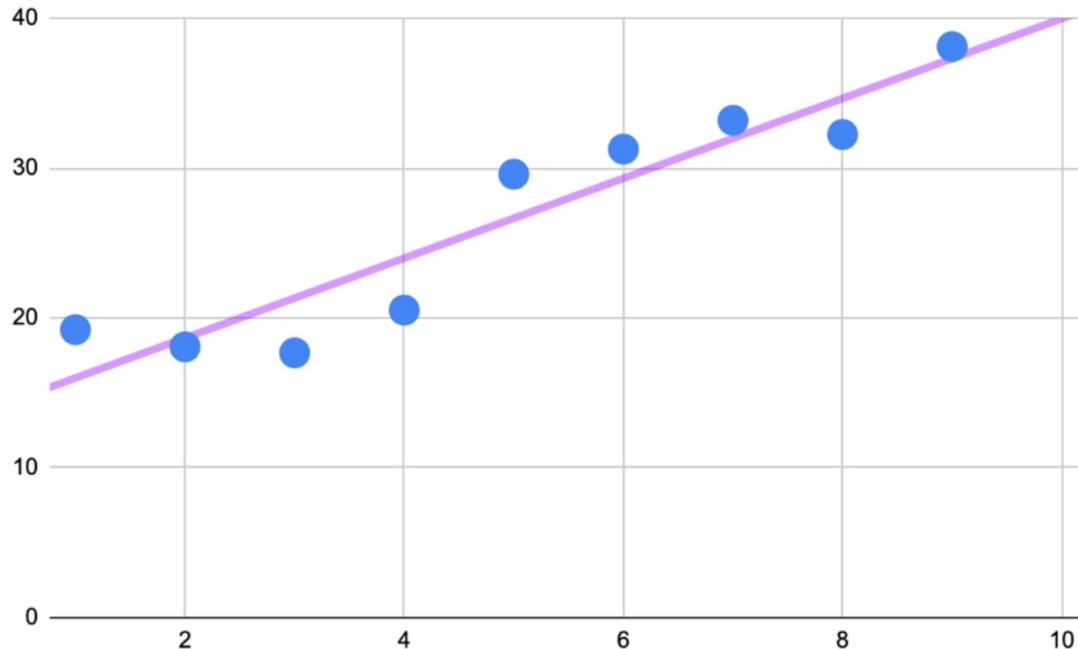


Figure 15: Linear regression hyperplane

As shown in Figure 15, this regression line neatly dissects the full company of data points. The technical term for the regression line is the *hyperplane*, and you'll see this term used throughout your study of machine learning. In a two-dimensional space, a hyperplane serves as a (flat) trendline, which is how Google Sheets titles linear regression in their scatterplot customization menu.

The goal of linear regression is to split the data in a way that minimizes the distance between the hyperplane and the observed values. This means that if you were to draw a perpendicular line (a straight line at an angle of 90 degrees) from the hyperplane to each data point on the plot, the aggregate distance of each point would equate to the smallest possible distance to the hyperplane. The distance between the best fit line and the observed values is called the residual or error and the closer those values are to the hyperplane, the more accurate the model's predictions.

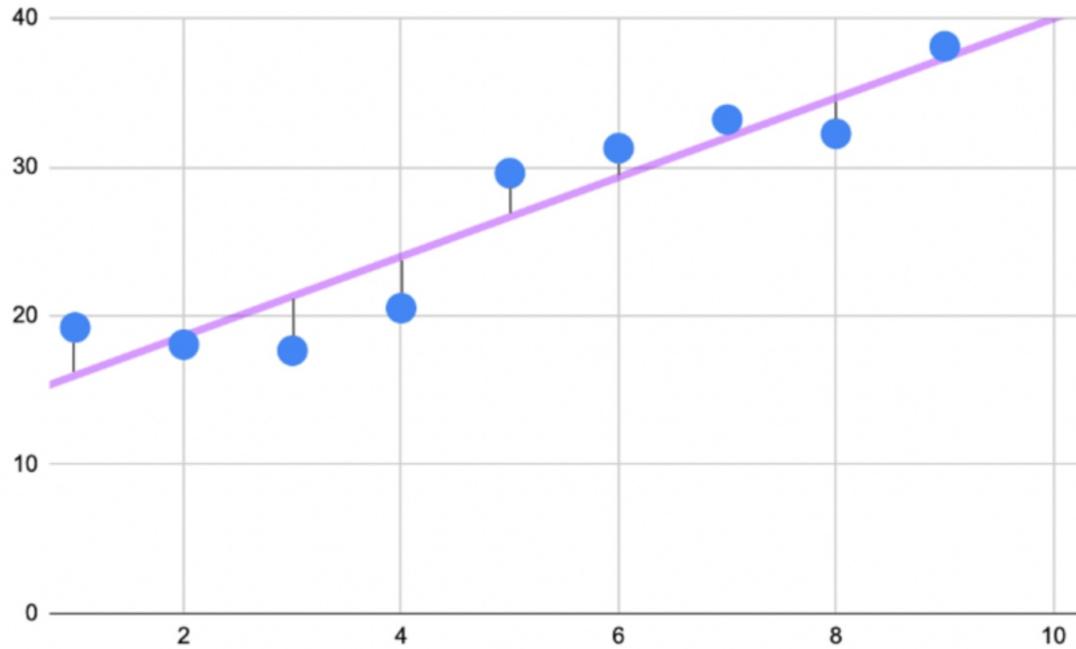


Figure 16: Error is the distance between the hyperplane and the observed value

## The Slope

An important part of linear regression is the *slope*, which can be conveniently calculated by referencing the hyperplane. As one variable increases, the other variable will increase by the average value denoted by the hyperplane. The slope is therefore helpful for formulating predictions, such as predicting the number of season viewers for a potential tenth season of Seinfeld. Using the slope, we can input 10 as the x coordinate and find the corresponding y value, which in this case, is approximately 40 million viewers.

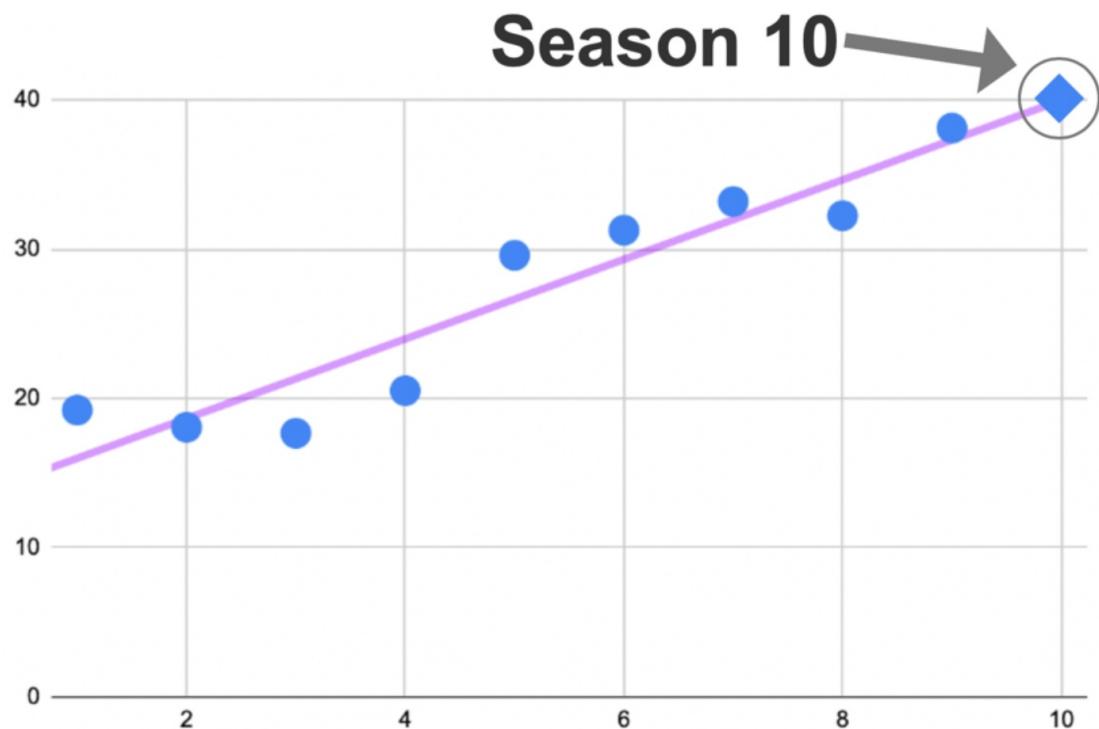


Figure 17: Using the slope/hyperplane to make a prediction

While linear regression isn't a fail-proof method for predicting trends, the trendline does offer a basic reference point for predicting unknown or future events.

### Linear Regression Formula

The formula [\[17\]](#) for linear regression is  $y = bx + a$ .

“y” represents the dependent variable and “x” represents the independent variable.

“a” is the point where the hyperplane crosses the y-axis, known as the *y-intercept* or the value of y when x = 0.

“b” dictates the steepness of the slope and explains the relationship between x and y (what change in y is predicted for 1 unit change in x).

### Calculation Example

Although your programming language takes care of this automatically, it's interesting to know how simple linear regression works. We'll use the following dataset to break down the formula.

	<b>(X)</b>	<b>(Y)</b>	<b>XY</b>	<b>X<sup>2</sup></b>
<b>1</b>	1	3	3	1
<b>2</b>	2	4	8	4
<b>3</b>	1	2	2	1
<b>4</b>	4	7	28	16
<b>5</b>	3	5	15	9
<b>Σ (Total)</b>	<b>11</b>	<b>21</b>	<b>56</b>	<b>31</b>

Table 12: Sample dataset

# The final two columns of the table are not part of the original dataset and have been added for reference to complete the following formula.

$$a = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

### Where:

$\Sigma$  = Total sum

$\Sigma x$  = Total sum of all x values ( $1 + 2 + 1 + 4 + 3 = 11$ )

$\Sigma y$  = Total sum of all y values ( $3 + 4 + 2 + 7 + 5 = 21$ )

$\Sigma xy$  = Total sum of  $x*y$  for each row ( $3 + 8 + 2 + 28 + 15 = 56$ )

$\Sigma x^2$  = Total sum of  $x*x$  for each row ( $1 + 4 + 1 + 16 + 9 = 31$ )

n = Total number of rows. In the case of this example, n is equal to 5.

$$a = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$a = \frac{(21)(31) - (11)(56)}{5(31) - (11)^2}$$

$$b = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b = \frac{5(56) - (11)(21)}{5(31) - (11)^2}$$

a =

$$((21 \times 31) - (11 \times 56)) / (5(31) - 11^2)$$

$$(651 - 616) / (155 - 121)$$

$$35 / 34 = 1.029$$

$$\mathbf{b} =$$

$$(5(56) - (11 \times 21)) / (5(31) - 11^2)$$

$$(280 - 231) / (155 - 121)$$

$$49 / 34 = 1.441$$

Insert the “a” and “b” values into the linear formula.

$$y = bx + a$$

$$y = 1.441x + 1.029$$

The linear formula  $y = 1.441x + 1.029$  dictates how to draw the hyperplane.

Let's now test the linear regression model by looking up the coordinates for  $x = 2$ .

$$y = 1.441(x) + 1.029$$

$$y = 1.441(2) + 1.029$$

$$y = 3.911$$

In this case, the prediction is very close to the actual result of 4.0.

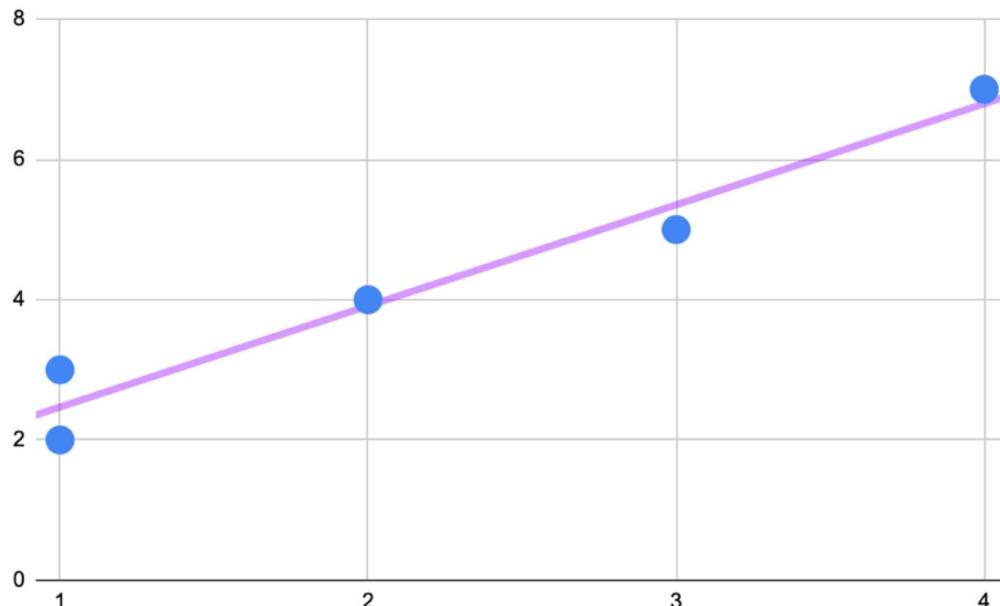


Figure 18:  $y = 1.441x + 1.029$  plotted on the scatterplot

## Multiple Linear Regression

Having summarized simple linear regression using a single independent variable, we can now look at multiple linear regression. This second technique is more applicable to machine learning given organizations use more than one independent variable to make decisions.

Multiple linear regression is simple linear regression but with more than one independent variable as the following formula shows.

$$y = a + b_1x_1 + b_2x_2 + b_3x_3 + \dots$$

The  $y$ -intercept is still expressed as  $a$ , but now there are multiple independent variables (represented as  $x_1$ ,  $x_2$ ,  $x_3$ , etc.) and each with their own respected coefficient ( $b_1$ ,  $b_2$ ,  $b_3$ , etc).

As with simple linear regression, various sums of  $X$  and  $y$  values (including squared values) from the training data are used to solve for a ( $y$ -intercept) and  $b$  (coefficient values).

Once a model has been built using the  $X$  and  $y$  values from the training data, the multiple linear regression formula can be used to make a prediction ( $y$ ) using the  $X$  values from the test data (to assess accuracy).

## Discrete Variables

While the output (dependent variable) of linear regression must be continuous in the form of a floating-point or integer (whole number) value, the input (independent variables) can be continuous or categorical. For categorical variables, i.e. gender, these variables must be expressed numerically using one-hot encoding (0 or 1) and not as a string of letters (male, female).

## Variable Selection

Before finishing this chapter, it's important to address the dilemma of variable selection and choosing an appropriate number of independent variables. On the one hand, adding more variables helps to account for more potential factors that control patterns in the data. On the other hand, this rationale only holds if the variables are relevant and possess some correlation/linear relationship with the dependent variable.

The expansion of independent variables also creates more relationships to consider. In simple linear regression, we saw a one-to-one relationship between two variables, whereas in multiple linear regression there is a many-to-one relationship. In multiple linear regression, not only are the independent variables potentially related to the dependent variable, but they are also potentially related to each other.

## ONE-TO-ONE



## MANY-TO-ONE

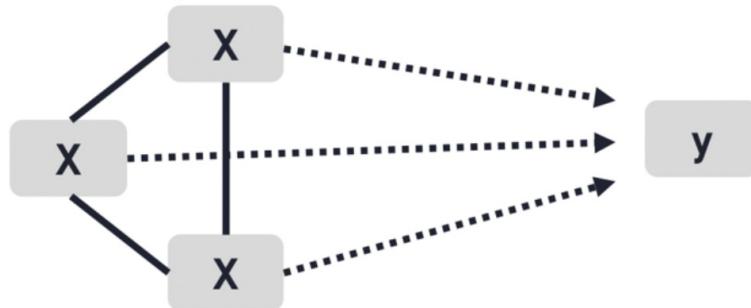


Figure 19: Simple linear regression (above) and multiple linear regression (below)

If a strong linear correlation exists between two independent variables, this can lead to a problem called multi-collinearity. When two independent variables are strongly correlated, they have a tendency to cancel each other out and provide the model with little to no unique information.

An example of two multi-collinear variables are liters of fuel consumed and liters of fuel in the tank to predict how far a jet plane will fly. Both independent variables are directly correlated, and in this case negatively correlated; as one variable increases, the other variable decreases and vice versa. When both variables are used to predict the dependent variable of how far the jet will fly, one effectively cancels the other out. It's still worthwhile to include one of these variables in the model, but it would be redundant to include both variables.

To avoid multi-collinearity, we need to check the relationship between each combination of independent variables using a scatterplot, pairplot (a matrix of relationships between variables), or correlation score.

If we look at the pairplot in Figure 20, we can analyze the relationship between all three variables (total\_bill, tip, and size). If we set tip as the dependent variable, then we need to assess whether the two independent variables (total\_bill and size) are strongly correlated. Using our pairplot, we can see there are two scatterplots visualizing the relationship between total\_bill and size (row 1 on the right, and row 3 on the left). These two plots are not identical (as the x- and y-axis are inverted), but you can refer to either one.

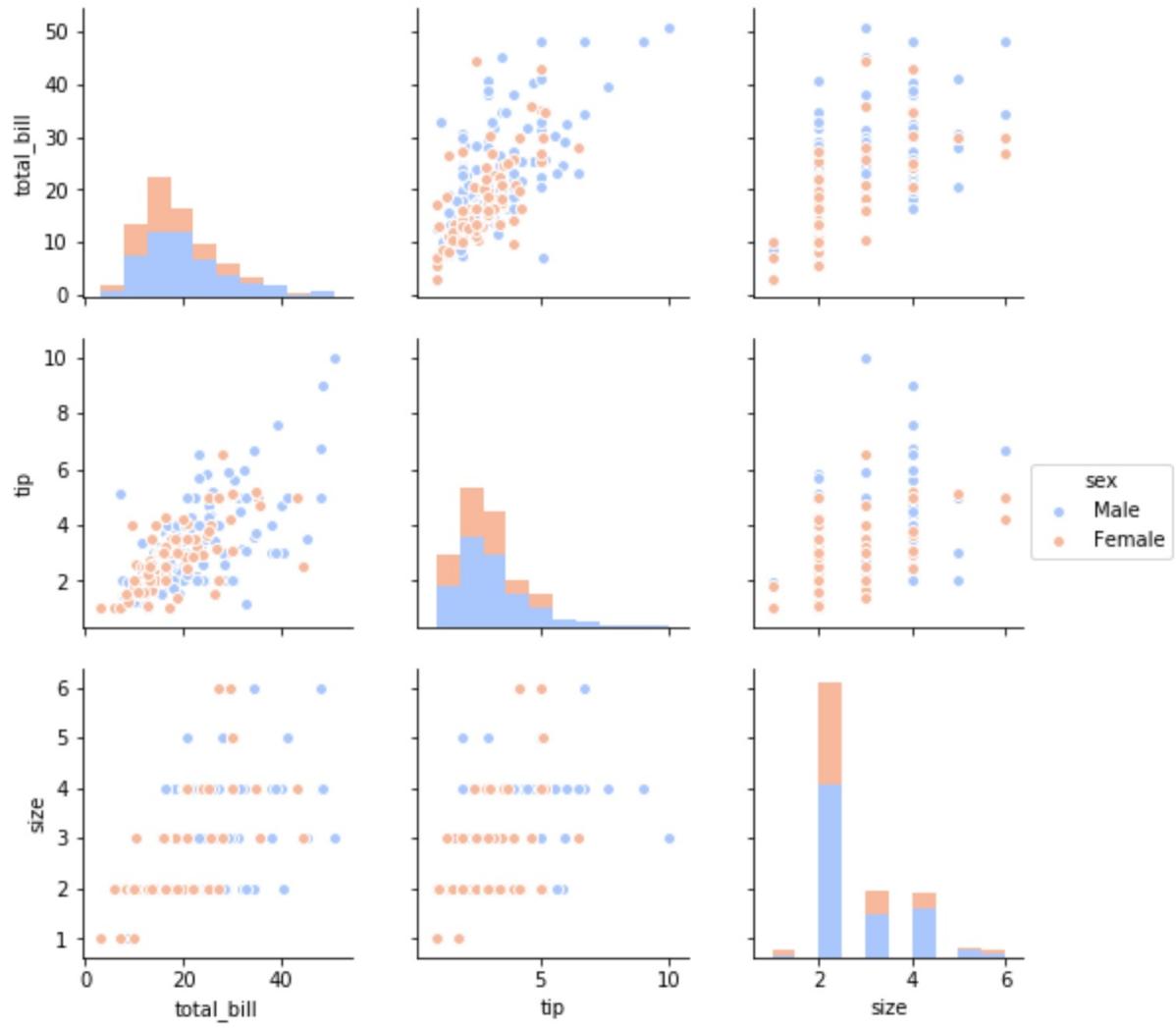


Figure 20: Pairplot with three variables

Judging by the upward linear trend, we can see that these two variables are partly correlated. However, if we were to insert a linear regression hyperplane, there would be significant residuals/error on both sides of the hyperplane to confirm that these two variables aren't strongly or directly correlated and we can definitely include both these variables in our regression model.

The following heatmap, shown in Figure 21, also confirms a modest correlation score of 0.6 between total\_bill and size.

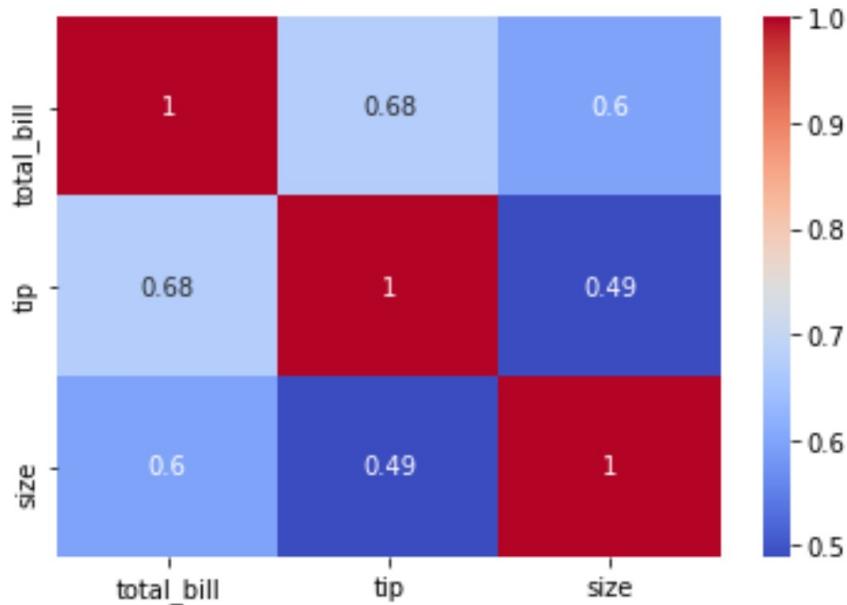


Figure 21: Heatmap with three variables

We can also use a pairplot, heatmap or correlation score to check if the independent variables are correlated to the dependent variable (and therefore relevant to the prediction outcome). In Figure 21, we can see that total\_bill (0.68) and size (0.49) show some correlation with the dependent variable of tip. (Correlation is measured between -1 and 1, with a correlation of 1 describing a perfect positive relationship and a correlation of -1 indicating a perfect negative relationship. A coefficient of 0, meanwhile, denotes no relationship between two variables.)

In summary, the objective of multiple linear regression is for all the independent variables to be correlated with the dependent variable but not with each other.

# CHAPTER QUIZ

Using **multiple linear regression**, your task is to create a model to predict the tip amount guests will leave the restaurant when paying for their meal. Note that this is a snippet of the actual dataset and the full dataset has 244 rows (diners).

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2

- 1) The dependent variable for this model should be which variable?
  - A) size
  - B) total\_bill and tip
  - C) total\_bill
  - D) tip
- 2) From looking only at the data preview above, which variable(s) appear to have a linear relationship with total\_bill?
  - A) smoker
  - B) total\_bill and size
  - C) time
  - D) smoker
- 3) It's important for the independent variables to be strongly correlated

**with the dependent variable and one or more of the other independent variables. True or False?**

# ANSWERS

1) D, tip

2) B, total\_bill and size

(When there is an increase in both of these variables, we see a general increase in the tip for most rows. Other variables might be correlated to tip, but it's not clear to judge using only these 10 rows.)

3) False

(Ideally, the independent variables should not be strongly correlated with each other.)

# LOGISTIC REGRESSION

As demonstrated in the previous chapter, linear regression is useful for quantifying relationships between variables to predict a continuous outcome. Total bill and size (number of guests) are both examples of continuous variables. However, what if we want to predict a categorical variable such as “new customer” or “returning customer”? Unlike linear regression, the dependent variable ( $y$ ) is no longer a continuous variable (such as total tip) but rather a discrete categorical variable.

Rather than quantify the linear relationship between variables, we need to use a classification technique such as logistic regression.

Logistic regression is still a supervised learning technique but produces a qualitative prediction rather than a quantitative prediction. This algorithm is often used to predict two discrete classes, e.g., *pregnant* or *not pregnant*. Given its strength in binary classification, logistic regression is used in many fields including fraud detection, disease diagnosis, emergency detection, loan default detection, or to identify spam email through the process of discerning specific classes, e.g., non-spam and spam.

Using the sigmoid function, logistic regression finds the probability of independent variables ( $X$ ) producing a discrete dependent variable ( $y$ ) such as “spam” or “non-spam.”

$$y = \frac{1}{1+e^{-x}}$$

Where:

$x$  = the independent variable you wish to transform

$e$  = Euler's constant, 2.718

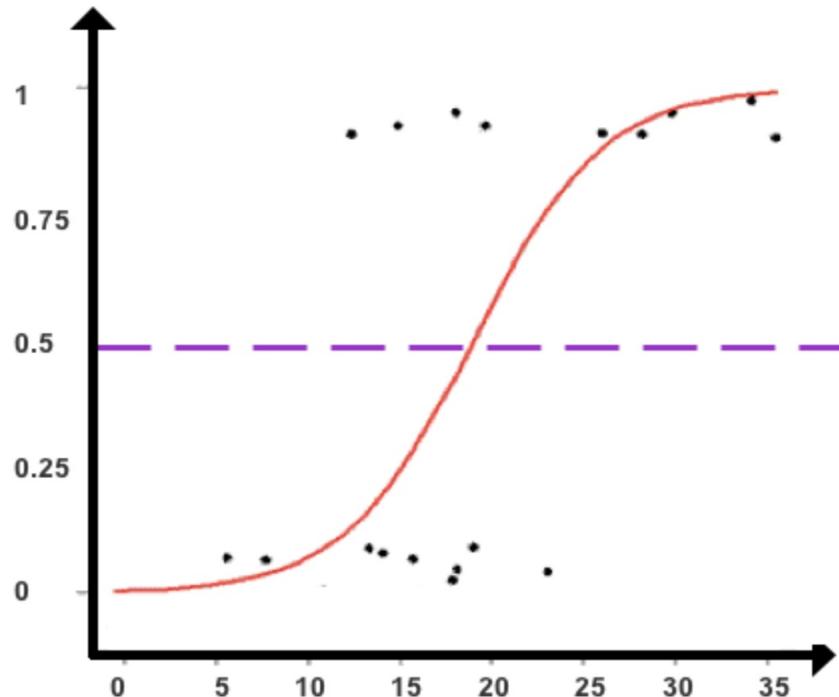


Figure 22: A sigmoid function used to classify data points

The sigmoid function produces an S-shaped curve that can convert any number and map it into a numerical value between 0 and 1 but without ever reaching those exact limits. Applying this formula, the sigmoid function converts independent variables into an expression of probability between 0 and 1 in relation to the dependent variable. In a binary case, a value of 0 represents no chance of occurring, and 1 represents a certain chance of occurring. The degree of probability for values located between 0 and 1 can be found according to how close they rest to 0 (impossible) or 1 (certain possibility).

Based on the found probabilities of the independent variables, logistic regression assigns each data point to a discrete class. In the case of binary classification (shown in Figure 22), the cut-off line to classify data points is 0.5. Data points that record a value above 0.5 are classified as Class A, and data points below 0.5 are classified as Class B. Data points that record a result of precisely 0.5 are unclassifiable but such instances are rare due to the mathematical component of the sigmoid function.

Following the logistic transformation using the Sigmoid function, the data points are assigned to one of two classes as presented in Figure 23.

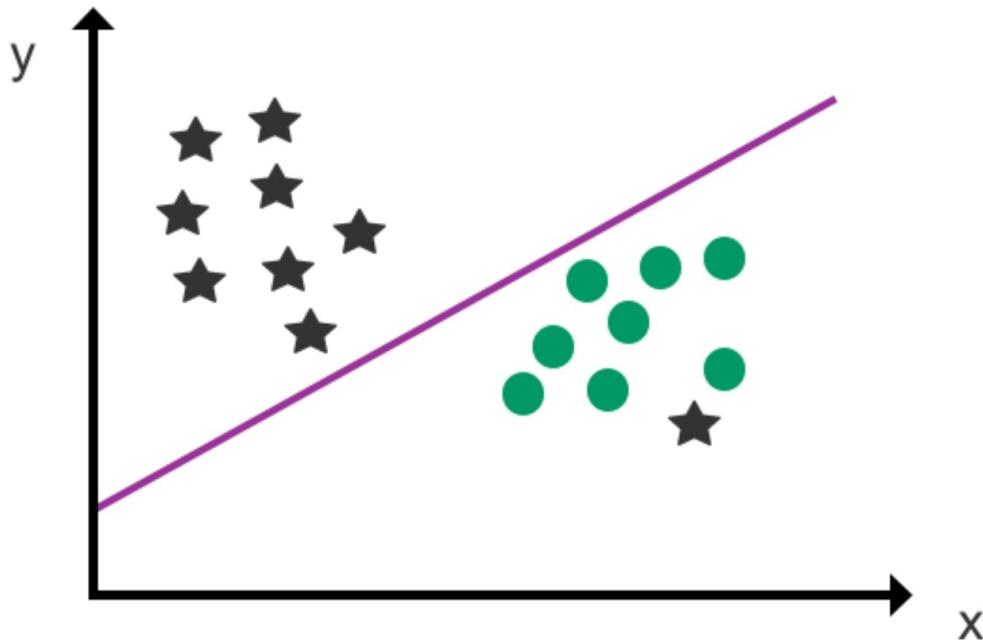


Figure 23: An example of logistic regression

Similar to linear regression, the independent variables, used as input to predict the dependent variable, can be categorical or continuous as long as they are expressed as numbers and not as strings of letters. In the case of discrete categorical variables, this involves using one-hot encoding to create a new set of variables to represent the original variable numerically.

Although logistic regression shares a visual resemblance to linear regression, the logistic hyperplane represents a classification/decision boundary rather than a prediction trendline. Thus, instead of using the hyperplane to make numeric predictions, the hyperplane is used to divide the dataset into classes.

The other distinction between logistic and linear regression is that the dependent variable ( $y$ ) isn't placed along the  $y$ -axis in logistic regression. Instead, independent variables can be plotted along both axes, and the class (output) of the dependent variable is determined by the position of the data point in relation to the decision boundary. Data points on one side of the decision boundary are classified as Class A, and data points on the opposing side of the decision boundary are Class B.

For classification scenarios with more than two possible discrete outcomes, multinomial logistic regression can be used as shown in Figure 24.

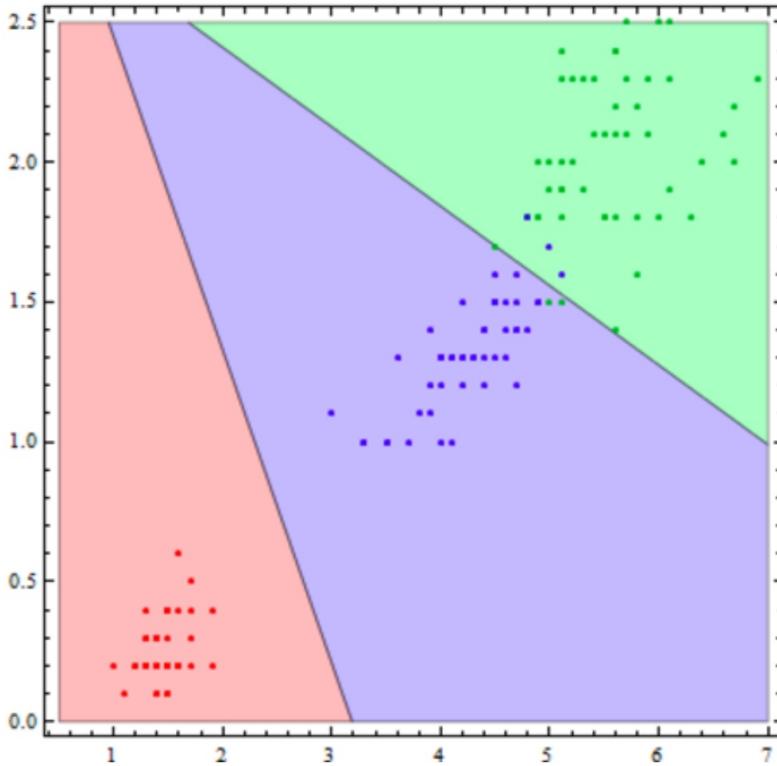


Figure 24: An example of multinomial logistic regression

As a similar classification method, multinomial logistic regression solves multiclass problems with more than two possible discrete outcomes. Multinomial logistic regression can also be applied to ordinal cases where there are a set number of discrete outcomes, e.g., pre-undergraduate, undergraduate, and postgraduate. Keep in mind, though, that logistic regression's core strength lies in binary prediction, and other classification algorithms including decision trees or support vector machines may be a better option for solving multiclass problems.

Two tips to remember when using logistic regression are that the dataset should be free of missing values and that all independent variables are independent and not strongly correlated with each other. There should also be sufficient data for each output variable to ensure high accuracy. A good starting point would be approximately 30-50 data points for each output, i.e., 60-100 total data points for binary logistic regression. In general, logistic regression normally doesn't work so well with large datasets, and especially messy data containing outliers, complex relationships, and missing values.

If you would like to learn more about the mathematical foundation of logistic regression, you can check out *Statistics 101: Logistic Regression* series on YouTube by Brandon Foltz. [\[18\]](#)

# CHAPTER QUIZ

Using **logistic regression**, your task is to classify penguins into different classes based on the following dataset. Please note that this dataset has 244 rows and the following screenshot is a snippet of the full dataset.

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	MALE
6	Adelie	Torgersen	38.9	17.8	181.0	3625.0	FEMALE
7	Adelie	Torgersen	39.2	19.6	195.0	4675.0	MALE
8	Adelie	Torgersen	34.1	18.1	193.0	3475.0	NaN
9	Adelie	Torgersen	42.0	20.2	190.0	4250.0	NaN

- 1) Which three variables (in their current form) could we use as the dependent variable to classify penguins?
- 2) Which row(s) contains missing values?
- 3) Which variable in the dataset preview is binary?

# ANSWERS

**1)** species, island, or sex

**2)** Row 3, 8, and 9

(NaN = missing value)

**3)** sex

(Species and island might also be binary but we can't judge from the screenshot alone.)

# **$k$ -NEAREST NEIGHBORS**

Another popular classification technique in machine learning is  $k$  -nearest neighbors ( $k$  -NN). As a supervised learning algorithm,  $k$  -NN classifies new data points based on their position to nearby data points.

In many ways,  $k$  -NN is similar to a voting system or a popularity contest. Imagine you're the new kid at school and you need to know how to dress in order to fit in with the rest of the class. On your first day at school, you see six of the nine students sitting closest to you with their sleeves rolled-up. Based on numerical supremacy and close proximity, the following day you also make the decision to roll up your sleeves.

Let's now look at another example.

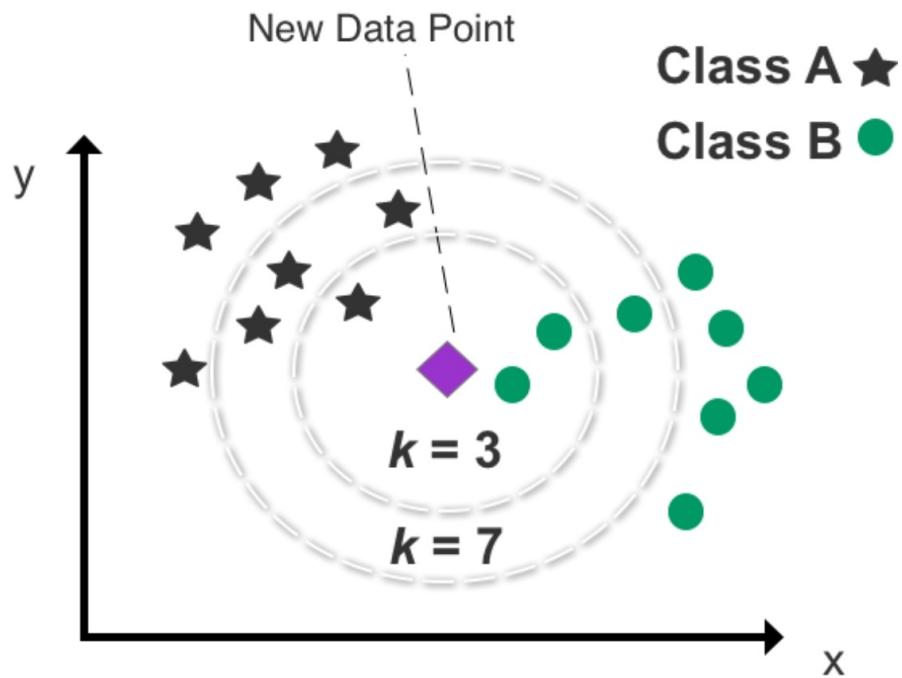


Figure 25: An example of  $k$ - NN clustering used to predict the class of a new data point

Here in Figure 25, the data points have been classified into two classes, and a new data point, whose class is unknown, is added to the plot. Using  $k$  -NN, we

can predict the category of the new data point based on its position to the existing data points.

First, though, we need to set “ $k$ ” to determine how many data points we want to use to classify the new data point. If we set  $k$  to 3,  $k$ -NN analyzes the new data point’s position in respect to the three nearest data points (neighbors). The outcome of selecting the three closest neighbors returns two Class B data points and one Class A data point. Defined by  $k$  (3), the model’s prediction for determining the category of the new data point is Class B as it returns two out of the three nearest neighbors.

The chosen number of neighbors identified, defined by  $k$ , is crucial in determining the results. In Figure 25, you can see that the outcome of classification changes by altering  $k$  from “3” to “7.” It’s therefore useful to test numerous  $k$  combinations to find the best fit and avoid setting  $k$  too low or too high. Setting  $k$  too low will increase bias and lead to misclassification and setting  $k$  too high will make it computationally expensive. Setting  $k$  to an uneven number will also help to eliminate the possibility of a statistical stalemate and an invalid result. Five is the default number of neighbors for this algorithm using Scikit-learn.

Given that the scale of the individual variables has a major impact on the output of  $k$ -NN, the dataset usually needs to be scaled to standardize variance as discussed in Chapter 5. This transformation will help to avoid one or more variables with a high range unfairly pulling the focus of the  $k$ -NN model.

In regards to what type of data to use with  $k$ -NN, this algorithm works best with continuous variables. It is still possible to use binary categorical variables represented as 0 and 1, but the results are more likely to be informed by the binary splits relative to the dispersion across other variables as visualized in Figure 26.

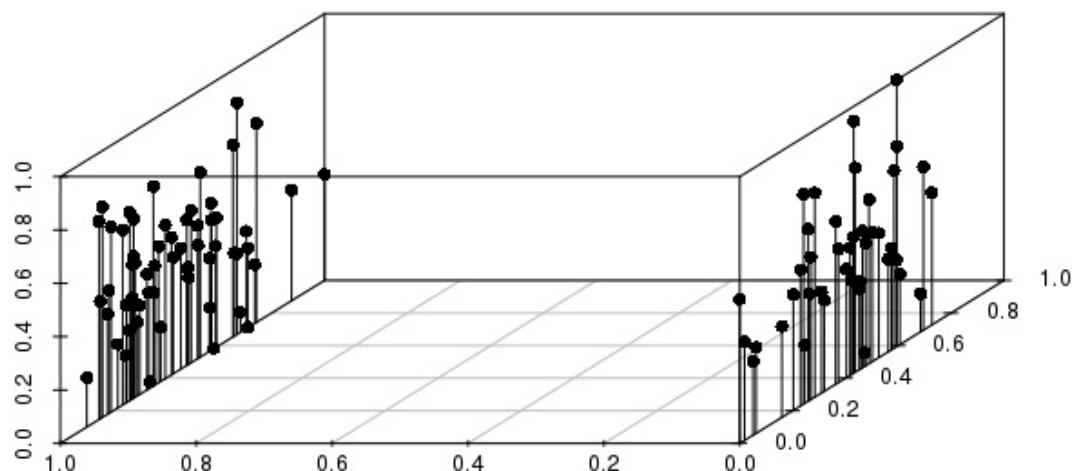


Figure 26: One binary variable and two continuous variables

Above, we can see that the horizontal x-axis is binary (0 or 1), which splits the data into two distinct sides. Moreover, if we switch one of the existing continuous variables to a binary variable (as shown in Figure 27), we can see that the distance between variables is influenced even more greatly by the outcome of the binary variables.

If you do wish to examine binary variables, it's therefore best to only include critical binary variables for  $k$ -NN analysis.

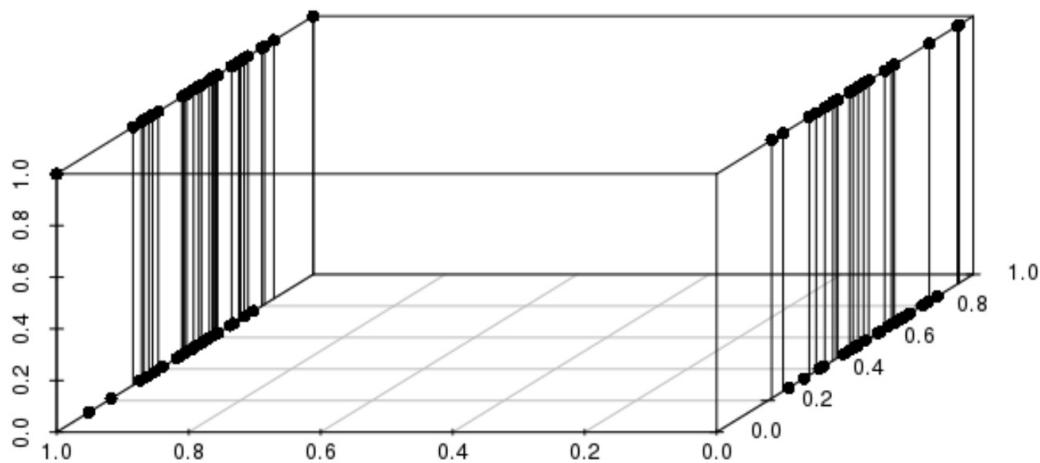


Figure 27: Two binary variable and one continuous variable

While  $k$ -NN is generally accurate and easy to comprehend, storing an entire dataset and calculating the distance between each new data point and all existing data points puts a heavy burden on computing resources. This means that the number of data points in the dataset is proportional to the time it takes to execute a single prediction, which can lead to slower processing times. For this reason,  $k$ -NN is generally not recommended for analyzing large datasets.

Another downside is that it can be challenging to apply  $k$ -NN to high-dimensional data with a high number of features. Measuring multiple distances between data points in a high-dimensional space is also taxing on computing resources and it becomes more difficult to perform accurate classification.

# CHAPTER QUIZ

Your task is to classify penguins into different species using the  **$k$ -nearest neighbors** algorithm, with  $k$  set to 3 (neighbors). Please note that this dataset has 244 rows and the following is a preview of the full dataset.

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	39.1	18.7	181.0	3750.0	MALE
1	Adelie	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	NaN	NaN	NaN	NaN	NaN
4	Adelie	36.7	19.3	193.0	3450.0	FEMALE
5	Adelie	39.3	20.6	190.0	3650.0	MALE
6	Adelie	38.9	17.8	181.0	3625.0	FEMALE
7	Adelie	39.2	19.6	195.0	4675.0	MALE
8	Adelie	34.1	18.1	193.0	3475.0	NaN
9	Adelie	42.0	20.2	190.0	4250.0	NaN

- 1) Which of the following variables should we consider removing from our k-NN model?
  - A. sex
  - B. species
  - C. body\_mass\_g
  - D. bill\_depth\_mm
- 2) If we wanted to reduce the processing time of our model, which of the following methods is recommended?
  - A. Increase  $k$  from 5 to 10
  - B. Reduce  $k$  from 10 to 5
  - C. Re-run the model and hope for a faster result
  - D. Increase the size of the training data
- 3) To include the variable ‘sex’ in our model, which data scrubbing technique do we need to use?

# ANSWERS

**1)** A, sex

(Binary variables should only be used when critical to the model's accuracy.)

**2)** B, Reduce k from 10 to 5

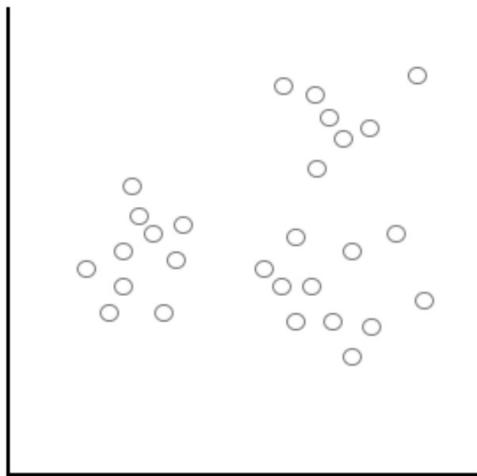
**3)** One-hot encoding (to convert the variable into a numerical identifier of 0 or 1)

## ***k* -MEANS CLUSTERING**

The next method of analysis involves grouping or clustering data points that share similar attributes using unsupervised learning. An online business, for example, wants to examine a segment of customers that purchase at the same time of the year and discern what factors influence their purchasing behavior. By understanding a given cluster of customers, they can then form decisions regarding which products to recommend to customer groups using promotions and personalized offers. Outside of market research, clustering can also be applied to other scenarios, including pattern recognition, fraud detection, and image processing.

One of the most popular clustering techniques is  $k$  -means clustering. As an unsupervised learning algorithm,  $k$  -means clustering attempts to divide data into  $k$  number of discrete groups and is highly effective at uncovering new patterns. Examples of potential groupings include animal species, customers with similar features, and housing market segmentation.

**Original Data**



**Clustered Data**

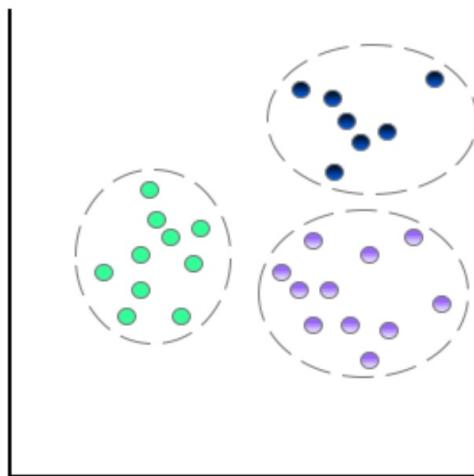


Figure 28: Comparison of original data and clustered data using  $k$ - means

The  $k$  -means clustering algorithm works by first splitting data into  $k$  number of

clusters, with  $k$  representing the number of clusters you wish to create. If you choose to split your dataset into three clusters, for example, then  $k$  should be set to 3. In Figure 28, we can see that the original data has been transformed into three clusters ( $k = 3$ ). If we were to set  $k$  to 4, an additional cluster would be derived from the dataset to produce four clusters.

How does  $k$ -means clustering separate the data points? The first step is to examine the unclustered data and manually select a centroid for each cluster. That centroid then forms the epicenter of an individual cluster.

Centroids can be chosen at random, which means you can nominate any data point on the scatterplot to act as a centroid. However, you can save time by selecting centroids dispersed across the scatterplot and not directly adjacent to each other. In other words, start by guessing where you think the centroids for each cluster might be positioned. The remaining data points on the scatterplot are then assigned to the nearest centroid by measuring the Euclidean distance.

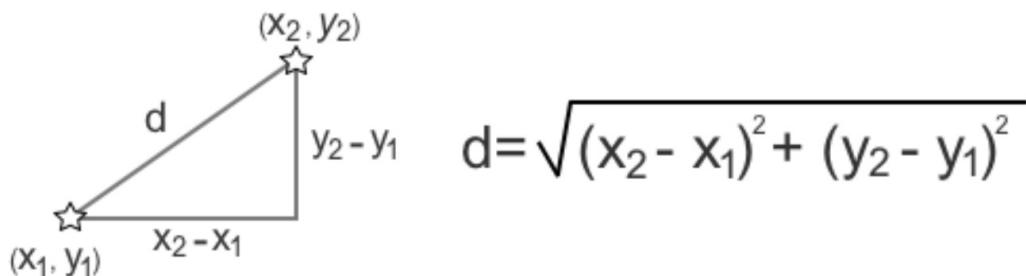


Figure 29: Calculating Euclidean distance

Each data point can be assigned to only one cluster, and each cluster is discrete. This means that there's no overlap between clusters and no case of nesting a cluster inside another cluster. Also, all data points, including anomalies, are assigned to a centroid irrespective of how they impact the final shape of the cluster. However, due to the statistical force that pulls nearby data points to a central point, clusters will typically form an elliptical or spherical shape.

After all data points have been allocated to a centroid, the next step is to aggregate the mean value of the data points in each cluster, which can be found by calculating the average  $x$  and  $y$  values of the data points contained in each cluster.

Next, take the mean value of the data points in each cluster and plug in those  $x$  and  $y$  values to update your centroid coordinates. This will most likely result in one or more changes to the location of your centroid(s). The total number of clusters, however, remains the same as you are not creating new clusters but rather updating their position on the scatterplot. Like musical chairs, the

remaining data points rush to the closest centroid to form  $k$  number of clusters. Should any data point on the scatterplot switch clusters with the changing of centroids, the previous step is repeated. This means, again, calculating the average mean value of the cluster and updating the x and y values of each centroid to reflect the average coordinates of the data points in that cluster. Once you reach a stage where the data points no longer switch clusters after an update in centroid coordinates, the algorithm is complete, and you have your final set of clusters.

The following diagrams break down the full algorithmic process.

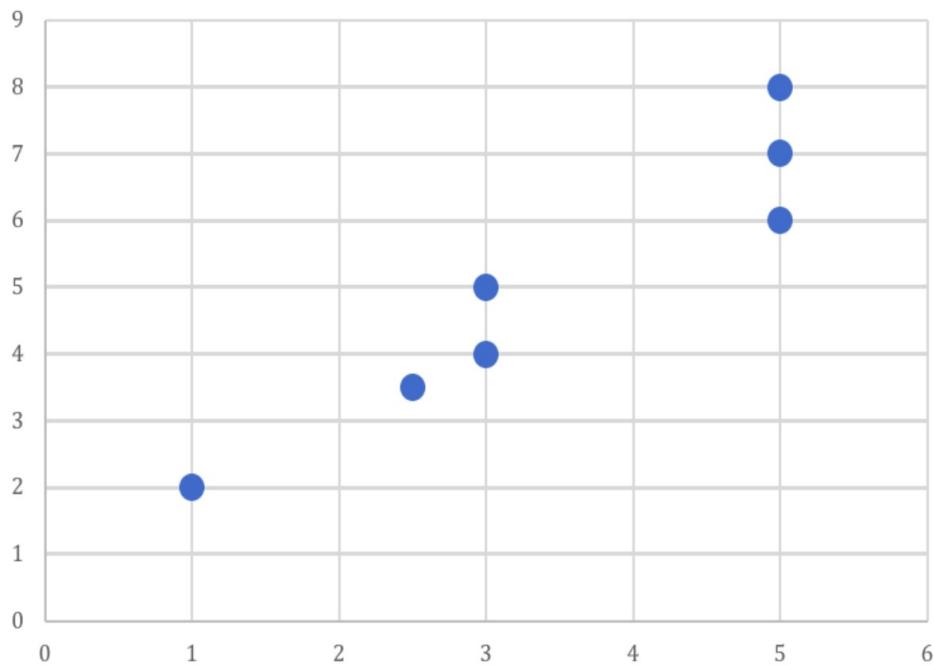


Figure 30: Sample data points are plotted on a scatterplot

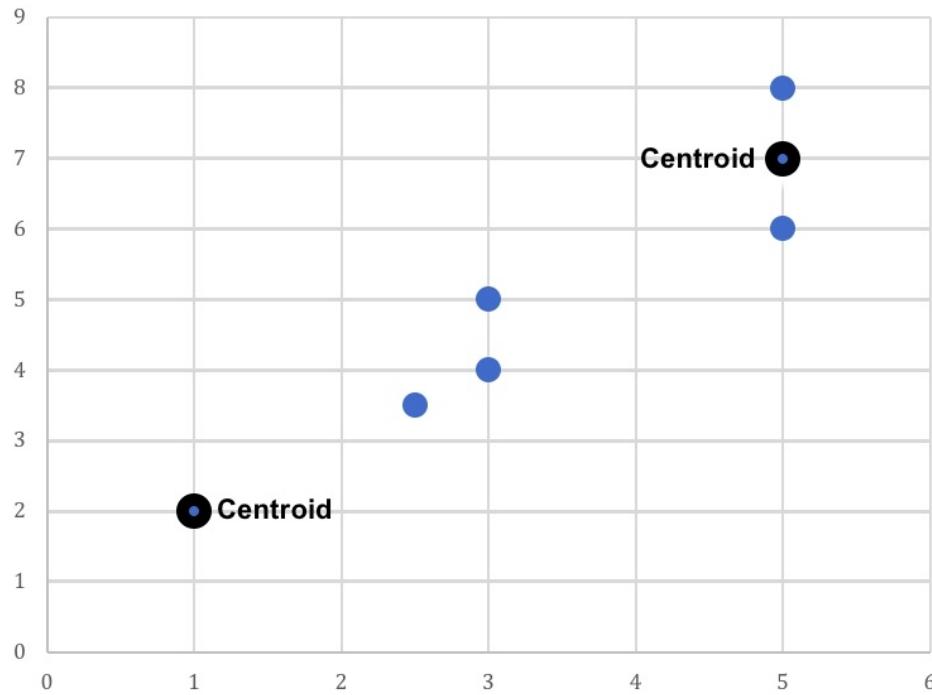


Figure 31: Two existing data points are nominated as the centroids

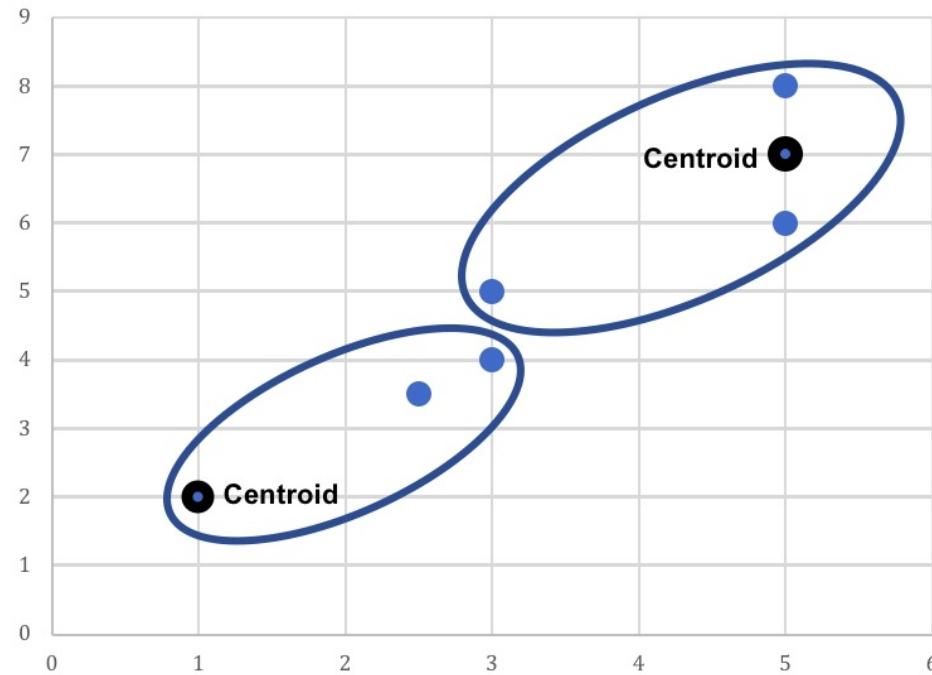


Figure 32: Two clusters are formed after calculating the Euclidean distance of the remaining data points to the centroids.

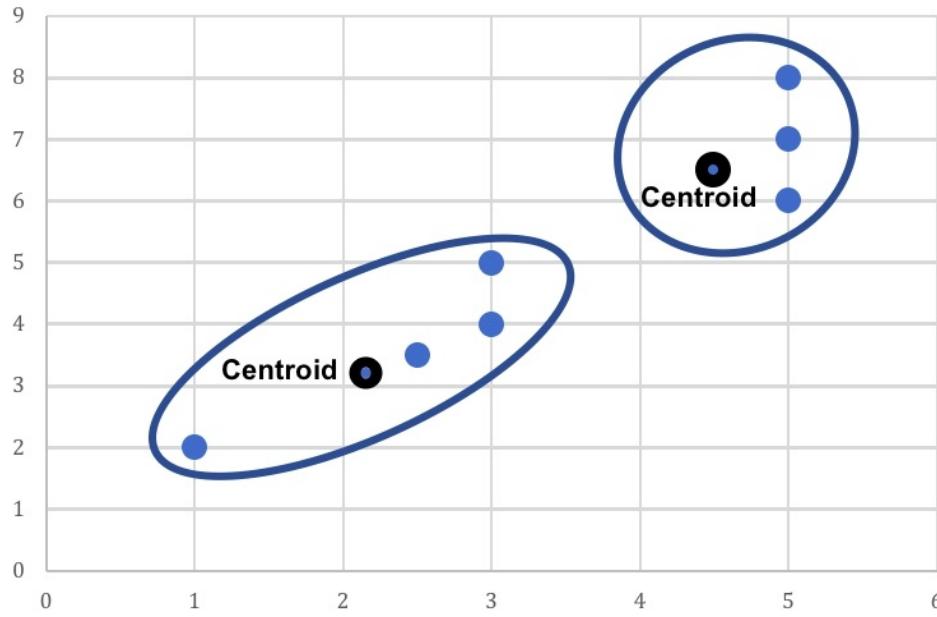


Figure 33: The centroid coordinates for each cluster are updated to reflect the cluster's mean value. The two previous centroids stay in their original position and two new centroids are added to the scatterplot . Lastly, as one data point has switched from the right cluster to the left cluster, the centroids of both clusters need to be updated one last time.

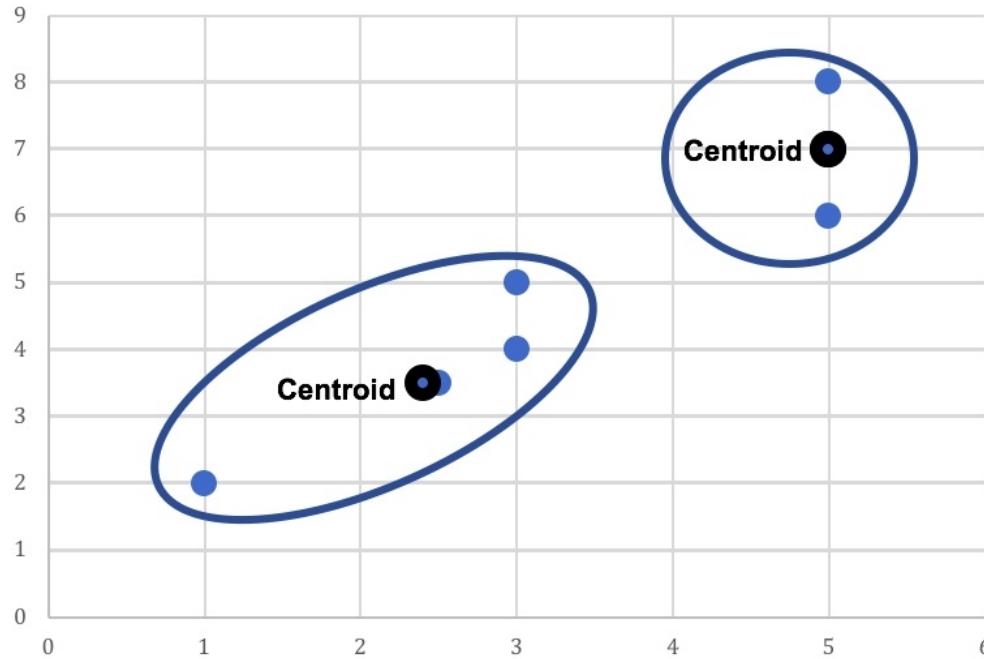


Figure 34: Two final clusters are produced based on the updated centroids for each cluster

For this example, it took two iterations to successfully create our two clusters. However,  $k$  -means clustering is not always able to reliably identify a final combination of clusters. In such cases, you will need to switch tactics and utilize

another algorithm to formulate your classification model.

Also, be aware that you may need to rescale the input features using standardization before running the  $k$ -means algorithm. This will help to preserve the true shape of the clusters and avoid exaggerated variance from affecting the final output (i.e. over-stretched clusters).

## Setting $k$

When setting “ $k$ ” for  $k$ -means clustering, it’s important to find the right number of clusters. In general, as  $k$  increases, clusters become smaller and variance falls. However, the downside is that neighboring clusters become less distinct from one another as  $k$  increases. If you set  $k$  to the same number of data points in your dataset, each data point automatically becomes a standalone cluster. Conversely, if you set  $k$  to 1, then all data points will be deemed as homogenous and fall inside one large cluster. Needless to say, setting  $k$  to either extreme does not provide any worthwhile insight.

In order to optimize  $k$ , you may wish to use a scree plot for guidance. A scree plot charts the degree of scattering (variance) inside a cluster as the total number of clusters increase. Scree plots are famous for their iconic “elbow,” which reflects several pronounced kinks in the plot’s curve. A scree plot compares the Sum of Squared Error (SSE) for each variation of total clusters. SSE is measured as the sum of the squared distance between the centroid and the other neighbors inside the cluster. In a nutshell, SSE drops as more clusters are produced.

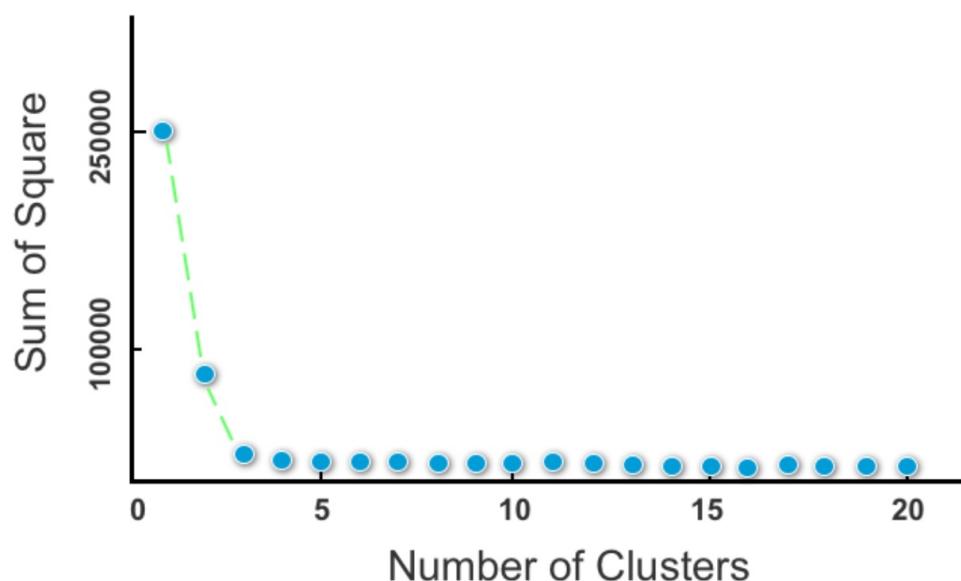


Figure 35: A scree plot