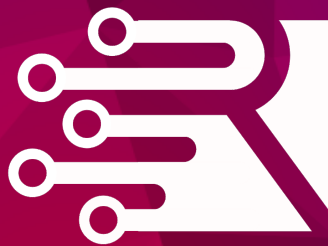




University of Texas at San Antonio

Rowdy Datathon
Supplementary Material



Juan B. Gutiérrez



©Juan B. Gutiérrez, Ph.D. Professor of Mathematics
University of Texas at San Antonio.
October 5, 2024

Preface

Data is everywhere, therefore data should be for everyone.

There is a constellation of resources that teach different aspects of data science, with promises ranging from learning skills instantly without mathematics, programming, or statistics, to mathematical methods that require a hopelessly long chain of prerequisites. Those resources fulfill specific needs and specialized niches. This book, however, attempts something different.

This book is intended for data analysts. We take a holistic approach trying to survey the most elementary building blocks in data analysis (quantitative methods), and their connection to the larger enterprise of data science. That is why we refer to the contents of this book as *Data Analytics*: the quantitative aspects of data analysis plus all else that makes data analysis possible.

The collection of knowledge skills and abilities required to become a competent data analyst is broad and difficult to acquire. It has foundational components related to mathematics, statistics, and computer science. It also has operational components related to information technology, scientific data analysis, and data engineering. Finally, there is a translational dimension related to ethical and legal considerations, as well as interdisciplinary collaborations.

The dominating paradigm in interdisciplinary education is inherently inefficient. It consists of teaching the same thing to each student at the same pace within a classroom with well defined initial and ending points analogous to a tree structure; however: (i) students in these settings usually come from different disciplines, thus have different (often non-overlapping) backgrounds, and (ii) curricula in interdisciplinary fields are comprised by subject matters drawn from different (often traditionally disconnected) areas. This book presents a model of experiential learning for data science in which the metaphor of the tree is replaced by a dense rhizome-like network that does not privilege a particular path, but instead offers a milieu

for traversal. In practice, it is the student during the learning process who makes an abstract knowledge network come to a unique realization.

There are multiple data projects associated with this book. They are referenced throughout the chapters, but are contained in different data volumes. Each data volume has to do with complex and heterogeneous public data in instances in which either data reveals an injustice, or the way data is collected and/or used is conducive to injustice. The rationale for this approach is that ethics problems, particularly related to injustice, tend to provoke a decisive response from students. Note that the concept of justice is rooted in specific moral systems, which vary in time and space. Within any given society there are groups that consider a given outcome just whereas a different group might consider the same outcome to be the apex of injustice. Experience has shown that most students are driven to learn about instances of injustice either to fight it, to try to debunk it, or to simply learn about the world. However, students must learn the techniques of data science to uncover the facts. We could call this method *“learning through passion.”*

A competent data analyst never claims to have the true answer; they claim to have a correct answer, that is, a solution and insight generated by following a well-documented and reproducible set of steps. They know that insight and knowledge can change in time as perspectives evolve, new data becomes available, or additional interpretation informs new routes of analysis. Informed data analysts recognize the pitfalls of introducing one’s values into data analysis or presentation, even when such values are aligned with the mission in which they find themselves in, and they take steps to make such biases explicit.

The material presented here assumes that students have foundational knowledge of linear algebra, calculus and computer programming. The first example presents a computer program to multiply matrices, not as step-by-step operations on rows and columns of matrices, but instead using defined operators. After that, all data analysis problems are error minimization problems, which necessitates the use of calculus. Ideally, the reader will know multivariate calculus, but given a foundation of calculus it is possible to acquire the basic multivariable skills here.

This book is programming language-independent. When a programming task is presented, examples will be presented in MAT-

LAB/Octave, Python, R, and C++. Data analysts are encouraged to learn in depth at least one programming language, and become familiar with as many programming languages as needed for the environment in which they will operate. Furthermore, a data analysis pipeline might be composed of elements created with different programming languages.

In closing, the author thanks the reader for attempting the voyage that follows. It might not be comfortable. The fact that injustice was chosen as the backbone to articulate concepts does not mean that we adopt a grim perspective of reality. Nor is this a book of lamentations. In borrowing from poetry, we embrace the Ulysses from Konstantinos Kavafis' "Ithaca" (1911), "*When you depart for Ithaca, wish for the road to be long, full of adventure, full of knowledge. Fear not the Laestrygonians and the Cyclopes, nor the angry Poseidon.*" Let us depart now from safe harbors.

Prof. Juan B. Gutiérrez,
San Antonio, Texas, 2023.

Contents

1	A Roadmap for Data Analytics	1
1.1	Data collection	3
1.2	Computational Machines	4
1.3	Bits, Bytes, and Floating Point	7
1.3.1	The IEEE Standard for Floating-Point Arithmetic	8
1.3.2	A Bestiary of Floating Point Anomalies	10
2	Non-coding Options for Data Analysis	13
2.1	Visualizing Individual Items on a Map using CODAP	15
2.2	Visualizing County Occurrences using CODAP and Excel	18
3	Introduction to Programming Environments	22
3.1	Introduction	22
3.1.1	Compiled Languages	22
3.1.2	Scripting Languages	23
3.1.3	Comparative Examples	24
3.1.4	What is Python?	25
3.1.5	Python Installation	26
3.1.6	Executing Python Commands	27
3.1.7	The VSC Python Editor: Executing A Simple Program	29
3.1.8	Executing Functions	32
3.2	Introduction to NumPy	33
3.2.1	Matrix Operations	35
4	Data from Web Pages	38
4.1	Introduction to HTML	38
4.2	Selecting HTML Elements Programmatically	39

4.3	Handling Challenges and Storing Data	40
5	Linear Operations with Data	42
5.1	Principal Component Analysis	42
5.2	Numerical Implementation of Principal Component Analysis	44
6	Configuration Management	46
6.1	Introduction to Configuration Management	46
6.2	Source Control	47
6.3	Software and Libraries	48
6.4	Environment variables and system configurations	49
6.5	Data Sources and Versions in Data Analysis	50
6.6	Algorithm and Model Parameters in Data Analysis	51
7	Processing Time Series	53
7.1	Time Series	53
7.2	Pseudoinverse	56
7.3	Fourier Transform	57
8	Basic Operations in SQL	61
8.1	Introduction to PostgreSQL	61
8.2	Basic operations in a RDMS	62
8.3	How to import a CSV file	63
8.4	Character Encoding. A Challenge Importing Data into a RDBMS	64
8.5	Handling Encoding Issues in PostgreSQL across Operating Systems	65
9	Using Relational Databases in Data Analysis	67
9.1	Sending SQL statements from Python	67
9.2	Accelerating SQL statements	68
10	Nonlinear Discriminants	70
10.1	Neural Networks	70
10.1.1	The single neuron	71
10.1.2	A Didactic Implementation	74
11	The Role of Generative AI in Data Analytics	75
11.1	Generative AI in the Life Cycle of Data Analytics	76
11.2	Enhancing Predictive Analytics	78

11.3 Conclusion 79

11.4 Using APIs 80

11.5 Strategies and Tactics to Use Generative AI 80

 11.5.1 Strategies for Enhanced Model Interaction . . 82

 11.5.2 Tactical Implementation 82

Chapter 1

A Roadmap for Data Analytics

What is Data Analytics? In the context of this book, we use this name to refer to data analysis (quantitative methods), and their connection to the larger enterprise of data science through other areas such as ethics, code versioning (or, more general, configuration management), solution architecture, reporting, etc. In brief, *Data Analytics* is the quantitative aspects of data analysis plus all else that makes that analysis possible.

Data analysts employs advanced mathematical, statistical and computational techniques to tackle problems, evaluates alternatives, and executes solutions. They consume extensive data sources to address questions through exploratory data analysis, probabilistic modeling, or predictive analysis. Effectively communicating insights to stakeholders requires statistically rigorous visualizations and narratives. The role on occasion demands algorithm design to unearth data insights. Additionally, quantitative methods and software tools are utilized to construct computer programs, while designing coherent data structures for data access and analytics needs. Ethical and legal considerations are imperative throughout a project's duration.

The typical components that define data analytics start with data collection, where raw data is gathered from diverse, often multi-modal (e.g. text, video, time series, etc.) sources. This is followed by the step of data cleaning, where inaccuracies and inconsistencies are weeded out. Once cleaned, the data is then explored to understand its structure, discern trends, and identify potential anomalies. The subsequent phase involves data modeling, where statistical or computational algorithms are applied to extract specific insights or predictions.

There are several categories in data analytics. Descriptive analytics focuses on shedding light on past events, answering the question of what happened. Diagnostic analytics delves deeper, probing why a particular event occurred. Predictive analytics, as the name suggests, forecasts future events or trends. Lastly, prescriptive analytics provides recommendations on actions that should be taken based on the data's insights.

A myriad of tools and technologies facilitate data analytics. Simple tasks may utilize spreadsheets such as Microsoft Excel or Google Sheets. For more intricate tasks, programming languages like Python and R are prevalent. Visualization tools such as Tableau and Power BI assist in representing data in understandable formats. Furthermore, databases like SQL and NoSQL variants (e.g., MongoDB, Cassandra) play a role in storing and querying vast amounts of data.

Data analytics finds applications in a vast array of sectors. In the business domain, it's pivotal for sales forecasting, customer segmentation, and inventory management. Healthcare professionals employ it for predicting disease outbreaks and patient outcomes. The finance sector leverages it for fraud detection and credit scoring. Even in sports, it's harnessed to analyze player performances and optimize game strategies.

However, the journey in data analytics isn't devoid of challenges. Data privacy and security concerns loom large, emphasizing the need for ethical data usage and protection. Ensuring data quality, i.e., its accuracy and consistency, remains a persistent challenge. Moreover, there's a palpable skill gap, necessitating the need for professionals who can analyze increasingly intricate datasets.

Looking ahead, the future of data analytics is promising and dynamic. Big data analytics is emerging as a field focusing on handling and analyzing exceptionally large datasets. Real-time analytics, which pertains to analyzing data as it is generated or procured, is gaining traction. Additionally, augmented analytics, which utilizes machine learning to automate insights, is another promising frontier.

Data analytics is inherently value-driven, vital for extracting value from vast datasets. It's an ever-evolving field, adapting and growing with technological and business advancements. Its applicability, spanning various sectors and industries, underlines its significance in today's data-driven world.

Describe in your own words what is data analytics and why and how it is different from data analysis. **Record your explanation as**

answer #1.

1.1 Data collection

The oldest known artifact is the Blombos ochre plaque, named after the Blombosfontein Nature Reserve, 300 km east of Cape Town on coastline of South Africa. In 2002, two pieces of ochre, an iron-rich mineral, were recovered exhibiting deliberate and complex geometric patterns dated to 70,000 to 100,000 years old [12]. Although we will never know for sure whether this particular instrument was used to count celestial events, livestock, time, or nothing at all, the following 80,000 years would see an increased level of sophistication in quantity representation, with varying degree of accuracy.



Figure 1.1-1: Engraved ochre found in the Blombos Cave, South Africa. This stone is suspected to be one of the earliest counting devices [7].

The concept and utility of data have evolved significantly throughout human history [17, 20, 18]. In ancient civilizations, data primarily existed in the form of physical tallies or rudimentary records. The ancient Sumerians, for instance, used cuneiform script on clay tablets to record trade transactions and inventories. With the invention of the printing press in the 15th century, data dissemination became more widespread, leading to an era of information sharing and record-keeping [9].

The 20th century ushered in groundbreaking advancements in data storage and computation. The invention of the electronic computer in the 1940s provided a platform for structured data storage and complex computations [2]. By the late 20th century, the rise of the internet and the proliferation of digital devices produced an explosion in the volume of data generated, stored, and analyzed. This digital revolution laid the groundwork for what is now known as “Big Data” [14].

Today, with the emergence of technologies such as the Internet of Things (IoT), artificial intelligence, and cloud computing, data generation and processing capabilities are expanding at an unprecedented rate. The evolution from rudimentary tallies to complex digital datasets has revolutionized various sectors, including business, healthcare, and science, to name a few [11].

1.2 Computational Machines

Humans have been using devices to aid in computations for millennia. A machine explicitly designated to aid computations was the abacus. Its exact origins are unknown, although there are clear archeological findings in Mesopotamia (2700-2300 BCE), Persia (600 BCE), Greece (300 BCE), and China (200 BCE). The latter, known as Suan Pan, is still in use today; methods have been developed to perform multiplication, division, addition, subtraction, square root, and cube root operations.

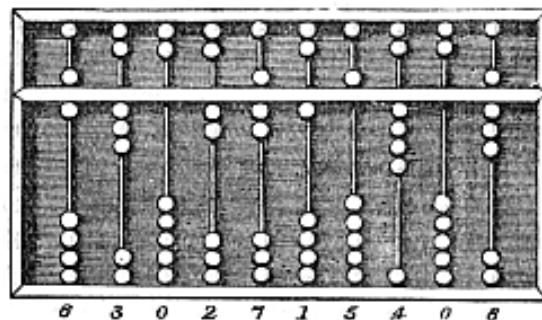


Figure 1.2-2: Suan Pan (the number represented in the picture is 6,302,715,408). This image appeared in the article for “abacus”, 9th edition Encyclopedia Britannica, volume 1 (1875) [6].

The Antikythera mechanism (150-100 BCE) was a machine designed to predict solar eclipses, along with the astronomical positions of several planets and the moon according to the theories of its day [10]. The results of the manipulation of this mechanism are now known to be inaccurate due to the limited understanding about planetary movement two millennia ago. The Antikythera mechanism is the first computing machine for which we can claim that computational errors rendered its use limited, whether the first users were aware of this limitation or not. The sophistication of the Antikythera mechanism was lost and would not occur again for at least

14 centuries, with the emergence of watch makers in Europe.



Figure 1.2-3: The Antikythera mechanism. This image shows the largest gear in the mechanism, approximately 140 millimeters (5.5 in) in diameter [5]. This mechanism is suspected to be the first mechanical computer.

The industrial revolution in Great Britain imposed demands on engineers to perform more and better calculations. This came with the certainty that dominion over the entire universe was within reach thanks to technification, and that calculations would lift off the veil of the mysteries of the universe. The zeitgeist was accurately captured by Pierre-Simon Laplace, in his *Essai philosophique sur les probabilités*, 1814 [13, pages 3-4]:

We may regard the present state of the universe as the effect of its past and the cause of its future. An intellect which at a certain moment would know all forces that set nature in motion, and all positions of all items of which nature is composed, if this intellect were also vast enough to submit these data to analysis, it would embrace in a single formula the movements of the greatest bodies of the universe and those of the tiniest atom; for such an intellect nothing would be uncertain and the future just like the past would be present before its eyes.

This paragraph, usually known as *Laplace's demon*, was the first mention of scientific determinism. It provides at the same time the notion of the computation of everything (later embraced by Konrad Zuse, see below), and its possible implausibility.

It was until 1822 CE that a fully mechanized computing machine was designed by Charles Babbage in Great Britain, 1822 [1]. Babbage's differential engine was intended to compute divided differences, an algorithm used at that time to calculate tables of logarithm and trigonometric functions. The design called the attention of governmental agencies, but Babbage was unable to deliver the promised product due to his interest in a more general analytical engine capable of more general operations. In the end, Babbage built neither but is commonly credited with the invention of both; this is, perhaps, the definition of privilege. The Difference Engine was finished in 1991 by the London Science Museum, and was shown in the exhibit *Computing* until September 1st, 2015 [15].

It took over a century for the next major development to occur. On the one hand, Alan Turing proposed in 1936 the concept of a computer of general purpose, what we call today a Turing Machine. Turing formalized the concepts of algorithm and computation. He worked in a computer used to decrypt German communications during World War II. Shortly after, Konrad Zuse created the Z1 in 1938 in his parents' living room in Hamburg. It was an electro-mechanical computer. The third version of this machine, named Z3 was the first programmable, fully automatic computer, with 2,000 relays, 22-bit word, operating at 10 Hz. Its development was suspended due to WWII. Zuse was the first to propose in 1967 that the entire universe is a computational machine; this is not far from Laplace's demon, although Zuse never credited him. Likewise, nearly half a century later, Stephen Wolfram proposed the idea of computational equivalence, not too different from Zuse and Laplace's ideas, but gave credit to neither. It seems that for some men, in matters of the universe, the larger they go the smaller they become.

Turing is often cited as the father of computer science, although the numerical representation used in Zuse's Z3 is nearly identical to the current standard for floating point operations [24], as described later in this chapter.

World War II catalyzed a fast advance of computing. International Business Machines' (IBM) Automatic Sequence Controlled Calculator (ASCC), known as Mark by Harvard staff (hence the name Harvard Mark I, II, III, and IV), was completed in 1944. The ENIAC is regarded as the first electronic computer. It was completed in 1946 for the US Army, Ordnance Corps.

The age of modern computing starts with the IBM System/360

(S/360), a family of commercial computers sold between 1965 and 1978. It was important because it standardized the use of 8 bits = 1 byte as the unit of memory, as described in the next section. Shortly after, Intel launched the first 8-bit microprocessor, named the 8008, in 1972; it was followed by the first 16-bit microprocessor, named the 8080, in 1974. In 1978, Intel released the 8086, a 16-bit processor that gave rise to the x86 architecture. In 2003, AMD launched the first x86 64-bit processor. As of 2017, the vast majority of computers (personal and servers) are based on the x86 architecture. Phones and tablets use a different architecture, but also based on either 32 or 62-bit platforms.

1.3 Bits, Bytes, and Floating Point

Computers have a finite number of ways to represent numbers. That is, they can only approximate a small subset of real numbers. Understanding this is an essential component of conducting numerical operations: You need to understand how and why numerical errors accumulate, and when they interfere with a calculation.

The unit of computing representation is a **bit** (unit represented by b), i.e. a circuit that is either open or closed. Eight bits are grouped into one **byte** (unit represented by B), as shown in Figure 1.3-4. This number evolved from computer architectures in the early days of computing, and was first standardized by ISO/IEC 2382-1:1993. In one byte, positions go from 0 to 7, usually represented from right to left. A 1 in the first position represents 2^0 , in the second position is 2^1 , and so on.

A naive way to represent numbers with a computer is to allocate N decimal digits, for each number. For example:

012.345

This example represents a 6-digit fixed-point system. This 6-digit fixed-point system results in errors in some sum cases:

$$999.999 + 999.999 = \text{OVERFLOW}$$

Also errors occur in multiplication and division

$$013.000/003.000 = 004.333$$

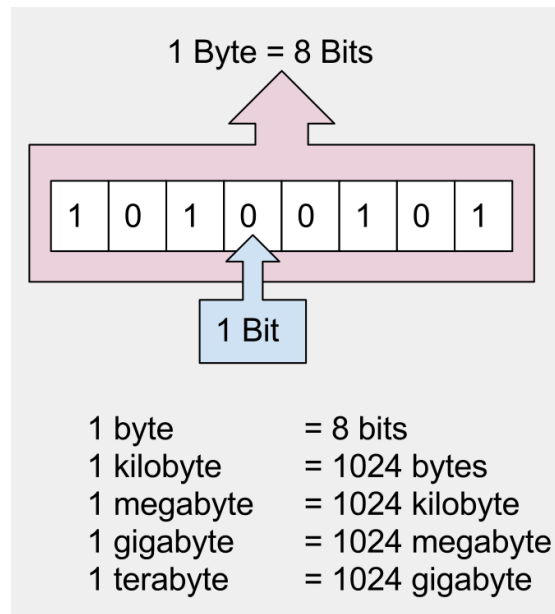


Figure 1.3-4: Bits and Bytes [8]. Reading from left to right, the example represents: $2^0 + 2^2 + 2^5 + 2^7 = 1 + 4 + 32 + 128 = 165$. If you read from right to left (this is the standard way to represent bits), you would obtain $2^0 + 2^2 + 2^5 + 2^7 = 165$. In this case, left-to-right and right-to-left produce the same result. But in general they don't have to be the same. What happens if you replace any zero with a one, and compute both directions again? Thus, representation matters, and you need to know the convention.

In this case, we lose all digits after the third decimal.

We can allow the decimal point to “float” to gain accuracy with the same number of digits. For example:

$$013.000/003.000 = 4.33333$$

We could even represent larger numbers than 6 digits allowed us before:

$$999.999 \times 999.999 = 9.99998 \times 10^5$$

1.3.1 The IEEE Standard for Floating-Point Arithmetic

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE).

- Finite numbers may be either base 2 (binary) or base 10 (decimal).

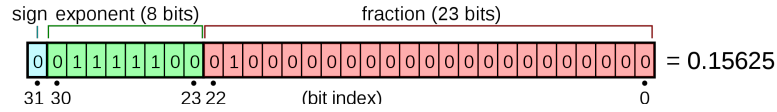


Figure 1.3-5: Representation of a single precision number [3]

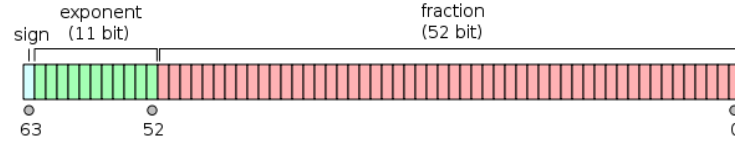


Figure 1.3-6: Representation of a double precision number [4].

- Each finite number is described by three integers:
 1. s = a sign (zero or one),
 2. c = a significand (or ‘coefficient’),
 3. q = an exponent.
- The numerical value of a finite number is $(-1)^s \times c \times b^q$ where b is the base (2 or 10), also called radix.

The three fields in an IEEE-754 float. Binary floating-point numbers are stored in a sign-magnitude form where the most significant bit is the *sign* bit, *exponent* is the biased exponent, and *fraction* is the significand minus the most significant bit. Figure 1.3-5 shows a 32-bit number, also known as a single-precision floating number, whereas Figure 1.3-6 depicts a 64-bit number, also known as double-precision floating number.

If we want to represent a real number with a finite number N of memory positions, we can reserve one position for the sign, $N - k - 1$ for the integer part, and k for the digits beyond the separator or fractional part. The notation conventionally adopted is [19]:

$$x = (-1)^s [a_{N-2}a_{N-3}\dots a_k.a_{k-1}a_{k-2}\dots a_0] = (-1)^s \cdot \beta^{-k} \sum_{j=0}^{N-2} a_j \beta^j \quad (1.1)$$

where β is the base of the representation (it could be any number, but usually it is 2 or 10), s is either 1 or 0, N is the number of digits in the significand, a_j is the j^{th} digit, k is the exponent.

The floating point representation in equation 1.1 limits the minimum or maximum number unless some scaling is allowed. In such

case, a *floating point* representation is given by

$$x = (-1)^s \cdot [0.a_1a_2\dots a_t] \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t} \quad (1.2)$$

where $t \in \mathbb{N}$ is the number of significant digits a_i with $0 \leq a_i \leq \beta - 1$, $m = a_1a_2\dots a_t$ is an integer number called *mantissa* with $0 \leq m \leq \beta^t - 1$ and e is an integer number called *exponent* with $L \leq e \leq U$ (typically $L \leq 0$ and $U \geq 0$). If we require $a_1 \neq 0$ and $m \geq \beta^{t-1}$ then the representation in equation 1.2 is called *normalized*.

The term *mantissa* in computer science commonly refers to the significant digits of a floating-point number, essentially synonymous with *significant*. This usage diverges from its historical mathematical meaning, where *mantissa* denoted the fractional part of a logarithm. Notably, among the many authors that followed John Napier's early logarithmic work, "*Mirifici Logarithmorum Canonis Descriptio*" (1614) [16], it was John Wallis who used the word "*appendage*" in the English version of "*A Treatise of Algebra*," [22, p. 38] (1658), translated into Latin as "*mantiffam*" (in modern form *mantissa*) [23, p. 41] (1681) (see Figure 1.3-7), to describe the decimal part of a logarithm, not the logarithm of a significand. This distinction highlights the evolution of mathematical terminology over time.

1.3.2 A Bestiary of Floating Point Anomalies

One of the most surprising aspects of floating-point arithmetic is that simple operations do not result in expected results. For instance, the addition $1 - 1/3 - 2/3$ results algebraically in zero. However,

$$1 - 1/3 - 2/3 = 1.1102e - 16,$$

$$1 - 2/3 - 1/3 = 5.5511e - 17.$$

Thus, not only the operation is not zero, but also the order of summands matter, i.e. floating point addition is not commutative. Hence, $1 - 1/3 - 2/3 \neq 1 - 2/3 - 1/3 \neq 0$

Because floating point numbers cannot represent all real numbers, errors of representation occur in many cases. But why is the operation noncommutative? **Record your explanation as answer #2.**

The following instruction compares whether $10^{30} + 1 - 10^{30}$ equals the value 1. Using elementary arithmetic, the answer should be a

Cap. X.

ALGEBRA.

41

variatur. Et quidem Augetur, si majore multiplicatore multiplicetur Numerator, quam Denominator: Minuitur, si contra.

Sive (quod eodem recidit)

Si Fractionis duo termini, (Numerator & Denominator,) ita (addendo) augentur ambo, ut augmenta sint ipsis terminis proportionalia; idem qui prius manet fractionis Valor: Augetur vero, si Numeratoris augmentum, ad augmentum Denominatoris, majorem rationem habeat, quam habet ipse Numerator ad Denominator: Minuitur, si minorem.

Quodque de Numeratore & Denominatore Fractionis dictum est, similiter intelligendum est de Rationis Antecedente & Consequente, & pariter in sequentibus.

$$\begin{array}{l}
 \text{Put, } \frac{1}{3} = \frac{1 \times 2}{3 \times 2} = \frac{2}{6} = \frac{3}{9} = \frac{4}{12} \text{ & c. } \quad \frac{n}{d} = \frac{2n}{2d} = \frac{3n}{3d} = \frac{an}{ad} \\
 \text{Sen, } \frac{1}{3} = \frac{1+1}{3+3} = \frac{1+2}{3+6} = \frac{1+3}{3+9} \text{ & c. } \quad \frac{n}{d} = \frac{n+n}{d+d} = \frac{n+2n}{d+2d} = \frac{n+3n}{d+3d} \\
 \text{Sed, } \frac{1}{3} < \frac{1 \times 3}{3 \times 2} = \frac{3}{6} = \frac{1+2}{3+3} \quad \frac{n}{d} < \frac{3n}{2d} = \frac{n+2n}{d+d} \\
 \text{Et, } \frac{1}{3} > \frac{1 \times 2}{3 \times 3} = \frac{2}{9} = \frac{1+1}{3+6} \quad \frac{n}{d} > \frac{2n}{3d} = \frac{n+n}{d+2d}
 \end{array}$$

S O L U T I O.

His præmissis; si datæ fractionis, ad minimos terminos (communi divisione per maximam communem mensuram) reductæ, tum Numerator, tum Denominator, dato numero non sit major: fit quod imperatur. Quippe Fractio est in minimis terminis, & qualis requiritur.

Si Fractio, sic reductæ, terminus vel alter vel uterque sit exposito numero major: querenda est fractio quæ sit quam exposita vel proxime major, vel proxime minor, terminos habens exposito numero non majores; ea quæ sequitur methodo.

Pro Proxime-Majori.

Pro invenienda proxime majori, sic procedo.

Denominatorem fractionis expositæ (aut ejus ad quam reducitur, si præcesserit reductio,) per Numeratorem divido; ut habeam, qui Numeratori 1 respondeat, Denominator, in integris cum annexis partibus decimalibus, tam accuratum quam videbitur expedire.

Put, 2684769) 8376571 (3.12003416 +.

Nam, ut 2684769 ad 8376571, sic 1 ad 3.12003416 +.

Adeoque, pro exposita fractione $\frac{8376571}{2684769}$, hanc habeo ei (quam prope censetur necessarium) æquipollentem $\frac{8376571}{2684769} +$ quæ numeratorem habet 1. Quam Fractionem primam completam appello; eandemque, neglectis partibus decimalibus, appello fractionem primam truncatam. Ejusque partes decimales abscissas, 0.12003416, Appendicem voco, sive Mantissam fractionis primæ, seu Mantissam primam.

Estque hæc truncata Fractio $\frac{1}{3}$, tum justo major (propter denominatorem justo minorem, utpote partibus decimalibus mulctatum;) tum proxime-major, omnium habentium numeratorem 1, & denominatorem integrum, (nam $\frac{1}{3}$ est adhuc major, & $\frac{1}{4}$ justo minor, & de reliquis similiter:) Neque, ex justo majoribus, propius ad verum valorem accedere potest ullus, nisi aucto numeratore 1.

F

Loco

Figure 1.3-7: Page 41 of "A Treatise of Algebra," [22, p. 38] (1658), where the word mantissa is introduced.

resounding *yes*. In the Python language, the exponent is represented by a double asterisk, like in Fortran, instead of the carat sign (^) used in most other computing platforms. The double equal sign tests whether the two sides are equal, and returns a Boolean value of *true* or *false*. Hence, the question is translated into the following instruction:

```
10**30 + 1 - 10**30 == 1
```

This particular instruction returns a Boolean value of *true* in Python and *false* in Matlab (the correct Matlab syntax is $10^{30} + 1 - 10^{30} == 1$). However, the following statement:

```
10**30 + 1. - 10**30 == 1
```

returns *false* in Python. Why? **Record your explanation as answer #3.**

An operation like

```
10**20 + 1 - 10**20 == 1
```

returns different results in Matlab, C, Fortran (*false*) vs. Python (*true*). It might be tempting to declare Python the winner in numerical computation, but what truly happens is that Python does not follow the IEEE-754 standard strictly. To learn about more subtleties of floating point in Python, go to <https://docs.python.org/3/tutorial/floatingpoint.html>. In most cases, deviations from the standard in the Python interpreter are harmless and have an ideological origin rather than a technical one. However, it is conceivable that extreme algorithms developed under the IEEE standard would have to be carefully adapted in Python. There are alternatives, but they are beyond the scope of this chapter. For now, it suffices to say that there are landmines in the Python landscape.

We can explore further discrepancies in extreme floating point values by exploring in two numerical environments the results of $10 \cdot 10^b + 1 - 10 \cdot 10^b$ for different values of b . Download and install Python, Matlab, R, or compile a C/C++ program that executes the same operations. Compare what happens in these environments when you choose $b \in \{10, 1e2, 1e10, 1e100, 1e1000\}$. Describe the discrepancies, and analyze. **Record your explanation as answer #4.**

Chapter 2

Non-coding Options for Data Analysis

There are numerous non-coding alternatives available for conducting data analysis. These tools offer graphical user interfaces (GUIs) that enable users to perform data analysis without the need to write code. They are popular among individuals who may not have coding experience or who prefer a more visual approach. Here are some of the prominent non-coding alternatives for data analysis:

1. Microsoft Excel: It is one of the most widely-used tools for basic data analysis. Offers functions for statistics, data visualization, and data manipulation. Pivot tables allow for powerful data summarization and aggregation.
2. Google Sheets: Similar to Excel but Web browser-based. Provides data analysis functions, charting, and collaboration features.
3. Tableau: A powerful data visualization and business intelligence tool. It enables users to create interactive dashboards and visualizations. It can connect to a variety of data sources.
4. Power BI: Microsoft's business analytics tool. Offers data visualization, report sharing, and enterprise-level capabilities. Integrates well with other Microsoft products.
5. SPSS: Statistical software widely used in social sciences and marketing research. Provides a GUI for data manipulation, statistical tests, and modeling.

6. QlikView/Qlik Sense: Data visualization and business intelligence tools. Known for their associative data modeling and in-memory data processing, allowing for interactive dashboards.
7. Knime: Open-source data analytics, reporting, and integration platform. Uses a modular data pipelining concept, allowing users to visually create data flows.
8. RapidMiner: A platform for data science and machine learning. Offers a visual workflow designer for data preprocessing, modeling, and deployment.
9. Stata: Statistical software widely used in economics and social sciences. While it has a coding aspect, many of its functions can be accessed and executed via menus and dialogs.
10. SAS Enterprise Guide: GUI-based interface to the SAS system. Allows users to conduct advanced analytics, business intelligence, and data management without writing SAS code.
11. CODAP (Common Online Data Analysis Platform): Web-based platform for data exploration primarily used for educational purposes. Offers interactive, dynamic, and visual interfaces for data analysis.
12. TIBCO Spotfire: Data visualization and analytics software. Supports creating dashboards, analytical applications, and sophisticated visualizations without coding.

Many of these tools provide a combination of drag-and-drop interfaces, menus, and dialogs that abstract away the coding aspect. While they are user-friendly and can be powerful, it's essential to understand that the flexibility and customization potential might be limited compared to coding-based solutions. For advanced analyses or specific tasks, coding might still be necessary, even with these tools. However, they offer a great starting point and cover a wide range of analytical needs for many users.

2.1 Visualizing Individual Items on a Map using CODAP

CODAP (Common Online Data Analysis Platform) is a free, open-source platform for exploring and analyzing data, particularly for educational purposes. It's web-based, so users don't need to install anything to use it. CODAP is designed with a focus on dynamic, interactive, and visual interfaces to support students in learning data analysis and statistics.

CODAP offers draggable data points, sliders, and other dynamic features that make it engaging for learners to play with data. While it can be used in various educational contexts, one of its primary goals is to support science and math education in middle and high schools.

Teachers and educators often use CODAP as an integral part of their curriculum to help students gain a hands-on understanding of data analysis concepts and techniques.

To visualize a list of items on a map using a list of coordinates in CODAP, follow these steps:

1. Importing the Data

- (a) Open the CODAP website and create a new document.
- (b) If you have your data in an external file (like a CSV), drag and drop it into CODAP, or use the "Import Data" feature. Your data should include columns for latitude, longitude, and any other attributes of the items you want to display.

2. Organizing the Data

- (a) Ensure that your dataset is visible in the data table view (Figure 2.1-1). It should appear as a table with rows representing each item and columns representing attributes like latitude, longitude, etc.

3. Adding a Map

- (a) From the toolbar at the top, click the "Map" button to add a new map to the document (Figure 2.1-2).
- (b) The map should automatically recognize the latitude and longitude columns and begin to plot points for each item.

Index	LEAID	NAME	OPSTFIP	STREET	CITY	STATE	ZIP	STFIP	CNTY	NMCNTY	LAT
18824	5900183	Via Rio L...	59	11110 CO...	Cuyamaca	VIA	96313	59	59067	11110 RIO...	
18825	5900186	Casa BL...	59	P.O. Box...	Bapchu...	AZ	85121	4	4021	Pinal C...	
18826	5900187	Hannah...	59	NI4911...	Wilson	MI	48896	26	26109	Menom...	
18827	5900188	Beclabl...	59	P.O. Box...	Shiprock	NM	87420	35	35045	San Jua...	
18828	5900189	Mandar...	59	P.O. Box...	Mandar...	ND	58757	38	38053	McKen...	
18829	5900190	Tiospay...	59	HC 76 B...	Ridgevi...	SD	57652	46	46041	Dewey...	
18830	5900191	Tohajili...	59	P.O. Box...	Tohajili...	NM	87026	35	35001	Bernal...	
18831	5900192	Cila Cro...	59	4465 W...	Leaven	AZ	85339	4	4013	Maricop...	
18832	5900193	Saba D...	59	HC 63 B...	Winslow	AZ	86047	4	4017	Navajo...	
18833	5900194	Bogue...	59	11241 H...	Philadel...	MS	39150	28	28099	Neshob...	
18834	5900195	Sky City...	59	P.O. Box...	Pueblo...	NM	87034	35	35006	Cibola...	
18835	5900196	Meskuwa...	59	1608 SO...	Tama	IA	52339	19	19171	Tama C...	
18836	5900197	Noli Sc...	59	P.O. Box...	San Jac...	CA	92581	6	6065	Riversid...	
18837	5900200	Jones A...	59	HCR 74...	Hartsho...	OK	74547	40	40121	Pittsbur...	
18838	6000030	Americ...	60	Box DOE	Pago Pa...	AS	96799	60	60010	Eastern...	
18839	6600002	Guam D...	66	500 Ma...	Barriga...	CU	96913	66	66010	Guam	
18840	7200030	PUERT...	72	CALLE F...	HATO R...	PR	919	72	72127	San Jua...	
18841	7800002	Saint Cr...	78	2133 Ho...	Saint Cr...	VI	820	78	78010	St. Cro...	
18842	7800030	Saint Th...	78	386 An...	Saint Th...	VI	802	78	78030	St. Tho...	

Figure 2.1-1: CODAP's Data Table View

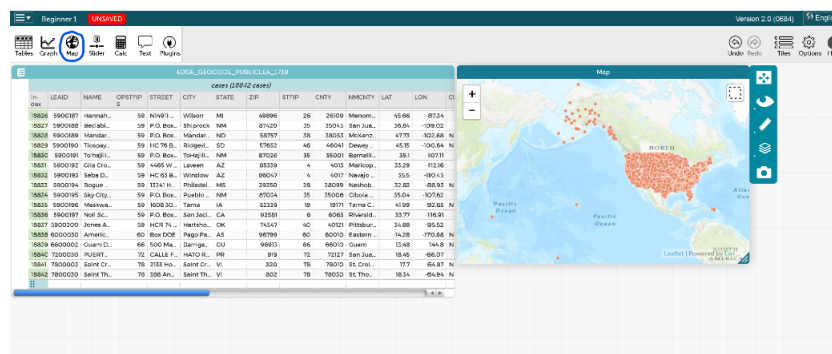


Figure 2.1-2: Adding a Map in CODAP

- (c) If the map doesn't automatically recognize your coordinates, manually set the attributes for latitude and longitude by following the instructions below to drag the 'LEAID' feature onto the map.

4. Mapping the Data

- (a) Drag the feature you wish to graph (click and drag the feature name like 'State' in Figure 2.1-3) onto the map.

5. Customizing the Map

- (a) Using the settings (the panel on the right side of the map), you can adjust the appearance of the points, set colors based on attributes, add labels, and more. This is shown in Figure 2.1-4.

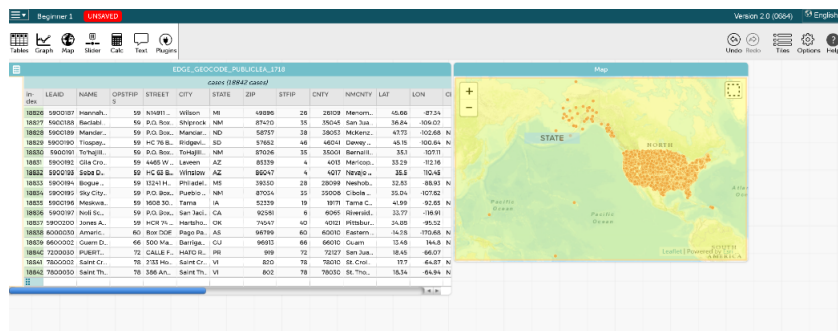


Figure 2.1-3: Graphing a Feature from a Data Table on a Map in CODAP

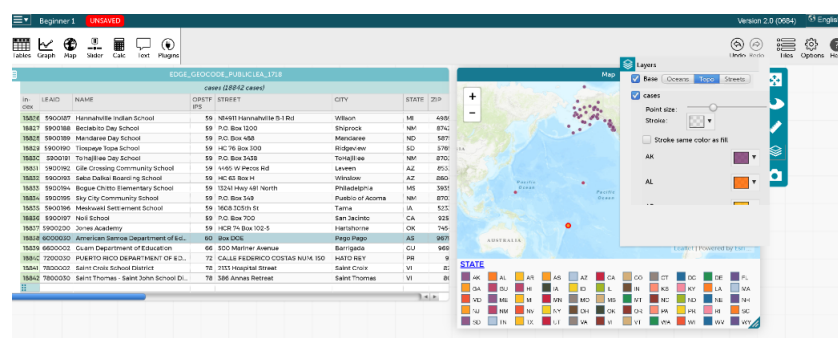


Figure 2.1-4: Customizing a Map in CODAP

- (b) If your data includes other attributes (like categories or numerical values), you can often use these to change the appearance of the data points on the map - for example, different colors for different categories.

6. Exploring and Interacting

- (a) You can zoom in and out of the map, click on individual data points to see more details, and even filter the data to only show certain items on the map. For example, you can see in Figure 2.1-4, a data point in America Samoa (near the bottom of the map) has been selected and the corresponding row in the data table is highlighted.
- (b) If your dataset includes time data, you can also create animations or sliders to see how the items change over time on the map.

CODAP is designed to be interactive, so it's encouraged to play around and explore different visualizations and settings to get the most insight from your data.

2.2 Visualizing County Occurrences using CODAP and Excel

A common task is to visualize quantities in the US by county. For this example, assume that you have an Excel file containing the columns **CNTY** and **STFIP**, which represent the county and state FIPS code.

The Federal Information Processing Standard Publication 6-4 (FIPS 6-4) was a five-digit Federal Information Processing Standards code which uniquely identified counties and county equivalents in the United States, certain U.S. possessions, and certain freely associated states.

FIPS codes are numbers which uniquely identify geographic areas. The number of digits in FIPS codes vary depending on the level of geography. State-level FIPS codes have two digits, county-level FIPS codes have five digits of which the first two are the FIPS code of the state to which the county belongs.

The US Department of Commerce announced in 2008 that FIPS 6-4 was one of ten FIPS standards withdrawn by the department's National Institute of Standards and Technology (NIST). Deemed obsolete, NIST replaced FIPS 6-4 with "INCITS 31 - 2009" codes. Many existing databases use FIPS codes; some newer databases use INCITS. You need to be aware of both.

1. Preparing the Excel File

- (a) Open your Excel file.
- (b) Make sure there are no blank rows or columns.
- (c) Save the file as a **.csv** (comma-separated values) file if it's not already in that format. You can do this by selecting **File > Save As**, choosing the **.csv** format from the dropdown list, and then clicking 'Save'.

2. Importing the Data into CODAP

- (a) Navigate to the CODAP website.
- (b) Click on **File > New Document** to start with a blank document.

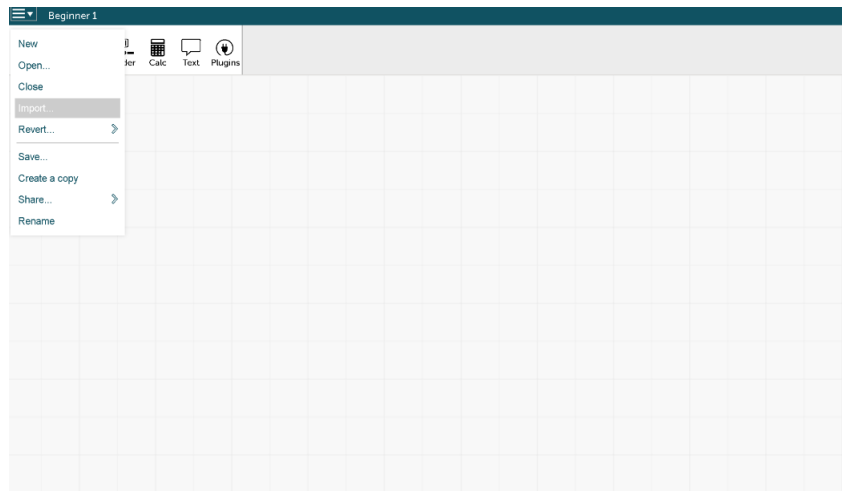


Figure 2.2-5: Importing Data into CODAP

- (c) Click on the button with three horizontal lines (usually on the top left corner of the screen).
 - (d) Choose **Import Data** as shown in Figure 2.2-5.
 - (e) Choose your `.csv` file and import it.
3. Counting the Number of Rows for Each County
- (a) Once your data is loaded, you should see the data table with the columns **CNTY** and **STFIP**.
 - (b) Click on the top of the **CNTY** column to select it.
 - (c) Drag the **CNTY** column to the leftmost part of the data table (as shown in Figure 2.2-6). This will group the data by county.
 - (d) A new attribute (or column) will appear, possibly named **Count**, showing the number of occurrences of each county.
4. Plotting the Data
- (a) Click on **Graph** on the toolbar on the upper lefthand side of your screen.
 - (b) Drag the **CNTY** attribute to the x-axis. The result is shown in Figure 2.2-7
 - (c) Drag the **CNTY** attribute to map. The counties will now be colored based on the number of occurrences, with the color scale reflecting the data distribution.

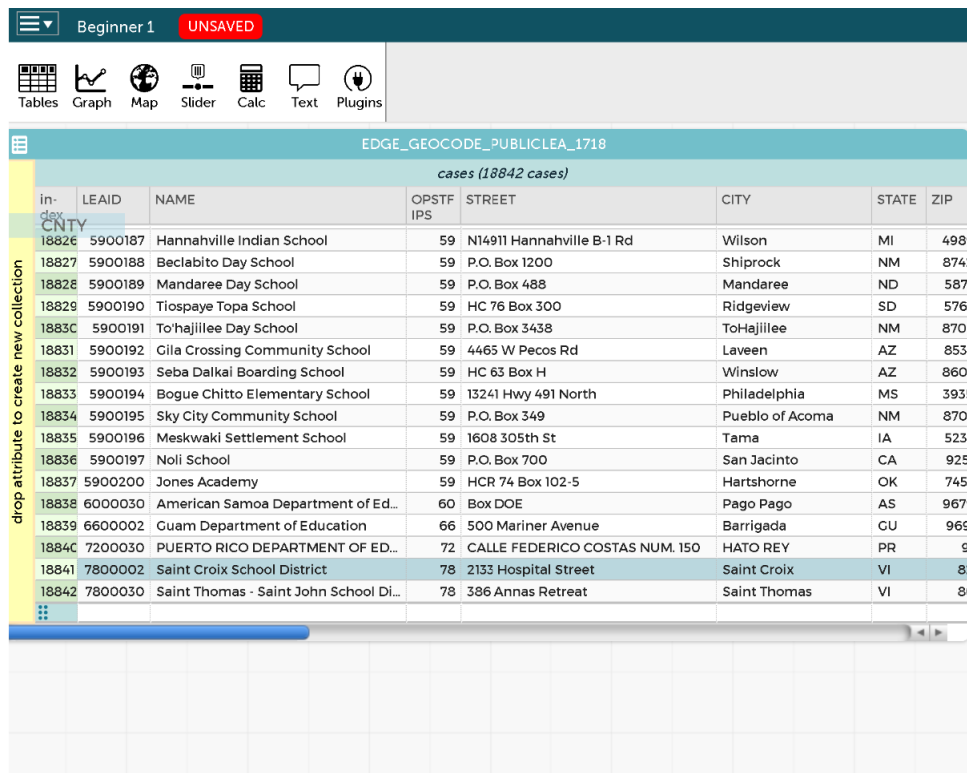


Figure 2.2-6: Create a Collection in CODAP by Dragging the Feature to the Left of the Table

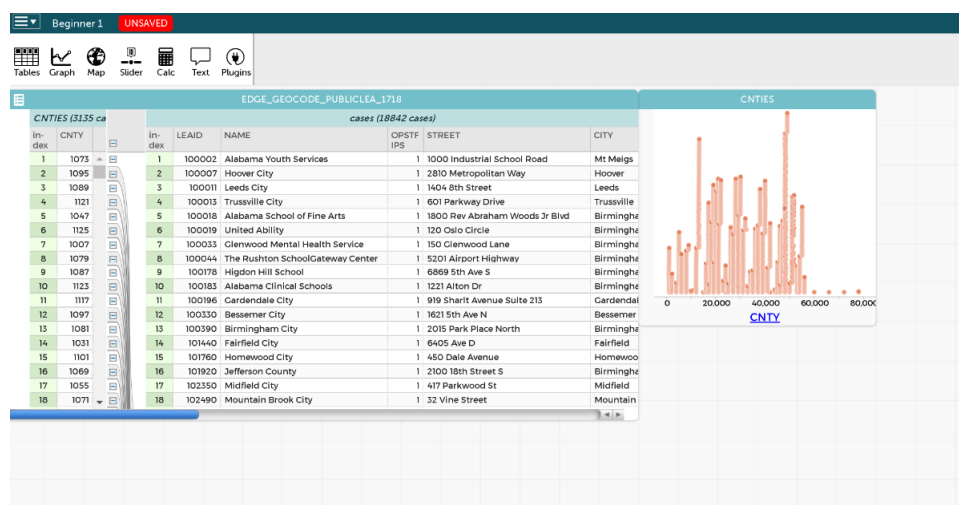


Figure 2.2-7: Create a Graph Showing a Distribution in CODAP by Dragging the Field Containing Counts onto a Graph Object

5. Customizing the Visualization (Optional)

- (a) You can further customize the graph by adjusting axis labels, color scales, etc., using the various options available in CODAP's interface.
- (b) There are also many ways to achieve this count of counties. Another method is to create a new "Count" feature using a function defined in the CODAP Documentation and graph the "Count" feature.

Chapter 3

Introduction to Programming Environments

3.1 Introduction

There is a large number of programming languages. For the purpose of data analytics, we will distinguish two main families of languages: compiled vs. scripting languages. They represent two different paradigms of programming language design, each with its own advantages and trade-offs. The key difference lies in when and how the source code you write is translated into machine code that the computer can understand and execute.

In the realm of data analytics, the choice between using a compiled language like C++ and a scripting language like Python often comes down to the specifics of the project at hand, including its complexity, required execution speed, and the need for direct hardware access.

3.1.1 Compiled Languages

Compiled languages are those where the source code is entirely translated into machine code before execution. This translation is done by a compiler, a program that takes the entire source code as an input and generates an executable file. Examples of compiled languages include C++, Java, Rust, Go, and Swift.

Compiled languages offer several important benefits including:

- **Performance:** Since the code is precompiled into machine code, compiled programs usually run faster and more efficiently than

interpreted ones.

- **Type checking:** Many compiled languages have strict type systems, which catch many type-related errors at compile time, before the code is run.
- **Security:** The source code is not distributed with the software, which makes it harder for others to reverse engineer the software.

However, compiled languages also have some drawbacks:

- **Development speed:** The edit-compile-run cycle can be time consuming, particularly for large software projects.
- **Portability:** The generated executable is usually specific to a type of processor and operating system. Availability for different operating systems or processors might require specific source code alterations and independent compilations.

Compiled languages like C++ are translated into machine code before execution, making them very fast because they're executed directly by the computer's hardware. This can be a significant advantage when dealing with huge datasets or when computation speed is a critical factor.

However, the syntax of languages like C++ is more complex and it may require more lines of code to achieve the same result as a scripting language. Moreover, C++ lacks many of the built-in data analysis functionalities found in languages such as Python, MATLAB or R, so it's necessary to use external libraries like Eigen or Armadillo for matrix operations and other complex computations.

3.1.2 Scripting Languages

Scripting languages, on the other hand, are typically interpreted from text into machine instructions, line-by-line of code just in time for execution. While this can make scripting languages slower than compiled languages, they're often easier to write and read because of their high-level syntax. Examples of scripting languages include Python, MATLAB, R, JavaScript, Ruby, and PHP.

Scripting languages offer several advantages:

- Ease of use: Scripting languages are usually easier to learn and use. They have less verbose syntax and more built-in functionality.
- Flexibility: Since they are interpreted line-by-line, it's easier to make changes to your code and see their effect immediately without having to recompile.
- Portability: The same script can be run on any system that has the correct interpreter installed, without needing to change the code.

Among the drawbacks of scripting languages there are issues related to:

- Performance: Interpreted languages are generally slower than compiled languages because of the overhead of interpreting code line-by-line during execution.
- Dependency on the interpreter: The code can only be run on systems that have the correct interpreter installed.

Python is particularly well-suited for data analytics because it has numerous powerful libraries for data analysis (like Pandas, NumPy, and SciPy) and visualization (like Matplotlib and Seaborn). In addition, Python's simple, straightforward syntax makes it an excellent choice for quick prototyping and exploratory data analysis.

It must be noted that even with all the benefits of Python, the simplified MATLAB syntax for matrix operations facilitates algorithm development since the source code closely resembles mathematical notation.

3.1.3 Comparative Examples

Let us consider a simple example where we want to multiply two matrices A and B. The language examples are listed below in alphabetical order by the name of the language.

In C++:

```
1 #include <Eigen/Dense>
2 #include <iostream>
3
4 int main() {
```



```

5     Eigen::Matrix2f A;
6     A << 1, 2,
7         3, 4;
8     Eigen::Matrix2f B;
9     B << 5, 6,
10        7, 8;
11     std::cout << "The product AB is:\n" << A * B;
12 }

```

../source/IntroProgMatMul.c

In MATLAB:

```

1 A = [1, 2; 3, 4];
2 B = [5, 6; 7, 8];
3 disp('The product AB is:')
4 disp(A * B)

```

../source/IntroProgMatMul.m

In R:

```

1 A <- matrix(c(1, 3, 2, 4), nrow=2, ncol=2, byrow = TRUE)
2 B <- matrix(c(5, 7, 6, 8), nrow=2, ncol=2, byrow = TRUE)
3
4 print("The product AB is:")
5 print(A %*% B)

```

../source/IntroProgMatMul.r

In Python:

```

1 import numpy as np
2
3 A = np.array([[1, 2], [3, 4]])
4 B = np.array([[5, 6], [7, 8]])
5 print("The product AB is:\n", np.matmul(A, B))

```

../source/IntroProgMatMul.py

From the above example, it's clear that MATLAB code is much more concise and easier to understand than C++. However, if we were dealing with very large matrices or needed to perform this operation many times in a loop, the C++ code might run faster due to being a compiled language.

3.1.4 What is Python?

Python is an *interpreted high-level* programming language for general-purpose computing.

- *Interpreted* means that a program in Python has to be decoded line by line by another program called the Python interpreter and translated into something a computer can understand.
- *High-level* means that there is a strong abstraction of the elements of the hardware of the computer; for instance, memory does not need to be pre-allocated in Python to create a matrix, whereas the C language (a low-level language) requires it. You can think of the level of the language as how close the instructions are to the metal of the machine.
- The process of language interpretation is called *compilation*, that is, the translation of one computer language into another language. This could happen multiple times until the instructions are reduced to a language called *assembler* or *assembly*; this code matches closely the architecture of the computer executing the program. For instance, Java programs are compiled into a language called Java bytecode, which the Java Virtual Machine translates into assembler.

3.1.5 Python Installation

Refer now to the software installation guide provided in a separate document.

Being an interpreted language, it means that Python requires an *interpreter*. This is what you install when you “*install Python*”. A Python program is simply a text file that contains instructions the interpreter can understand one line (or block of lines) at a time. Since Python is open source, it has many contributors; this results in a robust ecosystem, but the drawback is that there are numerous software libraries that have to be installed while tracking a complex web of co-dependencies. Hence, there is some difficulty in installing a working Python environment to conduct quantitative analysis.

An easy way to use the Python language is through an *Integrated Development Environment*. There are many IDEs that can execute Python code. Being familiar with more than one environment is important as each has strengths and weaknesses. Furthermore, it is a common occurrence that Python programs appear to fail in one environment but work properly in another; this, of course, is a matter of configuration, but it can be a source of frustration in absence of

awareness of the potential sources of conflicts in software. Particularly, you want to know how to use tools frequently used in software development, and you need exposure to the multiple options you will find in academia, government and industry.

3.1.6 Executing Python Commands

Commands in Python are case-sensitive, that is, you must respect the capitalization of instructions as they appear in the documentation. The Python interpreter offers multiple mechanisms to interact with the environment. The most commonly used is the Python command line. You can start it from a command shell. Simply type `python`. The command line prompt will change to `>>>`

You can enter Python commands in the command line prompt. Press *Enter* or *return* to execute commands. Like most other programming languages, Python provides mathematical expressions. The building blocks of expressions are variables, numbers, operators, scripts and functions

The basic operations are listed below.

Symbol	Operation
+	Addition
−	Subtraction
★	Multiplication
/	Division
★★	Power

Some commands are just like you would type things in a calculator. If you want to refer the result of the command, you need to give it a name, or in other words, assign the result to a variable. The following command assigns the result to the variable `s`. (Note that Python didn't save the true answer $7/12$ but rather a decimal approximation.)

```
>>> s = 1/3 + 1/4
```

If you want to know what the variable `s` contains, simply type its name in the command prompt:

```
>>> s
0.5833333333333333
```

If a command does not fit on one line, use a backslash (`\`) followed by Return/Enter to indicate that the statement continues on the

next line. For example,

```
>>> s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 \
... :- 1/8 + 1/9 - 1/10 + 1/11 - 1/12
```

Type `s` and Enter to see the new value of `s`.

```
>>> s
0.6532106782106782
```

Use the double star(`**`) to raise something to an exponent.

```
>>> 89**2
7921
```

Imaginary numbers can be used with the constant `j`, which stands for $\sqrt{-1}$ by default.

```
>>> (1j)**2
(-1+0j)
```

Hence, you must be careful in using the variable `j` when dealing with complex numbers. What do you expect from the following operation? Can you explain what you observe? You might need to complete the entire chapter before you can answer this question.

Record your explanation as answer #5.

```
>>> j = 10
>>> j = j*(-1)**(0.5)
>>> j
```

Does the result stored in the variable `j` match your expectation? Explain. **Record your explanation as answer #6.**

Python uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter `e` or `E` to specify a power-of-ten scale factor. Imaginary numbers use `i` as a suffix. (This course does not use complex numbers, but you might generate errors with one.)

Some examples of legal numbers are: 1, -1, 0.0001, 9.87654321, 1.2345e10, 1.2345*10**10, 1j, -2j, 3e5j.

When assigning names to variables or expressions, it is good practice to make the names useful. For example, in the second command above, we gave the result the name `rhoSq` to imply “rho squared”. Observe that you **CAN NOT** have spaces in variable names (i.e. no spaces in names on the left hand side of the equals sign). Giving

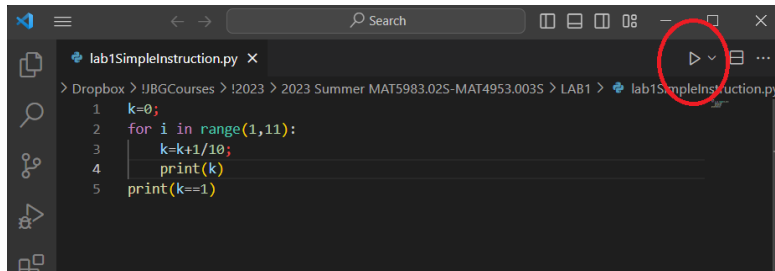


Figure 3.1-1: To execute a Python file in VSC, simply click on the Run button, represented by a triangle pointing to the right. It has been highlighted with a circle.

useful names to your variables or output will help you keep track of what you are doing.

You can use the up and down arrow keys to easily recall and then edit a command with the left and right arrows. Try it. This is the fastest way to repeat Python commands you might have used in the past.

3.1.7 The VSC Python Editor: Executing A Simple Program

Download the file `lab1SimpleInstruction.py`. Open the file in VSC. The following instructions will appear in the editor:

```
1 k=0;
2 for i in range(1,11):
3     k=k+1/10;
4     print(k)
5 print(k==1)
```

You can click the *Run file* button on the toolbar (as shown in Figure 3.1-1), or press F5. We call all these actions *to execute a program*. The program `lab1SimpleInstruction.py` adds ten times the number 0.1. The comparison of the number 1 against the sum returns a value of *false*, whereas elementary arithmetic tells us that adding the number 0.1 ten times should result in 1, and the returned value should be *true*. Why is this result false? **Record your explanation as answer #7.** Once the program is executed, the bottom portion of VSC shows a panel labeled TERMINAL, as well as other tabs labeled PROBLEMS, OUTPUT, and DEBUG CONSOLE, as shown in Figure 3.1-2

Note that you can enter commands in the terminal. For example, type `1+1` and hit enter. Try mathematical operations such as square

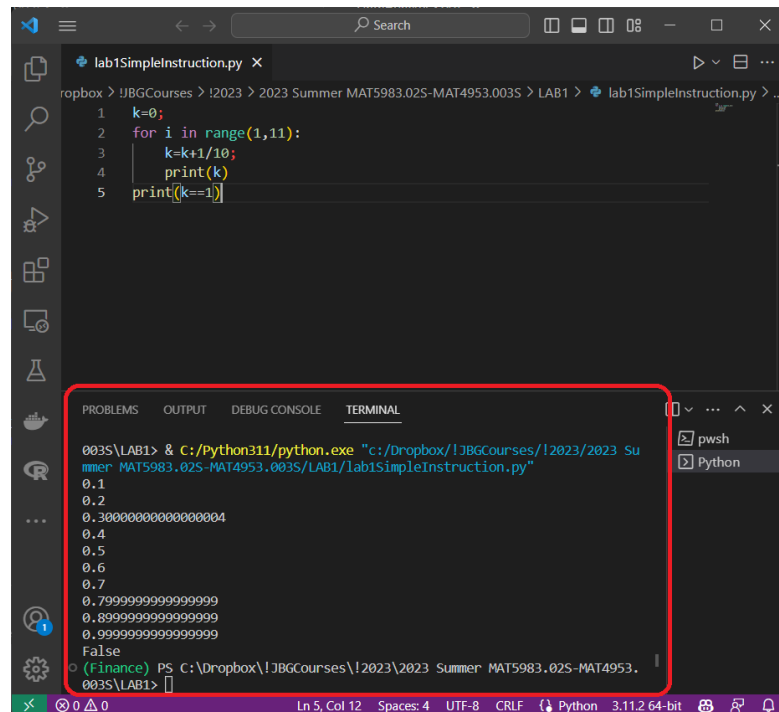


Figure 3.1-2: Executing a python program activates the command line. Results are printed in that panel. The command line is highlighted by the round box.

root, or trigonometric functions. What challenges do you encounter?
Record your explanation as answer #8.

In the program `lab1SimpleInstruction.py` all lines of code were executed sequentially, one at a time. This is what we call a **script file**. That is, when you execute a script file, you expect something tangible to happen, whether it is a message on the screen, a file altered, communication taking place, etc.

There is a way to use source code files to hold code that will be executed later, without necessarily executing any instruction. At this point, open the file `lab1Function.py`. You will see the following instructions:

```

1 def sum1(n):
2     # sum1(n) computes the sum of 1 + 2 + ... + n
3     total = 0;
4     for i in range(1,n):
5         total = total + i
6     return total

```

This file defines a function named `sum1(n)`. The command `sum(3)` returns 6 since $1 + 2 + 3 = 6$. It is instructive to see how this code does its work.

- The instruction `def` indicates the beginning of a *function*. Note that all instructions that are executed inside that function have a vertical alignment shifted to the right to indicate that all these commands are subordinate to `def`. This shift in vertical alignment is called indentation. It is part of the syntax of the Python programming language, therefore you must always treat indentations strictly.
- The instruction `# sum1(n)...` is a *comment*. Lines of code that start with the symbol `#` are not executed. The semicolon at the end is optional.
- The instruction `total = 0;` is an *assignment*. This line of code creates the variable `total` and assigns to it the value zero.
- The instruction `for i in range(1,n):` starts a *loop*. The subordinate instructions, identified by indentation, will be repeated n times. Particularly, the instruction `range(1,n)` creates a sequence of integers starting in 1 and ending in n . What is the outcome of `range(10,13)` ?
- The instruction `total = total + i` is an assignment. It is very important to emphasize that this is not an equation; if it were, `total` would cancel and `i` would be zero. What takes place is that for every repetition of the loop, the variable `total` is redefined to contain its previous value plus the value of `i`.
- The instruction `return total` is the output of the function. That is, if somehow we could execute this function like `x = sum1(3)`, the variable `x` would contain the value 6, because `sum1(3) = 1 + 2 + 3 = 6`.

Now, we will modify this file. Append at the end a program called `sum2` in which we sum integers squared. The next question has to be: How do we execute these functions?

The command `python [filename]` can be used to execute a Python program from the terminal, or any command line prompt. But if we run the following in the command prompt, nothing happens:

```
python lab1Function.py
```

Why? Record your explanation as answer #9.

Now, add the following line of code to the end of the file, **without indentation**:

```
print(sum1(2))
```

Execute the program. What was the result of `sum1(2)`? Is this what you expected? Why? **Record your explanation as answer #10.** Compute by hand what the output of `sum1(3)` should be. Now change the last line in the program to read `print(sum1(2))`

What we did was simply to define a function and then use it. Note that if you place the previous line of code at the beginning of the file, the Python interpreter would raise an exception because the function `sum1` is not known to the interpreter yet... keep in mind that the Python interpreter executes one line of code at a time in sequential order (perhaps with branching and repetition, as we will explore later). Therefore, functions must be defined before they can be invoked within a file.

Did the program produce the result you were expecting? In case it did not, enter the following instruction in the terminal: `man range`. This command shows the manual (`man`) for the function `range`. The fact that `range` behaves in certain ways does not force you to abide by that particular design. In fact, you can modify `sum1` so that the outcome corresponds to what a naive user would expect. To achieve this, you will need to change the limit of range inside the function. Implement it and explain. **Record your explanation as answer #11.**

3.1.8 Executing Functions

In the previous section, we added a single line of code at the end of the program to compute the function we programmed. However, there is a more interesting use case: Create a function that other programs can reuse. In order to use an existing function from a file, we must import that function into the Python environment. We already did something like this when we imported NumPy.

We will now import the function `sum1` from the file `lab1Function.py`. To do this, execute the following command:

```
import lab1function as LAB1
```

This instruction loads into memory a directory of all functions present in that file. Now you can invoke the functions you created.

For example,

```
LAB1.sum2(3)
```

It is also possible to load into memory a single function from a file. We can accomplish this by executing

```
from lab1function import sum1
```

In this case, there is no alias associated with the function. To use, you can simply call the function with a parameter, as in `sum2(3)`

Do some research on how to load multiple functions with a single instruction. Explain. **Record your explanation as answer #12.**

3.2 Introduction to NumPy

Numerical operations can be programmed in Python. A library called `NumPy` encapsulates much needed functionality and has become the *de facto* standard. The basic data element in `NumPy` is a multidimensional array. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a low-level language such as C.

For example, at the Python prompt type in the following command and press Enter.

```
import numpy as np
```

This command loads into memory a reference to the NumPy library. Now you can access all functions in this library by using the prefix `np`; this is an *alias*. You can pick any alias you want, but it is customary to use `np` for NumPy. Now enter the following command:

```
np.sqrt(9)
```

The function `sqrt` computes the square root of a real number. Here is an example, and the resulting values.

```
>>> rho = (1+np.sqrt(5))/2
>>> rhoSq = rho**2
>>> a = rhoSq + rho
>>> a
4.23606797749979
```

However, you cannot use the function `sqrt` with negative numbers. What would you have to do to allow Python to compute the square root of a negative number and return a complex number? **Record your explanation as answer #13.**

The library NumPy offers easy access to commonly used constants, such as:

Constant	Value
<code>np.e</code>	Returns the number e
<code>np.pi</code>	Returns the number π
<code>np.inf</code>	Returns the number ∞
<code>np.nan</code>	NaN, not-a-number

Expressions use the arithmetic operators and precedence rules you already know by now. There are also functions, such as cosine or sine. You surround the input to the function by parenthesis. With NumPy, we can now access commonly used mathematical functions. Among others:

Function	Operation
<code>np.abs(x)</code>	Absolute value $ x $
<code>np.sqrt(x)</code>	Square root \sqrt{x}
<code>np.exp(x)</code>	Exponential function e^x
<code>np.log(x)</code>	$\log(x)$ where x is positive real number
<code>np.sin(x)</code>	$\sin(x)$ where x is assumed to be in radians
<code>np.cos(x)</code>	$\cos(x)$ where x is assumed to be in radians

The library NumPy has *many* functions. A comprehensive list can be found at <https://numpy.org/doc/stable/reference/>. Now enter the following command:

```
x = np.random.standard_normal(3)
```

There is no output. Why do you think this is the behavior? **Record your explanation as answer #14.** On the right-hand side of the Spyder window there is a tab called *Variable Explorer*. It will show the value of x . Alternatively, you can enter

```
print(x)
```

Alternatively, you can print the result directly without assigning it to a variable. Try

```
print(np.random.standard_normal(3))
```

Why are the results of this last operation different to the values stored in variable x ? **Record your explanation as answer #15.**

3.2.1 Matrix Operations

The basic data element in NumPy is a multi-dimensional array. Special meaning is sometimes attached to 1-by-1 matrices, which are called scalars, and to matrices with only one row or column, which are called vectors. Python has other ways of storing both numeric and non-numeric data, but in the beginning, it is usually best to think of everything as a matrix.

Row vectors are delimited in notation by square brackets. A comma separates individual numbers in a row vector. Multiple row vectors of the same size are separated by commas, and are delimited by square brackets to define a matrix. It is common practice to use a capital letter when naming matrices in order to distinguish them from a single vector, scalar or variable value.

Now, we will create a vector and a matrix to demonstrate simple matrix operations

Function	Operation
<code>x = np.array([1,2,3,4])</code>	Creates the row vector $\mathbf{x} = [1, 2, 3, 4]$
<code>b = np.array([1,2,3])</code>	Creates the row vector $\mathbf{b} = [1, 2, 3]$
<code>A = np.array([[1,2,3],[4,5,6],[7,8,9]])</code>	Creates the matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$
<code>B = np.ones([2,3])</code>	Creates the matrix $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
<code>C = np.zeros([2,3])</code>	Creates the matrix $A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

The following commands demonstrate basic vector manipulation cases:

Operation	Outcome
<code>x[-1]</code>	Returns 4, the last element of x
<code>x[1]</code>	Returns 2, the second element of x
<code>x[-2]</code>	Returns 3, the second to last element of x

The colon (:) operator is notable. In vector notation, it indicates a range of rows or columns. In Python, the first element of a vector has index 0; the second element has index 1, and so on. The colon operator $a : b$ starts at the row or column of index a and covers up to

an index less than b . Thus, $A[0 : 2, 0 : 2]$ returns a matrix composed of rows 1 through 2, and columns 1 through 2 with respect to the matrix A .

The following commands demonstrate basic matrix manipulation cases:

Operation	Outcome
$A[1, 2]$	Value of 2 st row and 3 rd column of A
$A[0:2, 0:2]$	Returns $A = \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix}$
$A[1:3, 1:3]$	Returns $A = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$

The matrix A has three rows and three columns. Then, why does $A[1 : 100, 1 : 100]$ not produce an error? **Record your explanation as answer #16.**

The size or dimension of a matrix refers to the number of rows and number of columns a matrix has. The **shape** command returns the number of rows and columns of a matrix. Notice that the result is a vector - a one-dimensional matrix with 2 values where the first is the number of rows and the second value is the number of columns. By accessing each element of this vector, we can access each dimension individually.

Using the matrix A of our previous examples,

```
>>> y = A.shape()
>>> y
(3,3)
>>> y[0]
3
>>> y[1]
3
```

The following example will illustrate a subtlety in copying matrices:

```

>>> D = A

array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> D[0,0] = 9
>>> A

array([[9, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

```

Note that by updating a value in D , the corresponding value in A was changed. Why? What would you need to do to prevent an update in D to change A ? **Record your explanation as answer #17.**

The basic matrix operations are listed below, using the matrices in the previous examples.

Operation	Outcome
$A.T$	Transpose of A .
$A.conj().T$	Conjugate transpose of A .
$A@B.T$	Matrix multiplication $A \cdot B$.
$A \star A$	Element-wise multiplication
A/A	Element-wise division
$A \star \star 2$	Element-wise exponentiation

NumPy gives an easy way of solving systems of linear equations, Given the matrix A and the vector \mathbf{b} from the previous examples, the linear system $A\mathbf{x} = \mathbf{b}$ can be solved by invoking `np.linalg.solve` as follows

```

>>> x = np.linalg.solve(A,b)
array([-0.23333333,  0.46666667,  0.1        ])

```

It is easy to use Matrix multiplication $A\mathbf{x}$ to check the answer

```

>>> A@x
array([1., 2., 3.])

```

Chapter 4

Data from Web Pages

Web scraping is a process that allows automatic data collection from websites, providing solutions in various fields including market research, data analysis, and competitive intelligence. By automating data retrieval from web pages, it eliminates the need for manual collection, saving time and resources.

Python is the preferred choice for web scraping due to its extensive library support and ease of use. Libraries like BeautifulSoup, Scrapy, and Selenium simplify the process of accessing, navigating, and extracting structured data efficiently. This chapter explores the essentials of web scraping through Python, beginning with an understanding of HTML, programmatically selecting HTML elements, and effectively loading and utilizing the extracted data.

4.1 Introduction to HTML

HTML, or Hypertext Markup Language, is the backbone of web content structuring. It's crucial for providing context and pathways when scraping web data. HTML documents are structured using a series of nested tags that define elements such as headings, paragraphs, and links. Understanding these elements allows us to pinpoint the exact locations of the data we wish to extract.

For instance, a typical HTML page might use `< div >`, `< span >`, and `< a >` tags extensively. These tags not only organize the content but also facilitate styling and linking. Consider the following basic HTML structure:

```
1 <!DOCTYPE html>  
2 <html>
```

```
3 <head>
4   <title>Sample Web Page</title>
5 </head>
6 <body>
7   <h1>Welcome to the Sample Website</h1>
8   <div class="content-wrapper">
9     <p class="intro-text">This paragraph provides
      an introduction.</p>
10    <span class="highlight">Highlighted text
      snippet.</span>
11  </div>
12  <a href="https://www.sample.com">Visit
    Sample.com</a>
13 </body>
14 </html>
```

In this example, `< div >` groups page content, `< span >` is used for inline elements that need styling, and `< a >` provides hyperlinked resources. Understanding this structure is the key to successful web scraping, as it guides you in identifying which elements to extract.

4.2 Selecting HTML Elements Programmatically

Navigating through HTML to extract data requires selecting specific elements programmatically. BeautifulSoup, a Python library, assists by parsing HTML and XML documents to locate and extract content based on specified criteria such as tag names, classes, or attributes.

The process of selecting elements programmatically involves creating a parsed version of the web page and using various methods to search and extract data. Here's a detailed example demonstrating BeautifulSoup in action:

```
1 from bs4 import BeautifulSoup
2
3 htmlDoc = """
4 <html>
5 <head>
6   <title>Sample Web Page</title>
7 </head>
8 <body>
9   <h1>Welcome to the Sample Website</h1>
10  <div class="content-wrapper">
```



```
11         <p class="intro-text">This paragraph provides
           an introduction.</p>
12         <a href="https://www.sample.com">Visit
           Sample.com</a>
13     </div>
14 </body>
15 </html>
16 """
17
18 # Initialize BeautifulSoup with the HTML and parser
19 soup = BeautifulSoup(htmlDoc, 'html.parser')
20
21 # Select and print the title element
22 title = soup.title
23 print(title.text) # Output: Sample Web Page
24
25 # Find and print the first paragraph with class
   "intro-text"
26 paragraph = soup.find('p', class_='intro-text')
27 print(paragraph.text) # Output: This paragraph
   provides an introduction.
28
29 # Retrieve and print all links
30 links = soup.find_all('a')
31 for link in links:
32     print(link.get('href')) # Output:
   https://www.sample.com
```

In the example above, the 'html.parser' argument is specified to use Python's built-in HTML parser. Methods like 'find()' and 'find_all()' are employed to retrieve elements by matching specified criteria, simplifying data extraction from documents.

4.3 Handling Challenges and Storing Data

Web scraping encompasses challenges such as dealing with dynamic content, pagination issues in multi-page sites, and ethical considerations including adherence to robots.txt files to respect scraping boundaries. Dynamic content, loaded via JavaScript, may require using Selenium, which automates browser interaction—essential for scraping data not directly available in HTML source.

When building larger-scale scraping solutions, exploring alternative libraries like Scrapy provides advantages in efficiency and scalability. Scrapy offers an all-in-one framework designed specifically for

scraping, handling data pipelines and concurrency effectively.

Once data is scraped, it must be properly stored and processed. Utilizing libraries like Pandas is beneficial for converting data into DataFrames, allowing for powerful manipulation and cleaning. Data can be stored in CSV files, databases, or further processed for analysis. Here's an illustration of storing scraped data:

```
1 import pandas as pd
2
3 # Sample data from scraping
4 data = {
5     'titles': ['Sample Web Page'],
6     'links': ['https://www.sample.com']
7 }
8
9 # Convert data into a Pandas DataFrame
10 df = pd.DataFrame(data)
11
12 # Display the DataFrame
13 print(df)
```

Storing results efficiently ensures that scraped data can be readily accessed and analyzed, optimizing the scraping process.

This chapter outlines the necessary steps in web scraping with Python, emphasizing how to overcome challenges and efficiently handle data extraction to leverage insights. For further examples and detailed scenarios such as scraping a real website, additional sections might focus on more specific and comprehensive case studies. If further exploration or assistance is needed, feel free to inquire.

Chapter 5

Linear Operations with Data

Data can usually be represented by a *feature matrix* in which each row is an observation and each column is a variable. A trivial example is a physician's office; annual exams for every patient include variables such as glucose level, cholesterol levels, etc. In total dozens of variables are defined. In this case, all patients in the physician's office could be represented by a table, i.e., a matrix, in which each patient would be a row, and the value for each clinical variable would be represented by individual columns. In this section we explore the most basic technique for linear manipulation of this feature matrix.

5.1 Principal Component Analysis

Let us try to represent data in a feature matrix with the best possible *projection* of all observations $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k, \dots, \mathbf{x}_n\}$, along a unitary vector \mathbf{e} onto a vector \mathbf{m} that occupies the same dimensional space. Here the subtlety is that asking about the optimal projection is not the same as asking about the single data vector (i.e. observation) that represents best the data; the latter would simply be the average of all observations in \mathbf{X} .

The projection of an observation \mathbf{x}_k onto a vector \mathbf{m} along a unitary vector \mathbf{e} does not necessarily produce an equality. An amount of error, denoted by ξ , might occur, as shown in in Figure 5.1-1. This can be characterized by $\mathbf{x}_k + \xi = \mathbf{m} + a_k \mathbf{e}$, where a_k is a scalar that stretches the unitary vector \mathbf{e} from \mathbf{x}_k to the direction of the vector \mathbf{m} . It is evident that when $\xi = 0$, then

$$a_k = \mathbf{e}^t(\mathbf{x}_k - \mathbf{m}). \quad (5.1)$$

Our goal is to minimize the error ξ for all elements of \mathbf{X} . We can find an optimal set of coefficients a_k by minimizing the error criterion function given by

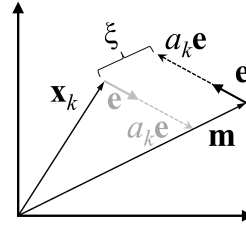


Figure 5.1-1: Projection of an observation \mathbf{x}_k along an arbitrary unitary vector \mathbf{e} onto a vector \mathbf{m} .

$$\begin{aligned}
 J_0(a_1, a_2, \dots, a_n, \mathbf{e}) &= \sum_{k=1}^n \|(\mathbf{m} + a_k \mathbf{e}) - \mathbf{x}_k\|^2 \\
 &= \sum_{k=1}^n \|a_k \mathbf{e} - (\mathbf{x}_k - \mathbf{m})\|^2 \\
 &= \sum_{k=1}^n a_k^2 \|\mathbf{e}\|^2 - 2 \sum_{k=1}^n a_k \mathbf{e}^t (\mathbf{x}_k - \mathbf{m}) + \sum_{k=1}^n \|\mathbf{x}_k - \mathbf{m}\|^2.
 \end{aligned}$$

We will seek the direction of \mathbf{e} that results in no error. Thus, replacing a_k given by Equation 5.1, and since $\|\mathbf{e}\|^2 = 1$, then J is transformed into a function that no longer depends on a_k

$$\begin{aligned}
 J_0(\mathbf{e}) &= \sum_{k=1}^n (\mathbf{e}^t (\mathbf{x}_k - \mathbf{m}))^2 - 2 \sum_{k=1}^n (\mathbf{e}^t (\mathbf{x}_k - \mathbf{m}))^2 + \sum_{k=1}^n \|\mathbf{x}_k - \mathbf{m}\|^2 \\
 &= - \sum_{k=1}^n (\mathbf{e}^t (\mathbf{x}_k - \mathbf{m}))^2 + \sum_{k=1}^n \|\mathbf{x}_k - \mathbf{m}\|^2. \\
 &= - \sum_{k=1}^n \mathbf{e}^t (\mathbf{x}_k - \mathbf{m})(\mathbf{x}_k - \mathbf{m})^t \mathbf{e} + \sum_{k=1}^n \|\mathbf{x}_k - \mathbf{m}\|^2.
 \end{aligned}$$

Recall that the scatter matrix, defined as $n - 1$ times the covariance matrix, is given by $\mathbf{S} = \sum_{k=1}^n (\mathbf{x}_k - \mathbf{m})(\mathbf{x}_k - \mathbf{m})^t$. Thus, the objective function J is the constrained optimization problem

$$J_0(\mathbf{e}) = \mathbf{e}^t \mathbf{S} \mathbf{e} + \sum_{k=1}^n \|\mathbf{x}_k - \mathbf{m}\|^2$$

subject to the constraint $\|\mathbf{e}\|^2 = 1$. Since the term $\sum_{k=1}^n \|\mathbf{x}_k - \mathbf{m}\|^2$ is always positive, the optimal \mathbf{e} is found by minimizing

$$J_0(\mathbf{e}) = \mathbf{e}^t \mathbf{S} \mathbf{e}, \text{ subject to } \|\mathbf{e}\|^2 = 1. \quad (5.2)$$

Equation 5.3 is a constrained optimization problem, which can be converted into an unconstrained optimization problem by adding to the original problem the constraint equated to zero, multiplied by a Lagrange multiplier. In this case, the constraint $\|\mathbf{e}\|^2 = 1$ can be expressed as $\|\mathbf{e}\|^2 - 1 = \mathbf{e}^t \mathbf{e} - 1 = 0$. Thus, we transform the problem of minimizing $J_0(\mathbf{e})$ into the problem of minimizing the following unconstrained problem:

$$L_0(\mathbf{e}, \lambda) = \mathbf{e}^t \mathbf{S} \mathbf{e} - \lambda(\mathbf{e}^t \mathbf{e} - 1). \quad (5.3)$$

Now that we have an unconstrained optimization problem, we can find the extrema by simply deriving and equating to zero, as follows:

$$\frac{\partial L_0(\mathbf{e}, \lambda)}{\partial \mathbf{e}} = 2\mathbf{S}\mathbf{e} - 2\lambda\mathbf{e} = 0 \quad (5.4)$$

From Equation 5.4 we see that $\mathbf{S}\mathbf{e} = \lambda\mathbf{e}$, thus \mathbf{e} must be an eigenvector, and the minimization problem is simply an eigenvalue problem.

5.2 Numerical Implementation of Principal Component Analysis

The first step is to generate as an example two distributions of size 5×50 .

```
1 import numpy as np
2
3 np.random.seed(42)
4 X1 = np.random.rand(5, 50)
5 X2 = X1 + 1
```

For PCA to work efficiently, the data must be standardized.

```
1 def standardize_data(X):
2     mean = np.mean(X, axis=1, keepdims=True)
3     std_dev = np.std(X, axis=1, keepdims=True)
4     return (X - mean) / std_dev
5
6 X1_standardized = standardize_data(X1)
7 X2_standardized = standardize_data(X2)
```

Compute the covariance matrix from the standardized data.

```
1 def compute_covariance_matrix(X):
2     return np.dot(X, X.T) / X.shape[1]
```

```
3
4 cov_matrix1 = compute_covariance_matrix(X1_standardized)
5 cov_matrix2 = compute_covariance_matrix(X2_standardized)
```

Obtain the eigenvalues and eigenvectors from the covariance matrix.

```
1 eigenvalues1, eigenvectors1 = np.linalg.eig(cov_matrix1)
2 eigenvalues2, eigenvectors2 = np.linalg.eig(cov_matrix2)
```

Sort the eigenvalues and choose the top eigenvectors for dimensionality reduction.

```
1 sorted_indices1 = np.argsort(eigenvalues1)[::-1]
2 top_2_eigenvectors1 = eigenvectors1[:,
    sorted_indices1[:2]]
3
4 sorted_indices2 = np.argsort(eigenvalues2)[::-1]
5 top_2_eigenvectors2 = eigenvectors2[:,
    sorted_indices2[:2]]
```

Transform the data into the new subspace defined by the top eigenvectors.

```
1 Y1 = np.dot(top_2_eigenvectors1.T, X1_standardized)
2 Y2 = np.dot(top_2_eigenvectors2.T, X2_standardized)
```

Finally, visualize the transformed data using plots.

```
1 import matplotlib.pyplot as plt
2
3 plt.scatter(Y1[0, :], Y1[1, :], label="Class 1")
4 plt.scatter(Y2[0, :], Y2[1, :], label="Class 2")
5 plt.legend()
6 plt.xlabel('Principal Component 1')
7 plt.ylabel('Principal Component 2')
8 plt.title('2D PCA')
9 plt.show()
```

To visualize in 3D, take the top 3 eigenvectors and use the 3D plotting tools in Matplotlib.

Chapter 6

Configuration Management

6.1 Introduction to Configuration Management

In data analysis, rigorous management of system configurations, which include both hardware and software components, is essential. This management is termed ‘configuration management’ (CM). CM ensures the optimal operation of systems and applications by organizing and documenting all software, hardware, settings, and associated records.

Data analysis tools and methodologies depend upon an integration of various software libraries, environment variables, system configurations, and data sources. A change in any of these elements can influence the results of a data analysis project. Therefore, maintaining records of system states and managing modifications is a *sine qua non* condition for the principles of reproducibility and scalability in data analysis projects.

Configuration management integrates multiple facets: (i) Source Control pertains to the versions and conditions of the source code used in data analysis projects, (ii) Software and Libraries concern the versions and configurations of software utilities, libraries, and programs employed in the data analysis projects, (iii) Environment Variables and System Configurations include the hardware framework, operating systems, runtime environments, and the established parameters within these arenas, (iv) Data Sources and Versions address the datasets employed for the training and evaluation of data analysis models, (v) Algorithm and Model Parameters cover the parameters and configurations within the machine learning algorithms

or statistical techniques implemented, and (vi) Documentation of the aforementioned elements is imperative to ensure the reproducibility, transferability, and scalability of the project.

As data analysis projects progress, CM protocols facilitate structured development and oversight of modifications across these elements. This ensures clarity among team members and allows replication of work and revisitation of past projects. Configuration management acts as a comprehensive version control throughout the data analysis project continuum.

Subsequent sections will explore the significance of configuration management, its methodologies, challenges, and solutions within data analysis.

6.2 Source Control

Git, a widely used version control system, facilitates collaboration and code management in software development. Two prevalent strategies for incorporating changes in Git repositories are:

(i) **Direct Merge into the Main Version:** This approach involves making changes directly to the main branch (typically called ‘master’ or ‘main’) and then pushing those changes.

```
1 # Clone the repository
2 git clone <repository-url>
3
4 # Navigate to the repository directory
5 cd <repository-name>
6
7 # Make changes in the code
8
9 # Stage the changes
10 git add .
11
12 # Commit the changes
13 git commit -m "Commit message describing the changes"
14
15 # Push the changes directly to the main branch
16 git push origin main
```

(ii) **By Creating Branches:** Rather than making changes directly to the main branch, developers create a separate branch, make changes within that branch, and subsequently merge the branch back into the main branch.

```
1 # Clone the repository
2 git clone <repository-url>
3
4 # Navigate to the repository directory
5 cd <repository-name>
6
7 # Create a new branch
8 git checkout -b <branch-name>
9
10 # Make changes in the code
11
12 # Stage the changes
13 git add .
14
15 # Commit the changes
16 git commit -m "Commit message describing the changes"
17
18 # Push the branch to the remote repository
19 git push origin <branch-name>
20
21 # To merge the branch into the main branch
22 # First, switch to the main branch
23 git checkout main
24
25 # Merge the feature branch into the main branch
26 git merge <branch-name>
27
28 # Push the merged changes to the remote main branch
29 git push origin main
```

The decision between these methods often rests on the specific workflow requirements, team preferences, and the nature of the changes being introduced. Utilizing branches can be advantageous in larger projects or teams as it compartmentalizes changes, thereby minimizing conflicts and ensuring a stable main branch.

6.3 Software and Libraries

Software and libraries form the basis of most data analysis projects. They are tools that provide pre-written code to perform certain functions, thereby saving time and resources in the process of development. Understanding the role of software and libraries in the context of configuration management involves two main aspects: versions and configurations.

Versions: As developers improve and update software and libraries, they often release new versions. These new versions may include feature enhancements, bug fixes, security improvements, or even modifications in how certain functions operate. It is important to keep track of which version of a software or library used in a data analysis project because different versions may produce different results.

Configurations: Apart from versions, the configurations of software and libraries can significantly affect the results of a data analysis project. Configurations may involve settings that determine how a software or library behaves, or parameters that influence the operation of certain functions. Ensuring consistency in configurations is essential for the replicability and reliability of a project's outcomes.

Careful management of software and library versions and configurations ensures that data analysis work is reliable, replicable, and less prone to unexpected bugs or inconsistencies.

6.4 Environment variables and system configurations

Environment variables and system configurations are a wide range of settings and parameters that dictate how a system behaves when running a data analysis project. They are an essential part of configuration management because they impact the performance, reproducibility, and even the results of data analysis tasks.

- **Environment Variables:** These are dynamic variables that can affect the way running processes behave on a computer. They are part of the environment in which a process runs and can include anything from paths to certain directories, locale settings that determine language and format, or custom variables that specific data analysis projects might need. For instance, a project may need to connect to a database, and the connection string could be stored in an environment variable for security and ease of management.
- **System Configurations:** These refer to the specifics of the hardware environment and operating system (OS) in which a project runs. They can include the type of processor, the amount

of RAM, the operating system version, and more. The system configurations can impact the execution of a project. For example, an algorithm may run smoothly on a system with a large amount of RAM but crash on a system with less memory.

- **Runtime Environments:** These refer to the environment in which a computer program runs. This includes the type of interpreter for scripting languages, the versions of these interpreters, and other settings. Changes in the runtime environment can lead to different execution behavior for the same piece of code.
- **Parameters and Configurations within these environments:** These are specific settings that influence the behavior of software and libraries within the environments, like the number of threads allocated for an operation, memory limits for certain processes, or specific flags set in the operating environment.

In data analysis, being able to reproduce an analysis is absolutely essential, and this extends to being able to recreate the environment in which the analysis was originally run. Minor differences in these variables and configurations can sometimes lead to major differences in outcomes, making the project non-reproducible.

Effective configuration management ensures that these environment variables and system configurations are carefully controlled and documented. This means that anyone trying to replicate a project, or when moving a project to a different system, will have a record of the environment in which the code was originally run, reducing the chance of unexpected issues or results.

6.5 Data Sources and Versions in Data Analysis

This refers to the specific datasets used in various stages of the data analysis project, such as training, testing, and validation of models. This involves maintaining an understanding of not only where the data is sourced from, but also the specific versions or states of these datasets used at different points in time.

- **Data Sources:** These are the origins of your datasets. They can be databases, APIs, CSV files, Excel files, data scraped from

websites, or any other place where data can be stored. Keeping track of where your data is sourced from is necessary for reproducibility and transparency.

- **Data Versions:** This refers to the specific state of your dataset at a certain point in time. Datasets, especially in a live environment, can change over time as new data is added, erroneous entries are corrected, or outdated entries are removed. Each unique state of a dataset is a different version. Tracking these versions is critical because changes in data can lead to different model training results, and thus, different model performance. This is especially crucial in situations where a model is trained on a particular version of a dataset and is expected to perform consistently when the data is updated or changes.

Managing data sources and versions effectively ensures that the project is robust, reproducible, and reliable. For this purpose, tools like version control systems (e.g., Git), dataset versioning tools (e.g., DVC), or data cataloging tools can be utilized to keep track of the various versions of datasets used throughout a data analysis project.

6.6 Algorithm and Model Parameters in Data Analysis

This area refers to the specific parameters and configurations used within the machine learning models or statistical methods applied in a project.

- **Algorithm Parameters:** These are the settings or options chosen when running an algorithm that can affect its behavior or outcome. This could be the step size for an integrator, the number of clusters in a clustering algorithm, the seed of a randomizer, or the learning rate in a neural network. Different configurations of these parameters can lead to different models, each with potentially different performance characteristics.
- **Model Parameters:** These are the parameters that the model learns from the training data. In a linear regression model, for example, the coefficients of the regression equation are model parameters that the algorithm estimates from the data. In a

neural network, the weights and biases are model parameters that get updated during the learning process.

- **Pipeline Configurations:** These refer to the settings and sequences of a data analysis pipeline. A pipeline consists of several steps such as data cleaning, feature extraction, model training, and model evaluation. Each step could have its own specific configurations. For instance, a pipeline configuration could define how missing data is handled during the cleaning process, how features are scaled or transformed, or how model performance is evaluated.

Effective configuration management of algorithm parameters, model parameters, and pipeline configurations is essential and necessary in data analysis. It ensures that experiments are repeatable and that different models and pipelines can be compared fairly. It can also facilitate collaboration, as others can replicate and build upon your work.

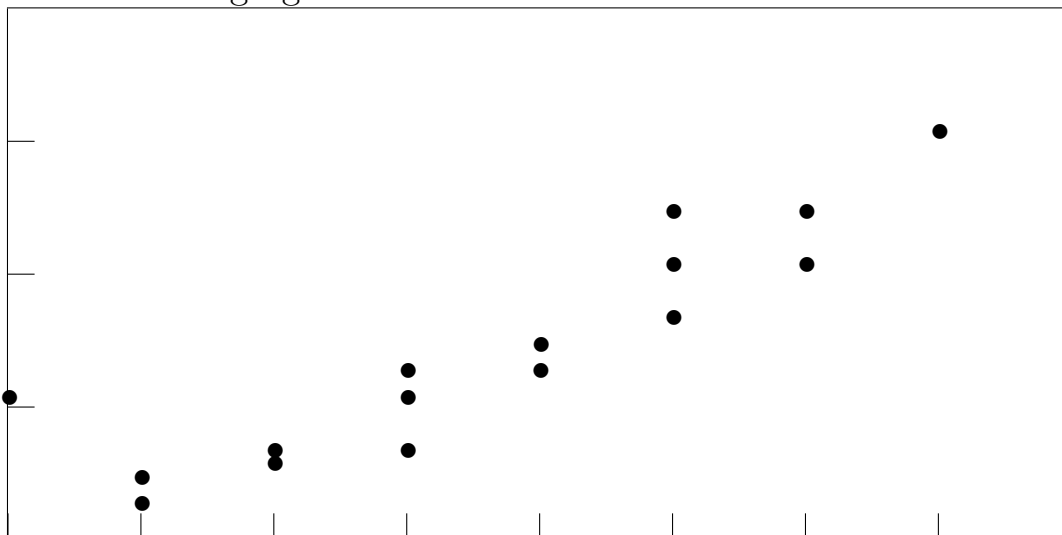
Chapter 7

Processing Time Series

7.1 Time Series

A quantity evolving in time is one of the most common ways to represent data available in public data sets. Let $i, m, n \in \mathbb{N}$, a *time series* consisting of n time points and m variables is a totally ordered set $T := \{(t_i, \mathbf{x}_i)\}$, such that $\mathbf{x}_i \in \mathbb{C}^m$ for $i \in [0, n]$.

Suppose that we obtain a data set from an experiment as shown in the following figure:



It is clear that there is no function which produces the given data.

Suppose we want to approximate a function $f(t)$ with a sum of a given set of functions multiplied by a constant, that is,

$$f(t) = c_1 u_1(t) + c_2 u_2(t) + \cdots + c_k u_k(t) + \cdots + c_n u_n(t) = \sum_{k=1}^n c_k u_k(t).$$

We call $u_k(t)$ a **basis function**. It is possible to determine optimal constants c_k by minimizing the error function:

$$J(c_1, \dots, c_n) = \min_{c_1, \dots, c_n} \int_{\Omega} \left(\sum_{k=1}^n c_k u_k(t) - f(t) \right)^2 dt.$$

We say that $J(c_1, \dots, c_n)$ is optimal in a *least square error sense*.

To find this minimum, derive with respect to each c_k , and set the equation equal to zero:

$$\begin{aligned} \frac{\partial J}{\partial c_k} &= \frac{\partial}{\partial c_k} \left(\int_{\Omega} \left(\sum_{k=1}^n c_k u_k(t) - f(t) \right)^2 dt \right) \\ &= \int_{\Omega} \left(\frac{\partial}{\partial c_k} \left(\sum_{k=1}^n c_k u_k(t) - f(t) \right)^2 dt \right) \\ &= \int_{\Omega} \left(2u_k(t) \left(\sum_{k=1}^n c_k u_k(t) - f(t) \right) dt \right) \\ &= 2 \int_{\Omega} c_1 u_1(t) u_k(t) dt + 2 \int_{\Omega} c_2 u_2(t) u_k(t) dt + \dots + 2 \int_{\Omega} c_n u_n(t) u_k(t) dt - 2 \int_{\Omega} u_k(t) f(t) dt. \\ &= 0 \end{aligned}$$

Let $(u_j, u_k) = \int_{\Omega} u_j(t) u_k(t) dt$. Rearranging the previous terms,

$$c_1 (u_1(t), u_1(t)) + \dots + c_k (u_1(t), u_k(t)) + \dots + c_n (u_1(t), u_n(t)) = (u_k(t), f(t)).$$

Thus, in matrix form,

$$\begin{bmatrix} (u_1, u_1) & \dots & (u_1, u_n) \\ \vdots & \ddots & \vdots \\ (u_n, u_1) & \dots & (u_n, u_n) \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} (u_1, f) \\ \vdots \\ (u_n, f) \end{bmatrix}.$$

The basis functions $u_k(t)$ can be arbitrary. If we select monomials, then $f(t) = c_0 + c_1 t + c_2 t^2 + \dots + c_k t^k + \dots + c_n t^n$. This results in traditional linear regression if the basis set is $\{c_0, c_1 t\}$, traditional quadratic regression if the basis set is $\{c_0, c_1 t, c_2 t^2\}$, Fourier series with basis $\{\cos(k\pi t/l)\}$, discrete Fourier transform follows with little

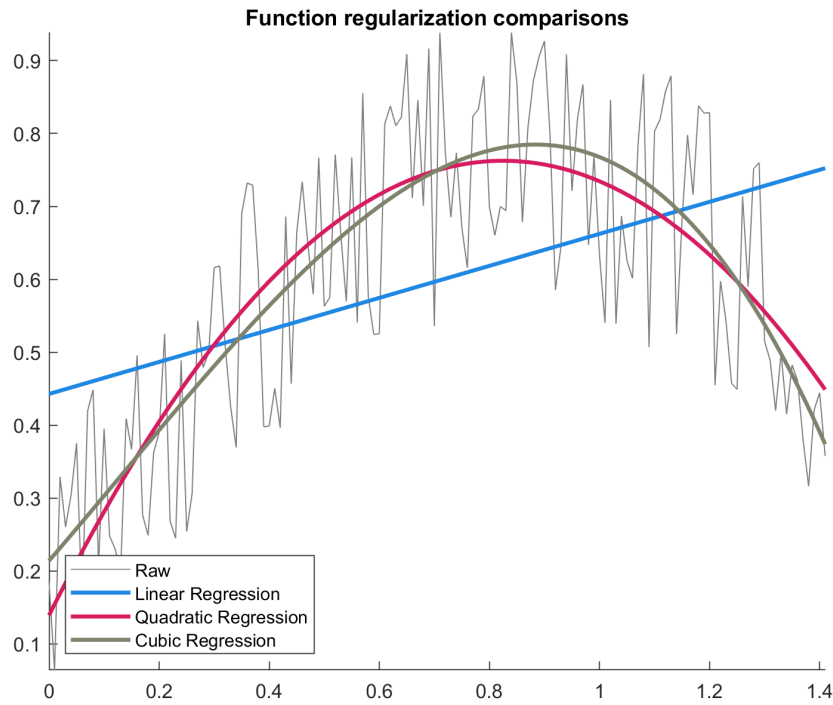


Figure 7.1-1: Multiple regression cases using monomials as basis functions

effort and so on. Figure 7.1-1 depicts multiple regression cases using monomials as basis functions.

We can select functions such that $(u_i, u_j) = 0, \forall i \neq j$. Such functions are called orthogonal functions, e.g. $\sin(t)$, $\cos(t)$, etc. Thus,

$$c_i = \frac{(f, u_i)}{(u_i, u_i)}. \quad (7.1)$$

The consequences of this method are far reaching. When $\sin(t)$ and $\cos(t)$ are used, this method produces the Fourier series for a finite interval of length l . If we study this problem in the limit when $l \rightarrow \infty$, the Fourier transform is produced.

Example 1. Given $f(t) = t^2, t \in [0, 1]$, find c such that the function $g(t) = ct$ approximates $f(t)$ in a *least square error sense*.

$$c = \frac{(t^2, t)}{(t, t)} = \frac{\int_0^1 t^3 dt}{\int_0^1 t^2 dt} = \frac{3}{4}$$

Thus, $\frac{3}{4}t$ approximates t^2 optimally in a *least square error sense* in the interval $[0, 1]$.

These examples lead us to find functions approximately fitting the given data. Suppose that $\{(x_i, b_i) : i = 1, \dots, n\}$ are given observation data from a physical model. Let $f(x) = a_1\phi_1(x) + \dots + a_m\phi_m(x)$ be a function to represent the physical model. We are looking for coefficients $a_i, i = 1, \dots, m$ such that

$$(1) \quad \min_{a_1, \dots, a_m} \max_{1 \leq i \leq n} \{|f(x_i) - b_i|\},$$

or

$$(2) \quad \min_{a_1, \dots, a_m} \sum_{i=1}^n |f(x_i) - b_i|,$$

or

$$(3) \quad \min_{a_1, \dots, a_m} \sum_{i=1}^n (f(x_i) - b_i)^2.$$

In this lab, we mainly study the problem of finding a_1, \dots, a_m satisfying (3) which is called *Least Square Problem*. The other cases lead to different approximate schemes and will not be covered in this chapter.

Let us first give a remark on choosing the model function f . Usually, people use their experiences and the pattern of observation data to choose $\phi_i, i = 1, \dots, m$. For example, if the pattern of the observation data behaviors like a periodic function, we choose $\sin kx$ and/or $\cos kx$ as ϕ_i . We may choose exponential functions $e^{c_i x}$ as ϕ_i if the asymptotic behavior of the physical model has exponential decay or growth.

7.2 Pseudoinverse

Time series, many times, as presented as a collection of values at specific time intervals with known associated outcomes. This problem is equivalent to the following matrix problem: find $\mathbf{x}^* \in \mathbb{R}^m$ such that

$$J(\mathbf{x}^*) = \|\mathbf{A}\mathbf{x}^* - \mathbf{b}\|_2^2 \leq \min_{\mathbf{x} \in \mathbb{R}^m} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

since we may write $\mathbf{x} = (a_1, \dots, a_m)^t$, \mathbf{A} is a matrix of size $n \times m$ with row entries $\phi_i(x_j), 1 \leq i \leq m, 1 \leq j \leq n$ and $\mathbf{b} = (b_1, \dots, b_n)^t$.

Now, for ease of notation, let's drop the $*$. The objective function $J(\mathbf{x})$ can be expressed as:

$$J(\mathbf{x}) = (\mathbf{A}\mathbf{x} - \mathbf{b})^T(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} - 2\mathbf{x}^T \mathbf{A}^T \mathbf{b} + \mathbf{b}^T \mathbf{b}.$$

Now minimize $J(\mathbf{x})$:

$$\nabla J(\mathbf{x}) = 2\mathbf{A}^T \mathbf{A} \mathbf{x} - 2\mathbf{A}^T \mathbf{b} = 0$$

Thus, we conclude that \mathbf{x} satisfies the following

$$\mathbf{A}^T \mathbf{A} \mathbf{x} - \mathbf{A}^T \mathbf{b} = 0.$$

which is called the normal equation.

It follows that

$$\begin{aligned} \mathbf{A}^T \mathbf{A} \mathbf{x} &= \mathbf{A}^T \mathbf{b} \\ \mathbf{x} &= (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} \\ \mathbf{x} &= \mathbf{A}^\dagger \mathbf{b} \end{aligned}$$

where $\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$ is known as the pseudo-inverse of \mathbf{A} .

7.3 Fourier Transform

In our first brush with the integral approach in the previous section, we suggested that it should be possible to express a function as an infinite linear combination of the sinusoids $\{\sin kt, \cos kt\}$. Note that Euler's formula allows us to express each exponential function e^{ikt} in terms of $\sin kt$ and $\cos kt$ and conversely we can express both $\sin kt$ and $\cos kt$ in terms of the exponentials e^{ikt}, e^{-ikt} . Thus it should be possible to express each reasonable 2π -periodic function $f: \mathbb{R} \rightarrow \mathbb{C}$ as an infinite series

$$f(t) = \sum_{k=-\infty}^{\infty} a_k e^{ikt}. \quad (7.2)$$

We have also introduced an inner product which makes the exponential functions appearing in Equation 7.2 mutually orthogonal, so it should be easy to recover the coefficients $\{a_k\}$ from f by taking inner products. Still there are delicate questions remaining, including the meaning of "reasonable" and how we even observe the infinitely

many values of f . The idea of the *discrete Fourier transform* is to make this whole process finite by restricting attention to the values of f at n equally spaced points in the interval $[0, 2\pi)$. This idea is known as *sampling*.

Example 1. Let's begin with the trivial case $n = 1$, i.e., we only want Equation 7.2 to hold when t is a multiple of 2π . Since all the functions f, e^{ikt} are 2π -periodic, Equation 7.2 reduces to $f(0) = \sum_{k=-\infty}^{\infty} a_k 1$. There are infinitely many unknowns here and we simplify the situation by taking $a_k = 0$ for $k \neq 0$. This leaves $a_0 = f(0)$.

and we get the solution $a_k = \begin{cases} f(0), & k = 0 \\ 0, & \text{otherwise} \end{cases}$.

Let's next consider the case $n = 2$, which means we want Equation 7.2 to hold when $t = 0$ and when $t = \pi$. Since $e^{ik\pi} = (-1)^k$, this translates to the system

$$f(0) = \sum_{k=-\infty}^{\infty} a_k (1)^k \quad (7.3)$$

$$f(\pi) = \sum_{k=-\infty}^{\infty} a_k (-1)^k \quad (7.4)$$

Since there are only two equations here, we again simplify the situation by taking all a_k except a_0, a_1 to be zero. This leaves the system

$$f(0) = a_0 + a_1 \quad f(\pi) = a_0 - a_1 \quad (7.5)$$

which easily yields the solution $a_k = \begin{cases} \frac{f(0)+f(\pi)}{2}, & k = 0 \\ \frac{f(0)-f(\pi)}{2}, & k = 1 \\ 0, & \text{otherwise} \end{cases}$. which

can be expressed matricially as

$$\begin{bmatrix} f(0) \\ f(\pi) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}. \quad (7.6)$$

The coefficient matrix here, denoted F_2 , is referred to as a *discrete Fourier transform matrix*.

Example 2. Suppose $n = 4$, so we only care about the values of f at multiples of $\frac{\pi}{2}$. We can store this information as a vector in \mathbb{C}^4 ,

namely $v_f := \begin{bmatrix} f(0) \\ f(\frac{\pi}{2}) \\ f(\pi) \\ f(\frac{3\pi}{2}) \end{bmatrix}$. In particular, our basic function e^{ikt} is stored

as $v_k := \begin{bmatrix} e^0 \\ e^{\frac{ik\pi}{2}} \\ e^{ik\pi} \\ e^{\frac{3ik\pi}{2}} \end{bmatrix} = \begin{bmatrix} 1 \\ i^k \\ (-1)^k \\ (-i)^k \end{bmatrix}$. We should be able to express vectors

$v \in \mathbb{C}^4$ as infinite linear combinations of the $\{v_k\}$. But $v_0 = v_4 = v_8 \dots$ and in general v_k only depends on $k \bmod 4$. Thus most of the vectors $\{v_k\}$ are redundant and we might hope to express v_f as a linear combination of the four vectors v_0, v_1, v_2, v_3 . In other words, given v_f , we are looking for scalars a_0, a_1, a_2, a_3 satisfying

$$v_f = a_0 v_0 + a_1 v_1 + a_2 v_2 + a_3 v_3. \quad (7.7)$$

Take F_4 to be the four by four matrix whose columns are v_0, v_1, v_2, v_3 respectively; written out explicitly, we have $F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$, and the preceding display can be abbreviated as

$$\begin{bmatrix} f(0) \\ f(\frac{\pi}{2}) \\ f(\pi) \\ f(\frac{3\pi}{2}) \end{bmatrix} = F_4 \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (7.8)$$

In fact the columns of F_4 are mutually orthogonal in \mathbb{C}^4 , and thus they form a basis \mathcal{B} for \mathbb{C}^4 . We can summarize this discussion by noting that *every* $v \in \mathbb{C}^4$ satisfies $v = F_4[v]_{\mathcal{B}}$

(Multiplication by) F_4 is known as the *discrete Fourier transform* on \mathbb{Z}_4 . Why \mathbb{Z}_4 ? Because as we have seen, while we can define v_k for every integer k , the actual value of v_k only depends on the equivalence class $k \bmod 4$. For similar reasons, we will sometimes index entries of vectors in \mathbb{C}^n and matrices in $M_n(\mathbb{C})$ by integers modulo n . In the current section, it will also be convenient to begin our indexing at 0,

though we will not always do so later. Thus for the vector $\mathbf{x} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$, we think of a as its 0th or 3rd or 6th, etc. coordinate, b as its 1st or 4th, etc. coordinate, and c as its 2nd or 5th etc. coordinate. With this understanding, the (j, k) entry of the matrix F_4 is given by i^{jk} , with j, k ranging from 0 to 3 modulo 4.

More generally, we write $\omega = \omega_n := e^{\frac{2\pi i}{n}}$ for the primitive n th root of unity and set $F_n := [\omega^{jk}]_{j,k=0}^{n-1}$. The *discrete Fourier transform* (DFT) on \mathbb{Z}_n is implemented by multiplication by the matrix F_n . It allows us to satisfy Equation 7.2 when we only care about the values of f at multiples of $\frac{2\pi}{n}$.

Example 3. Compute F_3 .

Solution. The primitive third root of the identity is $\omega = \omega_3 = e^{\frac{2\pi i}{3}} = \cos(\frac{2\pi i}{3}) + i \sin(\frac{2\pi i}{3}) = \frac{-1+i\sqrt{3}}{2}$, so

$$F_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega^4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \frac{-1+i\sqrt{3}}{2} & \frac{-1-i\sqrt{3}}{2} \\ 1 & \frac{-1-i\sqrt{3}}{2} & \frac{-1+i\sqrt{3}}{2} \end{bmatrix}. \quad (7.9)$$

Chapter 8

Basic Operations in SQL

8.1 Introduction to PostgreSQL

Relational database management systems (RDMSs) are programs that manage databases. They have been the backbone for data operations in government and industry for several decades. The main sources for college-level training in RDMSs can be found in computer science departments, but most often (and usually at a greater depth) in a College of Business, in a major usually called Information Systems. At UTSA, the major is called *Bachelor of Business Administration Degree in Information Systems*.

Knowing how to use a relational database goes deeper than simply learning how to use a software tool. There are many open source and proprietary database management systems. The differences can be quite substantial in implementation, but all of them have a core of instructions that are compatible with Standard ISO/IEC 9075-1:2016¹ from the International Standards Organization (ISO) and the International Electrotechnical Commission (IEC).

The most important aspect of relational databases is information organization, and the architecture of a solution to access data. In this lab, we will explore basic operations related to importing data, querying data, and harmonizing multiple sources of heterogeneous data.

¹<https://www.iso.org/standard/63555.html> accessed on March 15, 2021

8.2 Basic operations in a RDMS

We will focus on RDMS that support a high-level access language known as Structured Query Language, or SQL. This language is declarative, that is, a statement specifies *what* needs to be obtained, not *how* the data retrieval must be executed. The RDBMS interprets statements of the SQL language to perform the requested database access using sophisticated algorithms for data retrieval, but without exhibiting this inherent complexity.

A RDMS can be organized in *databases*, each one of which might contain one or more *schemas*. Each *schema* contains one or more *tables*. Each *table* contains a set of unique *rows*. Each *row* describes a different *entity*. Each *entity* is defined by a unique combination of attributes organized in *columns*.

Table 8.1 illustrates a student data organized in a table; this table is named ‘STUDENT’. In this case, there is a row (or record) per student in ‘STUDENT’. Usually, students have multiple records in a ‘GRADE’ table, as illustrated in Table 8.2

Table 8.1: Example of a data table named ‘STUDENT’

<i>ID</i>	<i>NAME</i>	<i>ADDRESS</i>	<i>STATUS</i>
101	John	123 Merry Ln	fresh
102	Shawn	1 UTSA Circle	soph
110	Leonardo	100 Main St	soph
199	Sofia	200 High Rd	senior

Table 8.2: Example of a data table named ‘GRADE’

<i>ID</i>	<i>LETTER</i>	<i>POINTS</i>
101	C	2
101	B	3
101	A	4
102	C	2
110	A	4
110	A	4
199	A+	4

An SQL query to retrieve students named John would be

```
1 select * from STUDENT where NAME = 'John'
```

The * after the keyword ‘*select*’ means *all columns*; individual columns can be selected by name separated by commas, e.g. ID, NAME. The table name goes after the keyword ‘*from*’. Finally, a condition follows the keyword ‘*where*’ using the column name.

The result of this query is shown in Table 8.3

Table 8.3: Result of query

<i>ID</i>	<i>NAME</i>	<i>ADDRESS</i>	<i>STATUS</i>
101	John	123 Merry Ln	fresh

It is possible to link multiple tables in a single query. For example,

```
1 select      NAME , avg(POINTS)
2 from        STUDENT S
3             join GRADE G
4             on S.ID = G.ID
5 where       ID > 105
6 group by    NAME
7 order by    avg(POINTS)  ASCENDING
```

What would be the result of this query? Write down the expected table. **Record your explanation as answer #18.** Search for information and explain each element of this query. **Record your explanation as answer #19.**

8.3 How to import a CSV file

This is many times the most difficult step in data analysis. Data is often presented in comma-separated values (CSV) format. PostgreSQL offers an easy way to import CSV files through the *COPY* instruction, provided that a table exists to hold the data. Hence, the first step is to create a table.

For instance, if you have a CSV file with three columns, the table could be something like:

```
1 CREATE TABLE MyDataTable (
2     column1 SERIAL,
3     column2 VARCHAR(50),
```

```
4 column3 DATE ,  
5 PRIMARY KEY (column1)  
6 )
```

This instruction contains the statement *CREATE TABLE* which is part of a subset of the SQL language called Data Definition Language (DDL). Note that each column has to be defined with a data type. Once the table is created, you have to import the data with the *COPY* instruction.

```
1 COPY MyDataTable(column1, column2, dob, column3)  
2 FROM 'MyDataTable.csv'  
3 DELIMITER ','  
4 CSV HEADER;
```

8.4 Character Encoding. A Challenge Importing Data into a RDBMS

File encoding is a method by which information is stored or presented in a file. Specifically, it deals with character sets and their representation in a binary format. Encoding provides a mechanism that allows us to map these sequences of bits to human-readable characters. In doing so, we can efficiently read, write, and store text in a manner that computers can precisely retrieve.

One of the earliest character encoding standards is known as ASCII, which stands for the American Standard Code for Information Interchange. ASCII encapsulates each character within a 7-bit integer representation. Its primary design was intended for the English language, thus it envelops 128 characters which include English alphabets, numbers, and a variety of punctuation marks.

However, as the need for more diverse character sets arose, particularly for languages other than English, extensions to ASCII were introduced. The ISO-8859 series is an excellent example of this. Each variant of ISO-8859 was curated for different languages or sets of languages. A notable instance is ISO-8859-1, often referred to as Latin-1, which was crafted to cater to several Western European languages.

The advent of the Unicode standard brought with it UTF-8, a variable-length encoding with the capability to represent any character within the Unicode paradigm. A salient feature of UTF-8 is

its backward compatibility with ASCII, a trait that played a pivotal role in its rise to dominance, especially in the World Wide Web. The Unicode standard also brought forth other encoding forms, namely UTF-16 and UTF-32. While UTF-16 employs variable-length encoding, UTF-32 opts for a fixed-length approach. Both these forms, unlike UTF-8, are adept at representing characters beyond the ASCII spectrum more efficiently but sacrifice backward compatibility with ASCII in the process.

One might wonder about the ramifications of mismatched encoding. Characters may appear misplaced, be replaced with placeholders such as the ubiquitous `?`, or even result in gibberish output. This phenomenon is colloquially termed “mojibake”. Mismatched encoding can severely hinder the usability of data.

Contemporary text editors and Integrated Development Environments (IDEs) are equipped with auto-detection features for file encodings. Moreover, they grant users the autonomy to specify the encoding when accessing a file. For those who prefer the command line, tools like `file` on UNIX-based systems can sometimes discern the encoding of a given file. As data and text are increasingly shared and transferred between systems or software, it becomes paramount to maintain encoding consistency or to adeptly perform a conversion when needed.

Adhering to universal encodings, like UTF-8, is now considered a best practice.

8.5 Handling Encoding Issues in PostgreSQL across Operating Systems

When importing a CSV file into PostgreSQL, one might encounter an encoding issue. On Linux and macOS, you can use the `file` command to determine the encoding:

```
1 file -i /path/to/your/csvfile.csv
```

The output might indicate an encoding like `charset=iso-8859-1`.

On Windows (also Linux & Mac), you can use the Python tool `chardet`:

```
1 pip install chardet
2 chardetect /path/to/your/csvfile.csv
```

If the file isn't in UTF-8 encoding, convert it using the `iconv` command:

```
1 iconv -f ORIGINAL_ENCODING -t UTF-8  
  /path/to/your/csvfile.csv -o  
  /path/to/your/newfile.csv
```

Replace `ORIGINAL_ENCODING` with the encoding you identified.
If you don't have `iconv` on Windows:

- Open the file in Notepad++.
- From the top menu, select "Encoding" and take note of the current encoding.
- Choose "Convert to UTF-8" from the "Encoding" menu.
- Save the file.

After conversion, you can proceed with importing the file into PostgreSQL.

If you prefer not to change the file's encoding:

```
1 SET CLIENT_ENCODING TO 'ORIGINAL_ENCODING';  
2 COPY school_data FROM 'C:\path\to\your\csvfile.csv'  
  DELIMITER ',' CSV HEADER;  
3 SET CLIENT_ENCODING TO 'UTF8';
```

Chapter 9

Using Relational Databases in Data Analysis

9.1 Sending SQL statements from Python

PostgreSQL and Python –and in general relational database management systems (RDBMSs) and general-purpose programming languages– are different computer programs developed by separate communities for different purposes. However, they complement each other remarkably well for data analysis operations.

Managing a large text file to conduct data analysis is an immensely inefficient process, hence the need for an RDMS. At the same time, conducting relatively simple mathematical operations in a RDMS, e.g. matrix multiplication, becomes a hard task. The point of equilibrium with the tools we are using is to conduct data operations (i.e. search and retrieval, update, delete, aggregate) with an RDMS, and then perform data analysis operations with Python.

The first step needed to access data in PostgreSQL is to import the *psycopg2* library. We will also use *numpy*. This can be accomplished with the import command:

```
1 import psycopg2
2 import numpy as np
```

Assume you have created a schema called *USA*, and in it a table called *natalityConf*. To access data in that table, a connection has to be created to the database. Let's assume that there is a user called *postgresql* within PostgreSQL, with the associated password *SARS-CoV-2*. A connection to the database is created with the following

instruction:

```
1 conn = psycopg2.connect("dbname=USA user=postgres  
    password=SARS-CoV-2")
```

The connection by itself does nothing visible; however it is a data structure that exists in memory and contains all the information needed to exchange data with PostgreSQL. In order to use it, we need to create an instance of the *cursor*:

```
1 cur = conn.cursor()
```

At this point we have all we need to send a request for data to PostgreSQL from within a Python environment. In this case, we will request as an example the distinct values in the column *countyfips*, which contains the FIPS codes for every county in the database.

To obtain the unique FIPS codes in the table, execute the following instruction:

```
1 x = cur.execute('''select distinct countyfips,  
2                     county , "state" from  
                        "USA"."natalityConf";''')
```

To see the results of the previous query, you have to fetch the data. Execute the following instruction:

```
1 rows = cur.fetchall()
```

The variable *rows* contains the data as a list. The following instruction will take the first element of every row to create a numeric vector with which we will be able to conduct linear algebra operations:

```
1 # Now, how to extract all members of a tuple  
2 n = 0 # this is the column number  
3 y = [x[n] for x in rows]  
4  
5 # If the field is numeric, you might want to convert it  
    to a matrix  
6 z = np.asarray(y)
```

The variable *z* contains a numeric array. Now, data analysis can begin.

9.2 Accelerating SQL statements

To accelerate data retrieval, you will need to create an *index* in the database. Think of this as an analog to a phone guide; it simply

contains information sorted in a way that makes search and retrieval faster. With a phone guide, you might be able to find the phone of a specific business because the listing is sorted in alphabetical order. However, if you wanted to find all the business in e.g. Merry Street, the standard phone guide would not be the best instrument because you would have to read the entire guide line by line to ensure you capture all the entries that match the search criteria; you would need a different *index*.

The following command in our imaginary table

```
1 x = cur.execute('''CREATE INDEX "idxFIPS"
2     ON "USA"."natalityConf" USING btree
3     (countyfips ASC NULLS LAST)
4     TABLESPACE pg_default;''')
```

is equivalent to opening PGAdmin and executing the following command:

```
1 CREATE INDEX "idxFIPS"
2     ON "USA"."natalityConf" USING btree
3     (countyfips ASC NULLS LAST)
4     TABLESPACE pg_default;
```

Chapter 10

Nonlinear Discriminants

10.1 Neural Networks

This chapter¹ deals with simple problem: Given a family Ω of observations, $\mathbf{X}(t)$, and a finite number C of classes Ω can be assigned to, and assuming there is a feature vector \mathbf{x} in a n th dimensional feature space such that each element of \mathbf{x} represent a characteristic of $\mathbf{X}(t)$, we want to find a discriminant function $g_i(\mathbf{x})$, $i = 1, 2 \dots C$ that assigns \mathbf{x} to class w_i if $g_i(\mathbf{x}) < g_j(\mathbf{x})$ for all $j \neq i$. We define the *error* in this operation as the probability of assigning the wrong class w_i to \mathbf{x} . We define the *decision region* in the space of \mathbf{x} , as a set of boundaries that divide the feature space into segments such that each segment corresponds to either one class w_i or no class at all.

With a clever choice of nonlinear functions, it is possible to obtain arbitrary decision regions, in particular those leading to minimum error. These nonlinear functions are well defined by neural networks. In general, a *neural network* is composed of a group of connected *neurons*. A single neuron can be connected to many other neurons, so that the overall structure of the network can be very complex.

Neural network algorithms can be roughly divided into two classes.

- Supervised neural networks are given data in the form of inputs and targets, the targets being a *teacher's* specification of what the neural network's response to the input should be. The central idea of supervised neural networks is this: Given examples of a

¹This section is an excerpt from “*Information theory, inference, and learning algorithms*” by David J.C. MacKay, *in memoriam*, used with permission from the author.

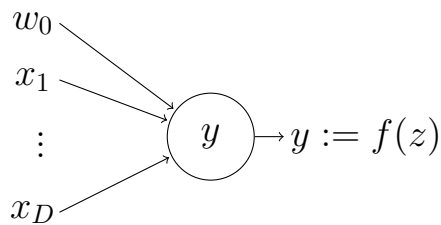


Figure 10.1-1: Single processing unit and its components. The activation function is denoted by f and applied on the actual input z of the unit to form its output $y = f(z)$. x_1, \dots, x_D represent input from other units within the network; w_0 is called bias and represents an external input to the unit. All inputs are mapped onto the actual input z using the propagation rule.

relationship between an input vector \mathbf{x} , and a target t , we hope to make the neural network ‘learn’ a model of the relationship between \mathbf{x} and t . A successfully trained network will, for any given \mathbf{x} , give an output y that is close (in some sense) to the target value t . Training the network involves searching in the weight space of the network for a value of \mathbf{w} that produces a function that fits the provided training data well.

- Unsupervised neural networks are given data in an undivided form – simply a set of examples. Some learning algorithms are intended simply to memorize these data in such a way that the examples can be recalled in the future. Other algorithms are intended to ‘generalize’, to discover ‘patterns’ in the data, or extract the underlying ‘features’ from them. Some unsupervised algorithms are able to make predictions, for example, some algorithms can ‘fill in’ missing variables in an example (and so can also be viewed as supervised networks).

10.1.1 The single neuron

A single neuron has a number I of *inputs* x_i and one output which we will here call y . (See figure 10.1-2.) Associated with each input is a weight w_i ($i = 1, \dots, I$). There may be an additional parameter w_0 of the neuron called a *bias* which we may view as being the weight associated with an input x_0 which is permanently set to 1. The single neuron is a *feed forward* device, i.e. the connections are directed from the inputs to the output of the neuron.

The activity rule has two steps.

- First, in response to the imposed inputs x , we compute the *ac-*

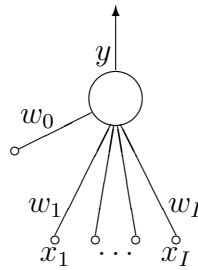


Figure 10.1-2: A single neuron

activation of the neuron,

$$a = \sum_i w_i x_i, \quad (10.1)$$

where the sum is over $i = 0, \dots, I$ if there is a bias and $i = 1, \dots, I$ otherwise.

- Second, the *output* y is set as a function $f(a)$ of the activation. The output is also called the *activity* of the neuron. Notice it is different to the activation a .

activation activity
 $a \rightarrow y(a)$

There are several possible *activation functions*; here are the most popular.

- Deterministic activation functions:

1. Linear.

$$y(a) = a.$$

2. Sigmoid (logistic function).

$$y(a) = \frac{1}{1 + e^{-a}} \quad (y \in (0, 1)).$$

3. Sigmoid (tanh).

$$y(a) = \tanh(a) \quad (y \in (-1, 1)).$$

4. Threshold function.

$$y(a) = \Theta(a) \equiv \begin{cases} 1 & a > 0 \\ -1 & a \leq 0. \end{cases}$$

- Stochastic activation functions: y is stochastically selected from ± 1 .

1. Heat bath.

$$y(a) = \begin{cases} 1 & \text{with probability } \frac{1}{1 + e^{-a}} \\ -1 & \text{otherwise.} \end{cases}$$

2. The Metropolis rule produces the output in a way that depends on the previous output state y :

Compute $\Delta = ay$

If $\Delta \leq 0$, flip y to the other state

Else flip y to the other state with probability $e^{-\Delta}$.

A neural network implements a function $y(\mathbf{x}; \mathbf{w})$; the ‘output’ of the network, y , is a nonlinear function of the ‘inputs’ \mathbf{x} ; this function is parameterized by ‘weights’ \mathbf{w} .

An example of a very simple neural network is the following function of \mathbf{x} , which produces an output between 0 and 1:

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}. \quad (10.2)$$

For convenience let us study the case where the input vector \mathbf{x} and the parameter vector \mathbf{w} are both two-dimensional: $\mathbf{x} = (x_1, x_2)$, $\mathbf{w} = (w_1, w_2)$. Then we can spell out the function performed by the neuron thus:

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}. \quad (10.3)$$

We now introduce the idea of *weight space*, that is, the parameter space of the network. In this case, there are two parameters w_1 and w_2 , so the weight space is two dimensional. Each *point* in \mathbf{w} space corresponds to a *function* of \mathbf{x} . Notice that the gain of the sigmoid function (the gradient of the ramp) increases as the magnitude of \mathbf{w} increases.

Typically an *objective function* or *error function* is defined, as a function of \mathbf{w} , to measure how well the network with weights set to \mathbf{w} solves the task. The objective function is a sum of terms, one for

each input/target pair \mathbf{x}, t , measuring how close the output $y(\mathbf{x}; \mathbf{w})$ is to the target t .

The training process is an exercise in function *minimization*, i.e., adjusting \mathbf{w} in such a way as to find a \mathbf{w} that minimizes the objective function. Many function-minimization algorithms make use not only of the objective function, but also its *gradient* with respect to the parameters \mathbf{w} . For general feed forward neural networks the *backpropagation* algorithm efficiently evaluates the gradient of the output y with respect to the parameters \mathbf{w} , and thence the gradient of the objective function with respect to \mathbf{w} .

10.1.2 A Didactic Implementation

Programming an artificial neural network is a trivial task. I have made available to you a MATLAB and a Python program that computes a simple ANN. Try to use it.

Chapter 11

The Role of Generative AI in Data Analytics

Generative AI refers to a subset of artificial intelligence technologies that generate new data instances by learning from existing data. Unlike traditional AI models that are primarily used for prediction or classification, generative AI models can create text, images, and other data forms that do not explicitly exist in the original dataset. This capability is facilitated by advanced neural network architectures like Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and Transformer models.

Generative AI and Large Language Models (LLMs) represent two distinct but overlapping technologies within the field of artificial intelligence. Generative AI encompasses a broader category of algorithms capable of creating new content that mimics real-world data, whereas LLMs specialize in processing and generating human language based on vast textual data.

Generative AI

Generative AI refers to models that can generate new data instances similar to the training data, encompassing not only text but also images, music, and other media. Key technologies under Generative AI include:

- **Generative Adversarial Networks (GANs):** These consist of two neural networks—the generator and the discriminator—that work against each other to produce data instances that

are indistinguishable from natural data.

- **Variational Autoencoders (VAEs):** Employed to compress data into a lower-dimensional representation from which new data can be generated, preserving essential characteristics of the original data.
- **Autoregressive Models:** Used to predict subsequent elements in a sequence based on preceding elements, facilitating the generation of sequential data such as text or music.

Large Language Models (LLMs)

LLMs are a subset of generative models specifically designed for understanding and generating human language. These models, trained on an extensive textual corpus, predict the next word in a sequence to generate coherent and contextually relevant text. They can also perform linguistic tasks such as translation, summarization, and content creation. Notable examples include:

- **GPT (Generative Pre-trained Transformer):** An autoregressive model capable of predicting the next word in a sentence and generating extensive text passages based on a given prompt.
- **BERT (Bidirectional Encoder Representations from Transformers) and its derivatives:** Initially designed for tasks like filling in missing words in sentences, these models can be adapted for generative tasks through fine-tuning.

In essence, while all LLMs can be considered a part of generative AI, not all generative AI models are LLMs. LLMs represent a highly specialized and complex subset focused specifically on linguistic tasks.

11.1 Generative AI in the Life Cycle of Data Analytics

Generative AI has transformative potential across various dimensions of data analytics, which includes enhancing ethical standards,

improving data collection and cleaning, supporting exploratory analysis, refining data modeling, optimizing reporting and communication, and bolstering reproducibility and configuration management.

Ethics: While the benefits of generative AI in data analytics are substantial, they also introduce new ethical considerations. The ability of generative models to create realistic data and patterns can lead to challenges related to data authenticity and the potential for misuse in generating misleading information. It is crucial to implement robust frameworks and guidelines to govern the use of generative AI, ensuring its application enhances rather than undermines trust in data analytics.

Generative AI can significantly influence the ethical handling of data. By creating synthetic data that mimics real-world data sets, generative AI models can ensure the privacy and anonymity of individuals, thereby adhering to legal and moral standards. For instance, in healthcare, generative models can produce de-identified patient data for research purposes, reducing the risk of privacy breaches while still providing valuable data for analysis.

LLMs can be instrumental in maintaining ethical standards in data analytics. By generating synthetic narratives or explanations that respect privacy and confidentiality, LLMs ensure adherence to ethical norms without sacrificing the depth of analysis.

Data Collection and Cleaning: In the realm of data collection and cleaning, generative AI can automate the generation of data dictionaries and schemas, thereby standardizing data collection processes across various sources. It can also contribute to data cleaning by identifying outliers or generating data to fill missing values, which helps in maintaining the quality and consistency of the data sets used for analysis.

LLMs can automate and enhance the data collection and cleaning process. By parsing and understanding large volumes of unstructured text data, LLMs can extract relevant information and transform it into a structured format suitable for analysis. Furthermore, LLMs can identify inconsistencies or errors in data entries by comparing them against learned language patterns.

Exploratory Analysis: Generative AI enhances exploratory analysis by providing tools to simulate and visualize different data scenarios and trends. By generating data based on existing patterns, these models can help analysts identify potential anomalies or validate hypotheses without the need for additional real-world testing.

This capability supports the elaboration of comprehensive data analysis plans and aids in understanding complex data structures and domains.

In exploratory analysis, LLMs can generate insightful summaries and detect underlying patterns in data. This is particularly useful in understanding complex datasets with subtle nuances in textual data.

Data Modeling: In data modeling, generative AI can be utilized to create advanced algorithms that generate predictive models based on existing data. These models can extrapolate from current trends to predict future outcomes, enhancing the decision-making process. Furthermore, generative AI can assist in the design and implementation of data analysis pipelines, thereby streamlining the workflow from data collection to insight extraction.

LLMs excel in data modeling by generating predictive models based on textual data inputs. These models can predict future trends or outcomes based on historical data interpreted by the LLM.

Reporting and Communication: Generative AI can revolutionize how findings are reported and communicated to stakeholders. By generating dynamic visualizations and narratives, it can tailor reports to the specific interests and understandings of different audiences, making complex data insights more accessible and actionable.

LLMs can dynamically generate reports and presentations tailored to the specific needs of stakeholders. By processing complex datasets and summarizing findings in a comprehensible language, these models facilitate effective communication.

Reproducibility and Configuration Management: LLMs contribute to reproducibility in data analytics by providing a means to automatically generate documentation and code for data analysis tasks. This ensures that analytical processes are transparent and can be replicated. An LLM could generate configuration files for a data analysis pipeline, ensuring consistency across different environments.

11.2 Enhancing Predictive Analytics

One of the primary roles of generative AI in data analytics is enhancing predictive analytics capabilities. By generating synthetic data, these models can simulate various scenarios and outcomes, providing analysts with a richer dataset to train predictive models. This is particularly valuable in fields where data is scarce or difficult to collect,

such as healthcare or finance. For instance, generative models can create synthetic patient data that mimics real patient demographics, symptoms, and treatment outcomes, thereby enabling the development of more accurate predictive models without compromising patient privacy.

Data augmentation is another significant application of generative AI. In contexts where the available data is unbalanced or insufficient for training robust machine learning models, generative AI can augment datasets by creating additional realistic, synthetic examples. This approach not only helps in balancing datasets but also enhances the generalizability of machine learning models, making them more effective across diverse scenarios.

Generative AI can also play a crucial role in anomaly detection. By learning the normal distribution of data, generative models can highlight instances that deviate from this norm. For example, in cybersecurity, a generative model trained on normal transaction data can generate expected patterns and flag transactions that are anomalous, helping prevent potential fraud and breaches.

11.3 Conclusion

Large Language Models are reshaping the landscape of data analytics by enhancing the automation, accuracy, and scope of analysis across various domains. Their ability to understand and generate human-like text makes them invaluable tools for data-driven decision-making, ensuring both the efficiency and integrity of analytical processes. The future of data analytics with LLMs promises even greater integration of AI in extracting actionable insights from complex data sets.

Looking ahead, the role of generative AI in data analytics is poised for significant growth. As these technologies become more sophisticated and their integration with cloud computing and IoT devices expands, their impact will become more profound across all sectors. The ongoing development of generative AI promises not only to enhance existing analytical processes but also to innovate new methods that could redefine how data is utilized in decision-making processes.

Generative AI is reshaping the field of data analytics by providing powerful tools for data generation, augmentation, and visualization. Its integration into data analytics marks a transition from tradi-

tional analysis techniques to more dynamic, predictive, and creative methodologies. As this field evolves, it will be essential to navigate the challenges it presents, particularly those related to ethics and data integrity, to fully realize its potential in enhancing and transforming data-driven decision-making.

11.4 Using APIs

At this point, you know how to connect to git. Clone the repository:

```
1 git clone  
   https://github.com/biomathematicus/CommandLineGPT.git
```

As described in the Environment Setup guide, create a virtual environment. Then, install the OpenAI, Anthropic, langchain, and langchain_groq libraries. Execute the programs Helper.py (OpenAI), ClaudeChatUL.py (Anthropic) and GroqChat.py (Groq). The program FOO.py creates an ensemble of agents.

11.5 Strategies and Tactics to Use Generative AI

Transformers, introduced in the seminal paper by Vaswani et al. (2017) [21], revolutionized natural language processing with their central innovation, the self-attention mechanism. This is the central component of OpenAI’s GPT models (Generalized Pre-trained Transformers). This mechanism computes a score for how much each element in the input sequence should attend to every other element, thereby enabling the model to capture relationships and dependencies in the data, regardless of their distance in the sequence. This is particularly advantageous for understanding the nuanced context of language, as it allows for a more dynamic handling of language structure than previous models. However, despite their capabilities, transformers have a limitation known as the fixed-length context window. Over the course of an extended conversation, this limitation may cause the model to appear to “forget” or fail to consistently apply information, directives, or context provided in earlier parts of the exchange.

There are various strategies and tactics that enhance user interactions with large language models like GPT-4, emphasizing the im-

portance of clear instructions and structured input to achieve better results.

ChatGPT Directives

Regarding ChatGPT custom instructions, the correct use of this tool can make the entire difference between a useless piece of information that is clearly produced by an incapable machine, or a masterpiece that reflects your state of mind.

The first thing you need to keep in mind is that **ChatGPT will not make a genius out of you. It will simply accelerate the process of finding answers within your reach.** Use it, since only practice will allow you to develop the skill to use it effectively.

I can share with you the directives I use. Enhance as you see fit.

- Directive R1: Unless I request otherwise, I need detailed answers without adjectives.
- Directive R2: Unless I request otherwise, I need answers in paragraphs instead of lists.
- Directive R3: I need responses with references. Never, ever generate references. Use only existing references for which there is a URL. If referencing a book, indicate the page number for the citation.
- Directive R4: When I ask for Jupyter markdown, I want you to produce mathematical notation enclosed ONLY in double dollar sign (\$\$). Avoid any other type of enclosure for equations. For inline formulas, use single dollar sign (\$).
- Directive R5: The following is an example of my writing style. Emulate it when generating answers: *“The dominating paradigm in interdisciplinary education is inherently inefficient. It consists of teaching the same thing to each student at the same pace within a classroom with well-defined initial end ending points analogous to a tree structure; however: (i) students interested in data often come from different disciplines, thus have different (often non-overlapping) backgrounds, and (ii) curricula in interdisciplinary fields are comprised by subject matters drawn from different (often traditionally disconnected) areas. I have designed an innovative model of experiential learning for data science in which*

the metaphor of the tree is replaced by a dense rhizome-like network that does not privilege a particular path, but instead offers a milieu for traversal.”

When beginning a new chat, start with the following request: *“Please echo back the directives for generation in this chat.”* Verify that the directives you have provided are all in place. A chat to complete a simple a complex task could have many interactions (hundreds); you must guide ChatGPT to refine the answers until you reach the level of achievement you are looking for. Sometimes, you will have to indicate *“this answer is wrong because... make the correction and regenerate the previous answer.”*

in a long interaction, previous steps will be forgotten. After ChatGPT gives a useful answer (keep in mind, not all answers will be useful), instruct *“Identify the previous answer as #”* where # is an indexing useful to you, e.g. 1, 2, 3, ... or a, b, c, ..., etc. As you interact with ChatGPT, you might need to refer to previous answers. Use these indices. For example, you could ask e.g. *“combine answer a.1 and b.3 into a single paragraph. Produce an analytical example and code in python to implement it. Generate all this answer in LaTeX.”*

11.5.1 Strategies for Enhanced Model Interaction

- Writing clear and precise instructions is crucial as it reduces the need for the model to infer user intentions, thus increasing response accuracy.
- Providing reference texts helps the model generate more accurate and less fabricated responses, which is particularly useful for specific or niche topics.
- Breaking down complex tasks into simpler, manageable subtasks can significantly decrease error rates and improve output quality from the model.

11.5.2 Tactical Implementation

- Use explicit and detailed queries to guide the model more effectively.

- Adopt persona-based responses to add a layer of customization and relevance to the model's replies. This involves instructing a language model to respond as if it has a specific character or identity. This can be particularly useful in creating more engaging and tailored interactions in applications such as chatbots, virtual assistants, or creative writing aids. When a model adopts a persona, it attempts to maintain a consistent style, tone, voice, and set of characteristics that match the defined persona throughout the interaction.
- Employ delimiters and structured formats to clearly distinguish different parts of the input or output.
- Encourage the model to "think" by developing a chain of thought, aiding in reasoning and reaching more accurate conclusions.
- Utilize external tools to complement the model's capabilities, especially for tasks that involve precise calculations or require up-to-date information.
- Test modifications systematically through a comprehensive suite of examples to ensure any changes improve the model's performance reliably.

Bibliography

Bibliography

- [1] C. Babbage. Note on the application of machinery to the computation of astronomical and mathematical tables. *Royal Astronomical Society*, 1822.
- [2] P. E. Ceruzzi. *A history of modern computing*. MIT press, 2003.
- [3] W. Commons. File:float example.svg — wikimedia commons, the free media repository, 2009. [Online; accessed 18-August-2017].
- [4] W. Commons. File:ieee 754 double floating point format.svg — wikimedia commons, the free media repository, 2015. [Online; accessed 18-August-2017].
- [5] W. Commons. File:nama machine d’anticythère 1.jpg — wikimedia commons, the free media repository, 2015. [Online; accessed 20-August-2017].
- [6] W. Commons. File:abacus 6.png — wikimedia commons, the free media repository, 2016. [Online; accessed 20-August-2017].
- [7] W. Commons. File:blombos cave engrave ochre 1.jpg — wikimedia commons, the free media repository, 2017. [Online; accessed 18-August-2017].
- [8] W. Commons. File:bits and bytes.svg — wikimedia commons, the free media repository, 2023. [Online; accessed 28-December-2023]. Modification: Border removed.
- [9] E. L. Eisenstein. *The printing press as an agent of change: Communications and cultural transformations in early-modern Europe*. Cambridge University Press, 1980.

- [10] T. Freeth, Y. Bitsakis, X. Moussas, J. H. Seiradakis, A. Tselikas, H. Mangou, M. Zafeiropoulou, R. Hadland, D. Bate, A. Ramsey, et al. Decoding the ancient Greek astronomical calculator known as the Antikythera Mechanism. *Nature*, 444(7119):587, 2006.
- [11] J. Gantz and D. Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007(2012):1–16, 2012.
- [12] C. S. Henshilwood, F. d’Errico, R. Yates, Z. Jacobs, C. Tribolo, G. A. Duller, N. Mercier, J. C. Sealy, H. Valladas, I. Watts, et al. Emergence of modern human behavior: Middle Stone Age engravings from South Africa. *Science*, 295(5558):1278–1280, 2002.
- [13] P. S. marquis de Laplace. *Essai philosophique sur les probabilités*. Bachelier, 1825.
- [14] V. Mayer-Schönberger and K. Cukier. *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2012.
- [15] L. S. Museum. Computing. 1991. [Online; accessed 20-August-2017].
- [16] J. Napier. *Mirifici Logarithmorum Canonis Descriptio, Eiusque Usus, in Utraque Trigonometria; ut etiam in omni Logistica Mathematica, Amplissimi, Facillimi, & Expeditissimi Explicatio*. Edinbvrge, Ex officina A. Hart, 1614.
- [17] S. A. Nefedov. A model of demographic cycles in a traditional society: The case of ancient china. *Social evolution & history*, 3(1):69–80, 2004.
- [18] E. Posner. *Archives in the ancient world*. Harvard University Press, 1972.
- [19] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.
- [20] V. P. Rindova and W. H. Starbuck. Ancient chinese theories of control. *Journal of Management Inquiry*, 6(2):144–159, 1997.

- [21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [22] J. Wallis. *A Treatise of Algebra*. Richard Davis, 1685.
- [23] J. Wallis. *Opera mathematica: De Algebra Tractatus*. Number v. 5 in *Opera mathematica: De Algebra Tractatus; Historicus & Practicus : Anno 1685 Anglice editus; Nunc Auctus Latine. Cum variis Appendicibus ... Operum Mathematicorum Volumen alterum*. Theatrum Sheldonianum, 1693.
- [24] K. Zuse. Rechnender raum (calculating space). 1969. Translation into English by Adrian German and Hector Zenil [Online; accessed 20-August-2017].