

Introduction to Data Science

STAT 3255/5255 @ UConn, Spring 2025

Jun Yan and Students in Spring 2025

2025-02-17

Table of contents

Preliminaries	1
Sources at GitHub	1
Compiling the Classnotes	2
Midterm Project	2
Final Project	3
Adapting to Rapid Skill Acquisition	3
Wishlist	4
Students in 3255	4
Students in 5255	8
Course Logistics	9
Presentation Orders	9
Presentation Task Board	10
Topic Presentation Schedule	11
Final Project Presentation Schedule	13
Contributing to the Class Notes	14
Homework Logistics	15
Quizzes about Syllabus	16
Practical Tips	16
Data analysis	16
Presentation	17
My Presentation Topic (Template)	17
Introduction	17
Sub Topic 1	17
Sub Topic 2	18
Sub Topic 3	18
Conclusion	18

Table of contents

Further Readings	18
1 Introduction	19
1.1 What Is Data Science?	19
1.2 Expectations from This Course	20
1.3 Computing Environment	21
1.3.1 Operating System	21
1.3.2 File System	22
1.3.3 Command Line Interface	23
1.3.4 Python	24
1.4 Data Science Ethics	25
1.4.1 Introduction	25
1.4.2 Principles of Ethical Data Science	25
1.4.3 Ensuring Ethics in Practice	28
1.4.4 Conclusion	30
2 Project Management	31
2.1 Set Up Git/GitHub	31
2.2 Most Frequently Used Git Commands	32
2.3 Tips on using Git:	33
2.4 Pull Request	33
3 Reproducible Data Science	37
3.1 Introduction to Quarto	37
3.2 Compiling the Classnotes	38
3.2.1 Set up your Python Virtual Environment	38
3.2.2 Clone the Repository	39
3.2.3 Render the Classnotes	40
3.2.4 Login Requirements	40
4 Python Refreshment	41
4.1 The Python World	41
4.2 Standard Library	41
4.3 Important Libraries	43

Table of contents

4.4	Writing a Function	44
4.4.1	Monty Hall	45
4.5	Variables versus Objects	46
4.6	Number Representation	49
4.6.1	Integers	49
4.6.2	Floating Number	50
4.7	Virtual Environment	52
4.8	Numpy	54
5	Data Manipulation	55
5.1	Introduction	55
5.2	NYC Crash Data	56
5.3	Accessing Census Data	64
5.3.1	Python Tools for Accessing Census Data	65
5.3.2	Zip-Code Level for NYC Crash Data	65
5.4	Cross-platform Data Format Arrow	73
6	Exercises	77
	References	85

Preliminaries

The notes were developed with Quarto; for details about Quarto, visit <https://quarto.org/docs/books>.

This book is free and is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 United States License.

Sources at GitHub

These lecture notes for STAT 3255/5255 in Spring 2025 represent a collaborative effort between Professor Jun Yan and the students enrolled in the course. This cooperative approach to education was facilitated through the use of GitHub, a platform that encourages collaborative coding and content development. To view these contributions and the lecture notes in their entirety, please visit our GitHub repository at <https://github.com/statds/ids-s25>.

Students contributed to the lecture notes by submitting pull requests to our GitHub repository. This method not only enriched the course material but also provided students with practical experience in collaborative software development and version control.

For those interested, class notes from Fall 2024, Spring 2024, Spring 2023, and Spring 2022 are also publicly accessible. These archives offer insights into the evolution of the course content and the different perspectives brought by successive student cohorts.

Compiling the Classnotes

To reproduce the classnotes output on your own computer, here are the necessary steps. See Section Section 3.2 for details.

- Clone the classnotes repository to an appropriate location on your computer; see Chapter Chapter 2 for using Git.
- Set up a Python virtual environment in the root folder of the source; see Section Section 4.7.
- Activate your virtual environment.
- Install all the packages specified in `requirements.txt` in your virtual environment:

```
pip install -r requirements.txt
```

- For some chapters that need to interact with certain sites that require account information. For example, for Google map services, you need to save your API key in a file named `api_key.txt` in the root folder of the source.
- Render the book with `quarto render` from the root folder on a terminal; the rendered book will be stored under `_book`.

Midterm Project

Reproduce NYC street flood research (Agonafir, Lakhankar, et al., 2022; Agonafir, Pabon, et al., 2022).

Four students will be selected to present their work in a workshop at the 2025 NYC Open Data Week. You are welcome to invite your family and friends to join the the workshop.

Final Project

Students are encouraged to start designing their final projects from the beginning of the semester. There are many open data that can be used. Here is a list of data challenges that you may find useful:

- ASA Data Challenge Expo: big data in 2025
- Kaggle
- DrivenData
- 15 Data Science Hackathons to Test Your Skills in 2025
- If you work on sports analytics, you are welcome to submit a poster to Connecticut Sports Analytics Symposium (CSAS) 2025.
- A good resource for sports analytics is ScoreNetwork.

Adapting to Rapid Skill Acquisition

In this course, students are expected to rapidly acquire new skills, a critical aspect of data science. To emphasize this, consider this insightful quote from VanderPlas (2016):

When a technologically-minded person is asked to help a friend, family member, or colleague with a computer problem, most of the time it's less a matter of knowing the answer as much as knowing how to quickly find an unknown answer. In data science it's the same: searchable web resources such as on-line documentation, mailing-list threads, and StackOverflow answers contain a wealth of information, even (especially?) if it is a topic you've found yourself searching before. Being an effective practitioner of data science is less about memorizing the tool or command you should use for every possible situation, and more about learning to effectively find the information you don't know, whether through a web search engine or another means.

Preliminaries

This quote captures the essence of what we aim to develop in our students: the ability to swiftly navigate and utilize the vast resources available to solve complex problems in data science. Examples tasks are: install needed software (or even hardware); search and find solutions to encountered problems.

Wishlist

This is a wish list from all members of the class (alphabetical order, last name first, comma, then first name). Here is an example.

- Yan, Jun
 - Make practical data science tools accessible to undergraduates.
 - Pass real-world data science project experience to students.
 - Co-develop a Quarto book in collaboration with the students.
 - Train students to participate in real data science competitions.

Add yours through a pull request; note the syntax of nested list in Markdown.

Students in 3255

- Ackerman, John
 - Get comfortable with command line interface
 - Hands-on experience with AI
 - Learn practical tools & tricks for professional data scientist
- Alsadadi, Ammar Shaker
 - Learn about the applications of Data Science in Finance
 - Learn more about time series and random walk
- Chen, Yifei

Wishlist

- Learn more advanced python programming skills.
 - Learn to use github for future projects
 - Get a good grade in this class.
- El Zein, Amer Hani
 - To gain a deeper understanding of data preparation.
 - To develop intuition on what the best tool for a given project is.
- Febles, Xavier Milan
 - Further develop skills with git
 - Learn more about specific tools used for data science
 - Become more comfortable with sql
- Horn, Alyssa Noelle
 - Be confident in using Git and Github
 - Learn how to collaborate with others on projects through Github
- Jun, Joann
 - Become proficient in using GitHub
 - Learn more about the applications of data science
- Kline, Daniel Esteban
- Lagutin, Vladislav
 - Learn how to do data science projects in python and interact with them using git
 - Learn how to do good visualizations of the data; explore appropriate libraries
- Lang, Lang
 - Become more proficient with python
 - Learn about the applications of Data Science

Preliminaries

- Learn how to make collaborative project by using GitHub
 - Have a good grade in this course
- Li, Shiyi
 - Learn to visualize the plots and results using the ggplot package.
 - Learn to use the common functions of the SciPy, scikit-learn, and statsmodels libraries in Python
 - Learn how to query, extract, and manipulate structured and unstructured data in a large database.
 - Learn the basics of artificial neural networks, CNNs for image data, NLP techniques.
 - Learn some of the data analysis models that will be commonly used in the workplace.
 - Learn some common applications of optimization techniques in data analysis.
 - Pass this course with an A grade.
- Lin, Selena
 - Get a good grade in this class.
 - Learn and get familiar with using GitHub.
 - Hands on experience with the data science skills learned in this class.
- Long, Ethan Kenneth
 - Become more comfortable using Git commands and CLI
 - Learn more about the data science field
 - Understand proper coding grammar
 - Develop good learning habits
- Nasejje, Ruth Nicole
 - Develop an organized coding style in python, quarto, & git
 - Learn various packages in python related to data science
 - Deepen knowledge in statistical modeling and data analysis

- Pfeifer, Nicholas Theodore
 - Learn about data science techniques in python
 - Learn and thoroughly practice using git and github
 - Get more comfortable with decision trees and random forests
- Reed, Kyle Daniel
 - Gain full confidence using Git/GitHub and corresponding applications.
 - Understand the workflow in professional data science projects.
 - Build on existing python skills.
- Roy, Luke William
 - Have fun
 - Develop skills in financial data analysis using python and relevant libraries like pandas and numpy.
 - Learn advanced data visualization techniques with a focus on the grammar of graphics.
 - Get an introduction to machine learning via scikit-learn, and explore applications in financial analysis and forensic accounting.
- Schittina, Thomas
 - Become more comfortable using git and GitHub
 - Become more familiar with popular data science packages in Python
- Symula, Sebastian
 - Learn SQL
 - Become better at working through each step in the data science pipeline to make better, cleaner looking projects
- Tamhane, Shubhan
- Tomaino, Mario Anthony

Preliminaries

- Xu, Peiwen
 - Learn some data analysis techniques
 - Learn how to use git and other essential tools for data science

Students in 5255

- Edo, Mezmur Wossenu
 - I hope to become adept working with github.
 - I hope to work on real-World data science projects.
 - I hope to learn about the different machine learning techniques.
- Mundiwala, Mohammad Moiz
 - Become more familiar with collaboration process of programming so that I can be more orderly while working with others.
 - I hope to become more efficient processing data that is messy, unstructured, or unlabeled.
 - Present engaging, intuitive, and interactive figures and animations for complex math and stat concepts.
- Vellore, Ajeeth Krishna
 - Understand the utility provided by GitHub and practice using its tools
 - Learn how to participate in a large-scale development project like how they are done in industry
 - Learn how to code properly and professionally instead of using “backyard” computer science techniques and formatting
 - Understand principles of coding documentation and readability while practicing their application
- Zhang, Gaofei
- Kravette, Noah
 - Become better at program collaboration.

- Become adept with git and github.
- Be able to quickly and efficiently process and analyze any data.
- Gain better skills at data prep, organization, and visualization.
- Learn new helpful statistical tools for data.

Course Logistics

Presentation Orders

The topic presentation order is set up in class.

```
with open('rosters/3255.txt', 'r') as file:
    ug = [line.strip() for line in file]
with open('rosters/5255.txt', 'r') as file:
    gr = [line.strip() for line in file]
presenters = ug + gr

import random
## seed jointly set by the class
random.seed(6895 + 283 + 3184 + 3078 + 5901 + 36)
random.sample(presenters, len(presenters))
## random.shuffle(presenters) # This would shuffle the list in place
```

```
['Li,Shiyi',
 'Jun,Joann',
 'Alsadadi,Ammar Shaker',
 'Lang,Lang',
 'Ackerman,John',
 'Horn,Alyssa Noelle',
 'Xu,Peiwen',
 'Schittina,Thomas',
 'Kline,Daniel Esteban',
```

Preliminaries

```
'Edo,Mezmur Wossenu',  
'Roy,Luke William',  
'Febles,Xavier Milan',  
'Tamhane,Shubhan',  
'Nasejje,Ruth Nicole',  
'Lagutin,Vladislav',  
'Zhang,Gaofei',  
'Long,Ethan Kenneth',  
'El Zein,Amer Hani',  
'Kravette,Noah',  
'Symula,Sebastian',  
'Tomaino,Mario Anthony',  
'Reed,Kyle Daniel',  
'Chen,Yifei',  
'Mundiwala,Mohammad Moiz',  
'Lin,Selena',  
'Pfeifer,Nicholas Theodore',  
'Vellore,Ajeeth Krishna']
```

Switching slots is allowed as long as you find someone who is willing to switch with you. In this case, make a pull request to switch the order and let me know.

You are welcome to choose a topic that you are interested the most, subject to some order restrictions. For example, decision tree should be presented before random forest or extreme gradient boosting. This justifies certain requests for switching slots.

Presentation Task Board

Talk to the professor about your topics. Here are some example tasks:

- Making presentations with Quarto

- Markdown jumpstart
- Effective data science communication
- Import/Export data
- Data manipulation with **Pandas**
- Using package **ace_tools**
- Accessing US census data
- Arrow as a cross-platform data format
- Statistical analysis for proportions and rates
- Database operation with Structured query language (SQL)
- Grammar of graphics
- Handling spatial data
- Spatial data with **GeoPandas**
- Visualize spatial data in a Google map
- Animation
- Support vector machine
- Random forest
- Naive Bayes
- Neural networks
- Deep learning
- TensorFlow
- Autoencoders
- Reinforcement learning
- Developing a Python package
- Web scraping
- Personal webpage on GitHub

Topic Presentation Schedule

The topic presentation is 20 points. It includes:

- Topic selection consultation on week in advance (4 points).
- Delivering the presentation in class (10 points).

Preliminaries

- Contribute to the class notes within two weeks following the presentation (6 points).

Tips on topic contribution:

- No plagiarism (see instructions on Contributing to Class Notes).
- Avoid external graphics.
- Use simulated data.
- Use data from homework assignments.
- Cite article/book references (learn how from our sources).
- Include a subsection of Further Readings.
- Test on your own computer before making a pull request.
- Send me your presentation two days in advance for feedbacks.

Please use the following table to sign up.

Date	Presenter	Topic
02/10	Li, Shiyi	A Primer of Markdown
02/12	Jun, Joann	Making Presentations with Quarto
02/17	Roy, Luke William	Grammar of Graphics with Plotnine
02/19	Lang, Lang	Data manipulation with Pandas
02/24	Ackerman, John	Database operation with Structured query language (SQL)
02/26	Horn, Alyssa Noelle	
03/03	Xu, Peiwen	
03/05	Schittina, Thomas	
03/10	Kline, Daniel	
	Esteban	
03/12	Edo, Mezmur	
	Wossenu	
03/24	Alsadadi, Ammar	
	Shaker	
03/24	Fables, Xavier Milan	
03/31	Tamhane, Shubhan	

Date	Presenter	Topic
03/31	Nasejje, Ruth Nicole	
04/02	Lagutin, Vladislav	
04/02	Zhang, Gaofei	
04/07	Long, Ethan	
	Kenneth	
04/07	El Zein, Amer Hani	
04/09	Kravette, Noah	
04/09	Symula, Sebastian	
04/09	Tomaino, Mario	
	Anthony	
04/12	Reed, Kyle Daniel	
04/12	Chen, Yifei	
04/12	Mundiwala, Mohammad Moiz	Math animations with manim
04/16	Lin, Selena	
04/16	Pfeifer, Nicholas	
	Theodore	
04/16	Vellore, Ajeeth	
	Krishna	

Final Project Presentation Schedule

We use the same order as the topic presentation for undergraduate final presentation. An introduction on how to use Quarto to prepare presentation slides is available under the `templates` directory in the classnotes source tree, thank to Zachary Blanchard, which can be used as a template to start with.

Preliminaries

Date	Presenter
04/21	Shiyi Li; Joann Jun; Ammar Alsadadi; Lang Lang; John Ackerman
04/23	Alyssa Horn; Peiwen Xu; Thomas Schittina; Daniel Kline; Luke Roy
04/28	Xavier Fables; Shubhan Tamhane; Ruth Nasejje; Vladislav Lagutin; Ethan Long
04/30	Amer El Zein; Sebastian Symula; Mario Tomiano; Kyle Reed; Yifei Chen
05/??	Selena Lin; Nick Pfeifer

Contributing to the Class Notes

Contribution to the class notes is through a ‘pull request’.

- Start a new branch and switch to the new branch.
- On the new branch, add a `qmd` file for your presentation
- If using Python, create and activate a virtual environment with `requirements.txt`
- Edit `_quarto.yml` add a line for your `qmd` file to include it in the notes.
- Work on your `qmd` file, test with `quarto render`.
- When satisfied, commit and make a pull request with your `quarto` files and an updated `requirements.txt`.

I have added a template file `mysection.qmd` and a new line to `_quarto.yml` as an example.

For more detailed style guidance, please see my notes on statistical writing.

Plagiarism is to be prevented. Remember that these class notes are publicly available online with your names attached. Here are some resources

on how to avoid plagiarism. In particular, in our course, one convenient way to avoid plagiarism is to use our own data (e.g., NYC Open Data). Combined with your own explanation of the code chunks, it would be hard to plagiarize.

Homework Logistics

Workflow of Submitting Homework Assignment

- Click the GitHub classroom assignment link in HuskCT announcement.
- Accept the assignment and follow the instructions to an empty repository.
- Make a clone of the repo at an appropriate folder on your own computer with `git clone`.
- Go to this folder, add your qmd source, work on it, and group your changes to different commits.
- Push your work to your GitHub repo with `git push`.
- Create a new release and put the generated pdf file in it for ease of grading.

Requirements

- Use the repo from Git Classroom to submit your work. See Chapter Chapter 2.
 - Keep the repo clean (no tracking generated files).
 - * Never “Upload” your files; use the git command lines.
 - * Make commit message informative (think about the readers).
 - Make at least 10 commits and form a style of frequent small commits.

Preliminaries

- Track `quarto` sources only in your repo. See Chapter Chapter 3.
- For the convenience of grading, add your standalone html or pdf output to a release in your repo.
- For standalone pdf output, you will need to have LaTeX installed.

Quizzes about Syllabus

- Do I accept late homework?
- Could you list a few examples of email etiquette?
- How would you lose style points?
- Would you use CLI and GUI?
- How many students will present at 2025 NYC ODW and when will the presentations be?
- What's the first date on which you have to complete something about your final project?
- Can you use AI for any task in this course?
- Anybody needs a reference letter? How could you help me to help you?

Practical Tips

Data analysis

- Use an IDE so you can play with the data interactively
- Collect codes that have tested out into a script for batch processing
- During data cleaning, keep in mind how each variable will be used later
- No keeping large data files in a repo; assume a reasonable location with your collaborators

Presentation

- Don't forget to introduce yourself if there is no moderator.
- Highlight your research questions and results, not code.
- Give an outline, carry it out, and summarize.
- Use your own examples to reduce the risk of plagiarism.

My Presentation Topic (Template)

This section was prepared by John Smith.

Use Markdown syntax. If not clear on what to do, learn from the class notes sources.

- Pay attention to the sectioning levels.
- Cite references with their bib key.
- In examples, maximize usage of data set that the class is familiar with.
- Could use datasets in Python packages or downloadable on the fly.
- Test your section by `quarto render <filename.qmd>`.

Introduction

Here is an overview.

Sub Topic 1

Put materials on topic 1 here

Python examples can be put into `python` code chunks:

Preliminaries

```
# import pandas as pd  
  
# do something
```

Sub Topic 2

Put materials on topic 2 here.

Sub Topic 3

Put materials on topic 3 here.

Conclusion

Put summaries here.

Further Readings

Put links to further materials.

1 Introduction

1.1 What Is Data Science?

Data science is a multifaceted field, often conceptualized as resting on three fundamental pillars: mathematics/statistics, computer science, and domain-specific knowledge. This framework helps to underscore the interdisciplinary nature of data science, where expertise in one area is often complemented by foundational knowledge in the others.

A compelling definition was offered by Prof. Bin Yu in her 2014 Presidential Address to the Institute of Mathematical Statistics. She defines

$$\text{Data Science} = \text{SDC}^3,$$

where

- ‘S’ represents Statistics, signifying the crucial role of statistical methods in understanding and interpreting data;
- ‘D’ stands for domain or science knowledge, indicating the importance of specialized expertise in a particular field of study;
- the three ‘C’s’ denotes computing, collaboration/teamwork, and communication to outsiders.

Computing underscores the need for proficiency in programming and algorithmic thinking, collaboration/teamwork reflects the inherently collaborative nature of data science projects, often requiring teams with diverse skill sets, and communication to outsiders emphasizes the importance of

1 Introduction

translating complex data insights into understandable and actionable information for non-experts.

This definition neatly captures the essence of data science, emphasizing a balance between technical skills, teamwork, and the ability to communicate effectively.

1.2 Expectations from This Course

In this course, students will be expected to achieve the following outcomes:

- **Proficiency in Project Management with Git:** Develop a solid understanding of Git for efficient and effective project management. This involves mastering version control, branching, and collaboration through this powerful tool.
- **Proficiency in Project Reporting with Quarto:** Gain expertise in using Quarto for professional-grade project reporting. This encompasses creating comprehensive and visually appealing reports that effectively communicate your findings.
- **Hands-On Experience with Real-World Data Science Projects:** Engage in practical data science projects that reflect real-world scenarios. This hands-on approach is designed to provide you with direct experience in tackling actual data science challenges.
- **Competency in Using Python and Its Extensions for Data Science:** Build strong skills in Python, focusing on its extensions relevant to data science. This includes libraries like Pandas, NumPy, and Matplotlib, among others, which are critical for data analysis and visualization.

- **Full Grasp of the Meaning of Results from Data Science Algorithms:** Learn to not only apply data science algorithms but also to deeply understand the implications and meanings of their results. This is crucial for making informed decisions based on these outcomes.
- **Basic Understanding of the Principles of Data Science Methods:** Acquire a foundational knowledge of the underlying principles of various data science methods. This understanding is key to effectively applying these methods in practice.
- **Commitment to the Ethics of Data Science:** Emphasize the importance of ethical considerations in data science. This includes understanding data privacy, bias in data and algorithms, and the broader social implications of data science work.

1.3 Computing Environment

All setups are operating system dependent. As soon as possible, stay away from Windows. Otherwise, good luck (you will need it).

1.3.1 Operating System

Your computer has an operating system (OS), which is responsible for managing the software packages on your computer. Each operating system has its own package management system. For example:

- **Linux:** Linux distributions have a variety of package managers depending on the distribution. For instance, Ubuntu uses APT (Advanced Package Tool), Fedora uses DNF (Dandified Yum), and Arch Linux uses Pacman. These package managers are integral to the Linux experience, allowing users to install, update, and manage software packages easily from repositories.

1 Introduction

- **macOS:** macOS uses Homebrew as its primary package manager. Homebrew simplifies the installation of software and tools that aren't included in the standard macOS installation, using simple commands in the terminal.
- **Windows:** Windows users often rely on the Microsoft Store for apps and software. For more developer-focused package management, tools like Chocolatey and Windows Package Manager (Winget) are used. Additionally, recent versions of Windows have introduced the Windows Subsystem for Linux (WSL). WSL allows Windows users to run a Linux environment directly on Windows, unifying Windows and Linux applications and tools. This is particularly useful for developers and data scientists who need to run Linux-specific software or scripts. It saves a lot of trouble Windows users used to have before its time.

Understanding the package management system of your operating system is crucial for effectively managing and installing software, especially for data science tools and applications.

1.3.2 File System

A file system is a fundamental aspect of a computer's operating system, responsible for managing how data is stored and retrieved on a storage device, such as a hard drive, SSD, or USB flash drive. Essentially, it provides a way for the OS and users to organize and keep track of files. Different operating systems typically use different file systems. For instance, NTFS and FAT32 are common in Windows, APFS and HFS+ in macOS, and Ext4 in many Linux distributions. Each file system has its own set of rules for controlling the allocation of space on the drive and the naming, storage, and access of files, which impacts performance, security, and compatibility. Understanding file systems is crucial for tasks such as data recovery, disk partitioning, and managing file permissions, making it an important

1.3 Computing Environment

concept for anyone working with computers, especially in data science and IT fields.

Navigating through folders in the command line, especially in Unix-like environments such as Linux or macOS, and Windows Subsystem for Linux (WSL), is an essential skill for effective file management. The command `cd` (change directory) is central to this process. To move into a specific directory, you use `cd` followed by the directory name, like `cd Documents`. To go up one level in the directory hierarchy, you use `cd ..`. To return to the home directory, simply typing `cd` or `cd ~` will suffice. The `ls` command lists all files and folders in the current directory, providing a clear view of your options for navigation. Mastering these commands, along with others like `pwd` (print working directory), which displays your current directory, equips you with the basics of moving around the file system in the command line, an indispensable skill for a wide range of computing tasks in Unix-like systems.

1.3.3 Command Line Interface

On Linux or MacOS, simply open a terminal.

On Windows, several options can be considered.

- Windows Subsystem Linux (WSL): <https://learn.microsoft.com/en-us/windows/wsl/>
- Cygwin (with X): <https://x.cygwin.com>
- Git Bash: <https://www.gitkraken.com/blog/what-is-git-bash>

To jump start, here is a tutorial: Ubuntu Linux for beginners.

At least, you need to know how to handle files and traverse across directories. The tab completion and introspection supports are very useful.

Here are several commonly used shell commands:

- `cd`: change directory; `..` means parent directory.

1 Introduction

- **pwd**: present working directory.
- **ls**: list the content of a folder; **-l** long version; **-a** show hidden files; **-t** ordered by modification time.
- **mkdir**: create a new directory.
- **cp**: copy file/folder from a source to a target.
- **mv**: move file/folder from a source to a target.
- **rm**: remove a file a folder.

1.3.4 Python

Set up Python on your computer:

- Python 3.
- Python package manager **miniconda** or **pip**.
- Integrated Development Environment (IDE) (Jupyter Notebook; RStudio; VS Code; Emacs; etc.)

I will be using VS Code in class.

Readability is important! Check your Python coding styles against the recommended styles: <https://peps.python.org/pep-0008/>. A good place to start is the Section on “Code Lay-out”.

Online books on Python for data science:

- “Python Data Science Handbook: Essential Tools for Working with Data,” First Edition, by Jake VanderPlas, O’Reilly Media, 2016.
2. “Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython.” Third Edition, by Wes McK-inney, O’Reilly Media, 2022.

1.4 Data Science Ethics

1.4.1 Introduction

Ethics in data science is a fundamental consideration throughout the lifecycle of any project. Data science ethics refers to the principles and practices that guide responsible and fair use of data to ensure that individual rights are respected, societal welfare is prioritized, and harmful outcomes are avoided. Ethical frameworks like the Belmont Report (Protection of Human Subjects of Biomedical & Research, 1979)} and regulations such as the Health Insurance Portability and Accountability Act (HIPAA) (Health & Services, 1996) have established foundational principles that inspire ethical considerations in research and data use. This section explores key principles of ethical data science and provides guidance on implementing these principles in practice.

1.4.2 Principles of Ethical Data Science

1.4.2.1 Respect for Privacy

Safeguarding privacy is critical in data science. Projects should comply with data protection regulations, such as the General Data Protection Regulation (GDPR) or the California Consumer Privacy Act (CCPA). Techniques like anonymization and pseudonymization must be applied to protect sensitive information. Beyond legal compliance, data scientists should consider the ethical implications of using personal data.

The principles established by the Belmont Report emphasize respect for persons, which aligns with safeguarding individual privacy. Protecting privacy also involves limiting data collection to what is strictly necessary. Minimizing the use of identifiable information and implementing secure data storage practices are essential steps. Transparency about how data is used further builds trust with stakeholders.

1 Introduction

1.4.2.2 Commitment to Fairness

Bias can arise at any stage of the data science pipeline, from data collection to algorithm development. Ethical practice requires actively identifying and addressing biases to prevent harm to underrepresented groups. Fairness should guide the design and deployment of models, ensuring equitable treatment across diverse populations.

To achieve fairness, data scientists must assess datasets for representativeness and use tools to detect potential biases. Regular evaluation of model outcomes against fairness metrics helps ensure that systems remain non-discriminatory. The Americans with Disabilities Act (ADA) (Congress, 1990) provides a legal framework emphasizing equitable access, which can inspire fairness in algorithmic design. Collaborating with domain experts and stakeholders can provide additional insights into fairness issues.

1.4.2.3 Emphasis on Transparency

Transparency builds trust and accountability in data science. Models should be interpretable, with clear documentation explaining their design, assumptions, and decision-making processes. Data scientists must communicate results in a way that stakeholders can understand, avoiding unnecessary complexity or obfuscation.

Transparent practices include providing stakeholders access to relevant information about model performance and limitations. The Federal Data Strategy (Team, 2019) calls for transparency in public sector data use, offering inspiration for practices in broader contexts. Visualizing decision pathways and using tools like LIME or SHAP can enhance interpretability. Establishing clear communication protocols ensures that non-technical audiences can engage with the findings effectively.

1.4.2.4 Focus on Social Responsibility

Data science projects must align with ethical goals and anticipate their broader societal and environmental impacts. This includes considering how outputs may be used or misused and avoiding harm to vulnerable populations. Data scientists should aim to use their expertise to promote public welfare, addressing critical societal challenges such as health disparities, climate change, and education access.

Engaging with diverse perspectives helps align projects with societal values. Ethical codes, such as those from the Association for Computing Machinery (ACM) (Computing Machinery (ACM), 2018), offer guidance on using technology for social good. Collaborating with policymakers and community representatives ensures that data-driven initiatives address real needs and avoid unintended consequences. Regular impact assessments help measure whether projects meet their ethical objectives.

1.4.2.5 Adherence to Professional Integrity

Professional integrity underpins all ethical practices in data science. Adhering to established ethical guidelines, such as those from the American Statistical Association (ASA) ((ASA), 2018), ensures accountability. Practices like maintaining informed consent, avoiding data manipulation, and upholding rigor in analyses are essential for maintaining public trust in the field.

Ethical integrity also involves fostering a culture of honesty and openness within data science teams. Peer review and independent validation of findings can help identify potential errors or biases. Documenting methodologies and maintaining transparency in reporting further strengthen trust.

1.4.3 Ensuring Ethics in Practice

1.4.3.1 Building Ethical Awareness

Promoting ethical awareness begins with education and training. Institutions should integrate ethics into data science curricula, emphasizing real-world scenarios and decision-making. Organizations should conduct regular training to ensure their teams remain informed about emerging ethical challenges.

Workshops and case studies can help data scientists understand the complexities of ethical decision-making. Providing access to resources, such as ethical guidelines and tools, supports continuous learning. Leadership support is critical for embedding ethics into organizational culture.

1.4.3.2 Embedding Ethics in Workflows

Ethics must be embedded into every stage of the data science pipeline. Establishing frameworks for ethical review, such as ethics boards or peer-review processes, helps identify potential issues early. Tools for bias detection, explainability, and privacy protection should be standard components of workflows.

Standard operating procedures for ethical reviews can formalize the consideration of ethics in project planning. Developing templates for documenting ethical decisions ensures consistency and accountability. Collaboration across teams enhances the ability to address ethical challenges comprehensively.

1.4.3.3 Establishing Accountability Mechanisms

Clear accountability mechanisms are essential for ethical governance. This includes maintaining documentation for all decisions, establishing audit

1.4 Data Science Ethics

trails, and assigning responsibility for the outputs of data-driven systems. Organizations should encourage open dialogue about ethical concerns and support whistleblowers who raise issues.

Periodic audits of data science projects help ensure compliance with ethical standards. Organizations can benefit from external reviews to identify blind spots and improve their practices. Accountability fosters trust and aligns teams with ethical objectives.

1.4.3.4 Engaging Stakeholders

Ethical data science requires collaboration with diverse stakeholders. Including perspectives from affected communities, policymakers, and interdisciplinary experts ensures that projects address real needs and avoid unintended consequences. Stakeholder engagement fosters trust and aligns projects with societal values.

Public consultations and focus groups can provide valuable feedback on the potential impacts of data science projects. Engaging with regulators and advocacy groups helps align projects with legal and ethical expectations. Transparent communication with stakeholders builds long-term relationships.

1.4.3.5 Continuous Improvement

Ethics in data science is not static; it evolves with technology and societal expectations. Continuous improvement requires regular review of ethical practices, learning from past projects, and adapting to new challenges. Organizations should foster a culture of reflection and growth to remain aligned with ethical best practices.

Establishing mechanisms for feedback on ethical practices can identify areas for development. Sharing lessons learned through conferences and

1 Introduction

publications helps the broader community advance its understanding of ethics in data science.

1.4.4 Conclusion

Data science ethics is a dynamic and integral aspect of the discipline. By adhering to principles of privacy, fairness, transparency, social responsibility, and integrity, data scientists can ensure their work contributes positively to society. Implementing these principles through structured workflows, stakeholder engagement, and continuous improvement establishes a foundation for trustworthy and impactful data science.

2 Project Management

Many tutorials are available in different formats. Here is a YouTube video “Git and GitHub for Beginners — Crash Course”. The video also covers GitHub, a cloud service for Git which provides a cloud back up of your work and makes collaboration with co-workers easy. Similar services are, for example, bitbucket and GitLab.

There are tools that make learning Git easy.

- Here is a collection of online Git exercises that I used for Git training in other courses that I taught.
- Here is a game called *Oh My Git*, an open source game about learning Git!

2.1 Set Up Git/GitHub

Download Git if you don't have it already.

To set up GitHub (other services like Bitbucket or GitLab are similar), you need to

- Generate an SSH key if you don't have one already.
- Sign up an GitHub account.
- Add the SSH key to your GitHub account

See how to get started with GitHub account.

2.2 Most Frequently Used Git Commands

The following seven commands will get you started and they may be all that you need most of the time.

- `git clone`:
 - Used to clone a repository to a local folder.
 - Requires either HTTPS link or SSH key to authenticate.
- `git pull`:
 - Downloads any updates made to the remote repository and automatically updates the local repository.
- `git status`:
 - Returns the state of the working directory.
 - Lists the files that have been modified, and are yet to be or have been staged and/or committed.
 - Shows if the local repository is behind or ahead a remote branch.
- `git add`:
 - Adds new or modified files to the Git staging area.
 - Gives the option to select which files are to be sent to the remote repository
- `git rm`:
 - Used to remove files from the staging index or the local repository.
- `git commit`:
 - Commits changes made to the local repository and saves it like a snapshot.
 - A message is recommended with every commit to keep track of changes made.

2.3 Tips on using Git:

- `git push`:
 - Used to send commits made on local repository to the remote repository.

For more advanced usages:

- `git diff`
- `git branch`
- `git reset`

2.3 Tips on using Git:

- Use the command line interface instead of the web interface (e.g., upload on GitHub)
- Make frequent small commits instead of rare large commits.
- Make commit messages informative and meaningful.
- Name your files/folders by some reasonable convention.
 - Lower cases are better than upper cases.
 - No blanks in file/folder names.
- Keep the repo clean by not tracking generated files.
- Create a `.gitignore` file for better output from `git status`.
- Keep the linewidth of sources to under 80 for better `git diff` view.

2.4 Pull Request

To contribute to an open source project (e.g., our classnotes), use pull requests. Pull requests “let you tell others about changes you’ve pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.”

2 Project Management

Watch this YouTube video: GitHub pull requests in 100 seconds.

The following are step-by-step instructions on how to make a pull request to the class notes contributed by **Nick Pfeifer**..

- Create a fork of the class repository on the GitHub website.
 - Make sure your fork is up to date by clicking **Sync fork** if necessary.
- Clone your fork into a folder on your computer.
 - `git clone https://github.com/GitHub_Username/ids-s25.git`
 - Replace **GitHub_Username** with your personal GitHub Username.
- Check to see if you can access the folder/cloned repository in your code editor.
 - The class notes home page is located in the `index.qmd` file.
- Make a branch and give it a good name.
 - Move into the directory with the cloned repository.
 - Create a branch using:
 - * `git checkout -b branch_name`
 - * Replace **branch_name** with a more descriptive name.
 - You can check your branches using:
 - * `git branch`
 - * The branch in use will have an asterisk to the left of it.
 - If you are not in the right branch you can use the following command:
 - * `git checkout existing-branch`
 - * Replace **existing-branch** with the name of the branch you want to use.
- Run `git status` to verify that no changes have been made.

2.4 Pull Request

- Make changes to a file in the class notes repository.
 - For example: add your wishes to the Wishlist in index.qmd using nested list syntax in markdown.
 - Remember to save your changes.
- Run `git status` again to see that changes have been made.
- Use the add command.
 - `git add filename`
 - Example usage: `git add index.qmd`
- Make a commit.
 - `git commit -m "Informative Message"`
 - Be clear about what you changed and perhaps include your name in the message.
- Push the files to GitHub.
 - `git push origin branch-name`
 - Replace **branch-name** with the name of your current branch.
- Go to your forked repository on GitHub and refresh the page, you should see a button that says **Compare and Pull Request**.
 - Describe the changes you made in the pull request.
 - Click **Create pull request**.

3 Reproducible Data Science

Data science projects should be reproducible to be trustworthy. Dynamic documents facilitate reproducibility. Quarto is an open-source dynamic document preparation system, ideal for scientific and technical publishing. From the official websites, Quarto can be used to:

- Create dynamic content with Python, R, Julia, and Observable.
- Author documents as plain text markdown or Jupyter notebooks.
- Publish high-quality articles, reports, presentations, websites, blogs, and books in HTML, PDF, MS Word, ePub, and more.
- Author with scientific markdown, including equations, citations, cross references, figure panels, callouts, advanced layout, and more.

3.1 Introduction to Quarto

To get started with Quarto, see documentation at [Quarto](#).

For a clean style, I suggest that you use `VS Code` as your IDE. The `ipynb` files have extra formats in plain texts, which are not as clean as `qmd` files. There are, of course, tools to convert between the two representations of a notebook. For example:

```
quarto convert hello.ipynb # converts to qmd
quarto convert hello.qmd   # converts to ipynb
```

3 Reproducible Data Science

We will use Quarto for homework assignments, classnotes, and presentations. You will see them in action through in-class demonstrations. The following sections in the Quarto Guide are immediately useful.

- Markdown basics
- Using Python
- Presentations

A template for homework is in this repo (`hwtemp.qmd`) to get you started with homework assignments.

3.2 Compiling the Classnotes

The sources of the classnotes are at <https://github.com/statds/ids-s25>. This is also the source tree that you will contribute to this semester. I expect that you clone the repository to your own computer, update it frequently, and compile the latest version on your computer (reproducibility).

To compile the classnotes, you need the following tools: Git, Quarto, and Python.

3.2.1 Set up your Python Virtual Environment

I suggest that a Python virtual environment for the classnotes be set up in the current directory for reproducibility. A Python virtual environment is simply a directory with a particular file structure, which contains a specific Python interpreter and software libraries and binaries needed to support a project. It allows us to isolate our Python development projects from our system installed Python and other Python environments.

To create a Python virtual environment for our classnotes:

3.2 Compiling the Classnotes

```
python3 -m venv .ids-s25-venv
```

Here `.ids-s25-venv` is the name of the virtual environment to be created. Choose an informative name. This only needs to be set up once.

To activate this virtual environment:

```
.. .ids-s25-venv/bin/activate
```

After activating the virtual environment, you will see `(.ids-s25-venv)` at the beginning of your shell prompt. Then, the Python interpreter and packages needed will be the local versions in this virtual environment without interfering your system-wide installation or other virtual environments.

To install the Python packages that are needed to compile the classnotes, we have a `requirements.txt` file that specifies the packages and their versions. They can be installed easily with:

```
pip install -r requirements.txt
```

If you are interested in learning how to create the `requirements.txt` file, just put your question into a Google search.

To exit the virtual environment, simply type `deactivate` in your command line. This will return you to your system's global Python environment.

3.2.2 Clone the Repository

Clone the repository to your own computer. In a terminal (command line), go to an appropriate directory (folder), and clone the repo. For example, if you use `ssh` for authentication:

```
git clone git@github.com:statds/ids-s25.git
```

3.2.3 Render the Classnotes

Assuming `quarto` has been set up, we render the classnotes in the cloned repository

```
cd ids-s25
quarto render
```

If there are error messages, search and find solutions to clear them. Otherwise, the html version of the notes will be available under `_book/index.html`, which is default location of the output.

3.2.4 Login Requirements

For some illustrations, you need to interact with certain sites that require account information. For example, for Google map services, you need to save your API key in a file named `api_key.txt` in the root folder of the source. Another example is to access the US Census API, where you would need to register an account and get your Census API Key.

4 Python Refreshment

You have programmed in Python. Regardless of your skill level, let us do some refreshing.

4.1 The Python World

- Function: a block of organized, reusable code to complete certain task.
- Module: a file containing a collection of functions, variables, and statements.
- Package: a structured directory containing collections of modules and an `__init.py__` file by which the directory is interpreted as a package.
- Library: a collection of related functionality of codes. It is a reusable chunk of code that we can use by importing it in our program, we can just use it by importing that library and calling the method of that library with `period(.)`.

See, for example, how to build a Python library.

4.2 Standard Library

Python's has an extensive standard library that offers a wide range of facilities as indicated by the long table of contents listed below. See documentation online.

4 Python Refreshment

The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

Question: How to get the constant e to an arbitrary precision?

The constant is only represented by a given double precision.

```
import math
print("%0.20f" % math.e)
print("%0.80f" % math.e)
```

2.71828182845904509080

2.71828182845904509079559829842764884233474731445312500000000000000000000000

Now use package `decimal` to export with an arbitrary precision.

```
import decimal # for what?

## set the required number digits to 150
decimal.getcontext().prec = 150
decimal.Decimal(1).exp().to_eng_string()
decimal.Decimal(1).exp().to_eng_string()[2:]
```

'7182818284590452353602874713526624977572470936999595749669676277240766303535

4.3 Important Libraries

- NumPy
- pandas
- matplotlib
- IPython/Jupyter
- SciPy
- scikit-learn
- statsmodels

Question: how to draw a random sample from a normal distribution and evaluate the density and distributions at these points?

```
from scipy.stats import norm

mu, sigma = 2, 4
mean, var, skew, kurt = norm.stats(mu, sigma, moments='mvsk')
print(mean, var, skew, kurt)
x = norm.rvs(loc = mu, scale = sigma, size = 10)
x
```

```
2.0 16.0 0.0 0.0
```

```
array([ 4.76746436,  4.63357066, 12.1066937 , -0.28276448, -3.21835845,
        5.03085553,  3.63765044,  2.12102122,  8.33377118, 10.29012074])
```

The pdf and cdf can be evaluated:

```
norm.pdf(x, loc = mu, scale = sigma)
```

```
array([0.07850663, 0.08030099, 0.00409793, 0.08474767, 0.04258684,
       0.07484784, 0.09171749, 0.09968993, 0.02847074, 0.01164447])
```

4.4 Writing a Function

Consider the Fibonacci Sequence 1, 1, 2, 3, 5, 8, 13, 21, 34, The next number is found by adding up the two numbers before it. We are going to use 3 ways to solve the problems.

The first is a recursive solution.

```
def fib_rs(n):
    if (n==1 or n==2):
        return 1
    else:
        return fib_rs(n - 1) + fib_rs(n - 2)

%timeit fib_rs(10)
```

9.02 s ± 93.9 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)

The second uses dynamic programming memoization.

```
def fib_dm_helper(n, mem):
    if mem[n] is not None:
        return mem[n]
    elif (n == 1 or n == 2):
        result = 1
    else:
        result = fib_dm_helper(n - 1, mem) + fib_dm_helper(n - 2, mem)
    mem[n] = result
    return result

def fib_dm(n):
    mem = [None] * (n + 1)
    return fib_dm_helper(n, mem)
```

4.4 Writing a Function

```
%timeit fib_dm(10)
```

2.03 s \pm 82.9 ns per loop (mean \pm std. dev. of 7 runs, 1,000,000 loops each)

The third is still dynamic programming but bottom-up.

```
def fib_dbu(n):
    mem = [None] * (n + 1)
    mem[1] = 1;
    mem[2] = 1;
    for i in range(3, n + 1):
        mem[i] = mem[i - 1] + mem[i - 2]
    return mem[n]

%timeit fib_dbu(500)
```

68.2 s \pm 1.88 s per loop (mean \pm std. dev. of 7 runs, 10,000 loops each)

Apparently, the three solutions have very different performance for larger n .

4.4.1 Monty Hall

Here is a function that performs the Monty Hall experiments. In this version, the host opens only one empty door.

4 Python Refreshment

```
import numpy as np

def montyhall(ndoors, ntrials):
    doors = np.arange(1, ndoors + 1) / 10
    prize = np.random.choice(doors, size=ntrials)
    player = np.random.choice(doors, size=ntrials)
    host = np.array([np.random.choice([d for d in doors
                                      if d not in [player[x], prize[x]]])
                    for x in range(ntrials)])
    player2 = np.array([np.random.choice([d for d in doors
                                      if d not in [player[x], host[x]]])
                       for x in range(ntrials)])
    return {'noswitch': np.sum(prize == player), 'switch': np.sum(prize == p
```

Test it out:

```
montyhall(3, 1000)
montyhall(4, 1000)
```

```
{'noswitch': np.int64(276), 'switch': np.int64(363)}
```

The true value for the two strategies with n doors are, respectively, $1/n$ and $\frac{n-1}{n(n-2)}$.

In the homework exercise, the host opens m doors that are empty. An argument `nempty` could be added to the function.

4.5 Variables versus Objects

In Python, variables and the objects they point to actually live in two different places in the computer memory. Think of variables as pointers to

4.5 Variables versus Objects

the objects they're associated with, rather than being those objects. This matters when multiple variables point to the same object.

```
x = [1, 2, 3] # create a list; x points to the list
y = x        # y also points to the same list in the memory
y.append(4)   # append to y
x            # x changed!
```

```
[1, 2, 3, 4]
```

Now check their addresses

```
print(id(x)) # address of x
print(id(y)) # address of y
```

```
4713323520
```

```
4713323520
```

Nonetheless, some data types in Python are “immutable”, meaning that their values cannot be changed in place. One such example is strings.

```
x = "abc"
y = x
y = "xyz"
x
```

```
'abc'
```

Now check their addresses

4 Python Refreshment

```
print(id(x))    # address of x
print(id(y))    # address of y
```

```
4518906736
4602501648
```

Question: What's mutable and what's immutable?

Anything that is a collection of other objects is mutable, except **tuples**.

Not all manipulations of mutable objects change the object rather than create a new object. Sometimes when you do something to a mutable object, you get back a new object. Manipulations that change an existing object, rather than create a new one, are referred to as “in-place mutations” or just “mutations.” So:

- **All** manipulations of immutable types create new objects.
- **Some** manipulations of mutable types create new objects.

Different variables may all be pointing at the same object is preserved through function calls (a behavior known as “pass by object-reference”). So if you pass a list to a function, and that function manipulates that list using an in-place mutation, that change will affect any variable that was pointing to that same object outside the function.

```
x = [1, 2, 3]
y = x

def append_42(input_list):
    input_list.append(42)
    return input_list

append_42(x)
```

4.6 Number Representation

```
[1, 2, 3, 42]
```

Note that both `x` and `y` have been appended by 42.

4.6 Number Representation

Numbers in a computer's memory are represented by binary styles (on and off of bits).

4.6.1 Integers

If not careful, It is easy to be bitten by overflow with integers when using Numpy and Pandas in Python.

```
import numpy as np

x = np.array(2 ** 63 - 1 , dtype = 'int')
x
# This should be the largest number numpy can display, with
# the default int8 type (64 bits)
```

```
array(9223372036854775807)
```

Note: on Windows and other platforms, `dtype = 'int'` may have to be changed to `dtype = np.int64` for the code to execute. Source: Stackoverflow

What if we increment it by 1?

4 Python Refreshment

```
y = np.array(x + 1, dtype = 'int')  
y  
# Because of the overflow, it becomes negative!
```

```
array(-9223372036854775808)
```

For vanilla Python, the overflow errors are checked and more digits are allocated when needed, at the cost of being slow.

```
2 ** 63 * 1000
```

```
9223372036854775808000
```

This number is 1000 times larger than the prior number, but still displayed perfectly without any overflows

4.6.2 Floating Number

Standard double-precision floating point number uses 64 bits. Among them, 1 is for sign, 11 is for exponent, and 52 are fraction significand, See https://en.wikipedia.org/wiki/Double-precision_floating-point_format. The bottom line is that, of course, not every real number is exactly representable.

If you have played the Game 24, here is a tricky one:

```
8 / (3 - 8 / 3) == 24
```

```
False
```


4.6 Number Representation

Surprise?

There are more.

```
0.1 + 0.1 + 0.1 == 0.3
```

False

```
0.3 - 0.2 == 0.1
```

False

What is really going on?

```
import decimal
decimal.Decimal(0.1)
```

```
Decimal('0.1000000000000000055511151231257827021181583404541015625')
```

```
decimal.Decimal(8 / (3 - 8 / 3))
```

```
Decimal('23.99999999999999989341858963598497211933135986328125')
```

Because the mantissa bits are limited, it can not represent a floating point that's both very big and very precise. Most computers can represent all integers up to 2^{53} , after that it starts skipping numbers.

```
2.1 ** 53 + 1 == 2.1 ** 53
```

```
# Find a number larger than 2 to the 53rd
```

True

4 Python Refreshment

```
x = 2.1 ** 53
for i in range(1000000):
    x = x + 1
x == 2.1 ** 53
```

True

We add 1 to `x` by 1000000 times, but it still equal to its initial value, `2.1 ** 53`. This is because this number is too big that computer can't handle it with precision like add 1.

Machine epsilon is the smallest positive floating-point number `x` such that `1 + x != 1`.

```
print(np.finfo(float).eps)
print(np.finfo(np.float32).eps)
```

```
2.220446049250313e-16
1.1920929e-07
```

4.7 Virtual Environment

Virtual environments in Python are essential tools for managing dependencies and ensuring consistency across projects. They allow you to create isolated environments for each project, with its own set of installed packages, separate from the global Python installation. This isolation prevents conflicts between project dependencies and versions, making your projects more reliable and easier to manage. It's particularly useful when working on multiple projects with differing requirements, or when collaborating with others who may have different setups.

4.7 Virtual Environment

To set up a virtual environment, you first need to ensure that Python is installed on your system. Most modern Python installations come with the `venv` module, which is used to create virtual environments. Here's how to set one up:

- Open your command line interface.
- Navigate to your project directory.
- Run `python3 -m venv myenv`, where `myenv` is the name of the virtual environment to be created. Choose an informative name.

This command creates a new directory named `myenv` (or your chosen name) in your project directory, containing the virtual environment.

To start using this environment, you need to activate it. The activation command varies depending on your operating system:

- On Windows, run `myenv\Scripts\activate`.
- On Linux or MacOS, use `source myenv/bin/activate` or `. myenv/bin/activate`.

Once activated, your command line will typically show the name of the virtual environment, and you can then install and use packages within this isolated environment without affecting your global Python setup.

To exit the virtual environment, simply type `deactivate` in your command line. This will return you to your system's global Python environment.

As an example, let's install a package, like `numpy`, in this newly created virtual environment:

- Ensure your virtual environment is activated.
- Run `pip install numpy`.

This command installs the `requests` library in your virtual environment. You can verify the installation by running `pip list`, which should show `requests` along with its version.

4.8 Numpy

5 Data Manipulation

5.1 Introduction

Data manipulation is crucial for transforming raw data into a more analyzable format, essential for uncovering patterns and ensuring accurate analysis. This chapter introduces the core techniques for data manipulation in Python, utilizing the Pandas library, a cornerstone for data handling within Python's data science toolkit.

Python's ecosystem is rich with libraries that facilitate not just data manipulation but comprehensive data analysis. Pandas, in particular, provides extensive functionality for data manipulation tasks including reading, cleaning, transforming, and summarizing data. Using real-world datasets, we will explore how to leverage Python for practical data manipulation tasks.

By the end of this chapter, you will learn to:

- Import/export data from/to diverse sources.
- Clean and preprocess data efficiently.
- Transform and aggregate data to derive insights.
- Merge and concatenate datasets from various origins.
- Analyze real-world datasets using these techniques.

5.2 NYC Crash Data

Consider a subset of the NYC Crash Data, which contains all NYC motor vehicle collisions data with documentation from NYC Open Data. We downloaded the crash data for the week of June 30, 2024, on February 12, 2025, in CSC format.

```
import numpy as np
import pandas as pd

# Load the dataset
file_path = 'data/nyccrashes_2024w0630_by20250212.csv'
df = pd.read_csv(file_path,
                  dtype={'LATITUDE': np.float32,
                        'LONGITUDE': np.float32,
                        'ZIP CODE': str})

# Replace column names: convert to lowercase and replace spaces with underscores
df.columns = df.columns.str.lower().str.replace(' ', '_')

# Check for missing values
df.isnull().sum()
```

crash_date	0
crash_time	0
borough	542
zip_code	542
latitude	131
longitude	131
location	131
on_street_name	546
cross_street_name	932
off_street_name	1330

5.2 NYC Crash Data

```

number_of_persons_injured      0
number_of_persons_killed       0
number_of_pedestrians_injured  0
number_of_pedestrians_killed   0
number_of_cyclist_injured      0
number_of_cyclist_killed       0
number_of_motorist_injured     0
number_of_motorist_killed      0
contributing_factor_vehicle_1  11
contributing_factor_vehicle_2  450
contributing_factor_vehicle_3  1702
contributing_factor_vehicle_4  1824
contributing_factor_vehicle_5  1862
collision_id                   0
vehicle_type_code_1            33
vehicle_type_code_2            645
vehicle_type_code_3            1714
vehicle_type_code_4            1828
vehicle_type_code_5            1862
dtype: int64

```

Take a peek at the first five rows:

```
df.head()
```

	crash_date	crash_time	borough	zip_code	latitude	longitude	location	
0	06/30/2024	23:17	BRONX	10460	40.838844	-73.878166	(40.838844, -73.87817)	E
1	06/30/2024	8:30	BRONX	10468	40.862732	-73.903328	(40.862732, -73.90333)	V
2	06/30/2024	20:47	NaN	NaN	40.763630	-73.953300	(40.76363, -73.9533)	F
3	06/30/2024	13:10	BROOKLYN	11234	40.617031	-73.919891	(40.61703, -73.91989)	E
4	06/30/2024	16:42	NaN	NaN	NaN	NaN	NaN	3

5 Data Manipulation

A quick summary of the data types of the columns:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1876 entries, 0 to 1875
Data columns (total 29 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   crash_date                            1876 non-null   object
1   crash_time                            1876 non-null   object
2   borough                               1334 non-null   object
3   zip_code                             1334 non-null   object
4   latitude                             1745 non-null   float32
5   longitude                             1745 non-null   float32
6   location                             1745 non-null   object
7   on_street_name                       1330 non-null   object
8   cross_street_name                    944 non-null    object
9   off_street_name                      546 non-null    object
10  number_of_persons_injured             1876 non-null   int64
11  number_of_persons_killed              1876 non-null   int64
12  number_of_pedestrians_injured         1876 non-null   int64
13  number_of_pedestrians_killed         1876 non-null   int64
14  number_of_cyclist_injured             1876 non-null   int64
15  number_of_cyclist_killed              1876 non-null   int64
16  number_of_motorist_injured            1876 non-null   int64
17  number_of_motorist_killed             1876 non-null   int64
18  contributing_factor_vehicle_1         1865 non-null   object
19  contributing_factor_vehicle_2         1426 non-null   object
20  contributing_factor_vehicle_3         174 non-null    object
21  contributing_factor_vehicle_4         52 non-null     object
22  contributing_factor_vehicle_5         14 non-null     object
23  collision_id                          1876 non-null   int64
```


5.2 NYC Crash Data

```
24  vehicle_type_code_1          1843 non-null  object
25  vehicle_type_code_2          1231 non-null  object
26  vehicle_type_code_3          162 non-null  object
27  vehicle_type_code_4           48 non-null  object
28  vehicle_type_code_5           14 non-null  object
dtypes: float32(2), int64(9), object(18)
memory usage: 410.5+ KB
```

Now we can do some cleaning after a quick browse.

```
# Replace invalid coordinates (latitude=0, longitude=0 or NaN) with NaN
df.loc[(df['latitude'] == 0) & (df['longitude'] == 0),
       ['latitude', 'longitude']] = pd.NA
df['latitude'] = df['latitude'].replace(0, pd.NA)
df['longitude'] = df['longitude'].replace(0, pd.NA)

# Drop the redundant 'location' column
df = df.drop(columns=['location'])

# Converting 'crash_date' and 'crash_time' columns into a single datetime column
df['crash_datetime'] = pd.to_datetime(df['crash_date'] + ' '
                                     + df['crash_time'], format='%m/%d/%Y %H:%M', errors='coerce')

# Drop the original 'crash_date' and 'crash_time' columns
df = df.drop(columns=['crash_date', 'crash_time'])
```

Let's get some basic frequency tables of `borough` and `zip_code`, whose values could be used to check their validity against the legitimate values.

```
# Frequency table for 'borough' without filling missing values
borough_freq = df['borough'].value_counts(dropna=False).reset_index()
borough_freq.columns = ['borough', 'count']
```

5 Data Manipulation

```
# Frequency table for 'zip_code' without filling missing values
zip_code_freq = df['zip_code'].value_counts(dropna=False).reset_index()
zip_code_freq.columns = ['zip_code', 'count']
zip_code_freq
```

	zip_code	count
0	NaN	542
1	11207	31
2	11208	28
3	11236	28
4	11101	23
...
164	10470	1
165	11040	1
166	11693	1
167	11415	1
168	10025	1

A comprehensive list of ZIP codes by borough can be obtained, for example, from the New York City Department of Health's UHF Codes. We can use this list to check the validity of the zip codes in the data.

```
# List of valid NYC ZIP codes compiled from UHF codes
# Define all_valid_zips based on the earlier extracted ZIP codes
all_valid_zips = {
    10463, 10471, 10466, 10469, 10470, 10475, 10458, 10467, 10468,
    10461, 10462, 10464, 10465, 10472, 10473, 10453, 10457, 10460,
    10451, 10452, 10456, 10454, 10455, 10459, 10474, 11211, 11222,
    11201, 11205, 11215, 11217, 11231, 11213, 11212, 11216, 11233,
    11238, 11207, 11208, 11220, 11232, 11204, 11218, 11219, 11230,
    11203, 11210, 11225, 11226, 11234, 11236, 11239, 11209, 11214,
```

5.2 NYC Crash Data

```
11228, 11223, 11224, 11229, 11235, 11206, 11221, 11237, 10031,
10032, 10033, 10034, 10040, 10026, 10027, 10030, 10037, 10039,
10029, 10035, 10023, 10024, 10025, 10021, 10028, 10044, 10128,
10001, 10011, 10018, 10019, 10020, 10036, 10010, 10016, 10017,
10022, 10012, 10013, 10014, 10002, 10003, 10009, 10004, 10005,
10006, 10007, 10038, 10280, 11101, 11102, 11103, 11104, 11105,
11106, 11368, 11369, 11370, 11372, 11373, 11377, 11378, 11354,
11355, 11356, 11357, 11358, 11359, 11360, 11361, 11362, 11363,
11364, 11374, 11375, 11379, 11385, 11365, 11366, 11367, 11414,
11415, 11416, 11417, 11418, 11419, 11420, 11421, 11412, 11423,
11432, 11433, 11434, 11435, 11436, 11004, 11005, 11411, 11413,
11422, 11426, 11427, 11428, 11429, 11691, 11692, 11693, 11694,
11695, 11697, 10302, 10303, 10310, 10301, 10304, 10305, 10314,
10306, 10307, 10308, 10309, 10312
}

# Convert ZIP codes to numeric, invalid entries become NaN
df['zip_code_numeric'] = pd.to_numeric(df['zip_code'], errors='coerce')

# Identify invalid ZIP codes (including NaN)
invalid_zips = df[
    df['zip_code_numeric'].isna() | ~df['zip_code_numeric'].isin(all_valid_zips)
]['zip_code']

# Calculate frequency of invalid ZIP codes
invalid_zip_freq = invalid_zips.value_counts(dropna=False).reset_index()
invalid_zip_freq.columns = ['zip_code', 'frequency']

invalid_zip_freq
```

	zip_code	frequency
0	NaN	542

5 Data Manipulation

	zip_code	frequency
1	10065	7
2	11249	4
3	10112	1
4	11040	1

Are missing in zip code and borough always co-occur?

```
# Check if missing values in 'zip_code' and 'borough' always co-occur
# Count rows where both are missing
missing_cooccur = df[['zip_code', 'borough']].isnull().all(axis=1).sum()
# Count total missing in 'zip_code' and 'borough', respectively
total_missing_zip_code = df['zip_code'].isnull().sum()
total_missing_borough = df['borough'].isnull().sum()

# If missing in both columns always co-occur, the number of missing
# co-occurrences should be equal to the total missing in either column
np.array([missing_cooccur, total_missing_zip_code, total_missing_borough])
```

```
array([542, 542, 542])
```

Are there cases where zip_code and borough are missing but the geo codes are not missing? If so, fill in zip_code and borough using the geo codes by reverse geocoding.

First make sure geopy is installed.

```
pip install geopy
```

Now we use model Nominatim in package geopy to reverse geocode.

5.2 NYC Crash Data

```
from geopy.geocoders import Nominatim
import time

# Initialize the geocoder; the `user_agent` is your identifier
# when using the service. Be mindful not to crash the server
# by unlimited number of queries, especially invalid code.
geolocator = Nominatim(user_agent="jyGeopyTry")
```

We write a function to do the reverse geocoding given latitude and longitude.

```
# Function to fill missing zip_code
def get_zip_code(latitude, longitude):
    try:
        location = geolocator.reverse((latitude, longitude), timeout=10)
        if location:
            address = location.raw['address']
            zip_code = address.get('postcode', None)
            return zip_code
        else:
            return None
    except Exception as e:
        print(f"Error: {e} for coordinates {latitude}, {longitude}")
        return None
    finally:
        time.sleep(1) # Delay to avoid overwhelming the service
```

Let's try it out:

```
# Example usage
latitude = 40.730610
longitude = -73.935242
get_zip_code(latitude, longitude)
```

5 Data Manipulation

'11101'

The function `get_zip_code` can then be applied to rows where zip code is missing but geocodes are not to fill the missing zip code.

Once zip code is known, figuring out `borough` is simple because valid zip codes from each borough are known.

5.3 Accessing Census Data

The U.S. Census Bureau provides extensive demographic, economic, and social data through multiple surveys, including the decennial Census, the American Community Survey (ACS), and the Economic Census. These datasets offer valuable insights into population trends, economic conditions, and community characteristics at multiple geographic levels.

There are several ways to access Census data:

- **Census API:** The Census API allows programmatic access to various datasets. It supports queries for different geographic levels and time periods.
- **data.census.gov:** The official web interface for searching and downloading Census data.
- **IPUMS USA:** Provides harmonized microdata for longitudinal research. Available at IPUMS USA.
- **NHGIS:** Offers historical Census data with geographic information. Visit NHGIS.

In addition, Python tools simplify API access and data retrieval.

5.3.1 Python Tools for Accessing Census Data

Several Python libraries facilitate Census data retrieval:

- **censusapi**: The official API wrapper for direct access to Census datasets.
- **census**: A high-level interface to the Census API, supporting ACS and decennial Census queries. See [census on PyPI](#).
- **censusdata**: A package for downloading and processing Census data directly in Python. Available at [censusdata documentation](#).
- **uszipcode**: A library for retrieving Census and geographic information by ZIP code. See [uszipcode on PyPI](#).

5.3.2 Zip-Code Level for NYC Crash Data

Now that we have NYC crash data, we might want to analyze patterns at the zip-code level to understand whether certain demographic or economic factors correlate with traffic incidents. While the crash dataset provides details about individual accidents, such as location, time, and severity, it does not contain contextual information about the neighborhoods where these crashes occur.

To perform meaningful zip-code-level analysis, we need additional data sources that provide relevant demographic, economic, and geographic variables. For example, understanding whether high-income areas experience fewer accidents, or whether population density influences crash frequency, requires integrating Census data. Key variables such as population size, median household income, employment rate, and population density can provide valuable context for interpreting crash trends across different zip codes.

Since the Census Bureau provides detailed estimates for these variables at the zip-code level, we can use the Census API or other tools to retrieve relevant data and merge it with the NYC crash dataset. To access

5 Data Manipulation

the Census API, you need an API key, which is free and easy to obtain. Visit the Census API Request page and submit your email address to receive a key. Once you have the key, you must include it in your API requests to access Census data. The following demonstration assumes that you have registered, obtained your API key, and saved it in a file called `censusAPIkey.txt`.

```
# Import modules
import matplotlib.pyplot as plt
import pandas as pd
import geopandas as gpd
from census import Census
from us import states
import os
import io

with open("censusAPIkey.txt") as f:
    api_key = f.read().strip()
c = Census(api_key)
```

Suppose that we want to get some basic info from ACS data of the year of 2023 for all the NYC zip codes.

```
ACS_YEAR = 2023
ACS_DATASET = "acs/acs5"

# Important ACS variables (including land area for density calculation)
ACS_VARIABLES = {
    "B01003_001E": "Total Population",
    "B19013_001E": "Median Household Income",
    "B02001_002E": "White Population",
    "B02001_003E": "Black Population",
    "B02001_005E": "Asian Population",
```


5.3 Accessing Census Data

```
"B15003_022E": "Bachelor's Degree Holders",  
"B15003_025E": "Graduate Degree Holders",  
"B23025_002E": "Labor Force",  
"B23025_005E": "Unemployed",  
"B25077_001E": "Median Home Value"  
}
```

```
# Convert set to list of strings  
all_valid_zips = list(map(str, all_valid_zips))
```

Let's set up the query to request the ACS data, and process the returned data.

```
acs_data = c.acs5.get(  
    list(ACS_VARIABLES.keys()), {'for': 'zip code tabulation area:*'}  
)  
  
# Convert to DataFrame  
df_acs = pd.DataFrame(acs_data)  
  
# Filter to valid ZIPs only  
df_acs = df_acs[df_acs["zip code tabulation area"].isin(all_valid_zips)]  
  
# Rename columns  
df_acs.rename(columns=ACS_VARIABLES, inplace=True)  
df_acs.rename(columns={"zip code tabulation area": "ZIP Code"}, inplace=True)
```

The population density could be an important factor for crash likelihood. To obtain the population densities, we need the areas of the zip codes. The shape files can be obtained from NYC Open Data.

5 Data Manipulation

```
import requests
import zipfile
import geopandas as gpd

# Define the NYC MODZCTA shapefile URL and extraction directory
shapefile_url = "https://data.cityofnewyork.us/api/geospatial/pri4-ifjk?metho
extract_dir = "MODZCTA_Shapefile"

# Create the directory if it doesn't exist
os.makedirs(extract_dir, exist_ok=True)

# Step 1: Download and extract the shapefile
print("Downloading MODZCTA shapefile...")
response = requests.get(shapefile_url)
with zipfile.ZipFile(io.BytesIO(response.content), "r") as z:
    z.extractall(extract_dir)

print(f"Shapefile extracted to: {extract_dir}")
```

Downloading MODZCTA shapefile...
Shapefile extracted to: MODZCTA_Shapefile

Now we process the shape file to calculate the areas of the polygons.

```
# Step 2: Automatically detect the correct .shp file
shapefile_path = None
for file in os.listdir(extract_dir):
    if file.endswith(".shp"):
        shapefile_path = os.path.join(extract_dir, file)
        break # Use the first .shp file found

if not shapefile_path:
```

5.3 Accessing Census Data

```
raise FileNotFoundError("No .shp file found in extracted directory.")

print(f"Using shapefile: {shapefile_path}")

# Step 3: Load the shapefile into GeoPandas
gdf = gpd.read_file(shapefile_path)

# Step 4: Convert to CRS with meters for accurate area calculation
gdf = gdf.to_crs(epsg=3857)

# Step 5: Compute land area in square miles
gdf['Land Area (sq miles)'] = gdf['geometry'].area / 2_589_988.11
# 1 square mile = 2,589,988.11 square meters

print(gdf[['modzcta', 'Land Area (sq miles)']].head())
```

Using shapefile: MODZCTA_Shapefile/geo_export_d553a5b0-e77f-4879-9a0a-f779d7d2a564.shp

	modzcta	Land Area (sq miles)
0	10001	1.153516
1	10002	1.534509
2	10003	1.008318
3	10026	0.581848
4	10004	0.256876

Now we are ready to merge the two data frames.

```
# Merge ACS data (`df_acs`) directly with MODZCTA land area (`gdf`)
gdf = gdf.merge(df_acs, left_on='modzcta', right_on='ZIP Code', how='left')

# Calculate Population Density (people per square mile)
gdf['Population Density (per sq mile)'] = (
    gdf['Total Population'] / gdf['Land Area (sq miles)']
```

5 Data Manipulation

```
)

# Display first few rows
print(gdf[['modzcta', 'Total Population', 'Land Area (sq miles)',
          'Population Density (per sq mile)']].head())
```

	modzcta	Total Population	Land Area (sq miles)	\
0	10001	27004.0	1.153516	
1	10002	76518.0	1.534509	
2	10003	53877.0	1.008318	
3	10026	38265.0	0.581848	
4	10004	4579.0	0.256876	

	Population Density (per sq mile)
0	23410.171200
1	49864.797219
2	53432.563117
3	65764.650082
4	17825.700993

Some visualization of population density.

```
import matplotlib.pyplot as plt
import geopandas as gpd

# Set up figure and axis
fig, ax = plt.subplots(figsize=(10, 12))

# Plot the choropleth map
gdf.plot(column='Population Density (per sq mile)',
         cmap='viridis', # Use a visually appealing color map
         linewidth=0.8,
```

5.3 Accessing Census Data

```
        edgecolor='black',
        legend=True,
        legend_kwds={'label': "Population Density (per sq mile)",
                     'orientation': "horizontal"},
        ax=ax)

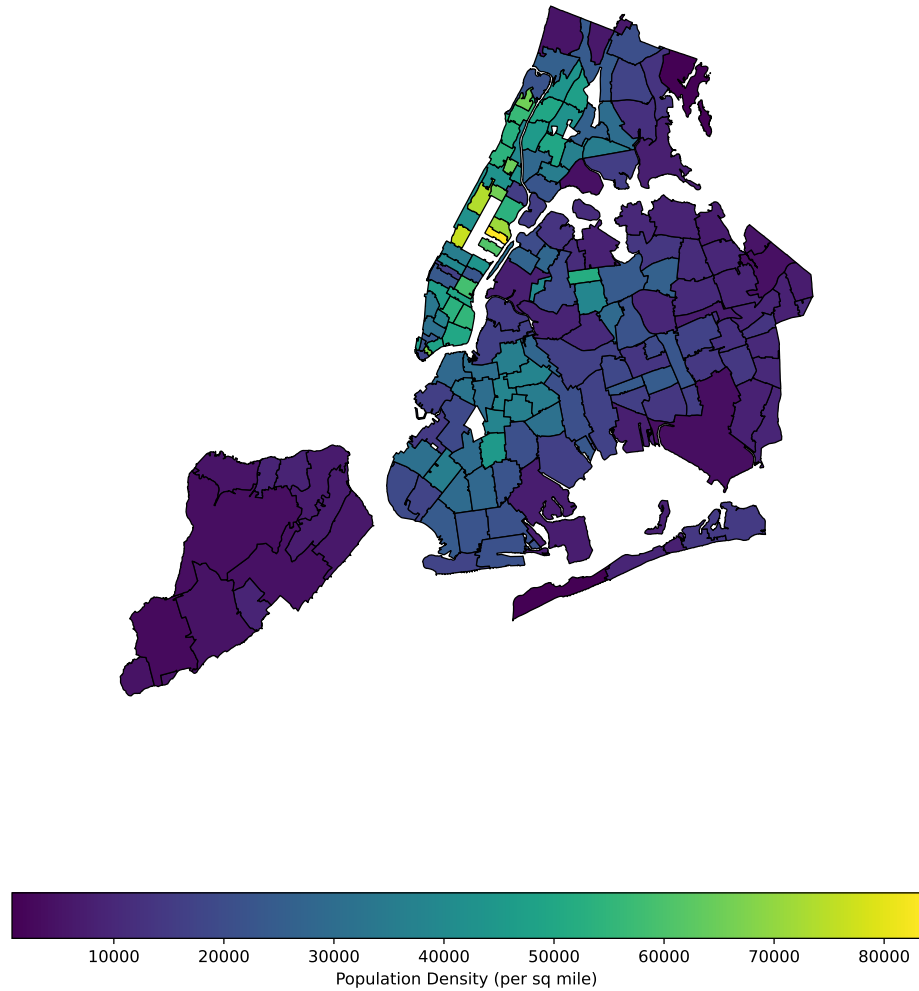
# Add a title
ax.set_title("Population Density by ZIP Code in NYC", fontsize=14)

# Remove axes
ax.set_xticks([])
ax.set_yticks([])
ax.set_frame_on(False)

# Show the plot
plt.show()
```

5 Data Manipulation

Population Density by ZIP Code in NYC



Let's save the merged data for future usage.

5.4 Cross-platform Data Format Arrow

```
csv_output = "data/NYC_Zipcode_Data.csv"

gdf[['modzcta', 'Total Population', 'Land Area (sq miles)',
     'Population Density (per sq mile)']].to_csv(csv_output, index=False)
print(f"CSV file saved: {csv_output}")
```

CSV file saved: data/NYC_Zipcode_Data.csv

5.4 Cross-platform Data Format Arrow

The CSV format (and related formats like TSV - tab-separated values) for data tables is ubiquitous, convenient, and can be read or written by many different data analysis environments, including spreadsheets. An advantage of the textual representation of the data in a CSV file is that the entire data table, or portions of it, can be previewed in a text editor. However, the textual representation can be ambiguous and inconsistent. The format of a particular column: Boolean, integer, floating-point, text, factor, etc. must be inferred from text representation, often at the expense of reading the entire file before these inferences can be made. Experienced data scientists are aware that a substantial part of an analysis or report generation is often the “data cleaning” involved in preparing the data for analysis. This can be an open-ended task — it required numerous trial-and-error iterations to create the list of different missing data representations we use for the sample CSV file and even now we are not sure we have them all.

To read and export data efficiently, leveraging the Apache **Arrow** library can significantly improve performance and storage efficiency, especially with large datasets. The IPC (Inter-Process Communication) file format in the context of Apache Arrow is a key component for efficiently sharing

5 Data Manipulation

data between different processes, potentially written in different programming languages. Arrow's IPC mechanism is designed around two main file formats:

- Stream Format: For sending an arbitrary length sequence of Arrow record batches (tables). The stream format is useful for real-time data exchange where the size of the data is not known upfront and can grow indefinitely.
- File (or Feather) Format: Optimized for storage and memory-mapped access, allowing for fast random access to different sections of the data. This format is ideal for scenarios where the entire dataset is available upfront and can be stored in a file system for repeated reads and writes.

Apache Arrow provides a columnar memory format for flat and hierarchical data, optimized for efficient data analytics. It can be used in Python through the `pyarrow` package. Here's how you can use Arrow to read, manipulate, and export data, including a demonstration of storage savings.

First, ensure you have `pyarrow` installed on your computer (and preferably, in your current virtual environment):

```
pip install pyarrow
```

Feather is a fast, lightweight, and easy-to-use binary file format for storing data frames, optimized for speed and efficiency, particularly for IPC and data sharing between Python and R or Julia.

```
#| eval: false
df.to_feather('data/nyccrashes_cleaned.feather')

# Compare the file sizes of the feather format and the CSV format
import os
```


5.4 Cross-platform Data Format Arrow

```
# File paths
csv_file = 'data/nyccrashes_2024w0630_by20250212.csv'
feather_file = 'data/nyccrashes_cleaned.feather'

# Get file sizes in bytes
csv_size = os.path.getsize(csv_file)
feather_size = os.path.getsize(feather_file)

# Convert bytes to a more readable format (e.g., MB)
csv_size_mb = csv_size / (1024 * 1024)
feather_size_mb = feather_size / (1024 * 1024)

# Print the file sizes
print(f"CSV file size: {csv_size_mb:.2f} MB")
print(f"Feather file size: {feather_size_mb:.2f} MB")
```

Read the feather file back in:

```
#| eval: false
dff = pd.read_feather("data/nyccrashes_cleaned.feather")
dff.shape
```


6 Exercises

1. **Quarto and Git setup** Quarto and Git are two important tools for data science. Get familiar with them through the following tasks. Please use the `templates/hw.qmd` template to document, for each step, what you did, the obstacles you encountered, and how you overcame them. Think of this as a user manual for students who are new to this. Use the command line interface.
 - a. Set up SSH authentication between your computer and your GitHub account.
 - b. Install Quarto onto your computer following the instructions of Get Started.
 - c. Pick a tool of your choice (e.g., VS Code, Jupyter Notebook, Emacs, etc.), follow the instructions to reproduce the example of line plot on polar axis.
 - d. Render the homework into a pdf file and put the file into a release in your GitHub repo.
2. **Working on Homework Problems** All the requirements on homework styles have reasons. Reviewing these questions help you to understand them.
 - a. What are the differences between binary and source files?
 - b. Why do we not want to track binary files in a repo?
 - c. Why do I require pdf output via release?
 - d. Why do I not want your files added via ‘upload’?
 - e. Why do I require line width under 80?
 - f. Why is it not a good idea to have spaces in file/folder names?

6 Exercises

3. **Contributing to the Class Notes** To contribute to the classnotes, you need to have a working copy of the sources on your computer. Document the following steps in a `qmd` file in the form of a step-by-step manual, as if you are explaining them to someone who wants to contribute too. Make at least 10 commits for this task, each with an informative message.
 - a. Create a fork of the notes repo into your own GitHub account.
 - b. Clone it to an appropriate folder on your computer.
 - c. Render the classnotes on your computer; document the obstacles and solutions.
 - d. Make a new branch (and name it appropriately) to experiment with your changes.
 - e. Checkout your branch and add your wishes to the wish list; commit with an informative message; and push the changes to your GitHub account.
 - f. Make a pull request to class notes repo from your fork at GitHub. Make sure you have clear messages to document the changes.
4. **Monty Hall** Consider a generalized Monty Hall experiment. Suppose that the game start with n doors; after you pick one, the host opens $m \leq n - 2$ doors, that show no award. Include sufficient text around the code chunks to explain them.
 - a. Write a function to simulate the experiment once. The function takes two arguments `ndoors` and `nempty`, which represent the number of doors and the number of empty doors showed by the host, respectively, It returns the result of two strategies, switch and no-switch, from playing this game.
 - b. Play this game with 3 doors and 1 empty a few times.
 - c. Play this game with 10 doors and 8 empty a few times.
 - d. Write a function to demonstrate the Monty Hall problem through simulation. The function takes three arguments

- `ndoors`, `nempty`, and `ntrials`, where `ntrial` is the number of trials in a simulation. The function should return the proportion of wins for both the switch and no-switch strategy.
- e. Apply your function with 3 doors (1 empty) and 10 doors (8 empty), both with 1000 trials. Summarize your results.
5. **Approximating π** Write a function to do a Monte Carlo approximation of π . The function takes a Monte Carlo sample size `n` as input, and returns a point estimate of π and a 95% confidence interval. Apply your function with sample size 1000, 2000, 4000, and 8000. Repeat the experiment 1000 times for each sample size and check the empirical probability that the confidence intervals cover the true value of π . Comment on the results.
 6. **Google Billboard Ad** Find the first 10-digit prime number occurring in consecutive digits of e . This was a Google recruiting ad.
 7. **Game 24** The math game 24 is one of the addictive games among number lovers. With four randomly selected cards from a deck of poker cards, use all four values and elementary arithmetic operations ($+$ $-$ \times $/$) to come up with 24. Let \square be one of the four numbers. Let \circ represent one of the four operators. For example,

$$(\square \circ \square) \circ (\square \circ \square)$$
 is one way to group the the operations.
 - a. List all the possible ways to group the four numbers.
 - b. How many possible ways are there to check for a solution?
 - c. Write a function to solve the problem in a brutal force way. The inputs of the function are four numbers. The function returns a list of solutions. Some of the solutions will be equivalent, but let us not worry about that for now.
 8. **NYC Crash Data Cleaning** The NYC motor vehicle collisions data with documentation is available from NYC Open Data. The raw data needs some cleaning.

6 Exercises

- a. Use the filter from the website to download the crash data of the week of June 30, 2024 in CSV format; save it under a directory `data` with an informative name (e.g., `nyccrashes_2024w0630_by20240916.csv`); read the data into a Panda data frame with careful handling of the date time variables.
- b. Clean up the variable names. Use lower cases and replace spaces with underscores.
- c. Check the crash date and time to see if they really match the filter we intended. Remove the extra rows if needed.
- d. Get the basic summaries of each variables: missing percentage; descriptive statistics for continuous variables; frequency tables for discrete variables.
- e. Are there invalid `longitude` and `latitude` in the data? If so, replace them with NA.
- f. Are there `zip_code` values that are not legit NYC zip codes? If so, replace them with NA.
- g. Are there missing in `zip_code` and `borough`? Do they always co-occur?
- h. Are there cases where `zip_code` and `borough` are missing but the geo codes are not missing? If so, fill in `zip_code` and `borough` using the geo codes.
- i. Is it redundant to keep both `location` and the `longitude/latitude` at the NYC Open Data server?
- j. Check the frequency of `crash_time` by hour. Is there a matter of bad luck at exactly midnight? How would you interpret this?
- k. Are the number of persons killed/injured the summation of the numbers of pedestrians, cyclist, and motorists killed/injured? If so, is it redundant to keep these two columns at the NYC Open Data server?
- l. Print the whole frequency table of `contributing_factor_vehicle_1`. Convert lower cases to uppercases and check the frequencies again.
- m. Provided an opportunity to meet the data provider, what sug-

gestions would you make based on your data exploration experience?

9. **NYC Crash Data Exploration** Except for the first question, use the cleaned crash data in feather format.

- a. Construct a contingency table for missing in geocode (latitude and longitude) by borough. Is the missing pattern the same across boroughs? Formulate a hypothesis and test it.
- b. Construct a **hour** variable with integer values from 0 to 23. Plot the histogram of the number of crashes by **hour**. Plot it by borough.
- c. Overlay the locations of the crashes on a map of NYC. The map could be a static map or Google map.
- d. Create a new variable **severe** which is one if the number of persons injured or deaths is 1 or more; and zero otherwise. Construct a cross table for **severe** versus borough. Is the severity of the crashes the same across boroughs? Test the null hypothesis that the two variables are not associated with an appropriate test.
- e. Merge the crash data with the zip code database.
- f. Fit a logistic model with **severe** as the outcome variable and covariates that are available in the data or can be engineered from the data. For example, zip code level covariates can be obtained by merging with the zip code database; crash hour; number of vehicles involved.

10. **NYC Crash severity modeling** Using the cleaned NYC crash data, merged with zipcode level information, predict **severe** of a crash.

- a. Set random seed to 1234. Randomly select 20% of the crashes as testing data and leave the rest 80% as training data.

6 Exercises

- b. Fit a logistic model on the training data and validate the performance on the testing data. Explain the confusion matrix result from the testing data. Compute the F1 score.
 - c. Fit a logistic model on the training data with L_1 regularization. Select the tuning parameter with 5-fold cross-validation in F1 score
 - d. Apply the regularized logistic regression to predict the severity of the crashes in the testing data. Compare the performance of the two logistic models in terms of accuracy, precision, recall, F1-score, and AUC.
11. **Midterm project: Noise complaints in NYC** The NYC Open Data of 311 Service Requests contains all requests from 2010 to present. We consider a subset of it with requests to NYPD on noise complaints that are created between 00:00:00 06/30/2024 and 24:00:00 07/06/2024. The subset is available in CSV format as `data/nypd311w063024noise_by100724.csv`. Read the data dictionary online to understand the meaning of the variables.
- a. Data cleaning.
 - i. Import the data, rename the columns with our preferred styles.
 - ii. Summarize the missing information. Are there variables that are close to completely missing?
 - iii. Are there redundant information in the data? Try storing the data using the Arrow format and comment on the efficiency gain.
 - iv. Are there invalid NYC zipcode or borough? Justify and clean them if yes.
 - v. Are there date errors? Examples are earlier `closed_date` than `created_date`; `closed_date` and `created_date` matching to the second; dates exactly at midnight or noon to the second; `action_update_date` after `closed_date`.

- vi. Summarize your suggestions to the data curator in several bullet points.
- b. Data exploration.
- i. If we suspect that response time may depend on the time of day when a complaint is made, we can compare the response times for complaints submitted during nighttime and daytime. To do this, we can visualize the comparison by complaint type, borough, and weekday (vs weekend/holiday).
 - ii. Perform a formal hypothesis test to confirm the observations from your visualization. Formally state your hypotheses and summarize your conclusions in plain English.
 - iii. Create a binary variable `over2h` to indicate that a service request took two hours or longer to close.
 - iv. Does `over2h` depend on the complaint type, borough, or weekday (vs weekend/holiday)? State your hypotheses and summarize your conclusions in plain English.
- c. Data analysis.
- i. The addresses of NYC police precincts are stored in `data/nypd_precincts.csv`. Use geocoding tools to find their geocode (longitude and latitude) from the addresses.
 - ii. Create a variable `dist2pp` which represent the distance from each request incidence to the nearest police precinct.
 - iii. Create zip code level variables by merging with data from package `uszipcode`.
 - iv. Randomly select 20% of the complaints as testing data with seeds 1234. Build a logistic model to predict `over2h` for the noise complaints with the training data, using all the variables you can engineer from the available data. If you have tuning parameters, justify how they were selected.
 - v. Assess the performance of your model in terms of commonly used metrics. Summarize your results to a New Yorker who is not data science savvy.

References

- Agonafir, C., Lakhankar, T., Khanbilvardi, R., Krakauer, N., Radell, D., & Devineni, N. (2022). A machine learning approach to evaluate the spatial variability of New York City’s 311 street flooding complaints. *Computers, Environment and Urban Systems*, 97, 101854.
- Agonafir, C., Pabon, A. R., Lakhankar, T., Khanbilvardi, R., & Devineni, N. (2022). Understanding New York City street flooding through 311 complaints. *Journal of Hydrology*, 605, 127300.
- (ASA), A. S. A. (2018). *Ethical guidelines for statistical practice*.
- Computing Machinery (ACM), A. for. (2018). *Code of ethics and professional conduct*.
- Congress, U. S. (1990). *Americans with disabilities act of 1990 (ADA)*.
- Health, U. S. D. of, & Services, H. (1996). *Health insurance portability and accountability act of 1996 (HIPAA)*.
- Protection of Human Subjects of Biomedical, N. C. for the, & Research, B. (1979). *The belmont report: Ethical principles and guidelines for the protection of human subjects of research*.
- Team, F. D. S. D. (2019). *Federal data strategy 2020 action plan*.
- VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data*. O’Reilly Media, Inc.

