Introduction to Data Science

STAT 3255/5255 @ UConn, Spring 2025

Jun Yan and Students in Spring 2025 Shiyi Li

2025-02-26

Table of contents

Preliminaries

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

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 if 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 online 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.

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

  - 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 undestanding of data preparation.
  - To develop intution 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

  - 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 familier 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
  - Learn intersection between SQL and Python for a data science project

  - Learn machine learning algorithms like random forest and clustering

- Tomaino, Mario Anthony
- 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
  - Gain confidence in using Git and GitHub for version control and collaboration.
  - Develop a structured approach to data cleaning and preprocessing for complex datasets.
  - Enhance skills in statistical modeling and machine learning techniques relevant to public health research.
  - Improve efficiency in working with large-scale data using Python and SQL.
- Kravette, Noah
  - Become better at program collaboration.
  - Become adept with git and github.
  - Be able to quickly and efficently process and analyze any data.
  - Gain better skills at data prep, organization, and visulization.
  - 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',
 '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 proporations and rates
- Database operation with Structured query language (SQL)
- Grammer 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).
- 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	Performing Statistical Tests (SciPy)

Date	Presenter	Topic
02/26	Horn, Alyssa Noelle	Database operation with Structured query language (SQL)
03/03	El Zein, Amer Hani	
03/05	Schittina, Thomas	Spatial Data With Geopandas & Google Maps
03/10	Kline, Daniel Esteban	
03/12	Edo, Mezmur Wossenu	
03/24	Alsadadi, Ammar Shaker	
03/24	Fables, Xavier Milan	
03/31	Tamhane, Shubhan	
03/31	Nasejje, Ruth Nicole	
04/02	Lagutin, Vladislav	
04/02	Zhang, Gaofei	Random forest
04/07	Long, Ethan Kenneth	
04/07	Xu, Peiwen	
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 availabe under the `templates` directory in the classnotes source tree, thank to Zachary Blanchard, which can be used as a template to start with.

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 Assisngment

- 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.

- 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:

```
# import pandas as pd

# do something
```

Sub Topic 2

Put materials on topic 2 here.

Sub Topic 3

Put matreials on topic 3 here.

Conclusion

Put sumaries 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

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.

- **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

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: Ubunto 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.

- `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.

1.4.2.2 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

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

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 exersices 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 begind 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.

- `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.
- Creat 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."

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.

- 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
```

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 contributed 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:

```
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.

3.3 A Primer of Markdown

This section was prepared by Shiyi Li. I am a senior majoring in Mathematics/ Statistics and minoring in CSE. I aim to graduate from the University of Connecticut in August and continue my master's in Data Science program.

3.3.1 Introduction

Today's presentation, I will introduce some basic syntax on markdown. Markdown is a plain text format, an easy way to write content for a web interface. It is widely used in open-source documentation, including GitHub, R Markdown, Quarto, Jupyter Notebooks, etc. Its syntax is easy to learn, write, read, and seamlessly convert to HTML, PDF, and DOCX formats. I will divide my presentation into three parts, Structuring the Document, Formatting Text, and Enhancing Content.

3.3.2 Structuring the Document

There are some elements can define sections and hierarchy to help organize content in a clear and structured manner.

3.3.2.1 Headings

1. To create heading levels for topics, sections, or subsections, you can add number signs, "#", before you write any content for them.

- Example:

You can code like this:

````
```{markdown}
# Heading 1 (Main Title)

## Heading 2 (Section)

### Heading 3 (Subsection)

#### Heading 4
````
````

```
##### Heading 5

###### Heading 6
```

Be careful, you need to make sure that there is a blank line before and after your heading levels and a space between your number sign "#" and the heading name.

2. Alternatively, you can also use any number of double equal sign, "==", or double dashes sign, "–", to create a heading.

- Example:

You can code like this:

````
```{markdown}
Heading 1 (Main Title)
======
Heading 2 (Section)
------
```
````

However, this method can only be used to create two heading levels like above.

3.3.2.2 Horizontal Rules - line Seperator

- A horizontal rule adds a visual break between parts of text by adding three or more asterisks, "***", dashes,"—", or underscores,"____", on a line by themselves.
- Example:

You can code like this:

````
```{markdown}
This is the part 1.

--------

This is the part 2.

********

This is the part 3.


---------

This is the part 4.
````

- This code chunk will output like this:

This is the part 1.

This is the part 2.

This is the part 3.

This is the part 4.

> Be careful. You need to make sure you add a blank line before and after your separator line.
````

3.3.2.3 Paragraphs

- To create paragraphs, you can add a blank line between two paragraphs just like you usually create paragraphs in an essay.
- Example:

You can code like this:

```
```{markdown}
This is the first paragraph .....................................
.................................................................
.................................................................
... end the first paragraph.

This is the second paragraph ..................................................
.................................................................
.................................................................
...end the second paragraph.
```

- This will output like this:

This is the first paragraph ……. …… ……. end the first paragraph.

This is the second paragraph …… ……. ……. ...end the second paragraph.

3.3.2.4 Blockquotes

1. To highlight multiple line important text, cite multiple line references, or quote multiple line important points, you can add a greater than sign, “>”, at the begining of each line of the text.

- Example:
```

You can code like this:

````
```{markdown}
> This is an important text.
>
> This is a citation/reference.
>
> This is a quotation.
```
````

- This code will output like this:

  This is an important text.

  This is a citation/reference.

  This is a quotation.

- Wrong Example:

If you code like this:

````
```{markdown}
> This is an important text.
> This is a citation/reference.
> This is a quotation.
```
````

- It will output like this:

  This is an important text. This is a citation/reference. This is a quotation.

Be sure to add a blank line with a greater-than sign, ">", otherwise the output will display multiple lines of text in a single line.

2. You can also nest blockquotes by adding two or more greater than signs, “»”.

- Example:

You can code like this:

```
```{markdown}
> This is an important text.
>
>> This is a citation/reference.
>>
>>> This is a quotation.
```

- This code will output like this:

  This is an important text.

  > This is a citation/reference.

  > > This is a quotation.

Be sure to add a blank line before the first line of your blockquotes, otherwise, it will not look right.

3.3.3 Formatting Text

There are some elements to make text stand out, improving emphasis and readability like Bold, Italic, and Strikethrough text.
```

3.3.3.1 Bold

- To bold text, you can add two asterisks, “**“, or underscores,”___“, before and after the text.
- Example:

You can code like this:

```
```{markdown}
**This is the first important text.**

__This is the second important text.__

This is the __third important__ text.

This is the **fourth important** text.

This is the f**if**th important text.
```

- This will output like this:

This is the first important text.

This is the second important text.

This is the **third important** text.

This is the **fourth important** text.

This is the fifth im**portan**t text.

> Be careful do not use underscores, “___“, to bold charachers inside a word, like this”This is the fifth im___portan___t text”.***
```

3.3.3.2 Italic

- To italicize a text, you can add one asterisk, "*", or one underscore,"_", before and after a text.
- Example:

You can code like this:

```
```{markdown}
*ThIs Is the fIrst Important TexT.*

_ThIs Is the second Important TexT._

This is the _Third ImportanT_ text.

This is the *fourth ImportanT* text.

Fifth im*PORtaN*t text.
```

- This code chunk will output like this:

ThIs Is the fIrst Important TexT.

ThIs Is the second Important TexT.

This is the *Third ImportanT* text.

This is the *fourth ImportanT* text.

Fifth im*PORtaN*t text.

> Be careful don't use underscores, "_", to italicize charachers inside a word, like this"Fifth im_porta_nt text".
```

3.3.3.3 Bold & Italic

To bold and italicize for the same text, you can use three asterisks, “***“, or three underscores,”___“, before and after a text.

- Example:

You can code like this:

```
```{markdown}
***This is the first important text.***

___This is the second important text.___

This is the ___third important___ text.

This is the ***fourth important*** text.

Fifth i***mportan***t text.
```

- This code chunk will output like this:

This is the first important text.

This is the second important text.

This is the ***third important*** text.

This is the ***fourth important*** text.

Fifth i***important***t text.

> Be careful don’t use underscores, “___“, to bold and italicize charachers inside a word, like this”Fifth im___porta___nt text”.
```

3.3.3.4 Highlight

- To highlight a text, you can add this sign, "<mark>", before the text and add this sign, "</mark>", after the text.
- Example:

You can code like this:

```
```{markdown}
<mark>This is a text that needs to be highlighted.</mark>

This is a te<mark>xt that needs to be high</mark>lighted.
```

- This code chunk will output like this:

This is a text that needs to be highlighted.

This is a text that needs to be highlighted.

3.3.3.5 Strikethrough - Deleted or Removed Text

- To show a deletion or correction on a text, you can add double tilde signs, "~~", before and after the part of the deletion or correction on your text.
- Example:

You can code like this:

```
```{markdown}
~~This text is struck through~~ This is the correct text.
```

- This code chunk will output like this:

~~This text is struck through~~ This is the correct text.

3.3.4 Enhancing Content

To enhance the illustrative capabilities of your content, there are several elements you can add to your document, including subscript, superscript, lists, tables, footnotes, links, images, math notations, etc.

3.3.4.1 Subscript

- To add a subscript before, after or within a number or word, you can add a tilde symbol, "~", before and after the text you want to subscript.
- Example:

You can code like this:

````
```{markdown}
This is a subscript before and after a word:

~subscript~word~subscript~

This is a subscript within a word:

Wor~111000~ds

This is a subscript before and after a number:

~7878~11111~7878~

This is a subscript within a number:

999~subscript~999
````

- This code chunk will output like this:
````

This is a subscript before and after a word:

$_{\text{subscript}}\text{word}_{\text{subscript}}$

This is a subscript within a word:

$\text{Wor}_{111000}\text{ds}$

This is a subscript before and after a number:

$_{7878}11111_{7878}$

This is a subscript within a number:

$999_{\text{subscript}}999$

> Be sure not to add any spaces or tabs between the two tilde symbols, "~ ~".

3.3.4.2 Superscript

- To add a superscript before, after or within a number or word, you can add a caret symbol, "^", before and after the text you want to superscript.
- Example:

You can code like this:

```
```{markdown}
This is a superscript before and after a word:

^787878^Words^787878^

This is a superscript within a word:

Wor^787878^ds
```
```

```
This is a superscript before and after a number:

^superscript^1111111^superscript^

This is a superscript within a number:

999^superscript^999
```

- This will output like this:

This is a superscript before and after a word:

^{787878}Words^{787878}

This is a superscript within a word:

Wor^{787878}ds

This is a superscript before and after a number:

^{superscript}1111111^{superscript}

This is a superscript within a number:

999^{superscript}999

> Be sure not to add any spaces or tabs between the two caret symbols, “^”.

3.3.4.3 Lists

To organize a list (nested list), you can use ordered or unordered numbers or alphabets followed by a period sign, “.”, dashes, “-”, asterisks, “*“, or plus signs,”+“, in front of line items. Markdown is smart, it will automatically detect and organize a list for you.

1. Using Ordered numbers followed by a period sign:

- Example:

You can code like this:

````
```{markdown}
1. First item
2. Second item
    1. Third item
    2. Fourth item
3. Fifth item
```
````

- This code chunk will output like this:

Using Ordered numbers followed by a period sign:

1. First item
2. Second item
    1. Third item
    2. Fourth item
3. Fifth item

2. Using Unordered numbers followed by a period sign:

- Example:

You can code like this:

````
```{markdown}
2. First item
2. Second item
    2. Third item
    2. Fourth item
2. Fifth item
```
````

```
2. First item
7. Second item
    9. Third item
    2. Fourth item
10. Fifth item
```

- This code chunk will output like this:

Using Unordered numbers followed by a period sign:

2. First item
3. Second item
    2. Third item
    3. Fourth item
4. Fifth item
5. First item
6. Second item
    9. Third item
    10. Fourth item
7. Fifth item

    Be careful, for an unordered list to work as you want, you need to take care of the first number or letter of the first item of your (nested) list, because markdown will order the list starting with the first number or alphabet of your (nested) list.

3. Using dashes:

- Example:

You can code like this:

````
```{markdown}
- First item
- Second item
    - Third item
    - Fourth item
- Fifth item
```
````

- This code chunk will output like this:

Using dashes:

- First item
- Second item
    - Third item
    - Fourth item
- Fifth item

4. Using asterisks:

- Example:

You can code like this:

````
```{markdown}
* First item
* Second item
    * Third item
    * Fourth item
* Fifth item
```
````

- This code chunk will output like this:

Using asterisks:

- First item
- Second item
    - Third item
    - Fourth item
- Fifth item

5. Using plus signs:

- Example:

You can code like this:

````
```{markdown}
+ First item
+ Second item
    + Third item
    + Fourth item
+ Fifth item
```
````

- This code chunk will output like this:

Using plus signs:

- First item
- Second item
    - Third item
    - Fourth item
- Fifth item

6. Using alphabets:

- Example:

You can code like this:

````
```{markdown}
a. First item
b. Second item
    a. Third item
    b. Fourth item
c. Fifth item

w. First item
a. Second item
    c. Third item
    y. Fourth item
a. Fifth item
```
````

- This code chunk will output like this:

Using alphabets:

a. First item

b. Second item

    a. Third item
    b. Fourth item

c. Fifth item

d. First item

e. Second item

    c. Third item
    d. Fourth item

f. Fifth item

7. Using different delimiters:

- Example:

You can code like this:

````
```{markdown}
+ First item
- Second item
    * Third item
    + Fourth item
* Fifth item
```
````

- This code chunk will output like this:

Using different delimiters:

- First item
- Second item
  - Third item
  - Fourth item
- Fifth item

  Using different delimiters in the same list has no effect on the list organized by markdown.

  Be careful, you need to make sure you add a blank line before the list starts.

  You can also use a backslash, “”, to escape a period,”.”, if you do not want to create a list and still need a period after a number or alphabet.

3.3.4.4 Task Lists - To-Do List

- To add a task/to-do list, you can add this sign, "- [ ]", or this sign, "- [x]", before each item in your task/to-do list.
- Example:

You can code like this:

```
```{markdown}
- [ ] Task not completed
- [x] Task completed
```

- This code chunk will output like this:

☐ Task not completed

☒ Task completed

In your rendered HTML file, you can check or uncheck the completion marks in the small box at the front of each task.

3.3.4.5 Links

1. You can add a link in your text by enclosing your added link with parentheses, "()". In addition, you can also optionally add a name or a short description for the link by enclosing them with brackets, "[]", before the link and this will appear as a tooltip when the user hovers over the link

1.

- Example:
```

You can code like this:

````
```{markdown}
We can use the
[Markdown Cheat Sheet](https://www.markdownguide.org/basic-syntax/#code-blocks)
to learn more generally used markdown syntax.
```
````

- This code chunk will output like this:

We can use the Markdown Cheat Sheet to learn more generally used markdown syntax.

2. You can add an Reference Style Link by enclosing the name or description of the link and a number/word pointed to the link with brackets, "[]".

- Example:

You can code like this:

````
```{markdown}
We can use the [Markdown Cheat Sheet][4] to learn more generally used
markdown syntax.

[4]: https://www.markdownguide.org/basic-syntax/#code-blocks
```
````

- This code chunk will output like this:

We can use the Markdown Cheat Sheet to learn more generally used markdown syntax.

> Remember to start a new line and add the link as a footnote after the number or word pointed to the reference enclosed by brackets, "[]".

3.3.4.6 Images

- To add images from your local computer or a website, you can add an exclamation mark, "!", followed by a description or other text enclosing with brackets, "[]", and a path/URL to the image enclosing with parentheses, "()".

- Example:

You can code like this:

```
```{markdown}
![This is a description to an online image](https://today.uconn.edu/2023/01/
uconn-on-campus-construction-update-january-2023/)

![This is a description to a local image](/Users/shiyili/
Desktop/UCONN.jpg)
```

- The render output will show like this:
```

Figure 3.1: This is a description to an online image.

3.3.4.7 Tables

- To create a table, you can use three or more hyphens, "—", and pipes, "|", to create and separate each column respectively.
- Example:

You can code like this:

```
```{markdown}
1. Each cell with the same width in code chunk:

|            | 1st Column | 2nd Column | 3rd Column | ...... |
```
```

```
| ---------- | ---------- | ---------- | ---------- | ------ |
| 1st Row    |    123     |     123    |     123    |   123  |
| 2nd Row    |    123     |     123    |     123    |   123  |
| 3rd Row    |    123     |     123    |     123    |   123  |
| ......     |    123     |     123    |     123    |   123  |

2. Each cell with vary width in code chunk:

|            | 1st Column | 2nd Column | 3rd Column | ...... |
| ------ | ---------- | ---------- | ---------- | ------ |
| 1st Row |     123       |      123      |     123       |    123    |
| 2nd Row      |     123       |     123       |      123      |   123     |
| 3rd Row  |      123      |      123      |     123       |    123    |
| ......          |      123      |     123       |      123      |    123
```

- This code chunk will output like this:

1. Each cell with the same width in code chunk:

	1st Column	2nd Column	3rd Column	……
1st Row	123	123	123	123
2nd Row	123	123	123	123
3rd Row	123	123	123	123
……	123	123	123	123

2. Each cell with vary width in code chunk:

	1st Column	2nd Column	3rd Column	……
1st Row	123	123	123	123
2nd Row	123	123	123	123
3rd Row	123	123	123	123
……	123	123	123	123

Cell width can vary, because markdown will automatically detect and organize the table for you with the same width.***

3. You can align text in the columns to the left, right, or center by adding a colon, ":", to the left, right, or on both side of the hyphens, "—", within the header (Cone, 2025).

- Example:

You can code like this:

```
```{markdown}
|            | 1st Column | 2nd Column | 3rd Column | ...... |
| :----------: | :---------- | :----------: | ----------: | ------: |
| 1st Row    |            123|       123     |  123          |        123|
| 2nd Row    |123            |    123        |           123 |123        |
| 3rd Row    |     123       |123            |     123       |   123     |
| ......     |   123         |            123| 123           |123        |
```

- This code chunk will output like this:

	1st Column	2nd Column	3rd Column	……
1st Row	123	123	123	123
2nd Row	123	123	123	123
3rd Row	123	123	123	123
……	123	123	123	123

3.3.4.8 Code

1. To add an inline code, you can add a backtick, " `", before and after a word or a text.

- Example:
```

You can code like this:

````
```{markdown}
This is my `inline` code.

`This is my inline code.`
```
````

- This will output like this:

This is my `inline` code.

`This is my inline code.`

2. If you want to display a text with a backtick, “ ‘”“, as an inline code, you can add a double backtick,”““, before and after the text.

- Example:

You can code like this:

````
```{markdown}
``This is a `text` with backticks.``
```
````

- This will output like this:

``This is a `text` with backticks.``

3. To add a code block, you can indent each line of the text with more than four spaces or one tab.

- Example:

You can code like this:

````
```{markdown}
    This is a code block.

        This is a code block.

            This is a code block.

    This is a code block.
```
````

- This will output like this:

```
This is a code block.

    This is a code block.

        This is a code block.

This is a code block.
```

4. To add a fenced code block, you can add a blank line begining with three backticks, “ “‘”, or three tilde signs, “~~~”, before and after your code blok.

- Example:

You can code like this:

````
```{markdown}
    ```
    {
    This is my fenced code block.
    This is my fenced code block.
    }
    ```
```
````

- This will output like this:

```
{
This is my fenced code block.
This is my fenced code block.
}
```

5. To add a nonexecutive Python code chunk, you can first add a fence code block for your code, then add a word, “python”, after the three backticks, “ “‘”, in the first line of your fence code block.

- Example :

You can code like this:

````
    ```python
    print("This is a Python code chunk.")
    ```
````

- This will output like this:

```
print("This is a Python code chunk.")
```

6. To add an executive Python code chunk, you can first add a fence code block for your code, then add a word, “python”, with brackets, “{}”after the three backticks, “ “‘”, in the first line of your fence code block.

- Example:

You can code like this:

````
    ```{python}
    print("This is a Python code chunk.")
    ```
````

- This will output like this:

```
print("This is a Python code chunk.")
```

```
This is a Python code chunk.
```

7. You can also make an executive Python code chunk not execute the commands inside the code chunk by adding "#| eval: false" to the first line of your code block.

- Example:

You can code like this:

````
    ```{python}
    #| eval: false

    print("This is a Python code chunk.")
    ```
````

- This will output like this:

```
print("This is a Python code chunk.")
```

If you add just "#| eval: true" to your code chunk, this code chunk will execute the commands as usual and output the results.

If you just add "#| echo: false" to your code chunk, then your code chunk will not be displayed in your rendered output, but the commands in your code chunk will still be executed as usual and the result of the code chunk will be displayed.

If you just add "#| output: false" to your code chunk, then the commands in your code chunk will be displayed as usual in the rendered output, but the result of your code chunk will not be displayed.

3.3.4.9 Math Notation - Using LaTeX in Markdown

1. To displace an inline math equation, you can add a dollar sign, "$", before and after the math equation.

- Example:

You can code like this:

```
```{markdown}
This is a quadratic equation, $ax^2+bx+c=0$.

This is a quadratic equation roots formula, $x = {(-b \pm \sqrt{b^2-4ac})}/{2
```

- This code chunk will output like this:

This is a quadratic equation, $ax^2 + bx + c = 0$.

This is a quadratic equation roots formula, $x = (-b \pm \sqrt{b^2 - 4ac})/2b$.

> In an inline math equation, make sure you use the division symbol instead of "\frac{}{}".

2. You can center an math equation by adding double dollar signs, "$$", before and after the math equation.

- Example:

You can code like this:

```
```{markdown}
This is a quadratic equation
$$
ax^2+bx+c=0.
$$
```

```
This is a quadratic equation roots formula
$$
x = {(-b \pm \sqrt{b^2-4ac})}/{2b}.
$$
```

- This code chunk will output like this:

This is a quadratic equation

$$ax^2 + bx + c = 0.$$

This is a quadratic equation roots formula

$$x = (-b \pm \sqrt{b^2 - 4ac})/2b.$$

Be careful not to add punctuation before a centered math equation.

If you want to add a period after a centered math equation, make sure you add it after the end of the math equation and before the second double dollar sign, "$$".

3.3.4.10 Footnotes

- To add notes and refercences without confusing for body content, you can add a caret sign, "^", follwed by a identifier number or words without any spaces and tabs inside a brackets, "[]".
- Example:

you can code like this:

````
```{markdown}
This is a body paragraph, and I have a note [^short_footnotes] want to
add here. [^long_footnotes]

[^short_footnotes]: This is a notes.

[^long_footnotes]: This is a notes with multiple paragraphs.
    You can include paragraphs in your footnotes using indentation like this
    ```
    {
        This is a fenced code block.
    }
    ```
    This is the end of this footnote.
````

- This code chunk will output like this:

This is a body paragraph, and I have two notes [1] want to add here. [2]

> Clicking on the footnote number in the body content, will take you to the location where the footnote exists in the document.
>
> All footnotes are automatically numbered sequentially at the end of the rendered HTML files and at the bottom of the page where the footnote exists in rendered PDF files.

[1] This is a notes.

[2] This is a notes with multiple paragraphs. You can include paragraphs in your footnotes using indentation like this.

```
print(This is a fenced code block.)

You can add any thing inside this fenced code block.
```

This is the end of this footnote.
```

In the rendered HTML file, each footnote displayed at the end of the document is followed by a link that, when clicked, takes you back to the specific location of the footnote in your body content. You can also move your mouse over the footnote number in the body content, and the content of the footnote will automatically appear below the footnote number.

3.3.5 Conclusion

Markdown is a simple yet powerful way to format text for documentation, blogging, and technical writing. With its easy-to-read syntax, you can structure documents, highlight key points, and present data effectively. It's widely used in open-source projects, academic writing, and web content due to its flexibility and seamless conversion to HTML, PDF, and DOCX. Mastering Markdown can help you create clear, well-organized content with minimal effort.

3.3.6 Further Readings

1. [Markdown Cheet Sheet] (Cone, 2025): A quick reference to the Markdown syntax.
2. [Quarto Markdown Basic] (Dervieux, 2025): Markdown basic for Quarto.
3. [GitHub Flavored Markdown (GFM)] (MacFarlane, 2019): Markdown features specific to GitHub.
4. [Jupyter Notebook Markdown] (MacFarlane, 2006): Use Markdown for interactive data science.

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 libratry.

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.

> 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 arbitary 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
2.71828182845904509079559829842764884233474731445312500000000000000000000000000
```

Now use package `decimal` to export with an arbitary 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:]
```

```
'718281828459045235360287471352662497757247093699959574966967627724076630353
```

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)
```

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

```
2.03 s ± 82.9 ns per loop (mean ± 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 ± 1.88 s per loop (mean ± 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.

```
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 == pl
```

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

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

```
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)
```

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

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

4.6 Number Representation

Numers 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?

```
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
```

```
92233720368547758080000
```

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
```

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('23.999999999999989341858963598497211933135986328125')
```

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
```

```
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.

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 undersco
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
```

```
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	c
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)	E
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

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
```

```
 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 `latitute` and `longitude` columns
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 legitmate values.

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

```
# 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,
```

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

# Identify invalid ZIP codes (including NaN)
invalid_zips = df[
    df['zip_code'].isna() | ~df['zip_code'].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
1	10065	7
2	11249	4
3	10112	1
4	11040	1

As it turns out, the collection of valid NYC zip codes differ from different sources. From United States Zip Codes, `10065` appears to be a valid NYC zip code. Under this circumstance, it might be safer to not remove any zip code from the data.

To be safe, let's concatenate valid and invalid zips.

```
# Convert invalid ZIP codes to a set of strings
invalid_zips_set = set(invalid_zip_freq['zip_code'].dropna().astype(str))

# Convert all_valid_zips to a set of strings (if not already)
valid_zips_set = set(map(str, all_valid_zips))

# Merge both sets
merged_zips = invalid_zips_set | valid_zips_set  # Union of both sets
```

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.

```
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 lattitude 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)
```

```
'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 `burough` 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 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

api_key = open("censusAPIkey.txt").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. The variable names can be found in the ACS variable documentation.

```
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",
    "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
merged_zips = list(map(str, merged_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': f'zip code tabulation area:{",".join(merged_zips)}'}
    )
```

```
# Convert to DataFrame
df_acs = pd.DataFrame(acs_data)

# Rename columns
df_acs.rename(columns=ACS_VARIABLES, inplace=True)
df_acs.rename(columns={"zip code tabulation area": "ZIP Code"}, inplace=True)
```

We could save the ACS data `df_acs` in feather format (see next Section).

```
#! eval: false
df_acs.to_feather("data/acs2023.feather")
```

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.

```
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:
    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_1daca795-0288-4cf1-be6c-e69d6ffefeee.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
```

Let's export this data frame for future usage in feather format (see next Section).

```
gdf[['modzcta', 'land_area_sq_miles']].to_feather('data/nyc_zip_areas.feather
```

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['popdensity_per_sq_mile'] = (
    gdf['Total Population'] / gdf['land_area_sq_miles']
    )

# Display first few rows
print(gdf[['modzcta', 'Total Population', 'land_area_sq_miles',
    'popdensity_per_sq_mile']].head())
```

```
  modzcta  Total Population  land_area_sq_miles  popdensity_per_sq_mile
0   10001           27004.0            1.153516            23410.171200
1   10002           76518.0            1.534509            49864.797219
2   10003           53877.0            1.008318            53432.563117
3   10026           38265.0            0.581848            65764.650082
4   10004            4579.0            0.256876            17825.700993
```

Some visualization of population density.

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

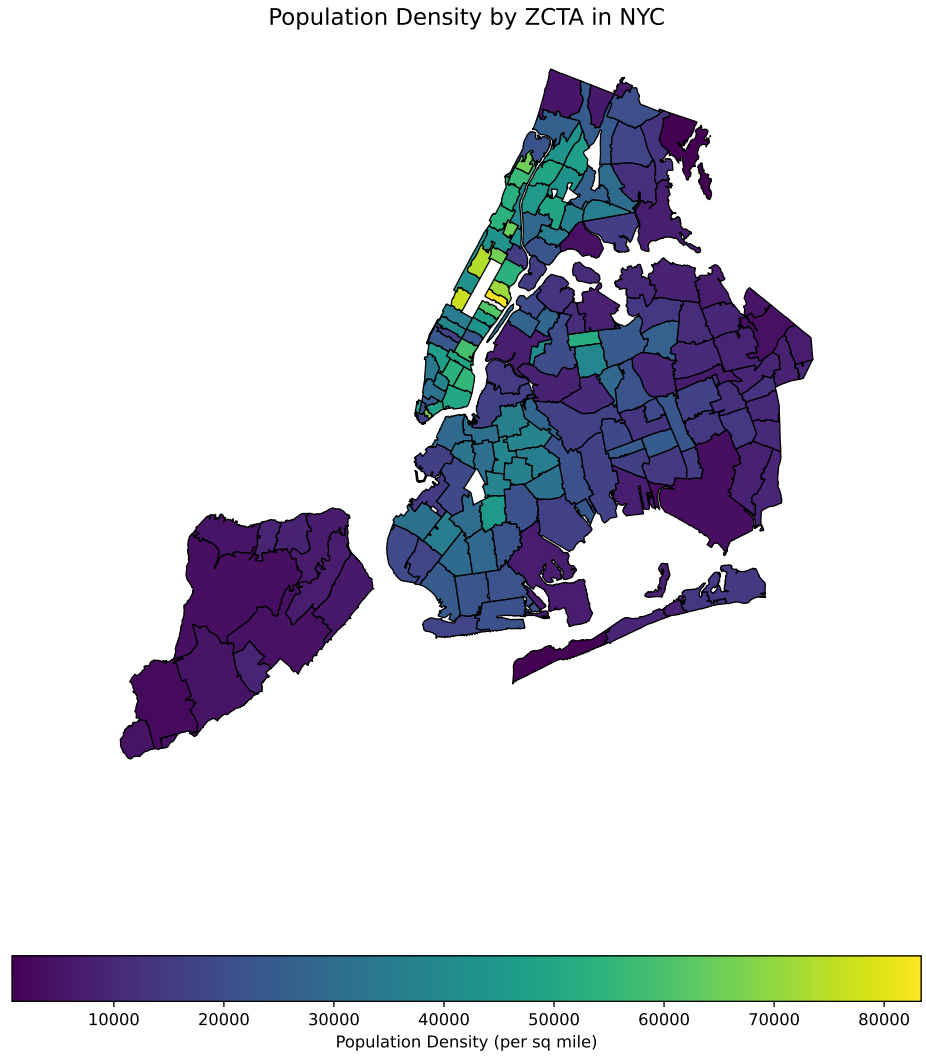
# Set up figure and axis
fig, ax = plt.subplots(figsize=(10, 12))

# Plot the choropleth map
gdf.plot(column='popdensity_per_sq_mile',
         cmap='viridis',  # Use a visually appealing color map
         linewidth=0.8,
         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 ZCTA in NYC", fontsize=14)

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

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


Population Density by ZCTA in NYC
10000
20000
30000
40000
50000
60000
70000
80000
Population Density (per sq mile)

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 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 preferrably, 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.

The following code processes the cleaned data in CSV format from Mohammad Mundiwala and write out in Arrow format.

```
import pandas as pd

# Read CSV, ensuring 'zip_code' is string and 'crash_datetime' is parsed as d
df = pd.read_csv('data/nyc_crashes_cleaned_mm.csv',
                 dtype={'zip_code': str},
                 parse_dates=['crash_datetime'])

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

# Move 'crash_datetime' to the first column
df = df[['crash_datetime'] + df.drop(columns=['crash_datetime']).columns.toli

df['zip_code'] = df['zip_code'].astype(str).str.rstrip('.0')
```

```
df = df.sort_values(by='crash_datetime')

df.to_feather('nyccrashes_cleaned.feather')
```

Let's compare the file sizes of the feather format and the CSV format.

```
import os

# 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")
```

```
CSV file size: 0.34 MB
Feather file size: 0.19 MB
```

Read the feather file back in:

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

6 Statistical Tests and Models

6.1 Tests for Exploratory Data Analysis

A collection of functions are available from `scipy.stats`.

- Comparing the locations of two samples
  - `ttest_ind`: t-test for two independent samples
  - `ttest_rel`: t-test for paired samples
  - `ranksums`: Wilcoxon rank-sum test for two independent samples
  - `wilcoxon`: Wilcoxon signed-rank test for paired samples
- Comparing the locations of multiple samples
  - `f_oneway`: one-way ANOVA
  - `kruskal`: Kruskal-Wallis H-test
- Tests for associations in contigency tables
  - `chi2_contingency`: Chi-square test of independence of variables
  - `fisher_exact`: Fisher exact test on a 2x2 contingency table
- Goodness of fit
  - `goodness_of_fit`: distribution could contain unspecified parameters
  - `anderson`: Anderson-Darling test
  - `kstest`: Kolmogorov-Smirnov test

- `chisquare`: one-way chi-square test
- `normaltest`: test for normality

Since R has a richer collections of statistical functions, we can call R function from Python with `rpy2`. See, for example, a blog on this subject.

For example, `fisher_exact` can only handle 2x2 contingency tables. For contingency tables larger than 2x2, we can call `fisher.test()` from R through `rpy2`. See this StackOverflow post. Note that the `.` in function names and arguments are replaced with `_`.

```
import pandas as pd
import numpy as np
import rpy2.robjects.numpy2ri
from rpy2.robjects.packages import importr
rpy2.robjects.numpy2ri.activate()

stats = importr('stats')

w0630 = pd.read_feather("data/nyccrashes_cleaned.feather")
w0630["injury"] = np.where(w0630["number_of_persons_injured"] > 0, 1, 0)
m = pd.crosstab(w0630["injury"], w0630["borough"])
print(m)

res = stats.fisher_test(m.to_numpy(), simulate_p_value = True)
print(res)
```

```
Loading custom .Rprofileborough  BRONX  BROOKLYN  MANHATTAN  QUEENS  STATEN I
injury
0           149       345        164     249             65
1           129       266        127     227             28

    Fisher's Exact Test for Count Data with simulated p-value (based on
    2000 replicates)
```

```
data:  structure(c(149L, 129L, 345L, 266L, 164L, 127L, 249L, 227L, 65L, 28L), dim = c(2L, 5L
p-value = 0.02949
alternative hypothesis: two.sided
```

6.2 Statistical Modeling

Statistical modeling is a cornerstone of data science, offering tools to understand complex relationships within data and to make predictions. Python, with its rich ecosystem for data analysis, features the `statsmodels` package— a comprehensive library designed for statistical modeling, tests, and data exploration. `statsmodels` stands out for its focus on classical statistical models and compatibility with the Python scientific stack (`numpy`, `scipy`, `pandas`).

6.2.1 Installation of `statsmodels`

To start with statistical modeling, ensure `statsmodels` is installed:

Using pip:

```
pip install statsmodels
```

6.2.2 Linear Model

Let's simulate some data for illustrations.

```
import numpy as np

nobs = 200
ncov = 5
np.random.seed(123)
x = np.random.random((nobs, ncov)) # Uniform over [0, 1)
beta = np.repeat(1, ncov)
y = 2 + np.dot(x, beta) + np.random.normal(size = nobs)
```

Check the shape of `y`:

```
y.shape
```

```
(200,)
```

Check the shape of `x`:

```
x.shape
```

```
(200, 5)
```

That is, the true linear regression model is

$$y = 2 + x_1 + x_2 + x_3 + x_4 + x_5 + \epsilon.$$

A regression model for the observed data can be fitted as

```
import statsmodels.api as sma
xmat = sma.add_constant(x)
mymod = sma.OLS(y, xmat)
myfit = mymod.fit()
myfit.summary()
```

Dep. Variable:	y	**R-squared:**	0.309
Model:	OLS	**Adj. R-squared:**	0.292
Method:	Least Squares	**F-statistic:**	17.38
Date:	Mon, 24 Feb 2025	**Prob (F-statistic):**	3.31e-14
Time:	11:56:32	**Log-Likelihood:**	-272.91
No. Observations:	200	**AIC:**	557.8
Df Residuals:	194	**BIC:**	577.6
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	1.8754	0.282	6.656	0.000	1.320	2.431
x1	1.1703	0.248	4.723	0.000	0.682	1.659
x2	0.8988	0.235	3.825	0.000	0.435	1.362
x3	0.9784	0.238	4.114	0.000	0.509	1.448
x4	1.3418	0.250	5.367	0.000	0.849	1.835
x5	0.6027	0.239	2.519	0.013	0.131	1.075

Omnibus:	0.810	**Durbin-Watson:**	1.978
Prob(Omnibus):	0.667	**Jarque-Bera (JB):**	0.903
Skew:	-0.144	**Prob(JB):**	0.637
Kurtosis:	2.839	**Cond. No.**	8.31

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Questions to review:

- How are the regression coefficients interpreted? Intercept?
- Why does it make sense to center the covariates?

Now we form a data frame with the variables

```
import pandas as pd
df = np.concatenate((y.reshape((nobs, 1)), x), axis = 1)
df = pd.DataFrame(data = df,
                  columns = ["y"] + ["x" + str(i) for i in range(1,
                  ncov + 1)])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   y       200 non-null    float64
 1   x1      200 non-null    float64
 2   x2      200 non-null    float64
 3   x3      200 non-null    float64
 4   x4      200 non-null    float64
 5   x5      200 non-null    float64
dtypes: float64(6)
memory usage: 9.5 KB
```

Let's use a formula to specify the regression model as in R, and fit a robust linear model (`rlm`) instead of OLS. Note that the model specification and the function interface is similar to R.

```
import statsmodels.formula.api as smf
mymod = smf.rlm(formula = "y ~ x1 + x2 + x3 + x4 + x5", data = df)
myfit = mymod.fit()
myfit.summary()
```

Dep. Variable:	y	**No. Observations:**	200
Model:	RLM	**Df Residuals:**	194
Method:	IRLS	**Df Model:**	5
Norm:	HuberT		
Scale Est.:	mad		
Cov Type:	H1		
Date:	Mon, 24 Feb 2025		
Time:	11:56:32		
No. Iterations:	16		

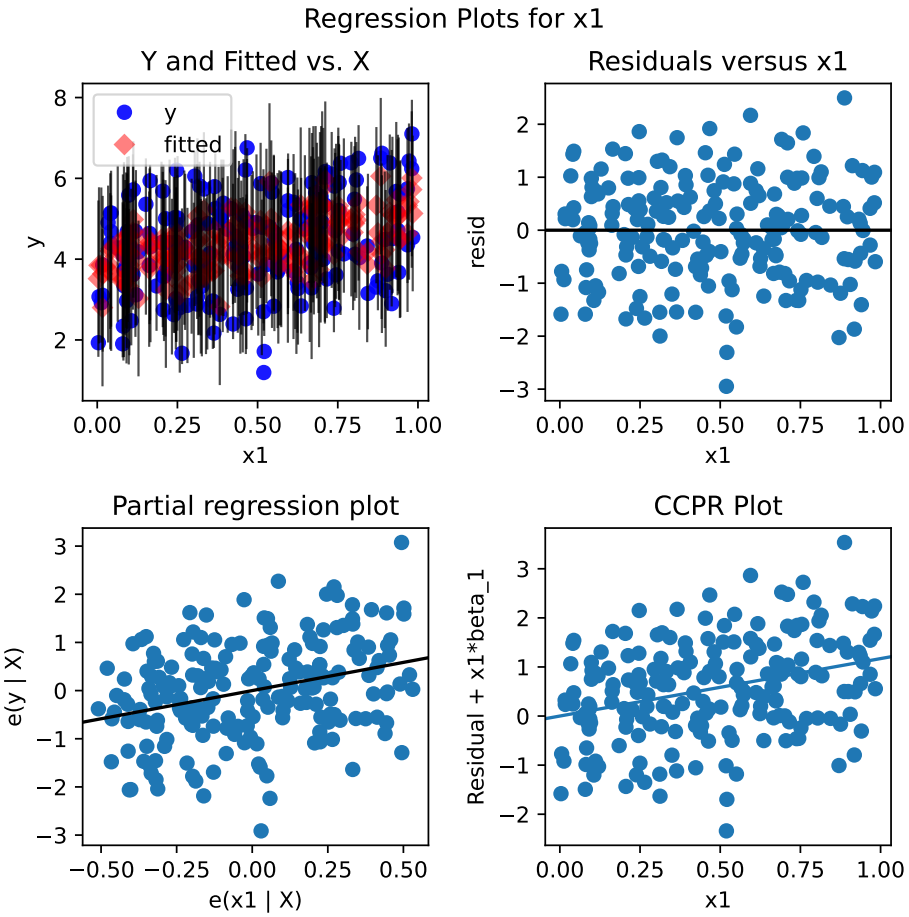
	coef	std err	z	P> |z|	[0.025	0.975]
Intercept	1.8353	0.294	6.246	0.000	1.259	2.411
x1	1.1254	0.258	4.355	0.000	0.619	1.632
x2	0.9664	0.245	3.944	0.000	0.486	1.447
x3	0.9995	0.248	4.029	0.000	0.513	1.486
x4	1.3275	0.261	5.091	0.000	0.816	1.839
x5	0.6768	0.250	2.712	0.007	0.188	1.166

If the model instance has been used for another fit with different fit parameters, then the fit options might not be the correct ones anymore .

For model diagnostics, one can check residual plots.

```
import matplotlib.pyplot as plt

myOlsFit = smf.ols(formula = "y ~ x1 + x2 + x3 + x4 + x5", data = df).fit()
fig = plt.figure(figsize = (6, 6))
## residual versus x1; can do the same for other covariates
fig = sma.graphics.plot_regress_exog(myOlsFit, 'x1', fig=fig)
```



See more on residual diagnostics and specification tests.

6.2.3 Generalized Linear Regression

A linear regression model cannot be applied to presence/absence or count data. Generalized Linear Models (GLM) extend the classical linear regres-

sion to accommodate such response variables, that follow distributions other than the normal distribution. GLMs consist of three main components:

- Random Component: This specifies the distribution of the response variable Y. It is assumed to be from the exponential family of distributions, such as Binomial for binary data and Poisson for count data.
- Systematic Component: This consists of the linear predictor, a linear combination of unknown parameters and explanatory variables. It is denoted as $\eta = X\beta$, where X represents the explanatory variables, and β represents the coefficients.
- Link Function: The link function, g, provides the relationship between the linear predictor and the mean of the distribution function. For a GLM, the mean of Y is related to the linear predictor through the link function as $\mu = g^{-1}(\eta)$.

GLMs adapt to various data types through the selection of appropriate link functions and probability distributions. Here, we outline four special cases of GLM: normal regression, logistic regression, Poisson regression, and gamma regression.

- Normal Regression (Linear Regression). In normal regression, the response variable has a normal distribution. The identity link function is typically used, making this case equivalent to classical linear regression.
  - Use Case: Modeling continuous data where residuals are normally distributed.
  - Link Function: Identity, $g(\mu) = \mu$.
  - Distribution: Normal.
- Logistic Regression. Logistic regression is used for binary response variables. It employs the logit link function to model the probability that an observation falls into one of two categories.

  - Use Case: Binary outcomes (e.g., success/failure).
  - Link Function: Logit, $g(\mu) = \log \frac{\mu}{1-\mu}$.
  - Distribution: Binomial.

- Poisson Regression. Poisson regression models count data using the Poisson distribution. It's ideal for modeling the rate at which events occur.

  - Use Case: Count data, such as the number of occurrences of an event.
  - Link Function: Log, $g(\mu) = \log(\mu)$
  - Distribution: Poisson.

- Gamma Regression. Gamma regression is suited for modeling positive continuous variables, especially when data are skewed and variance increases with the mean.

  - Use Case: Positive continuous outcomes with non-constant variance.
  - Link Function: Inverse $g(\mu) = \frac{1}{\mu}$.
  - Distribution: Gamma.

Each GLM variant addresses specific types of data and research questions, enabling precise modeling and inference based on the underlying data distribution. Prediction will need the inverse link function which transforms the linear predictor to the expectation of the outcome.

To demonstrate the validation of logistic regression models, we first create a simulated dataset with binary outcomes. This setup involves generating logistic probabilities and then drawing binary outcomes based on these probabilities.

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
```

```
# Set seed for reproducibility
np.random.seed(42)

# Create a DataFrame with random features named `simdat`
simdat = pd.DataFrame(np.random.randn(1000, 5), columns=['x1', 'x2', 'x3', 'x4', 'x5'])

# Calculating the linear combination of inputs plus an intercept
eta = simdat.dot([2, 2, 2, 0, 0]) - 5

# Applying the logistic function to get probabilities using statsmodels' logit link
p = sm.families.links.Logit().inverse(eta)

# Generating binary outcomes based on these probabilities and adding them to `simdat`
simdat['yb'] = np.random.binomial(1, p, p.size)

# Display the first few rows of the dataframe
print(simdat.head())
```

```
         x1        x2        x3        x4        x5  yb
0  0.496714 -0.138264  0.647689  1.523030 -0.234153   0
1 -0.234137  1.579213  0.767435 -0.469474  0.542560   0
2 -0.463418 -0.465730  0.241962 -1.913280 -1.724918   0
3 -0.562288 -1.012831  0.314247 -0.908024 -1.412304   0
4  1.465649 -0.225776  0.067528 -1.424748 -0.544383   0
```

Fit a logistic regression for `y1b` with the formula interface.

```
import statsmodels.formula.api as smf

# Specify the model formula
formula = 'yb ~ x1 + x2 + x3 + x4 + x5'
```

```
# Fit the logistic regression model using glm and a formula
fit = smf.glm(formula=formula, data=simdat, family=sm.families.Binomial()).fi

# Print the summary of the model
print(fit.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                     yb   No. Observations:                  100
Model:                            GLM   Df Residuals:                       99
Model Family:                Binomial   Df Model:
Link Function:                  Logit   Scale:                          1.000
Method:                          IRLS   Log-Likelihood:                -136.4
Date:                Mon, 24 Feb 2025   Deviance:                       272.8
Time:                        11:56:33   Pearson chi2:                 1.09e+0
No. Iterations:                     8   Pseudo R-squ. (CS):             0.279
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975
------------------------------------------------------------------------------
Intercept     -5.4564      0.453    -12.049      0.000      -6.344      -4.56
x1             2.1544      0.244      8.822      0.000       1.676       2.63
x2             2.0781      0.225      9.234      0.000       1.637       2.51
x3             1.9260      0.237      8.125      0.000       1.461       2.39
x4            -0.1085      0.166     -0.652      0.514      -0.434       0.21
x5             0.2672      0.158      1.695      0.090      -0.042       0.57
==============================================================================
```

6.3 Interpreting Logistic Regression Results

Once a logistic regression model is fitted, interpreting its results is crucial for understanding how predictor variables influence the probability of the

outcome. Logistic regression models the log-odds of the response variable as a linear function of the predictor variables. To ease the intrepretation, consider a logistic model with a single binary predictor (e.g., treatment indicator):

$$\log\left(\frac{\mu}{1-\mu}\right) = \beta_0 + \beta_1 X$$

where $\mu = E(Y \mid X)$ represents the probability of the positive class, and β_1 is the estimated coefficient for the binary predictor X.

6.3.1 Interpreting Coefficients

If X is a binary variable (e.g., 0 for "No" and 1 for "Yes"), β_1 represents the difference in log-odds between the two groups. Exponentiating β_1 gives the odds ratio:

$$\text{Odds Ratio} = \frac{\exp(\beta_0 + \beta_1)}{\exp(\beta)} = e^{\beta_1}.$$

- If $e^{\beta_1} > 1$, the outcome is more likely when $X = 1$ than when $X = 0$.
- If $e^{\beta_1} < 1$, the outcome is less likely when $X = 1$.
- If $e^{\beta_1} = 1$, there is no effect of X on the odds of the outcome.

Equivalently, β_1 is the log odds ratio between the two groups.

When there are multiple predictors, the intrepretation needs to state that all the other predictors are unchanged.

How would you intreprete the coefficient of a continuous predictor?

6.3.2 Probabilistic Interpretation

We can transform the linear predictor into a probability estimate using the inverse logit function:

$$\Pr(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}.$$

This allows for a direct interpretation of how being in one category of X influences the predicted probability of the outcome. By construction, this value is always in $(0, 1)$.

6.3.3 Evaluating Statistical Significance

The significance of β_1 is assessed using standard errors and p-values:

- A small p-value (e.g., < 0.05) suggests that X has a statistically significant effect on the outcome.
- Confidence intervals for e^{β_1} help understand the precision of odds ratio estimates.

6.4 Validating the Results of Logistic Regression

Validating the performance of logistic regression models is crucial to assess their effectiveness and reliability. This section explores key metrics used to evaluate the performance of logistic regression models, starting with the confusion matrix, then moving on to accuracy, precision, recall, F1 score, and the area under the ROC curve (AUC). Using simulated data, we will demonstrate how to calculate and interpret these metrics using Python.

6.4.1 Confusion Matrix

The confusion matrix is a fundamental tool used for calculating several other classification metrics. It is a table used to describe the performance of a classification model on a set of data for which the true values are known. The matrix displays the actual values against the predicted values, providing insight into the number of correct and incorrect predictions.

Actual	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Four entries in the confusion matrix:

- True Positive (TP): The cases in which the model correctly predicted the positive class.
- False Positive (FP): The cases in which the model incorrectly predicted the positive class (i.e., the model predicted positive, but the actual class was negative).
- True Negative (TN): The cases in which the model correctly predicted the negative class.
- False Negative (FN): The cases in which the model incorrectly predicted the negative class (i.e., the model predicted negative, but the actual class was positive).

Four rates from the confusion matrix with actual (row) margins:

- True positive rate (TPR): TP / (TP + FN). Also known as sensitivity.
- False negative rate (FNR): FN / (TP + FN). Also known as miss rate.
- False positive rate (FPR): FP / (FP + TN). Also known as false alarm, fall-out.

- True negative rate (TNR): TN / (FP + TN). Also known as specificity.

Note that TPR and FPR do not add up to one. Neither do FNR and FPR.

- Positive predictive value (PPV): TP / (TP + FP). Also known as precision.
- False discovery rate (FDR): FP / (TP + FP).
- False omission rate (FOR): FN / (FN + TN).
- Negative predictive value (NPV): TN / (FN + TN).

Note that PPV and NP do not add up to one.

6.4.2 Accuracy

Accuracy measures the overall correctness of the model and is defined as the ratio of correct predictions (both positive and negative) to the total number of cases examined.

```
Accuracy = (TP + TN) / (TP + TN + FP + FN)
```

- Imbalanced Classes: Accuracy can be misleading if there is a significant imbalance between the classes. For instance, in a dataset where 95% of the samples are of one class, a model that naively predicts the majority class for all instances will still achieve 95% accuracy, which does not reflect true predictive performance.
- Misleading Interpretations: High overall accuracy might hide the fact that the model is performing poorly on a smaller, yet important, segment of the data.

6.4.3 Precision

Precision (or PPV) measures the accuracy of positive predictions. It quantifies the number of correct positive predictions made.

```
Precision = TP / (TP + FP)
```

- Neglect of False Negatives: Precision focuses solely on the positive class predictions. It does not take into account false negatives (instances where the actual class is positive but predicted as negative). This can be problematic in cases like disease screening where missing a positive case (disease present) could be dangerous.
- Not a Standalone Metric: High precision alone does not indicate good model performance, especially if recall is low. This situation could mean the model is too conservative in predicting positives, thus missing out on a significant number of true positive instances.

6.4.4 Recall

Recall (Sensitivity or TPR) measures the ability of a model to find all relevant cases (all actual positives).

```
Recall = TP / (TP + FN)
```

- Neglect of False Positives: Recall does not consider false positives (instances where the actual class is negative but predicted as positive). High recall can be achieved at the expense of precision, leading to a large number of false positives which can be costly or undesirable in certain contexts, such as in spam detection.
- Trade-off with Precision: Often, increasing recall decreases precision. This trade-off needs to be managed carefully, especially in contexts where both false positives and false negatives carry significant costs or risks.

6.4.5 F-beta Score

The F-beta score is a weighted harmonic mean of precision and recall, taking into account a β parameter such that recall is considered β times as important as precision:

$$(1+\beta^2)\frac{\text{precision}\cdot\text{recall}}{\beta^2\text{precision}+\text{recall}}.$$

See stackexchange post for the motivation of β^2 instead of just β.

The F-beta score reaches its best value at 1 (perfect precision and recall) and worst at 0.

If reducing false negatives is more important (as might be the case in medical diagnostics where missing a positive diagnosis could be critical), you might choose a beta value greater than 1. If reducing false positives is more important (as in spam detection, where incorrectly classifying an email as spam could be inconvenient), a beta value less than 1 might be appropriate.

The F1 Score is a specific case of the F-beta score where beta is 1, giving equal weight to precision and recall. It is the harmonic mean of Precision and Recall and is a useful measure when you seek a balance between Precision and Recall and there is an uneven class distribution (large number of actual negatives).

6.4.6 Receiver Operating Characteristic (ROC) Curve

The Receiver Operating Characteristic (ROC) curve is a plot that illustrates the diagnostic ability of a binary classifier as its discrimination threshold is varied. It shows the trade-off between the TPR and FPR. The ROC plots TPR against FPR as the decision threshold is varied. It can be particularly useful in evaluating the performance of classifiers when the class distribution is imbalanced,

- Increasing from $(0, 0)$ to $(1, 1)$.
- Best classification passes $(0, 1)$.
- Classification by random guess gives the 45-degree line.
- Area between the ROC and the 45-degree line is the Gini coefficient, a measure of inequality.
- Area under the curve (AUC) of ROC thus provides an important metric of classification results.

The Area Under the ROC Curve (AUC) is a scalar value that summarizes the performance of a classifier. It measures the total area underneath the ROC curve, providing a single metric to compare models. The value of AUC ranges from 0 to 1:

- AUC $= 1$: A perfect classifier, which perfectly separates positive and negative classes.
- AUC $= 0.5$: A classifier that performs no better than random chance.
- AUC < 0.5: A classifier performing worse than random.

The AUC value provides insight into the model's ability to discriminate between positive and negative classes across all possible threshold values.

6.4.7 Demonstration

Let's apply these metrics to the `simdat` dataset to understand their practical implications. We will fit a logistic regression model, make predictions, and then compute accuracy, precision, and recall.

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, confusion_matrix,
    f1_score, roc_curve, auc
)
```

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification

# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra

# Fit the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict labels on the test set
y_pred = model.predict(X_test)

# Get predicted probabilities for ROC curve and AUC
y_scores = model.predict_proba(X_test)[:, 1]  # Probability for the positive

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Calculate accuracy, precision, and recall
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print confusion matrix and metrics
print("Confusion Matrix:\n", cm)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

```
Confusion Matrix:
 [[104  11]
 [ 26 109]]
Accuracy: 0.85
Precision: 0.91
Recall: 0.81
```

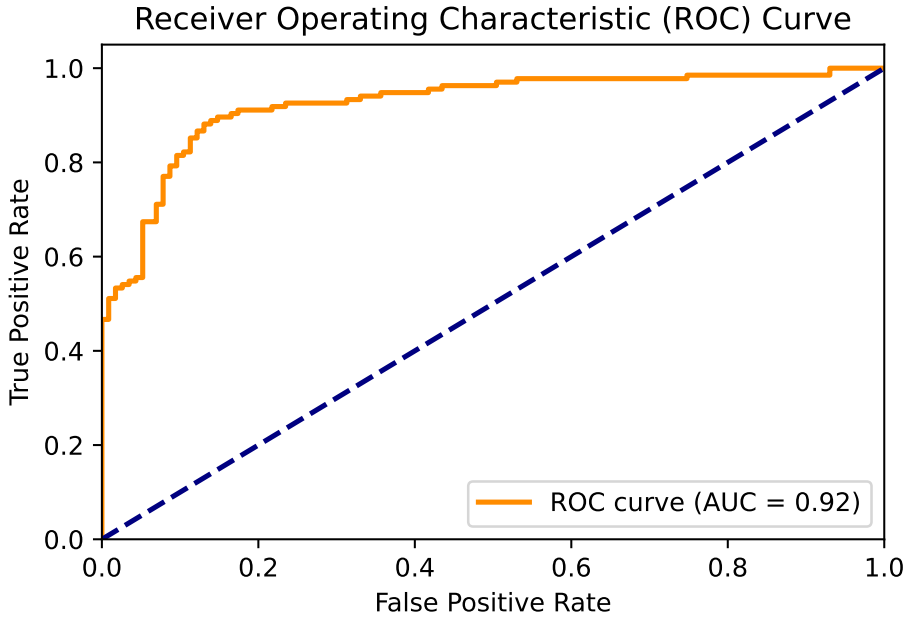
By varying threshold, one can plot the whole ROC curve.

```
# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Print AUC
print(f"AUC: {roc_auc:.2f}")

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line (random classi
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
AUC: 0.92
```



We could pick the best threshold that optmizes F1-score/

```
# Compute F1 score for each threshold
f1_scores = []
for thresh in thresholds:
    y_pred_thresh = (y_scores >= thresh).astype(int)  # Apply threshold to ge
    f1 = f1_score(y_test, y_pred_thresh)
    f1_scores.append(f1)

# Find the best threshold (the one that maximizes F1 score)
best_thresh = thresholds[np.argmax(f1_scores)]
best_f1 = max(f1_scores)

# Print the best threshold and corresponding F1 score
print(f"Best threshold: {best_thresh:.4f}")
```

```
print(f"Best F1 score: {best_f1:.2f}")
```

```
Best threshold: 0.3960
Best F1 score: 0.89
```

6.5 LASSO Logistic Models

The Least Absolute Shrinkage and Selection Operator (LASSO) (Tibshirani, 1996), is a regression method that performs both variable selection and regularization. LASSO imposes an L1 penalty on the regression coefficients, which has the effect of shrinking some coefficients exactly to zero. This results in simpler, more interpretable models, especially in situations where the number of predictors exceeds the number of observations.

6.5.1 Theoretical Formulation of the Problem

The objective function for LASSO logistic regression can be expressed as,

$$\min_{\beta} \left\{ -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)] + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

where:

- $\hat{p}_i = \frac{1}{1+e^{-X_i\beta}}$ is the predicted probability for the i-th sample.
- y_i represents the actual class label (binary: 0 or 1).
- X_i is the feature vector for the i-th observation.
- β is the vector of model coefficients (including the intercept).

- λ is the regularization parameter that controls the trade-off between model fit and sparsity (higher λ) encourages sparsity by shrinking more coefficients to zero).

The lasso penalty encourages the sum of the absolute values of the coefficients to be small, effectively shrinking some coefficients to zero. This results in sparser solutions, simplifying the model and reducing variance without substantial increase in bias.

Practical benefits of LASSO:

- Dimensionality Reduction: LASSO is particularly useful when the number of features p is large, potentially even larger than the number of observations n, as it automatically reduces the number of features.
- Preventing Overfitting: The L1 penalty helps prevent overfitting by constraining the model, especially when p is large or there is multicollinearity among features.
- Interpretability: By selecting only the most important features, LASSO makes the resulting model more interpretable, which is valuable in fields like bioinformatics, economics, and social sciences.

6.5.2 Solution Path

To illustrate the effect of the lasso penalty in logistic regression, we can plot the solution path of the coefficients as a function of the regularization parameter λ. This demonstration will use a simulated dataset to show how increasing λ leads to more coefficients being set to zero.

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

```
# Step 1: Generate a classification dataset
X, y = make_classification(n_samples=100, n_features=20, n_informative=2,
                           random_state=42)

# Step 2: Get a lambda grid given length of lambda and min_ratio of lambda_max
def get_lambda_l1(xs: np.ndarray, y: np.ndarray, nlambda: int, min_ratio: float):
    ybar = np.mean(y)
    xbar = np.mean(xs, axis=0)
    xs_centered = xs - xbar
    xty = np.dot(xs_centered.T, (y - ybar))
    lmax = np.max(np.abs(xty))
    lambdas = np.logspace(np.log10(lmax), np.log10(min_ratio * lmax),
                          num=nlambda)
    return lambdas

# Step 3: Calculate lambda values
nlambda = 100
min_ratio = 0.01
lambda_values = get_lambda_l1(X, y, nlambda, min_ratio)

# Step 4: Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

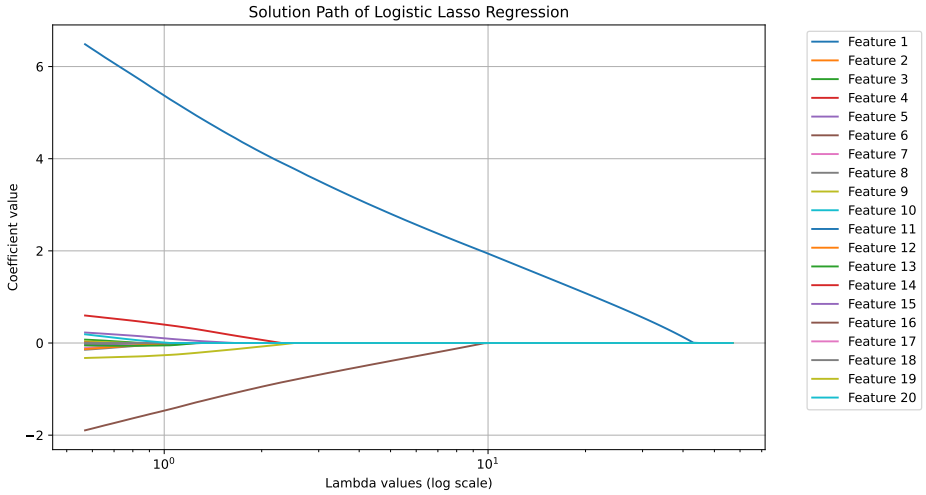
# Step 5: Initialize arrays to store the coefficients for each lambda value
coefficients = []

# Step 6: Fit logistic regression with L1 regularization (Lasso) for each lambda value
for lam in lambda_values:
    model = LogisticRegression(penalty='l1', solver='liblinear', C=1/lam, max_iter=1000)
    model.fit(X_scaled, y)
    coefficients.append(model.coef_.flatten())
```

```
# Convert coefficients list to a NumPy array for plotting
coefficients = np.array(coefficients)

# Step 7: Plot the solution path for each feature
plt.figure(figsize=(10, 6))
for i in range(coefficients.shape[1]):
    plt.plot(lambda_values, coefficients[:, i], label=f'Feature {i + 1}')

plt.xscale('log')
plt.xlabel('Lambda values (log scale)')
plt.ylabel('Coefficient value')
plt.title('Solution Path of Logistic Lasso Regression')
plt.grid(True)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



6.5.3 Selection the Tuning Parameter

In logistic regression with LASSO regularization, selecting the optimal value of the regularization parameter C (the inverse of λ) is crucial to balancing the model's bias and variance. A small C value (large λ) increases the regularization effect, shrinking more coefficients to zero and simplifying the model. Conversely, a large C (small λ) allows the model to fit the data more closely.

The best way to select the optimal C is through cross-validation. In cross-validation, the dataset is split into several folds, and the model is trained on some folds while evaluated on the remaining fold. This process is repeated for each fold, and the results are averaged to ensure the model generalizes well to unseen data. The C value that results in the best performance is selected.

The performance metric used in cross-validation can vary based on the task. Common metrics include:

- Log-loss: Measures how well the predicted probabilities match the actual outcomes.
- Accuracy: Measures the proportion of correctly classified instances.
- F1-Score: Balances precision and recall, especially useful for imbalanced classes.
- AUC-ROC: Evaluates how well the model discriminates between the positive and negative classes.

In Python, the `LogisticRegressionCV` class from `scikit-learn` automates cross-validation for logistic regression. It evaluates the model's performance for a range of C values and selects the best one.

```
import numpy as np
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
```

```
from sklearn.metrics import accuracy_score

# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra

# Initialize LogisticRegressionCV with L1 penalty for Lasso and cross-validat
log_reg_cv = LogisticRegressionCV(
    Cs=np.logspace(-4, 4, 20),  # Range of C values (inverse of lambda)
    cv=5,                       # 5-fold cross-validation
    penalty='l1',               # Lasso regularization (L1 penalty)
    solver='liblinear',         # Solver for L1 regularization
    scoring='accuracy',         # Optimize for accuracy
    max_iter=10000              # Ensure convergence
)

# Train the model with cross-validation
log_reg_cv.fit(X_train, y_train)

# Best C value (inverse of lambda)
print(f"Best C value: {log_reg_cv.C_[0]}")

# Evaluate the model on the test set
y_pred = log_reg_cv.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {test_accuracy:.2f}")

# Display the coefficients of the best model
print("Model Coefficients:\n", log_reg_cv.coef_)
```

```
Best C value: 0.08858667904100823
```

```
Test Accuracy: 0.86
Model Coefficients:
 [[ 0.          0.          0.05552448  0.          0.          1.90889734
   0.          0.          0.          0.          0.0096863   0.23541942
   0.          0.         -0.0268928   0.          0.          0.
   0.          0.        ]]
```

6.5.4 Preparing for Logistic Regression Fitting

The `LogisticRegression()` function in `scikit.learn` takes the design matrix of the regression as input, which needs to be prepared with care from the covariates or features that we have.

6.5.4.1 Continuous Variables

For continuous variables, it is often desirable to standardized them so that they have mean zero and standard deviation one. There are multiple advantages of doing so. It improves numerical stability in algorithms like logistic regression that rely on gradient descent, ensuring faster convergence and preventing features with large scales from dominating the optimization process. Standardization also enhances the interpretability of model coefficients by allowing for direct comparison of the effects of different features, as coefficients then represent the change in outcome for a one standard deviation increase in each variable. Additionally, it ensures that regularization techniques like Lasso and Ridge treat all features equally, allowing the model to select the most relevant ones without being biased by feature magnitude.

Moreover, standardization is essential for distance-based models such as k-Nearest Neighbors (k-NN) and Support Vector Machines (SVMs), where differences in feature scale can distort the calculations. It also prevents

models from being sensitive to arbitrary changes in the units of measurement, improving robustness and consistency. Finally, standardization facilitates better visualizations and diagnostics by putting all variables on a comparable scale, making patterns and residuals easier to interpret. Overall, it is a simple yet powerful preprocessing step that leads to better model performance and interpretability.

We have already seen this with `StandardScaler`.

6.5.4.2 Categorical Variables

Categorical variables can be classified into two types: nominal and ordinal. Nominal variables represent categories with no inherent order or ranking between them. Examples include variables like "gender" (male, female) or "color" (red, blue, green), where the categories are simply labels and one category does not carry more significance than another. Ordinal variables, on the other hand, represent categories with a meaningful order or ranking. For example, education levels such as "high school," "bachelor," "master," and "PhD" have a clear hierarchy, where each level is ranked higher than the previous one. However, the differences between the ranks are not necessarily uniform or quantifiable, making ordinal variables distinct from numerical variables. Understanding the distinction between nominal and ordinal variables is important when deciding how to encode and interpret them in statistical models.

Categorical variables needs to be coded into numrical values before further processing. In Python, nominal and ordinal variables are typically encoded differently to account for their unique properties. Nominal variables, which have no inherent order, are often encoded using One-Hot Encoding, where each category is transformed into a binary column (0 or 1). For example, the OneHotEncoder from scikit-learn can be used to convert a "color" variable with categories like "red," "blue," and "green" into separate columns color_red, color_blue, and color_green, with only one column being 1 for each observation. On the other hand, ordinal variables,

which have a meaningful order, are best encoded using Ordinal Encoding. This method assigns an integer to each category based on their rank. For example, an "education" variable with categories "high school," "bachelor," "master," and "PhD" can be encoded as 0, 1, 2, and 3, respectively. The OrdinalEncoder from scikit-learn can be used to implement this encoding, which ensures that the model respects the order of the categories during analysis.

6.5.4.3 An Example

Here is a demo with `pipeline` using a simulated dataset.

First we generate data with sample size 1000 from a logistic model with both categorical and numerical covariates.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
import numpy as np
from scipy.special import expit  # Sigmoid function

# Generate a dataset with the specified size
dataset_size = 1000
np.random.seed(20241014)

# Simulate categorical and numerical features
gender = np.random.choice(
    ['male', 'female'], size=dataset_size)  # Nominal variable
education = np.random.choice(
    ['high_school', 'bachelor', 'master', 'phd'], size=dataset_size)  # Ordinal variable
```

```
age = np.random.randint(18, 65, size=dataset_size)
income = np.random.randint(30000, 120000, size=dataset_size)

# Create a logistic relationship between the features and the outcome
gender_num = np.where(gender == 'male', 0, 1)

# Define the linear predictor with regression coefficients
linear_combination = (
    0.3 * gender_num - 0.02 * age + 0.00002 * income
)

# Apply sigmoid function to get probabilities
probabilities = expit(linear_combination)

# Generate binary outcome based on the probabilities
outcome = np.random.binomial(1, probabilities)

# Create a DataFrame
data = pd.DataFrame({
    'gender': gender,
    'education': education,
    'age': age,
    'income': income,
    'outcome': outcome
})
```

Next we split the data into features and target and define transformers for each types of feature columns.

```
# Split the dataset into features (X) and target (y)
X = data[['gender', 'education', 'age', 'income']]
y = data['outcome']
```

```
# Define categorical and numerical columns
categorical_cols = ['gender', 'education']
numerical_cols = ['age', 'income']

# Define transformations for categorical variable
categorical_transformer = OneHotEncoder(
    categories=[['male', 'female'], ['high_school', 'bachelor', 'master', 'phd']],
    drop='first')

# Define transformations for continuous variables
numerical_transformer = StandardScaler()

# Use ColumnTransformer to transform the columns
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', categorical_transformer, categorical_cols),
        ('num', numerical_transformer, numerical_cols)
    ]
)
```

Define a pipeline, which preprocess the data and then fits a logistic model.

```
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(penalty='l1', solver='liblinear',
    max_iter=1000))
])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=2024)
```

```
# Fit the pipeline to the training data
pipeline.fit(X_train, y_train)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('cat',
                                                  OneHotEncoder(categories=[
                                                                drop='first'
                                                  ['gender', 'education']),
                                                 ('num', StandardScaler(),
                                                  ['age', 'income'])])),
                ('classifier',
                 LogisticRegression(max_iter=1000, penalty='l1',
                                    solver='liblinear'))])
```

Check the coefficients of the fitted logistic regression model.

```
model = pipeline.named_steps['classifier']
intercept = model.intercept_
coefficients = model.coef_

# Check the preprocessor's encoding
encoded_columns = pipeline.named_steps['preprocessor']\
.transformers_[0][1].get_feature_names_out(categorical_cols)

# Show intercept, coefficients, and encoded feature names
intercept, coefficients, list(encoded_columns)
```

```
(array([0.66748582]),
 array([[ 0.30568894,  0.10069842,  0.12087311,  0.22576774, -0.24749201,
          0.55828424]]),
 ['gender_female', 'education_bachelor', 'education_master', 'education_phd'])
```

Note that the encoded columns has one for gender and three for education, with `male` and `high_school` as reference levels, respectively. The reference level was determined when calling `oneHotEncoder()` with `drop = 'first'`. If `categories` were not specified, the first level in alphabetical order would be dropped. With the default `drop = 'none'`, the estimated coefficients will have two columns that are not estimable and were set to zero. Obviously, if no level were dropped in forming the model matrix, the columns of the one hot encoding for each categorical variable would be perfectly linearly dependent because they would sum to one.

The regression coefficients returned by the logistic regression model in this case should be interpreted on the standardized scale of the numerical covariates (e.g., `age` and `income`). This is because we applied standardization to the numerical features using StandardScaler in the pipeline before fitting the model. For example, the coefficient for age would reflect the change in the log-odds of the outcome for a 1 standard deviation increase in age, rather than a 1-unit increase in years. The coefficients for the one-hot encoded categorical variables (gender and education) are on the original scale because one-hot encoding does not change the scale of the variables. For instance, the coefficient for `gender_female` tells us how much the log-odds of the outcome changes when the observation is male versus the reference category (`male`).

7 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?

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 $\bigcirc$ represent one of the four operators. For example,

$$(\square \bigcirc \square) \bigcirc (\square \bigcirc \square)$$

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.

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 intented. 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 their 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 contigency 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 a 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 Census zip code database which contains zip-code level demographic or socioeconomic variables.
    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 obtained from 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.

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*.

Cone, M. (2025). *Markdown cheat sheet | markdown guide*. https://www.markdownguide.org/cheat-sheet/

Congress, U. S. (1990). *Americans with disabilities act of 1990 (ADA)*.

Dervieux, C. (2025). *Markdown-basics*. https://quarto.org/docs/authoring/markdown-basics.html

Health, U. S. D. of, & Services, H. (1996). *Health insurance portability and accountability act of 1996 (HIPAA)*.

MacFarlane, J. (2006). *Pandoc user's guide*. https://pandoc.org/MANUAL.html#pandocs-markdown

MacFarlane, J. (2019). *GitHub flavored markdown spec*. https://github.github.com/gfm/

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*.

Tibshirani, R. (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288.

VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data.* O'Reilly Media, Inc.