# Introduction to Data Science

**STAT 3255/5255 @ UConn, Fall 2024**

Jun Yan and Students in Fall 2024:
Zachary Blanchard, Sara Clokey    Julia Mazzola

2024-10-23

# 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 Fall 2024 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 Spring 2024 repository at https://github.com/statds/ids-f24.

Students contributed to the lecture notes by submitting pull requests to our dedicated 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 in exploring the lecture notes from the previous years, the Spring 2024, Spring 2023 and Spring 2022 are both publicly accessible. These archives offer valuable insights into the evolution of the course content and the different perspectives brought by successive student cohorts.

## Midterm Project

NYC noise complaints made to NYPD in the week of July 4, 2024. See details in the exercises.

## 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
- Kaggle
- DrivenData
- Top 10 Data Science Competitions in 2024

If you work on sports analytics, you are welcome to submit a poster to UConn Sports Analytics Symposium (UCSAS) 2024.

## 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
  - Co-develop a Quarto book in collaboration with the students
  - Train students to participate real data science competitions

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

- Akach, Suha
  - Challenge and push myself to be better at python and all its libraries.
  - Be confident in my abilities of programming and making statistical inferences that are correct.

  - Be able to create my own personal project in class on time.

- Astle, Jaden
  - I've used Git before, but I'd like to become more comfortable using it and get more used to different issues that arise.
  - I'd like to learn more effective ways to "tell the story" of data analysis and show empowering visualizations.
  - I'd like to explore more methods that professional data scientists use in their model trainings to share with UConn's Data Science Club.

- Babiec, Owen
  - Become more comfortable with Git and Github and their applications
  - Better understand the Data Science pipeline and workflow
  - Learn how to show my skills I have learned in this class during interviews

- Baptista, Stef
  - Develop a project/presentation suitible enough for industry
  - Improve on my data science skills regarding pandas and numpy
  - Understanding the scope of packages in python as a language

- Bienvenue, Jack
  - Learn professional visualization techniques, particularly for geospatial data
  - Foster a high level of working knowledge of Git
  - Create a small portfolio of examples and projects for later reference

- Blanchard, Zachary
  - Gain experience working and collaborating on projects in Git
  - Improve computer programming skills and familiarity with Python

  - Teach other students about creating presentations using Quarto

- Borowski, Emily
  - Gain a greater understanding of Quarto and GitHub
  - Become more comfortable with my coding abilities
  - Acquire a deeper understanding of data science

- Clokey, Sara
  - Become more familiar with GitHub and Quarto
  - Execute a data science project from start to finish

- Desroches, Melanie
  - Explore the field of data science as a possible future career
  - Develope data science and machine learning skills
  - Become better at programming with Python and using Git/GitHub

- Febles, Xavier
  - Gain a further understanding of GitHub
  - Develop data visualization skills
  - Learn applications of skills learned in previous courses

- Jha, Aansh
  - Be a better student of the data science field
  - Hone git to work in colabarative workspaces
  - Learn better methods in data visualization

- Johnson, Dorothea
  - Enter data science contests
  - Familiarize myself with using Python for data Science
  - Develop a proficiency in Github

- Kashalapov, Olivia

  - Better understand neural networks
  - Machine learning utilizing Python
  - Creating and analyzing predictive models for informed decision making

- Manna, Rahul

  - Use knowledge gained and skills developed in class to study real-world problems such as climate change.
  - Obtain a basic understanding of machine learning

- Mazzola, Julia

  - Become proficient in Git and Github.
  - Have a better understanding of data science best practices and techniques.
  - Deepen my knowledge of Python programming concepts and libraries.

- Paricharak, Aditya

  - Master Commandline Interface
  - Apply my statistical knowladge and skills to course work
  - Understand how to work with datasets

- Parvez, Mohammad Shahriyar

  - Familiarizing myself with GitHub to effectively track and manage the entire data analysis process.
  - Adopting Quarto for improved documentation of my data workflows.
  - Exploring advanced techniques for data analysis and visualization.
  - Developing my personal Git repository and publishing data projects as a professional website.

- Tan, Qianruo

  - Learn how to use GitHub, and create my own page

  - Get a good grade on this class
  - Learn more about how to processing data

- Xu, Deyu
  - Be proficient in using Python to process data.
  - Learn the basics of machine learning.
  - Have a basic understanding of data scienc.
  - Lay a solid foundation for GNN and Bayes neural network.

## 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
target = "Blanchard"  # pre-arranged 1st presenter
presenters = [name for name in presenters if target not in name]

import random
## seed jointly set by the class
random.seed(5347 + 2896 + 9050 + 1687 + 63)
random.sample(presenters, len(presenters))
## random.shuffle(presenters) # This would shuffle the list in place
```

```
['Xu,Deyu',
 'Clokey,Sara Karen',
 'Johnson,Dorothea Trixie',
 'Febles,Xavier Milan',
 'Cai,Yizhan',
```

```
'Bienvenue,Jack Noel',
'Mazzola,Julia Cecelia',
'Akach,Suha',
'Manna,Rahul',
'Astle,Jaden Bryce',
'Kashalapov,Olivia',
'Borowski,Emily Helen',
'Tan,Qianruo',
'Desroches,Melanie',
'Paricharak,Aditya Sushant',
'Jha,Aansh',
'Babiec,Owen Thomas',
'Baptista,Stef Clare',
'Parvez,Mohammad Shahriyar']
```

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.

## Course Logistics

### Presentation Task Board

Here are some example tasks:

- Making presentations with Quarto
- Data science ethics
- Data science communication skills

- Import/Export data
- Arrow as a cross-platform data format
- Database operation with Structured query language (SQL)
- Grammer of graphics
- Handling spatial data
- Visualize spatial data in a Google map
- Animation
- Classification and regression trees
- Support vector machine
- Random forest
- Naive Bayes
- Bagging vs boosting
- Neural networks
- Deep learning
- TensorFlow
- Autoencoders
- Reinforcement learning
- Calling C/C++ from Python
- Calling R from Python and vice versa
- Developing a Python package

Please use the following table to sign up.

| Date | Presenter | Topic |
|---|---|---|
| 09/11 | Zachary Blanchard | Making presentation with Quarto |
| 09/16 | Deyu Xu | Import/Export data |
| 09/18 | Sara Clokey | Communications in data science |

| Date | Presenter | Topic |
|---|---|---|
| 09/23 | Dorathea Johnson | Database with SQL |
| 09/25 | Xavier Febles | |
| 09/30 | Jack Bienvenue | Visualizing spatial data in a Google Map |
| 10/02 | Julia Mazzola | Data Visualization with Plotnine |
| 10/07 | Suha Akach | |
| 10/09 | Rahul Manna | |
| 10/23 | Jaden Astle | Classification and regression trees |
| 10/23 | Olivia Kashalapov | SMOTE |
| 10/28 | Data science alumni panel | |
| 10/30 | Emily Borowski | Random Forest |
| 10/30 | Aditya Paricharak | Neural Networks |
| 11/04 | Melanie Desroches | |
| 11/06 | Qianruo Tan | Reinforcement Learning |
| 11/11 | Aansh Jha | |
| 11/11 | Owen Babiec | Calling R from Python and vice versa |
| 11/13 | Stef Baptista | |

| Date | Presenter | Topic |
|---|---|---|
| 11/13 | Mohammad Parvez | Developing a Python package |

### 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 |
|---|---|
| 11/18 | Sara Clokey; Dorothea Johnson; Xavier Febles; Jack Bienvenue |
| 11/20 | Julia Mazzola; Suha Akach; Rahul Manna; Jaden Astle |
| 12/02 | Olivia Kashalapov; Emily Borowski Qianruo Tan; Melanie Desroches |
| 12/04 | Aditya Paricharak; Aansh Jha; Owen Babiec; Stef Baptista |

### 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](https://usingsources.fas.harvard.edu/how-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 Requirements

- Use the repo from Git Classroom to submit your work. See Section 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).
- Use `quarto` source only. See Chapter 3.
- For the convenience of grading, add your html output to a release in your repo.
- For standalone pdf output, you will need to have LaTeX installed.

# 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)

## Introduction

Put an overview here. Use Markdown syntax.

## Sub Topic 1

Put materials on topic 1 here

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

```python
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.

# 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 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.

### 1.3.2 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.

# 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

- `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.

## 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.

# 3 Reproducibile 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-f24. 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-f24-venv
```

Here `.ids-f24-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-f24-venv/bin/activate
```

After activating the virtual environment, you will see `(.ids-f24-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 (floder), and clone the repo. For example, if you use `ssh` for authentication:

```
git clone git@github.com:statds/ids-f24.git
```

### 3.2.3 Render the Classnotes

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

```
cd ids-f24
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.

# 4 Python Refreshment

## 4.1 Know Your Computer

### 4.1.1 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.

### 4.1.2 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.

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

## 4.2 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.

## 4.3 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.

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

```
2.71828182845904509080
2.7182818284590450907955982984276488423347473144531250000000000000000000000000
```

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

```python
import decimal  # for what?

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

```
'7182818284590452353602874713526624977572470936999595749669676277240766303535
```

## 4.4 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([-2.19566959, -0.85974598, -4.74672631,  1.08303193,  1.56609321,
        4.25435655,  2.85994624,  0.43861787, -1.02268281, -1.87926448])
```

The pdf and cdf can be evaluated:

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

```
array([0.05753587, 0.07724292, 0.02404856, 0.09714905, 0.09915049,
       0.0850897 , 0.09745715, 0.09241946, 0.07496365, 0.06231803])
```

## 4.5 Writing a Function

Consider the Fibonacci Sequence $1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$. 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)
```

```
8.77  s ± 419 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)
```

```
2.01  s ± 42.9 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)
```

```
70.6  s ± 5.78  s per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

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

### 4.5.1 Monte Hall

Here is a function that performs the Monte Hall experiments.

```
import numpy as np

def montehall(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:

```
montehall(3, 1000)
montehall(4, 1000)
```

```
{'noswitch': np.int64(236), 'switch': np.int64(387)}
```

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

## 4.6 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
```

```
4590088832
4590088832
```

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
```

```
4483821816
4550248176
```

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.

## 4.7 Number Representation

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

### 4.7.1 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
```

```
9223372036854775808000
```

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

### 4.7.2 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
```

## 4.8 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.

# 5 Communication in Data Science

This chapter was written by Sara Clokey.

## 5.1 Introduction

Hi! My name is Sara, and I am a junior double majoring in Applied Data Analysis and Communication. The topic of my presentation today, Communication in Data Science, combines my academic and professional interests while underscoring the importance of 'soft skills' like public speaking, for example, within STEM fields like data science.

## 5.2 Importance & Application of Communication

Data science as a career path has exploded within the last decade. Some fields that offer data science positions include:

- Finance
- Healthcare
- Media production
- Sports
- Banking
- Insurance
- E-Commerce
- Energy

- Manufacturing
- Transportation
- Construction

Because data science is applicable in so many industries, it is essential that data scientists have the skills and experience to communicate their work with others who do not have the same technical education. As analyzed by Radovilsky et al. (2018), job listings within the field of data science often include qualifications like "strong interpersonal skills" and "demonstrated presentation and communication ability," highlighting the pervasive need for this skill set.

Within these industries, collaboration and teamwork are often at the forefront. Inexperience with data should not prevent your colleagues from being able to contribute to shared projects, and strong communication skills can mitigate this challenge!

## 5.3 General Communication Skills

### 5.3.1 Verbal Communication Skills

Verbal Communication: "The use of sounds and language to relay a message" (Yer (2018))

Verbal Communication Tips:

- Make a good first impression
- Use appropriate language (jargon, metaphors)
- Prioritize brevity
- Practice beforehand
- Allow room for questions

### 5.3.2 Non-Verbal Communication Skills

Non-Verbal Communication: "Information, emotion, a movement that is expressed without words and without the help of language" (Grillo & Enesi (2022))

Non-Verbal Communication Tips:

- Utilize vocal variety (pitch, rate, volume)
- Avoid distracting hand and body movements
- Make eye contact
- Pay attention to proxemics

### 5.3.3 Visual Communication Skills

Visual Communication: "Any communication that employs one's sense of sight to deliver a message without the usage of any verbal cues" (Fayaz (2022))

Visual Communication Tips:

- Prioritize clarity
- Use proper labeling and scaling
- Create visual contrast (colors, shapes, fonts)
- Choose the most appropriate visual representation

### 5.3.4 Written Communication Skills

Written Communication: "Any form of communication which is written and documented from the sender to the receiver" (Prabavathi & Nagasubramani (2018))

Written Communication Tips:

- Clearly state your goal with a thesis statement

- Maintain professionalism (contractions, slang)
- Proofread and utilize peer editing
- Follow a specific structure
- Balance concision with analysis

## 5.4 Communication in Data Science

Often, data scientists must communicate "technical conclusions to non-technical members" (Pragmatic (2024)); this may be colleagues in other departments, like marketing, or supervisors at the managerial level. Here are some tips for effectively communicating project results specifically in the field of data science.

### 5.4.1 Identify your Audience

Who are you sharing information with? Is it a room of data scientists like this one? Is it full of students who want to learn about data science? Is it a group of executives looking to make a funding decision?

- Consider the context and prior knowledge (technical jargon)
- Consider the motivation for listening

### 5.4.2 Utilize Data Visualization

One of the most effective methods of communicating results in data science, especially to those without technical coding knowledge, is data visualization techniques (Vandemeulebroecke et al. (2019)). Python uses the package 'matplotlib' to produce these visualizations, including:

- line plots
- bar plots

- box plots
- histograms
- heat maps
- pie charts

These visualizations allow complex statistical projects to be simplified into a single graphic, focusing on project results and implications rather than methodology. Ensure that data visualization techniques are free of technical jargon and clearly label all visual aspects.

### 5.4.3 Focus on Data Communication Skills

The following skill sets highlight technical data communication that will be more common in projects with other data scientists to communicate about your data.

- Coding communication: Python, R, Julia, JavaScript, SQL, etc.
- Analysis communication: creating a storyline, descriptive versus diagnostic versus predictive analytics, problem identification
- Data management: collection, cleaning/transformation, storage
- Data visualization

### 5.4.4 Give Space for Questions and Feedback

Within professional spaces, data scientists should provide time for their clients, supervisors, and colleagues to ask questions about their work and subsequent findings.

- Pause for questions throughout the presentation
- Offer contact information for continued collaboration
- Provide a structure for anonymous feedback
- Schedule follow-ups if necessary

Teamwork is often at the heart of data science projects within industries, and open communication makes this teamwork run much more smoothly.

## 5.5 Further Learning

While pursuing degrees in the data science field, consider taking Communication courses at UConn that can bolster your understanding and skill set. Some applicable communication courses include:

- COMM 2100: Professional Communication
- COMM 3110: Organizational Communication
- COMM 3430: Science Communication
- COMM 5655: Human-Computer Interaction
- COMM 5900: Professional Communication

Effective communication also requires practice. Here are some ways to practice these skills while earning your degree:

- Fully participate in group projects
- Seek presentation opportunities (class, conferences, etc.)
- Explain data science coursework to peers outside of your program
- Explore internship opportunities that involve collaboration with other departments

# 6 Data Manipulation

## 6.1 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.

## 6.2 Import/Export Data

This section was written by Deyu Xu, a MS student in Statistics at the time.

### 6.2.1 Summary

I would like to divide all of the content into five sections. The fisrt one is exporting data to a `.csv` file. The second one is importing common formats of data. The third one is importing data from other softwares. The forth one is viewing basic information of the data we have imported. The last one is finding null values.

### 6.2.2 Package `Pandas`

#### 6.2.2.1 Import data based on Package `Pandas`

We need to use the Package, `Pandas` provided by Python to import data. The first step is to install the Package, `Pandas`. Python allows us to install different versions of `Pandas`. We are able to use the following code to install the common cersion.

```
## install the common version of Pandas
pip install pandas
```

The code for installing the latest version is listed.

```
## install the latest version of Pandas
pip install --upgrade pandas
```

Different versions mean there are differnces in the code to achieve the same goal. We will see the specific example in the part of importing `.xlsx` files.

### 6.2.3 Export the data to a `.csv` file:

#### 6.2.3.1 Import the cleaned crashes data at first

Fisrtly, we need to import the file named "nyccrashes_cleaned.feather" data source. `.feather` file is a binary file format for storing and sharing data. It is especially suitable for large-scale data analysis and data science workflows. It uses Apache Arrow's columnar storage format, which can store data in binary form. The advantage of using this format of file is elvaluating the standard of reading and writing. We need to choose the function `read_feather` from `Pandas` to import the crashes data.

```
## Choose the Package Pandas
import pandas as pd
## Import the cleaned crashes data
## Choose the fcuntion, read_feather from Pandas
## Add the relative address of the data file to let your cooperator deal with the code smoot
df_feather = pd.read_feather("data/nyccrashes_cleaned.feather")
## Show the top 5 rows of data
## Determine whether we import the data successfully
print(df_feather.head(5)) # Use the fucntion "head()"
```

```
        crash_datetime    borough  zip_code  latitude  longitude  \
0  2024-06-30 17:30:00       None       NaN       NaN        NaN
1  2024-06-30 00:32:00       None       NaN       NaN        NaN
2  2024-06-30 07:05:00   BROOKLYN   11235.0  40.58106  -73.96744
3  2024-06-30 20:47:00  MANHATTAN   10021.0  40.76363  -73.95330
4  2024-06-30 10:14:00   BROOKLYN   11222.0  40.73046  -73.95149
```

```
                location     on_street_name      cross_street_name  \
0             (0.0, 0.0)               None                   None
1                   None  BELT PARKWAY RAMP                   None
2  (40.58106, -73.96744)               None                   None
3   (40.76363, -73.9533)          FDR DRIVE                   None
4  (40.73046, -73.95149)  GREENPOINT AVENUE  MC GUINNESS BOULEVARD

           off_street_name  number_of_persons_injured  ...  \
0              GOLD STREET                          0  ...
1                     None                          0  ...
2  2797      OCEAN PARKWAY                          0  ...
3                     None                          0  ...
4                     None                          0  ...

   contributing_factor_vehicle_2  contributing_factor_vehicle_3  \
0                    Unspecified                           None
1                    Unspecified                           None
2                           None                           None
3                           None                           None
4                    Unspecified                           None

   contributing_factor_vehicle_4  contributing_factor_vehicle_5  collision_id
0                           None                           None       4736746
1                           None                           None       4736768
2                           None                           None       4737060
3                           None                           None       4737510
4                           None                           None       4736759

                vehicle_type_code_1                  vehicle_type_code_2
0                             Sedan                                Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle
2  Station Wagon/Sport Utility Vehicle                                 None
3                             Sedan                                 None
```

```
4                                   Bus                              Box Truck

  vehicle_type_code_3 vehicle_type_code_4 vehicle_type_code_5
0                None                None                None
1                None                None                None
2                None                None                None
3                None                None                None
4                None                None                None

[5 rows x 28 columns]
```

- We have imported the cleaned crashes data successfully.
- We utilize the fucntion `head(5)` to show the top 5 rows of the data.

#### 6.2.3.2 Export the crashes data to a `.csv` file

It is easy to export the data. The fucntion that helps us to complete this goal is `to_csv` from `Pandas`.

```
## Choose the Package Pandas
## Choose the function "to_csv" from Pandas
## Use the argument, "df_feather" storing the data
## Export the data to the default working directory
df_feather.to_csv("nyccrashes_cleaned.csv") # Add the name of the CSV file
```

- We can check whether the corresponding `.csv` file is generated in the default working directory.

We have exported the data to a `.csv` file in the default working directory.

We will use this `.csv` file later.

### 6.2.4 Import files in common formats: `.csv/.xlsx/.txt`

#### 6.2.4.1 `.csv` files

We are familiar with `.csv` files as utilizing them to print some charts by R in the past courses. Now let us import this generated `.csv` file. We are supposed to choose the function `read_csv` from `Pandas`. The following code shows how to import it.

```
## Choose the Package Pandas
import pandas as pd
## Choose the function "read_csv"
## Add the relative address of the generated CSV file
df_csv = pd.read_csv("nyccrashes_cleaned.csv")
## Check the data we have imported
## Use the above function "head()" I have introduced
print(df_csv.head(2))
```

```
   Unnamed: 0       crash_datetime borough  zip_code  latitude  longitude  \
0           0  2024-06-30 17:30:00     NaN       NaN       NaN        NaN
1           1  2024-06-30 00:32:00     NaN       NaN       NaN        NaN

     location     on_street_name cross_street_name off_street_name  ...  \
0  (0.0, 0.0)                NaN               NaN     GOLD STREET  ...
1         NaN  BELT PARKWAY RAMP               NaN             NaN  ...

   contributing_factor_vehicle_2  contributing_factor_vehicle_3  \
0                    Unspecified                            NaN
1                    Unspecified                            NaN

   contributing_factor_vehicle_4  contributing_factor_vehicle_5  collision_ic
0                            NaN                            NaN       473674(
1                            NaN                            NaN       473676(
```

```
                   vehicle_type_code_1                  vehicle_type_code_2  \
0                                Sedan                                Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle

   vehicle_type_code_3 vehicle_type_code_4 vehicle_type_code_5
0                  NaN                 NaN                 NaN
1                  NaN                 NaN                 NaN

[2 rows x 29 columns]
```

#### 6.2.4.2 `.xlsx` files

We want to import `.xlsx` files but there are not suitable `.xlsx` files. We can transfer the CSV file to a `.xlsx` file by the fucntion `to_excel` from `Pandas`. Let's see how to achieve this goal according to the following code.

```
## Choose the Package Pandas
import pandas as pd
## Use the function "to_excel"
## Export the data to the default working directory
df_csv.to_excel("nyccrashes_cleaned.xlsx") # Add the name of the Excel file
```

- Check whether the corresponding `.xlsx` file is generated in the working directory

Now we have generated the `.xlsx` file covering the same data. And then we can learn how to import `.xlsx` files. The function wen use is `read_excel` no matter what `Pandas` version is.

- The latest version of `Pandas` corresponds to the following code.

```python
import pandas as pd
## Choose the function "read_excel"
## Add the command "engine" to read the file smoothly
df_excel = pd.read_excel("nyccrashes_cleaned.xlsx", engine = "openpyxl")
## Print top 2 rows of the data
print(df_excel.head(2))
```

```
   Unnamed: 0.1  Unnamed: 0       crash_datetime borough  zip_code  latitude
0             0           0  2024-06-30 17:30:00     NaN       NaN       NaN
1             1           1  2024-06-30 00:32:00     NaN       NaN       NaN

   longitude    location     on_street_name cross_street_name  ...  \
0        NaN  (0.0, 0.0)                NaN               NaN  ...
1        NaN         NaN  BELT PARKWAY RAMP               NaN  ...

  contributing_factor_vehicle_2  contributing_factor_vehicle_3  \
0                   Unspecified                            NaN
1                   Unspecified                            NaN

   contributing_factor_vehicle_4  contributing_factor_vehicle_5  collision_ic
0                            NaN                            NaN       4736746
1                            NaN                            NaN       4736768

                 vehicle_type_code_1                  vehicle_type_code_2
0                              Sedan                                Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle

   vehicle_type_code_3  vehicle_type_code_4 vehicle_type_code_5
0                  NaN                  NaN                 NaN
1                  NaN                  NaN                 NaN

[2 rows x 30 columns]
```

The code of the common `Pandas` version is below. What we need to adjust is to add correct `encoding`.

```
df_excel = pd.read_excel("nyccrashes_cleaned.xlsx", engine = "openpyxl",
encoding = "utf-8")
```

#### 6.2.4.3 `.txt` files

The last common kind of file is `.txt` files. We are able to generate the `.txt` file in the similar way as generatng the `.xlsx` file. We choose the function `to_csv` from `Pandas`. It is necessary to add the command `sep="\t"`. At the same time, we are supposed to add `index=False` to avoid the index of Dataframe. The specific code is following.

```
import pandas as pd
df_csv = pd.read_csv("nyccrashes_cleaned.csv")
## Choose the function "to_csv"
## Add the command "sep='\t'"
## Add the command "index=False"
## Export the data to the default working directory
df_csv.to_csv("nyccrashes_cleaned.txt", sep = "\t", index = False)
```

Now we get the corresponding `.txt` file successfully. The next step is to determine the correct encoding of this `.txt` file. This is because the computer will not read the file successfully without correct encoding. I have listed the code helping us to obtain the correct encoding. We use `with` statement to deal with data. And then we use the function `detect` from Package `Chardet`. The intention of `detect` is to detect character encoding of text files.

```
import chardet
## Use "with" statement
with open("nyccrashes_cleaned.txt", "rb") as f:
    # Execute the command "open"
    # And then assign the result to variable "f"
    raw_data = f.read() # Read the content from "f"
    result = chardet.detect(raw_data)
    encoding = result["encoding"]
    print(str(encoding))
```

```
ascii
```

- Warning: It is possible to generate the encoding which is not "utf-8".

Now we own the `.txt` file and its correct encoding. The last step is to use the function `read_table` to import the `.txt` file. We need to insert the correct encoding too. The corresponding code is following.

```
import pandas as pd
## Choose the function "read_table"
## Add the encoding behind the relative address
df_txt = pd.read_table("nyccrashes_cleaned.txt", encoding = "utf-8")
## The defualt of function "head()" is top five rows
print(df_txt.head())
```

```
   Unnamed: 0       crash_datetime    borough  zip_code  latitude  longitude
0           0  2024-06-30 17:30:00        NaN       NaN       NaN        NaN
1           1  2024-06-30 00:32:00        NaN       NaN       NaN        NaN
2           2  2024-06-30 07:05:00   BROOKLYN   11235.0  40.58106  -73.96744
3           3  2024-06-30 20:47:00  MANHATTAN   10021.0  40.76363  -73.95330
4           4  2024-06-30 10:14:00   BROOKLYN   11222.0  40.73046  -73.95149

                location     on_street_name      cross_street_name  \
```

```
0           (0.0, 0.0)                NaN                    NaN
1                  NaN  BELT PARKWAY RAMP                    NaN
2  (40.58106, -73.96744)              NaN                    NaN
3   (40.76363, -73.9533)        FDR DRIVE                    NaN
4  (40.73046, -73.95149)  GREENPOINT AVENUE  MC GUINNESS BOULEVARD

          off_street_name  ...  contributing_factor_vehicle_2  \
0            GOLD STREET  ...                    Unspecified
1                    NaN  ...                    Unspecified
2  2797      OCEAN PARKWAY  ...                            NaN
3                    NaN  ...                            NaN
4                    NaN  ...                    Unspecified

  contributing_factor_vehicle_3  contributing_factor_vehicle_4  \
0                           NaN                            NaN
1                           NaN                            NaN
2                           NaN                            NaN
3                           NaN                            NaN
4                           NaN                            NaN

  contributing_factor_vehicle_5  collision_id  \
0                           NaN       4736746
1                           NaN       4736768
2                           NaN       4737060
3                           NaN       4737510
4                           NaN       4736759

                  vehicle_type_code_1                  vehicle_type_code_2  \
0                               Sedan                                Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle
2  Station Wagon/Sport Utility Vehicle                                  NaN
3                               Sedan                                  NaN
4                                 Bus                            Box Truck
```

```
   vehicle_type_code_3 vehicle_type_code_4 vehicle_type_code_5
0                  NaN                 NaN                 NaN
1                  NaN                 NaN                 NaN
2                  NaN                 NaN                 NaN
3                  NaN                 NaN                 NaN
4                  NaN                 NaN                 NaN

[5 rows x 29 columns]
```

### 6.2.5 Import the data from other software

#### 6.2.5.1 SAS files

##### 6.2.5.1.1 Transfer the `.csv` file to a `.xpt` file

The reason why we choose `.xpt` file is to ensure data types remain consistent during conversion. Firstly, we need to process the data to ensure there is no space in the name of columns. If we don't do that, we will not achieve the goal. The code of dealing with data is following.

```python
import pandas as pd
df = pd.read_csv("nyccrashes_cleaned.csv")
df_csv_without_unnamed = df.loc[:, ~df.columns.str.contains("^Unnamed")]
df_csv_without_unnamed.columns=df_csv_without_unnamed.columns.str.replace(" "
df_csv_without_unnamed.to_csv("Without_Unamed.csv", index=False)
```

We use the Package `pyreadstat` and the function `write_xport` from `pyreadstat` to transfer the `.csv` file. The corresponding code is following.

```python
import pandas as pd
import pyreadstat
df_without_unnamed = pd.read_csv("Without_Unamed.csv")
```

```
sas_file = "SAS.xpt"
## Export the data to the default working directory
pyreadstat.write_xport(df_without_unnamed, "SAS.xpt")
```

#### 6.2.5.1.2 Import the generated .xpt file

We use the package `pyreadstat` too. We choose the function `read_xport` to import the data. Here is the code.

```
import pyreadstat
## Define the Dataframe and metadata
df_1, meta = pyreadstat.read_xport("SAS.xpt")
## Show the Dataframe
print(df_1.head(2))
## Show the metadata
print(meta)
```

```
        crash_datetime borough  zip_code  latitude  longitude    location  \
0  2024-06-30 17:30:00               NaN       NaN        NaN  (0.0, 0.0)
1  2024-06-30 00:32:00               NaN       NaN        NaN

      on_street_name cross_street_name off_street_name  \
0                                          GOLD STREET
1  BELT PARKWAY RAMP

   number_of_persons_injured  ...  contributing_factor_vehicle_2  \
0                        0.0  ...                    Unspecified
1                        0.0  ...                    Unspecified

   contributing_factor_vehicle_3  contributing_factor_vehicle_4  \
0
1
```

```
   contributing_factor_vehicle_5  collision_id  \
0                                    4736746.0
1                                    4736768.0

                   vehicle_type_code_1                  vehicle_type_code_2
0                                Sedan                                Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle

  vehicle_type_code_3 vehicle_type_code_4 vehicle_type_code_5
0
1

[2 rows x 28 columns]
<pyreadstat._readstat_parser.metadata_container object at 0x116fc7bc0>
```

#### 6.2.5.2 rdata files (the suffix of this file is `.RData`)

##### 6.2.5.2.1 Transfer the `.csv` file to a `.Rdata` file

We need to install the package `rpy2`

```
pip install rpy2
```

And then we choose the function `pandas2ri` The following code helps us achieve the goal.

```
import pandas as pd
## Use the Package rpy2
import rpy2.robjects as ro
from rpy2.robjects import pandas2ri
## Activate conversion between Pandas and R
pandas2ri.activate()
```

```
df_csv = pd.read_csv("nyccrashes_cleaned.csv")
## Transfer the Pandas DataFrame to R DataFrame
df_r = pandas2ri.py2rpy(df_csv)
## Save as .Rdata file
## Export the data to the default working directory
ro.globalenv["R"] = df_r
ro.r("save(R, file = 'nyccrashes_cleaned.Rdata')")
print("The CSV file has been transfered to a .Rdata file successfully.")
```

```
Loading custom .Rprofile
```

```
/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site-packages/rpy2/

Error while trying to convert the column "borough". Fall back to string conversion. The erro

/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site-packages/rpy2/

Error while trying to convert the column "on_street_name". Fall back to string conversion. T

/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site-packages/rpy2/

Error while trying to convert the column "cross_street_name". Fall back to string conversion

/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site-packages/rpy2/

Error while trying to convert the column "contributing_factor_vehicle_3". Fall back to strin

/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site-packages/rpy2/

Error while trying to convert the column "contributing_factor_vehicle_4". Fall back to strin

/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site-packages/rpy2/
```

```
Error while trying to convert the column "contributing_factor_vehicle_5". Fal

/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site

Error while trying to convert the column "vehicle_type_code_3". Fall back to

/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site

Error while trying to convert the column "vehicle_type_code_4". Fall back to

/Users/junyan/work/teaching/ids-f24/ids-f24/.ids-f24-venv/lib/python3.12/site

Error while trying to convert the column "vehicle_type_code_5". Fall back to
```

```
The CSV file has been transfered to a .Rdata file successfully.
```

- The error means : 1.Type conversion failure of columns. Some columns were not converted to the correct data typr in r as epected, and were instead coerced to strings. 2.The data is still saved, but there are potential data type issues. 3.These errores will not influence importing the `.Rdata` file.
- If you want to perform necessary type cpnversions, the following code is suitable.

```
df_csv["boroughs"] = df["boroughs"].astype(str)
```

#### 6.2.5.2.2 Import the generated `.Rdata` file

We also use the Package `rpy2`. We need the function `pandas2ri` too. The code is following

```
import pandas as pd
import rpy2.robjects as ro
## Load the .Rdata file
r_file_path = "nyccrashes_cleaned.Rdata"
ro.r["load"](r_file_path)
## View loaded variables
loaded_objects = ro.r("ls()")
## Show the loaded vatiables
print("Loaded R objects:", loaded_objects)
## We have set the name of dataframe as "R" above
r_dataframe = ro.r["R"]
from rpy2.robjects import pandas2ri
## Transfer R Dataframe to Pandas Dataframe
## Aim to deal with the data conveniently
pandas2ri.activate()
df_2 = pandas2ri.rpy2py(r_dataframe)
print(df_2)
```

```
Loaded R objects: ['R']
      Unnamed: 0        crash_datetime    borough  zip_code   latitude  \
0              0   2024-06-30 17:30:00       None       NaN        NaN
1              1   2024-06-30 00:32:00       None       NaN        NaN
2              2   2024-06-30 07:05:00   BROOKLYN   11235.0  40.581060
3              3   2024-06-30 20:47:00  MANHATTAN   10021.0  40.763630
4              4   2024-06-30 10:14:00   BROOKLYN   11222.0  40.730460
...          ...                   ...        ...       ...        ...
1870        1870   2024-07-07 21:25:00      BRONX   10457.0  40.852520
1871        1871   2024-07-07 10:31:00      BRONX   10460.0  40.843945
1872        1872   2024-07-07 20:15:00     QUEENS   11436.0  40.677982
1873        1873   2024-07-07 14:45:00      BRONX   10452.0  40.843822
1874        1874   2024-07-07 14:12:00      BRONX   10468.0  40.861084

      longitude                 location          on_street_name  \
```

```
0            NaN               (0.0, 0.0)                        None
1            NaN                     None           BELT PARKWAY RAMP
2     -73.967440    (40.58106, -73.96744)                        None
3     -73.953300     (40.76363, -73.9533)                   FDR DRIVE
4     -73.951490    (40.73046, -73.95149)           GREENPOINT AVENUE
...          ...                      ...                         ...
1870 -73.900020    (40.85252, -73.90002)             EAST 180 STREET
1871 -73.885800    (40.843945, -73.8858)                        None
1872 -73.791214  (40.677982, -73.791214)            SUTPHIN BOULEVARD
1873 -73.927500    (40.843822, -73.9275)  MAJOR DEEGAN EXPRESSWAY
1874 -73.911490   (40.861084, -73.91149)                        None

          cross_street_name            off_street_name  ...  \
0                      None                GOLD STREET  ...
1                      None                       None  ...
2                      None    2797      OCEAN PARKWAY  ...
3                      None                       None  ...
4     MC GUINNESS BOULEVARD                       None  ...
...                     ...                        ...  ...
1870                   None                       None  ...
1871                   None  855       EAST 178 STREET  ...
1872            120 AVENUE                        None  ...
1873                   None                       None  ...
1874                   None    2258      HAMPDEN PLACE  ...

      contributing_factor_vehicle_2  contributing_factor_vehicle_3  \
0                       Unspecified                           None
1                       Unspecified                           None
2                              None                           None
3                              None                           None
4                       Unspecified                           None
...                             ...                            ...
1870                    Unspecified                           None
1871                    Unspecified                           None
```

```
1872                 Unspecified                         None
1873                 Unspecified                         None
1874                        None                         None

      contributing_factor_vehicle_4  contributing_factor_vehicle_5  \
0                              None                          None
1                              None                          None
2                              None                          None
3                              None                          None
4                              None                          None
...                             ...                           ...
1870                           None                          None
1871                           None                          None
1872                           None                          None
1873                           None                          None
1874                           None                          None

      collision_id                  vehicle_type_code_1  \
0          4736746                                Sedan
1          4736768  Station Wagon/Sport Utility Vehicle
2          4737060  Station Wagon/Sport Utility Vehicle
3          4737510                                Sedan
4          4736759                                  Bus
...            ...                                  ...
1870       4744144                        Pick-up Truck
1871       4744576  Station Wagon/Sport Utility Vehicle
1872       4745391                                Sedan
1873       4746540                                Sedan
1874       4746320                                Sedan

                      vehicle_type_code_2  vehicle_type_code_3  \
0                                   Sedan                 None
1     Station Wagon/Sport Utility Vehicle                 None
2                                    None                 None
```

```
3                                        None          None
4                                   Box Truck          None
...                                       ...           ...
1870                                    Sedan          None
1871                                     None          None
1872                                    Sedan          None
1873                                    Sedan          None
1874                                     None          None

     vehicle_type_code_4 vehicle_type_code_5
0                   None                None
1                   None                None
2                   None                None
3                   None                None
4                   None                None
...                  ...                 ...
1870                None                None
1871                None                None
1872                None                None
1873                None                None
1874                None                None

[1875 rows x 29 columns]
```

#### 6.2.5.3 stata data (the suffix of this file is `.dta`)

##### 6.2.5.3.1 Transfer the `.csv` file to a `.dta` file

We can only use `Pandas`. We choose the fucntion `to_stata` to save the `.dta` file.

```
import pandas as pd
df_csv = pd.read_csv("nyccrashes_cleaned.csv")
```

```
## Export the data to the default working directory
df_csv.to_stata("stata.dta")
```

```
/var/folders/cq/5ysgnwfn7c3g0h46xyzvpj800000gn/T/ipykernel_33946/1977170459.py:4: InvalidCol
```

```
Not all pandas column names were valid Stata variable names.
The following replacements have been made:

    Unnamed: 0   ->   Unnamed__0

If this is not what you expect, please make sure you have Stata-compliant
column names in your DataFrame (strings only, max 32 characters, only
alphanumerics and underscores, no Stata reserved words)
```

#### 6.2.5.3.2 Import the `.dta` file

We use the function `read_stata` from `Pandas`. And here is the specific code.

```
import pandas as pd
df_3 = pd.read_stata("stata.dta")
print(df_3.head(2))
```

```
   index  Unnamed__0       crash_datetime borough  zip_code  latitude  \
0      0           0  2024-06-30 17:30:00               NaN       NaN
1      1           1  2024-06-30 00:32:00               NaN       NaN

   longitude    location     on_street_name cross_street_name  ...  \
0        NaN  (0.0, 0.0)                                        ...
1        NaN                BELT PARKWAY RAMP                   ...
```

```
  contributing_factor_vehicle_2  contributing_factor_vehicle_3  \
0                   Unspecified
1                   Unspecified

   contributing_factor_vehicle_4  contributing_factor_vehicle_5  collision_ic
0                                                                     473674(
1                                                                     473676(

                  vehicle_type_code_1                 vehicle_type_code_2
0                               Sedan                               Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle

   vehicle_type_code_3  vehicle_type_code_4 vehicle_type_code_5
0
1

[2 rows x 30 columns]
```

#### 6.2.5.4 spss data (the suffix of this file is `.sav`)

##### 6.2.5.4.1 Transfer the `.csv` file to a `.sav` file

We need to use the Package `pyreadstat` We choose the function `write_sav` form `pyreadstat` We are supposed to use the CSV file which is without space. We can uese the following code.

```
import pandas as pd
import pyreadstat
df_csv = pd.read_csv("Without_Unamed.csv")
## Export the data to the default working directory
pyreadstat.write_sav(df_csv, "SPSS.sav")
```

#### 6.2.5.4.2 Import the generated `.sav` file

We also use the Package `pyreadstat`. We utilize the function `read_sav` from `pyreadstat`. The following code helps us import the `.sav` file.

```python
import pandas as pd
import pyreadstat
df_4, meta = pyreadstat.read_sav("SPSS.sav")
print(df_4.head(2))
print(meta)
```

```
        crash_datetime borough  zip_code  latitude  longitude    location  \
0  2024-06-30 17:30:00               NaN       NaN        NaN  (0.0, 0.0)
1  2024-06-30 00:32:00               NaN       NaN        NaN

      on_street_name cross_street_name off_street_name  \
0                                          GOLD STREET
1  BELT PARKWAY RAMP

   number_of_persons_injured  ...  contributing_factor_vehicle_2  \
0                        0.0  ...                    Unspecified
1                        0.0  ...                    Unspecified

   contributing_factor_vehicle_3  contributing_factor_vehicle_4  \
0
1

   contributing_factor_vehicle_5  collision_id  \
0                                    4736746.0
1                                    4736768.0

                  vehicle_type_code_1                  vehicle_type_code_2  \
0                               Sedan                                Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle
```

```
  vehicle_type_code_3 vehicle_type_code_4 vehicle_type_code_5
0
1

[2 rows x 28 columns]
<pyreadstat._readstat_parser.metadata_container object at 0x1176b5400>
```

#### 6.2.5.5 Matlab files (the suffix of this file is `.mat`)

##### 6.2.5.5.1 Transfer the `.csv` file to a `.mat` file

We need to install the package `scipy.io`

```
pip install scipy
```

And then we choose the function `savemat` from `scipy`. The specific code is following.

```
import pandas as pd
from scipy.io import savemat
df_csv = pd.read_csv("nyccrashes_cleaned.csv")
## Convert DataFrame to dicotionary form
## MATLAB.mat require dictionary format
data_dict = {"data": df_csv.to_dict("list")}
## Save the dictionary as a .mat file
## Export the data to the default working directory
savemat("MATLAB.mat", data_dict)
```

##### 6.2.5.5.2 Import the generated `.mat` file

We use the Package `scipy.io` too. We choose the function `loadmat` from `spicy.io` And the corresponding code is following.

```python
import pandas as pd
from scipy.io import loadmat
df_csv = pd.read_csv("nyccrashes_cleaned.csv")
df_5 = loadmat("MATLAB.mat")
## Show the data keys
print(df_5.keys())
## Show the contents of the "data" keys
print(df_5["data"])
```

```
dict_keys(['__header__', '__version__', '__globals__', 'data'])
[[(array([[   0,    1,    2, ..., 1872, 1873, 1874]]), array(['2024-06-30 17:30:00', '2024-0
         '2024-06-30 07:05:00', ..., '2024-07-07 20:15:00',
         '2024-07-07 14:45:00', '2024-07-07 14:12:00'], dtype='<U19'), array(['nan
         'nan                             ',
         'BROOKLYN                        ', ...,
         'QUEENS                          ',
         'BRONX                           ',
         'BRONX                           '], dtype='<U32'), array([[   nan,    nan, 11235.,
          40.861084]]), array([[       nan,        nan, -73.96744 , ..., -73.791214, -73.927
         -73.91149 ]]), array(['(0.0, 0.0)                      ',
         'nan                             ',
         '(40.58106, -73.96744)           ', ...,
         '(40.677982, -73.791214)         ',
         '(40.843822, -73.9275)           ',
         '(40.861084, -73.91149)          '], dtype='<U32'), array(['nan
         'BELT PARKWAY RAMP               ',
         'nan                             ', ...,
         'SUTPHIN BOULEVARD               ',
         'MAJOR DEEGAN EXPRESSWAY         ',
         'nan                             '], dtype='<U32'), array(['nan
         'nan                             ',
         'nan                             ', ...,
         '120 AVENUE                      ',
```

```
 'nan                               ',
 'nan                               '], dtype='<U32'), array(['GOLD STR
 'nan                                  ',
 '2797      OCEAN PARKWAY              ', ...,
 'nan                                  ',
 'nan                                  ',
 '2258      HAMPDEN PLACE              '], dtype='<U35'), array([[0, 0
 'Unspecified                                                  ',
 'Unspecified                                                  ', ...,
 'Passing or Lane Usage Improper                               ',
 'Driver Inexperience                                          ',
 'Unspecified                                                  '],
dtype='<U53'), array(['Unspecified
 'Unspecified                                                  ',
 'nan                                                          ', ...,
 'Unspecified                                                  ',
 'Unspecified                                                  ',
 'nan                                                          '],
dtype='<U53'), array(['nan                                                 ',
 'nan                               ',
 'nan                               ', ...,
 'nan                               ',
 'nan                               ',
 'nan                               '], dtype='<U32'), array(['nan
 'nan                               ',
 'nan                               ', ...,
 'nan                               ',
 'nan                               ',
 'nan                               '], dtype='<U32'), array(['nan
 'nan                               ',
 'nan                               ', ...,
 'nan                               ',
 'nan                               ',
 'nan                               '], dtype='<U32'), array([[4736746
```

```
         'Station Wagon/Sport Utility Vehicle',
         'Station Wagon/Sport Utility Vehicle', ...,
         'Sedan                              ',
         'Sedan                              ',
         'Sedan                              '], dtype='<U35'), array(['Sedan
         'Station Wagon/Sport Utility Vehicle',
         'nan                                ', ...,
         'Sedan                              ',
         'Sedan                              ',
         'nan                                '], dtype='<U35'), array(['nan
         'nan                                ',
         'nan                                ', ...,
         'nan                                ',
         'nan                                ',
         'nan                                '], dtype='<U35'), array(['nan
         'nan                                ',
         'nan                                ', ...,
         'nan                                ',
         'nan                                ',
         'nan                                '], dtype='<U35'), array(['nan
         'nan                                ',
         'nan                                ', ...,
         'nan                                ',
         'nan                                ',
         'nan                                '], dtype='<U35'))
```

#### 6.2.5.6 HDF5 files (the suffix of this file is `.h5`)

##### 6.2.5.6.1 Transfer the `.csv` file to a `.h5` file

We can only use `Pandas`. At the same time, the function `to_hdf` helps us achive the goal. The code is following.

```
import pandas as pd
import tables
df_csv = pd.read_csv("nyccrashes_cleaned.csv")
## Export the data to the default working directory
df_csv.to_hdf("HDF5.h5", key = "data", mode = "w")
```

```
/var/folders/cq/5ysgnwfn7c3g0h46xyzvpj800000gn/T/ipykernel_33946/3411576893.p

your performance may suffer as PyTables will pickle object types that it cann
map directly to c-types [inferred_type->mixed,key->block2_values] [items->Ind
       'cross_street_name', 'off_street_name', 'contributing_factor_vehicle_1
       'contributing_factor_vehicle_2', 'contributing_factor_vehicle_3',
       'contributing_factor_vehicle_4', 'contributing_factor_vehicle_5',
       'vehicle_type_code_1', 'vehicle_type_code_2', 'vehicle_type_code_3',
       'vehicle_type_code_4', 'vehicle_type_code_5'],
      dtype='object')]
```

#### 6.2.5.6.2 Import the generated `.h5` file

We only use `Pandas` too. We need the function `read_h5`. The code of importing .h5 file is following.

```
import pandas as pd
df_6 = pd.read_hdf("HDF5.h5", key = "data")
print(df_6.head(2))
```

```
   Unnamed: 0       crash_datetime borough  zip_code  latitude  longitude  \
0           0  2024-06-30 17:30:00     NaN       NaN       NaN        NaN
1           1  2024-06-30 00:32:00     NaN       NaN       NaN        NaN
```

```
    location     on_street_name cross_street_name off_street_name  ...  \
0  (0.0, 0.0)               NaN               NaN     GOLD STREET  ...
1         NaN  BELT PARKWAY RAMP              NaN             NaN  ...

   contributing_factor_vehicle_2  contributing_factor_vehicle_3  \
0                    Unspecified                            NaN
1                    Unspecified                            NaN

   contributing_factor_vehicle_4  contributing_factor_vehicle_5  collision_id  \
0                            NaN                            NaN       4736746
1                            NaN                            NaN       4736768

                vehicle_type_code_1                 vehicle_type_code_2  \
0                             Sedan                               Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle

   vehicle_type_code_3 vehicle_type_code_4 vehicle_type_code_5
0                  NaN                 NaN                 NaN
1                  NaN                 NaN                 NaN

[2 rows x 29 columns]
```

#### 6.2.5.7 Import multiple files and merge them into a new file

I have introduced the method of importing single file of data. Python also allows us to import multiple files simultaneously. We choose the Package `glob` and Package `Pandas`

```
## Install Package Glob
pip install glob
```

The effect of Package `glob` is to find files and directories that match the specified pattern. We use the function `glob` from Package `glob`. The intention of function `glob` is to find all file paths that match a specitic pattern and return a list of file paths. The following fucntion is the corresponding code.

```
## Use the package globe and the package pandas
import glob
import pandas as pd
## Merge multiple arrays
## * means match any number of characters( including the null characters)
all_files = glob.glob("*.csv")
## Create a list to store the data
all_data = []
## Use "for" statement to import all of the csv files
for filename in all_files:
    df = pd.read_csv(filename, index_col=None, header=0)
    all_data.append(df)
## Combine multiple pandas objects into one along a fixed axis using some mer
data_merge = pd.concat(all_data, axis=0, ignore_index=True)
## Check the result
print(data_merge.head(2))
```

```
   Unnamed: 0       crash_datetime borough  zip_code  latitude  longitude  \
0         0.0  2024-06-30 17:30:00     NaN       NaN       NaN        NaN
1         1.0  2024-06-30 00:32:00     NaN       NaN       NaN        NaN

     location     on_street_name cross_street_name off_street_name  ...  \
0  (0.0, 0.0)                NaN               NaN     GOLD STREET  ...
1         NaN  BELT PARKWAY RAMP               NaN             NaN  ...

   contributing_factor_vehicle_2  contributing_factor_vehicle_3  \
0                    Unspecified                            NaN
1                    Unspecified                            NaN
```

```
   contributing_factor_vehicle_4  contributing_factor_vehicle_5  collision_id  \
0                            NaN                            NaN       4736746
1                            NaN                            NaN       4736768

                   vehicle_type_code_1                  vehicle_type_code_2  \
0                                Sedan                                Sedan
1  Station Wagon/Sport Utility Vehicle  Station Wagon/Sport Utility Vehicle

   vehicle_type_code_3 vehicle_type_code_4 vehicle_type_code_5
0                  NaN                 NaN                 NaN
1                  NaN                 NaN                 NaN

[2 rows x 29 columns]
```

### 6.2.6 View data information

It is natural for us to be interested in the fundamental information of the data we have imported. As a result, I have listed some useful functions to get the basic knowledge of the data.

The following code helps us know how much the data is. We choose the basic function `shape` from `pandas`.

```
## How much is the crashes data is
df_csv.shape
```

```
(1875, 29)
```

- There are 1875 data and 29 columns in the file.

The following code helps us check the type of each variable in data. The fucntion is `dtypes` from `Pandas`

```python
## Show all types of the crashes' variables
df_csv.dtypes
```

```
Unnamed: 0                          int64
crash_datetime                     object
borough                            object
zip_code                          float64
latitude                          float64
longitude                         float64
location                           object
on_street_name                     object
cross_street_name                  object
off_street_name                    object
number_of_persons_injured           int64
number_of_persons_killed            int64
number_of_pedestrians_injured       int64
number_of_pedestrians_killed        int64
number_of_cyclist_injured           int64
number_of_cyclist_killed            int64
number_of_motorist_injured          int64
number_of_motorist_killed           int64
contributing_factor_vehicle_1      object
contributing_factor_vehicle_2      object
contributing_factor_vehicle_3      object
contributing_factor_vehicle_4      object
contributing_factor_vehicle_5      object
collision_id                        int64
vehicle_type_code_1                object
vehicle_type_code_2                object
vehicle_type_code_3                object
vehicle_type_code_4                object
vehicle_type_code_5                object
dtype: object
```

- Our computer has listed 29 variables and their corresponding types.

The following code is suitable for viewing overall data information. We use the basic function `info` from `Pandas`

```
df_csv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1875 entries, 0 to 1874
Data columns (total 29 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Unnamed: 0                     1875 non-null   int64  
 1   crash_datetime                 1875 non-null   object 
 2   borough                        1749 non-null   object 
 3   zip_code                       1749 non-null   float64
 4   latitude                       1722 non-null   float64
 5   longitude                      1722 non-null   float64
 6   location                       1729 non-null   object 
 7   on_street_name                 1329 non-null   object 
 8   cross_street_name              943 non-null    object 
 9   off_street_name                546 non-null    object 
 10  number_of_persons_injured      1875 non-null   int64  
 11  number_of_persons_killed       1875 non-null   int64  
 12  number_of_pedestrians_injured  1875 non-null   int64  
 13  number_of_pedestrians_killed   1875 non-null   int64  
 14  number_of_cyclist_injured      1875 non-null   int64  
 15  number_of_cyclist_killed       1875 non-null   int64  
 16  number_of_motorist_injured     1875 non-null   int64  
 17  number_of_motorist_killed      1875 non-null   int64  
 18  contributing_factor_vehicle_1  1864 non-null   object 
 19  contributing_factor_vehicle_2  1425 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                   1875 non-null   int64
 24  vehicle_type_code_1            1842 non-null   object
 25  vehicle_type_code_2            1230 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: float64(3), int64(10), object(16)
memory usage: 424.9+ KB
```

- The basic information of crashes data has been listed.

The function `describe` from `Pandas` helps generate descriptive statistics.

```
## Show the basic descriptive statistics of crashes data
df_csv.describe()
```

| | Unnamed: 0 | zip_code | latitude | longitude | number_of_persons_in |
|---|---|---|---|---|---|
| count | 1875.000000 | 1749.000000 | 1722.000000 | 1722.000000 | 1875.000000 |
| mean | 937.000000 | 10892.563179 | 40.719287 | -73.919898 | 0.617067 |
| std | 541.410196 | 525.579066 | 0.081315 | 0.085191 | 0.915610 |
| min | 0.000000 | 10000.000000 | 40.513510 | -74.237366 | 0.000000 |
| 25% | 468.500000 | 10455.000000 | 40.662752 | -73.968543 | 0.000000 |
| 50% | 937.000000 | 11208.000000 | 40.712778 | -73.922933 | 0.000000 |
| 75% | 1405.500000 | 11239.000000 | 40.767641 | -73.869405 | 1.000000 |
| max | 1874.000000 | 11694.000000 | 40.907246 | -73.702190 | 11.000000 |

If we want to summarize the names of all columns, it is a good choice to use the function `columns` from `Pandas`

```
## Get all columns' names of crashes data
df_csv.columns
```

```
Index(['Unnamed: 0', 'crash_datetime', 'borough', 'zip_code', 'latitude',
       'longitude', 'location', 'on_street_name', 'cross_street_name',
       'off_street_name', 'number_of_persons_injured',
       'number_of_persons_killed', 'number_of_pedestrians_injured',
       'number_of_pedestrians_killed', 'number_of_cyclist_injured',
       'number_of_cyclist_killed', 'number_of_motorist_injured',
       'number_of_motorist_killed', 'contributing_factor_vehicle_1',
       'contributing_factor_vehicle_2', 'contributing_factor_vehicle_3',
       'contributing_factor_vehicle_4', 'contributing_factor_vehicle_5',
       'collision_id', 'vehicle_type_code_1', 'vehicle_type_code_2',
       'vehicle_type_code_3', 'vehicle_type_code_4', 'vehicle_type_code_5'],
      dtype='object')
```

The function `tail` from `Pandas` allow us to view the last rows.

```
## Show the last 2 rows of crashes data
df_csv.tail(n = 2)
```

| | Unnamed: 0 | crash_datetime | borough | zip_code | latitude | longitude | location |
|---|---|---|---|---|---|---|---|
| 1873 | 1873 | 2024-07-07 14:45:00 | BRONX | 10452.0 | 40.843822 | -73.92750 | (40.843822, -73.927 |
| 1874 | 1874 | 2024-07-07 14:12:00 | BRONX | 10468.0 | 40.861084 | -73.91149 | (40.861084, -73.911 |

The function `unique` from `Pandas` dedicates in unique values of one column.

```
## Show the unique values in the column named "crash_datetime"
df_csv["crash_datetime"].unique()
```

```
array(['2024-06-30 17:30:00', '2024-06-30 00:32:00',
       '2024-06-30 07:05:00', ..., '2024-07-07 09:13:00',
       '2024-07-07 10:31:00', '2024-07-07 14:12:00'], dtype=object)
```

We can get the values of one column(without deduplication). The choose of function is `values` from `Pandas` instead of `unique`

```
## Show all values of the column named "crash_datetime"
df_csv["crash_datetime"].values
```

```
array(['2024-06-30 17:30:00', '2024-06-30 00:32:00',
       '2024-06-30 07:05:00', ..., '2024-07-07 20:15:00',
       '2024-07-07 14:45:00', '2024-07-07 14:12:00'], dtype=object)
```

### 6.2.7 Find Null Values

It is necessary for us to find null values before we clean and preprocess the data. The following content covers how to find null values.

#### 6.2.7.1 Determine whether there are missing values.

We need to use the function `isnull` firstly. The aim is to detect missing values in data. And then we add the command `any` to determine whether there are missing values.

- Determine whether there are missing values in columns

```
## Determine whether there are missing values in columns of crashes data
df_csv.isnull().any(axis = 0) # "axis=0" means columns
```

```
Unnamed: 0                         False
crash_datetime                     False
borough                             True
zip_code                            True
latitude                            True
longitude                           True
location                            True
on_street_name                      True
cross_street_name                   True
off_street_name                     True
number_of_persons_injured          False
number_of_persons_killed           False
number_of_pedestrians_injured      False
number_of_pedestrians_killed       False
number_of_cyclist_injured          False
number_of_cyclist_killed           False
number_of_motorist_injured         False
number_of_motorist_killed          False
contributing_factor_vehicle_1       True
contributing_factor_vehicle_2       True
contributing_factor_vehicle_3       True
contributing_factor_vehicle_4       True
contributing_factor_vehicle_5       True
collision_id                       False
vehicle_type_code_1                 True
vehicle_type_code_2                 True
vehicle_type_code_3                 True
vehicle_type_code_4                 True
vehicle_type_code_5                 True
dtype: bool
```

- Determine whether there are missing values in rows.

```
## Determine whether there are missing values in rows of crashes data
df_csv.isnull().any(axis = 1) # "axis=1" means rows
```

```
0       True
1       True
2       True
3       True
4       True
        ...
1870    True
1871    True
1872    True
1873    True
1874    True
Length: 1875, dtype: bool
```

#### 6.2.7.2 Locate the missing values of rows/columns

We utilize the function `loc` from `Pandas`. The function of `loc` selects or modifies data in a DataFrame or Series. The slection and modification are based on labels.

- Locate the missing values of rows

```
## Locate the missing values in crashes data rows
df_csv.loc[df_csv.isnull().any(axis = 1)]
```

| | Unnamed: 0 | crash_datetime | borough | zip_code | latitude | longit |
|---|---|---|---|---|---|---|
| 0 | 0 | 2024-06-30 17:30:00 | NaN | NaN | NaN | NaN |
| 1 | 1 | 2024-06-30 00:32:00 | NaN | NaN | NaN | NaN |
| 2 | 2 | 2024-06-30 07:05:00 | BROOKLYN | 11235.0 | 40.581060 | -73.96 |

| | Unnamed: 0 | crash_datetime | borough | zip_code | latitude | longitude | location |
|---|---|---|---|---|---|---|---|
| 3 | 3 | 2024-06-30 20:47:00 | MANHATTAN | 10021.0 | 40.763630 | -73.953300 | (40.76363, - |
| 4 | 4 | 2024-06-30 10:14:00 | BROOKLYN | 11222.0 | 40.730460 | -73.951490 | (40.73046, - |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 1870 | 1870 | 2024-07-07 21:25:00 | BRONX | 10457.0 | 40.852520 | -73.900020 | (40.85252, - |
| 1871 | 1871 | 2024-07-07 10:31:00 | BRONX | 10460.0 | 40.843945 | -73.885800 | (40.843945, |
| 1872 | 1872 | 2024-07-07 20:15:00 | QUEENS | 11436.0 | 40.677982 | -73.791214 | (40.677982, |
| 1873 | 1873 | 2024-07-07 14:45:00 | BRONX | 10452.0 | 40.843822 | -73.927500 | (40.843822, |
| 1874 | 1874 | 2024-07-07 14:12:00 | BRONX | 10468.0 | 40.861084 | -73.911490 | (40.861084, |

#### 6.2.7.3 Determine the number of missing values.

We also use the function `isnull`. But this time we add the command `sum` rather than `any`.

```
## Calculate the number of missing values in crashes data columns
df_csv.isnull().sum(axis = 0)
```

```
Unnamed: 0                          0
crash_datetime                      0
borough                           126
zip_code                          126
latitude                          153
longitude                         153
location                          146
on_street_name                    546
cross_street_name                 932
off_street_name                  1329
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    1701
contributing_factor_vehicle_4    1823
contributing_factor_vehicle_5    1861
collision_id                        0
vehicle_type_code_1                33
vehicle_type_code_2               645
vehicle_type_code_3              1713
vehicle_type_code_4              1827
vehicle_type_code_5              1861
dtype: int64
```

## 6.3 SQL

This section was written by Thea Johnson, a senior in Statistics at the time.

### 6.3.1 Subsection 1

### 6.3.2 Subsection 2

### 6.3.3 Extended Reading

Give some references here.

## 6.4 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 September 16, 2024, in CSC format.

```
import pandas as pd

# Load the dataset
file_path = 'data/nyccrashes_2024w0630_by20240916.csv'
df = pd.read_csv(file_path)

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

# Display the first few rows of the dataset to understand its structure
df.head()
```

| | crash_date | crash_time | borough | zip_code | latitude | longitude | location | on_ |
|---|---|---|---|---|---|---|---|---|
| 0 | 06/30/2024 | 17:30 | NaN | NaN | 0.00000 | 0.00000 | (0.0, 0.0) | NaN |
| 1 | 06/30/2024 | 0:32 | NaN | NaN | NaN | NaN | NaN | BEL |
| 2 | 06/30/2024 | 7:05 | BROOKLYN | 11235.0 | 40.58106 | -73.96744 | (40.58106, -73.96744) | NaN |
| 3 | 06/30/2024 | 20:47 | NaN | NaN | 40.76363 | -73.95330 | (40.76363, -73.9533) | FDF |
| 4 | 06/30/2024 | 10:14 | BROOKLYN | 11222.0 | 40.73046 | -73.95149 | (40.73046, -73.95149) | GRE |

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)

# Longitude/latitude don't need double precision
df['latitude'] = df['latitude'].astype('float32', errors='ignore')
df['longitude'] = df['longitude'].astype('float32', errors='ignore')

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

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

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

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
missing_cooccur, total_missing_zip_code, total_missing_borough
```

```
(np.int64(541), np.int64(541), np.int64(541))
```

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
zip_code = get_zip_code(latitude, longitude)
```

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.

## 6.5 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.

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

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

# File paths
csv_file = 'data/nyccrashes_2024w0630_by20240916.csv'
feather_file = 'data/nyccrashes_cleaned.feather'

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

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

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

Read the feather file back in:

```
dff = pd.read_feather("data/nyccrashes_cleaned.feather")
dff.shape
```

## 6.6 Using the Census Data

The US Census provides a lot of useful data that could be merged with the NYC crash data for further analytics.

First, ensure the DataFrame (df) is ready for merging with census data. Specifically, check that the `zip_code` column is clean and consistent.and consistent.

```
import pandas as pd
df = pd.read_feather("data/nyccrashes_cleaned.feather")

valid_zip_df = df.dropna(subset=['zip_code']).copy()
valid_zip_df['zip_code'] = valid_zip_df['zip_code'].astype(int).astype(str).str.zfill(5)
unique_zips = valid_zip_df['zip_code'].unique()
```

We can use the `uszipcode` package to get basic demographic data for each zip code. For more detailed or specific census data, using the `CensusData` package or direct API calls to the Census Bureau's API.

The `uszipcode` package provides a range of information about ZIP codes in the United States. When you query a ZIP code using `uszipcode`, you can access various attributes related to demographic data, housing, geographic location, and more. Here are some of the key variables available at the ZIP code level:

**Demographic Information**

- `population`: The total population.
- `population_density`: The population per square kilometer.
- `housing_units`: The total number of housing units.
- `occupied_housing_units`: The number of occupied housing units.
- `median_home_value`: The median value of homes.
- `median_household_income`: The median household income.
- `age_distribution`: A breakdown of the population by age.

**Geographic Information**

- `zipcode`: The ZIP code.
- `zipcode_type`: The type of ZIP code (e.g., Standard, PO Box).

- `major_city`: The major city associated with the ZIP code.
- `post_office_city`: The city name recognized by the U.S. Postal Service.
- `common_city_list`: A list of common city names for the ZIP code.
- `county`: The county in which the ZIP code is located.
- `state`: The state in which the ZIP code is located.
- `lat`: The latitude of the approximate center of the ZIP code.
- `lng`: The longitude of the approximate center of the ZIP code.
- `timezone`: The timezone of the ZIP code.

**Economic and Housing Data**

- `land_area_in_sqmi`: The land area in square miles.
- `water_area_in_sqmi`: The water area in square miles.
- `occupancy_rate`: The rate of occupancy for housing units.
- `median_age`: The median age of the population.

Install the `uszipcode` package into the current virtual environment, if it has not been installd yet.

```
pip install uszipcode


``
We will first clean the `zip_code` column to ensure it only
contains valid ZIP codes. Then, we will use a vectorized
approach to fetch the required data for each unique zip code
and merge this information back into the original `DataFrame`.
`{python}
# Remove rows where 'zip_code' is missing or not a valid ZIP code format
valid_zip_df = df.dropna(subset=['zip_code']).copy()
valid_zip_df['zip_code'] = valid_zip_df['zip_code'].astype(str).str.zfill(5)
```

Since `uszipcode` doesn't valid_zip_dfrently support vectorized operations for multiple ZIP code queries, we'll optimize the process by querying each unique ZIP code once, then merging the results with the original `DataFrame`. This approach minimizes redundant queries for ZIP codes that appear multiple

times.

```
from uszipcode import SearchEngine

# Initialize the SearchEngine
search = SearchEngine()

# Fetch median home value and median household income for each unique ZIP code
zip_data = []
for zip_code in unique_zips:
    result = search.by_zipcode(zip_code)
    if result:  # Check if the result is not None
        zip_data.append({
            "zip_code": zip_code,
            "median_home_value": result.median_home_value,
            "median_household_income": result.median_household_income
        })
    else:  # Handle the case where the result is None
        zip_data.append({
            "zip_code": zip_code,
            "median_home_value": None,
            "median_household_income": None
        })

# Convert to DataFrame
zip_info_df = pd.DataFrame(zip_data)

# Merge this info back into the original DataFrame based on 'zip_code'
merged_df = pd.merge(valid_zip_df, zip_info_df, how="left", on="zip_code")
```

```
merged_df.head()
```

| | crash_datetime | borough | zip_code | latitude | longitude | location |
|---|---|---|---|---|---|---|
| 0 | 2024-06-30 07:05:00 | BROOKLYN | 11235 | 40.58106 | -73.96744 | (40.58106, -73 |
| 1 | 2024-06-30 20:47:00 | MANHATTAN | 10021 | 40.76363 | -73.95330 | (40.76363, -73 |
| 2 | 2024-06-30 10:14:00 | BROOKLYN | 11222 | 40.73046 | -73.95149 | (40.73046, -73 |
| 3 | 2024-06-30 15:52:00 | BRONX | 10468 | 40.86685 | -73.89597 | (40.86685, -73 |
| 4 | 2024-06-30 16:30:00 | BROOKLYN | 11226 | 40.63969 | -73.95321 | (40.63969, -73 |

# 7 Visualization

## 7.1 Data Visualization with Plotnine

This section was written by Julia Mazzola.

### 7.1.1 Introduction

Hi! My name is Julia, and I am a Senior double majoring in Statistical Data Science and Economics. I'm excited to show you the power of data visualization with `Plotnine`, a Python library inspired by R's `ggplot2`. Visualization is a crucial tool to effectively communicate your findings to your audience and `Plotnine` is a useful library to use.

### 7.1.2 What is `Plotnine`?

`Plotnine` uses grammer of graphics (Wilkinson, 2012) to create layered, customizable visualizations. Grammar of graphics is a framework that provides a systematic approach to creating visual representations of data by breaking down the plot into its fundamental components. To understand this better, think about how sentences have grammer, we can layer our graphics to create complex and detailed visulizations.

Components of the layered grammar of graphics:

- **Layer:** used to create the objects on a plot
- **Data:** defines the source of the information to be visualized

- **Mapping:** defines how the variables are represented in the plot
- **Statistical transformation (stat):** transforms the data, generally by summarizing the information
- **Geometric object (geom):** determines the type of plot type (e.g., points, lines, bars)
- **Position adjustment (position):** adjusts the display of overlapping points to improve clarity
- **Scale:** controls how values are mapped to aesthetic attributes (e.g., color, size)
- **Coordinate system (coord):** maps the position of objects onto the plane of the plot, and controls how the axes and grid lines are drawn
- **Faceting (facet):** used to split the data up into subsets of the entire dataset

You can make a wide array of different graphics with `Plotnine`. Some common examples are:

- Scatterplot `geom_point()`
- Bar Chart `geom_bar()`
- Histogram `geom_histogram()`
- Line Chart `geom_line()`

### 7.1.3 Installing Plotnine

To use `Plotnine` you must install it into your venv first. The instructions are as follows:

Type this command into either conda, your terminal, gitbash, or whatever you use for package install for your venv.

For **pip:**

```
pip install plotnine
```

For **conda:**

```
conda install -c conda-forge plotnine
```

You can import `Plotnine` without a prefix:

```
from plotnine import *
```

Or with with a prefix to access each component such as:

```
import plotnine as p9
```

This way is generally recommended for larger projects or when collaborating with others for better code maintainability. But for simplicity in this section I will use the first method.

For the examples we will be using NYC open data to visualize motor vehicle crashes from the week of June 30, 2024.

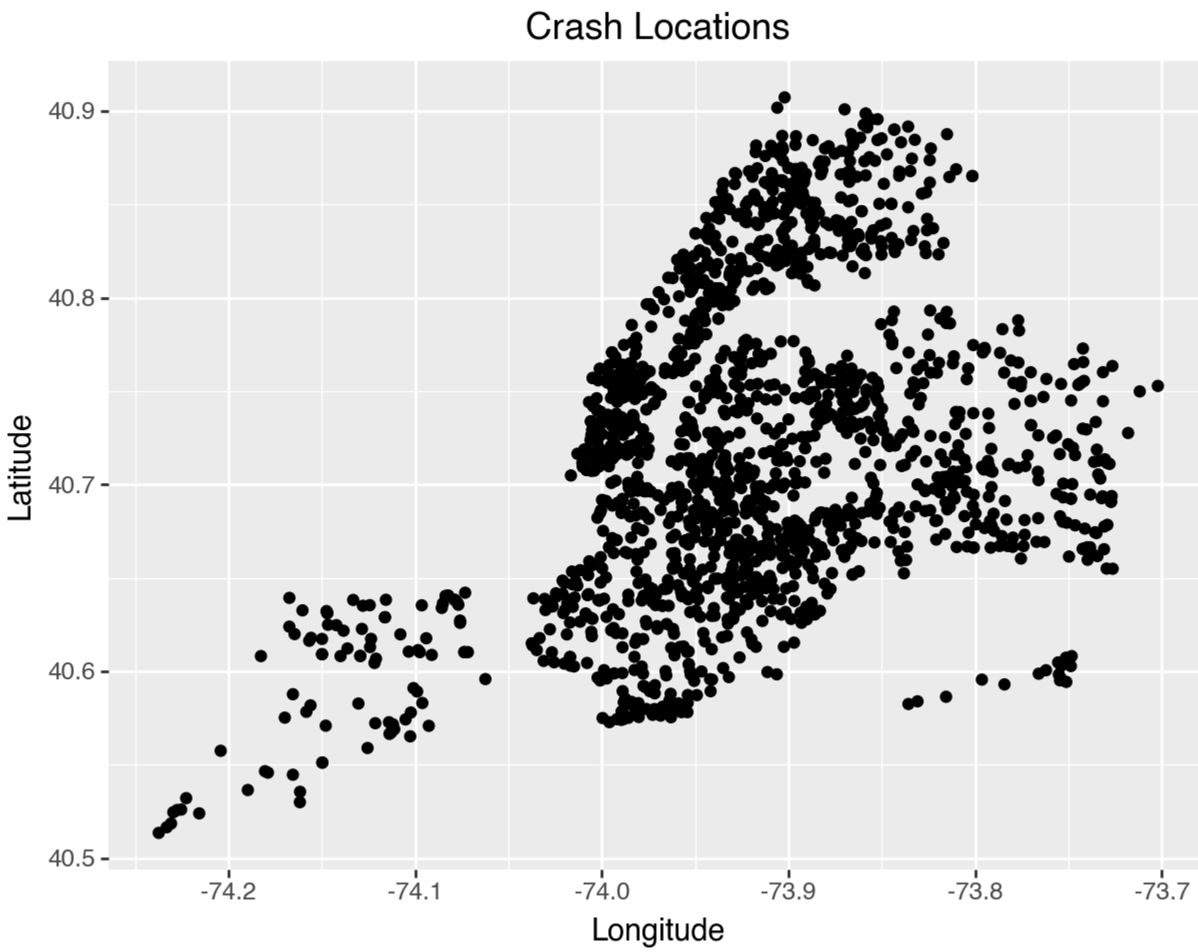
```
import pandas as pd

nyc_crash = pd.read_feather('data/nyccrashes_cleaned.feather').dropna(subset=['borough'])
```

### 7.1.4 Scatterplot

Firstly, we will be creating a scatterplot. This can be done with `geom_point()`. Our scatterplot will be displaying Crash Locations based on the longitude and latitude of the crash sites.

**Creating a Basic Scatterplot**

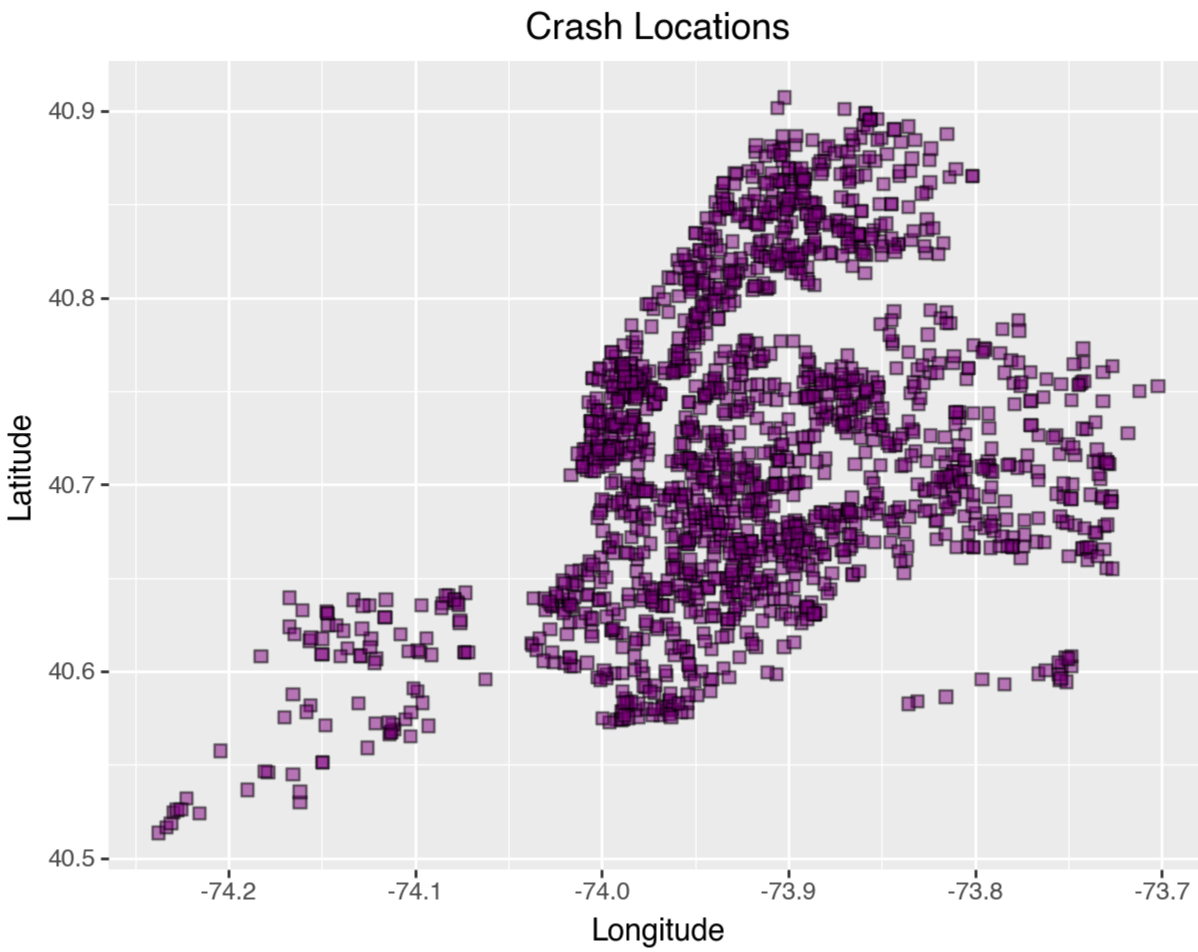
```python
import warnings
warnings.filterwarnings("ignore", category=UserWarning)

(ggplot(nyc_crash, aes(x='longitude', y='latitude')) +
# Specifies graph type
    geom_point() +
# Creates labels for graphic
    labs(title='Crash Locations',
         x='Longitude',
         y='Latitude') +
# Because we are plotting maps we want 1:1 ratio
# coord_fixed(): changes the ratio of the x and y axis
    coord_fixed(ratio = 1))
```




**Customizing a Scatterplot**

You can customize your plot further by changing the color, edge color, transparency, size, or shape of your points. This is done in geom_point().

```
(ggplot(nyc_crash, aes(x='longitude', y='latitude')) +
# Changes what our points look like
# color= changes the outline color
# fill= changes the fill color
# alpha= changes transparency
# size= changes size
# shape= chanegs shape (s = square)
    geom_point(color = 'black', fill = 'purple',
        alpha = 0.5, size = 2, shape = 's') +
    labs(title='Crash Locations',
         x='Longitude',
         y='Latitude') +
    coord_fixed(ratio = 1))
```



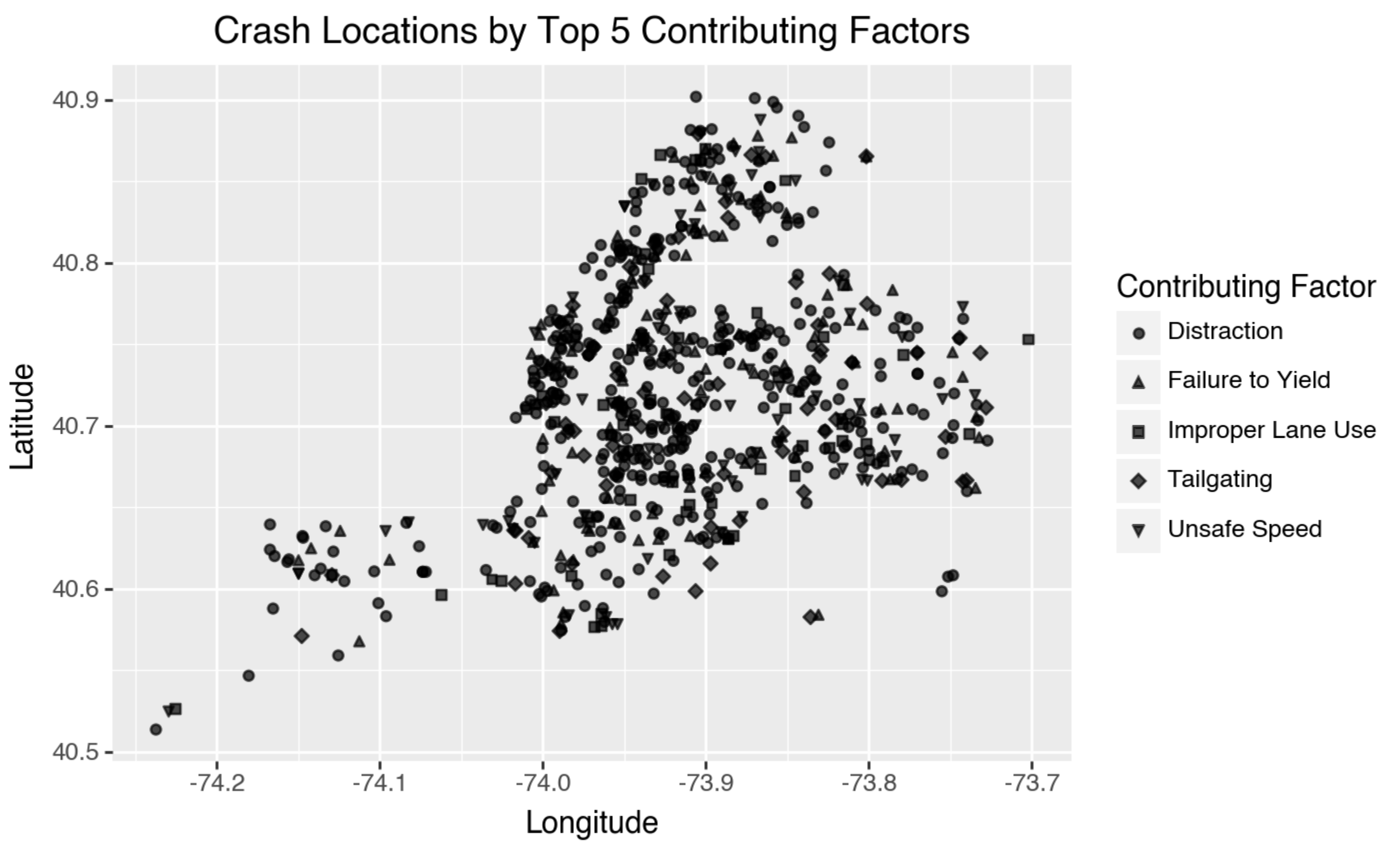

This scatterplot provides a lot of information, yet there are ways we can customize our plot to be more informative for our audience. We can create a scatterplot that differentiates by contributing factor.

**Changing Shape by Variables**

Changing shape of points by `contributing_factor_vehicle_1`:

```
# List of top 5 reasons for the contributing facor
# Abbreviating names for clairity
factor1 = {"Driver Inattention/Distraction": "Distraction",
            "Failure to Yield Right-of-Way": "Failure to Yield",
            "Following Too Closely": "Tailgating",
            "Unsafe Speed": "Unsafe Speed",
            "Passing or Lane Usage Improper": "Improper Lane Use"}

# Filter the data to only include valid contributing factors
confact = nyc_crash.loc[nyc_crash['contributing_factor_vehicle_1'].isin(facto

# Change to shortened names for better visability
confact.loc[:, 'contributing_factor_vehicle_1'] = confact[
                                                        'contributing_factor_
                                                        ].replace(factor1)
```

```
# Changes shape of point according to 'contributing_factor_vehicle_1'
(ggplot(confact, aes(x='longitude', y='latitude',
    shape ='contributing_factor_vehicle_1')) +
    geom_point(alpha = 0.7) +
    labs(title='Crash Locations by Top 5 Contributing Factors',
        x='Longitude',
        y='Latitude',
        shape = 'Contributing Factor',
        color= 'Contributing Factor') +
    coord_fixed(ratio = 1) +
    theme(figure_size = (7,5)))
```

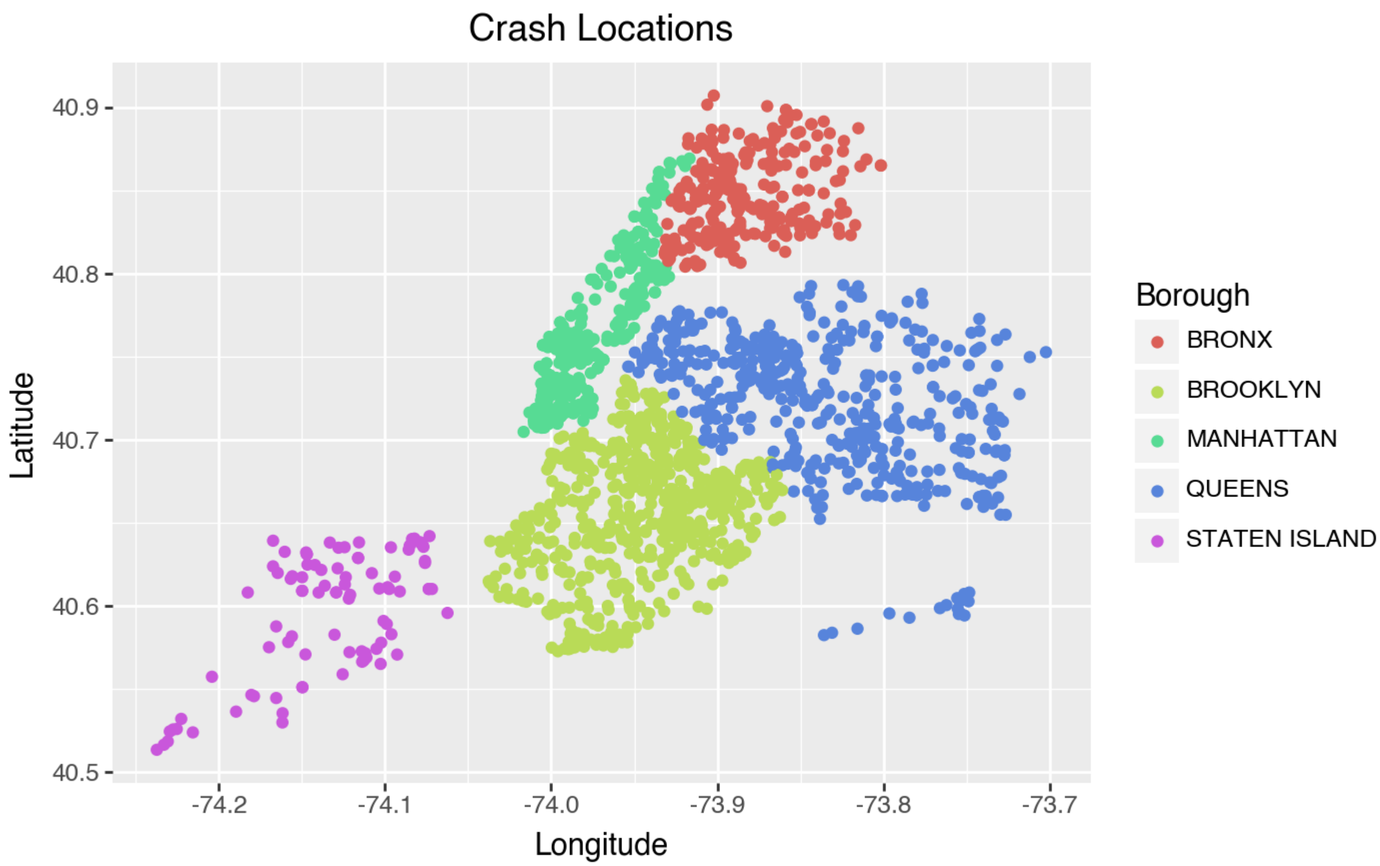



### Changing Color by Variables

To add color coordination to your plot in `Plotnine`, specify the variable you want to use for coloring by including `color='variable'` within the `aes()` function. This enables you to visually distinguish different categories in your dataset, enhancing the clarity and interpretability of your plot.

Changing color of point according to borough:

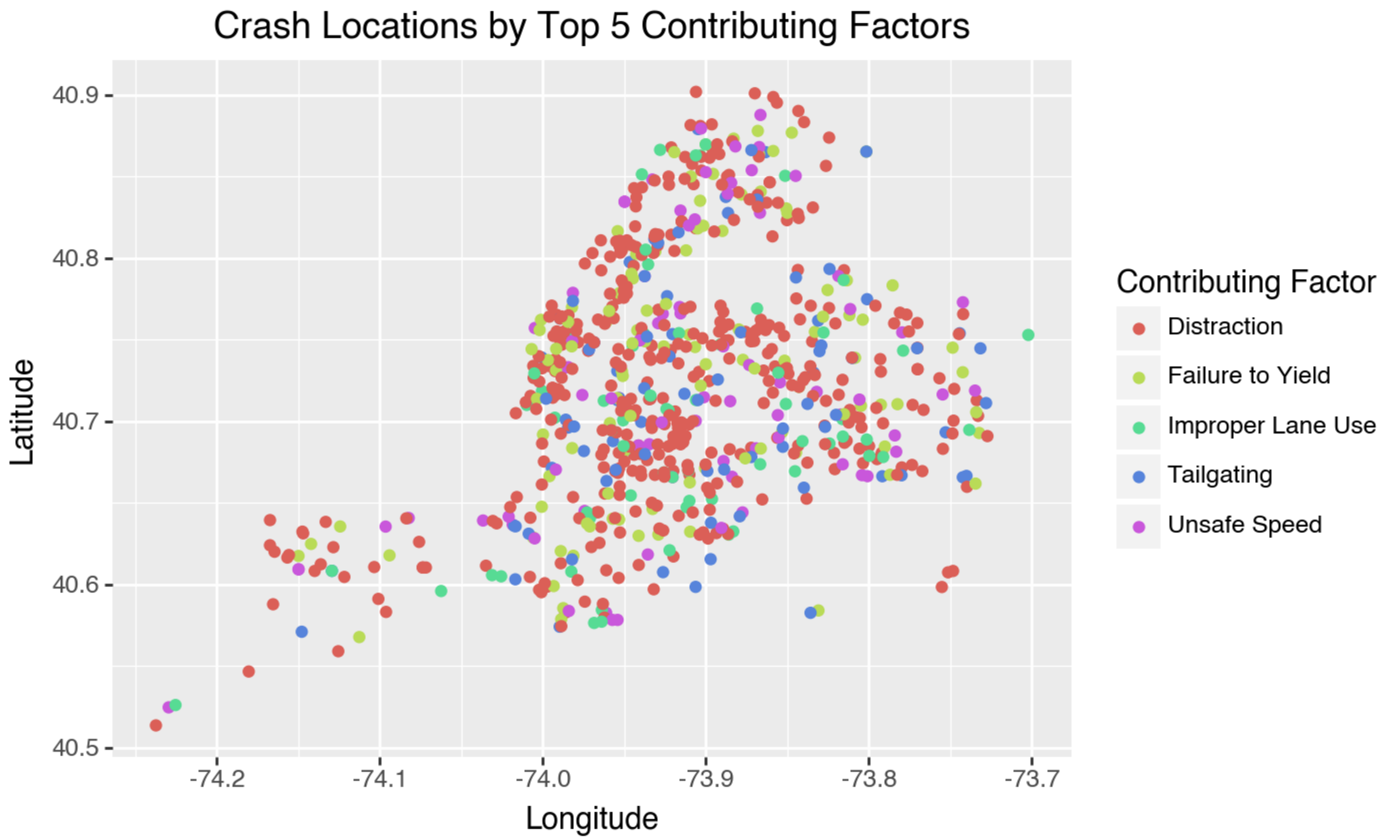
```
# color= changhes color according to 'borough'
(ggplot(nyc_crash, aes(x='longitude', y='latitude', color = 'borough')) +
    geom_point() +
    labs(title='Crash Locations',
        x='Longitude',
        y='Latitude',
# Changes key title to 'Borough'
        color= 'Borough') +
    coord_fixed(ratio = 1) +
    theme(figure_size = (7,5)))
```




As you can see, each borough is represented by its own color, allowing the audience to easily identify which borough the crash occurred in.

**Changing color of points by `contributing_factor_vehicle_1`:**

```
# color= changes color according to 'contributing_factor_vehicle_1'
(ggplot(confact, aes(x='longitude', y='latitude',
    color ='contributing_factor_vehicle_1')) +
    geom_point() +
    labs(title='Crash Locations by Top 5 Contributing Factors',
        x='Longitude',
        y='Latitude',
        color= 'Contributing Factor') +
        coord_fixed(ratio = 1) +
# Changes plot size to be larger
    theme(figure_size = (7,5)))
```




This graph uses color to distinguish what contributing factor caused the crash.

**Adding Linear Regression Line to Plot**

If you want to fit a linear regression line, use `geom_smooth()`. Adding this to your plot can be really helpful to visualize trends of your data easier. To add a linear regression line to your scatterplot, you would include the following line of code:

```
geom_smooth(method='lm', se=False, color='red')
```

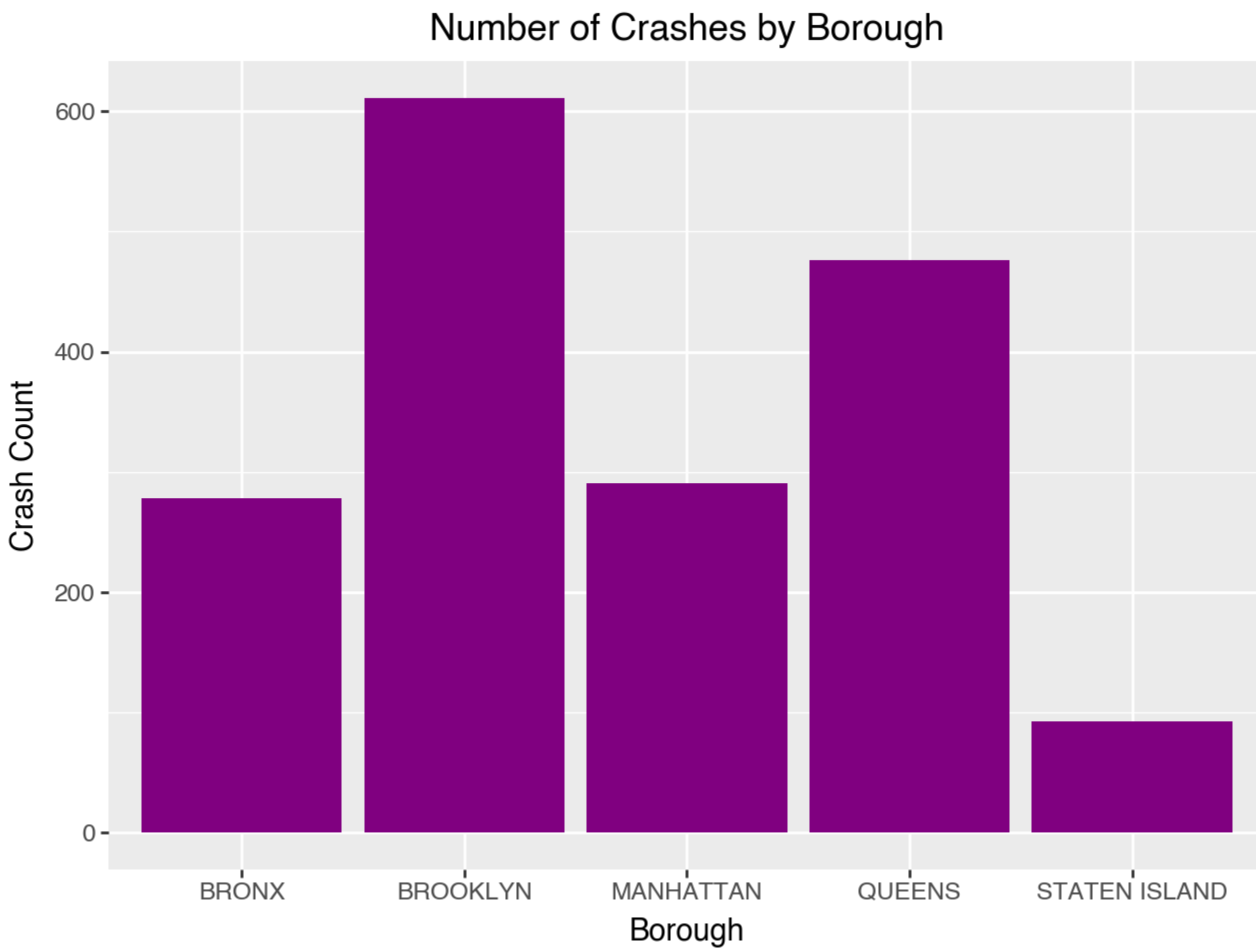
```
<plotnine.geoms.geom_smooth.geom_smooth at 0x11b2dd5e0>
```

### 7.1.5 Bar Chart

Another common use for displaying data is a bar chart. You can create one with `geom_bar()`. We will start with a simple chart of crashes by borough.

#### Creating a Basic Bar Chart

```
(ggplot(nyc_crash, aes(x='borough')) + # Use 'borough' for the x-axis
    geom_bar(fill='purple') +
    labs(title='Number of Crashes by Borough',
        x='Borough',
        y='Crash Count'))
```

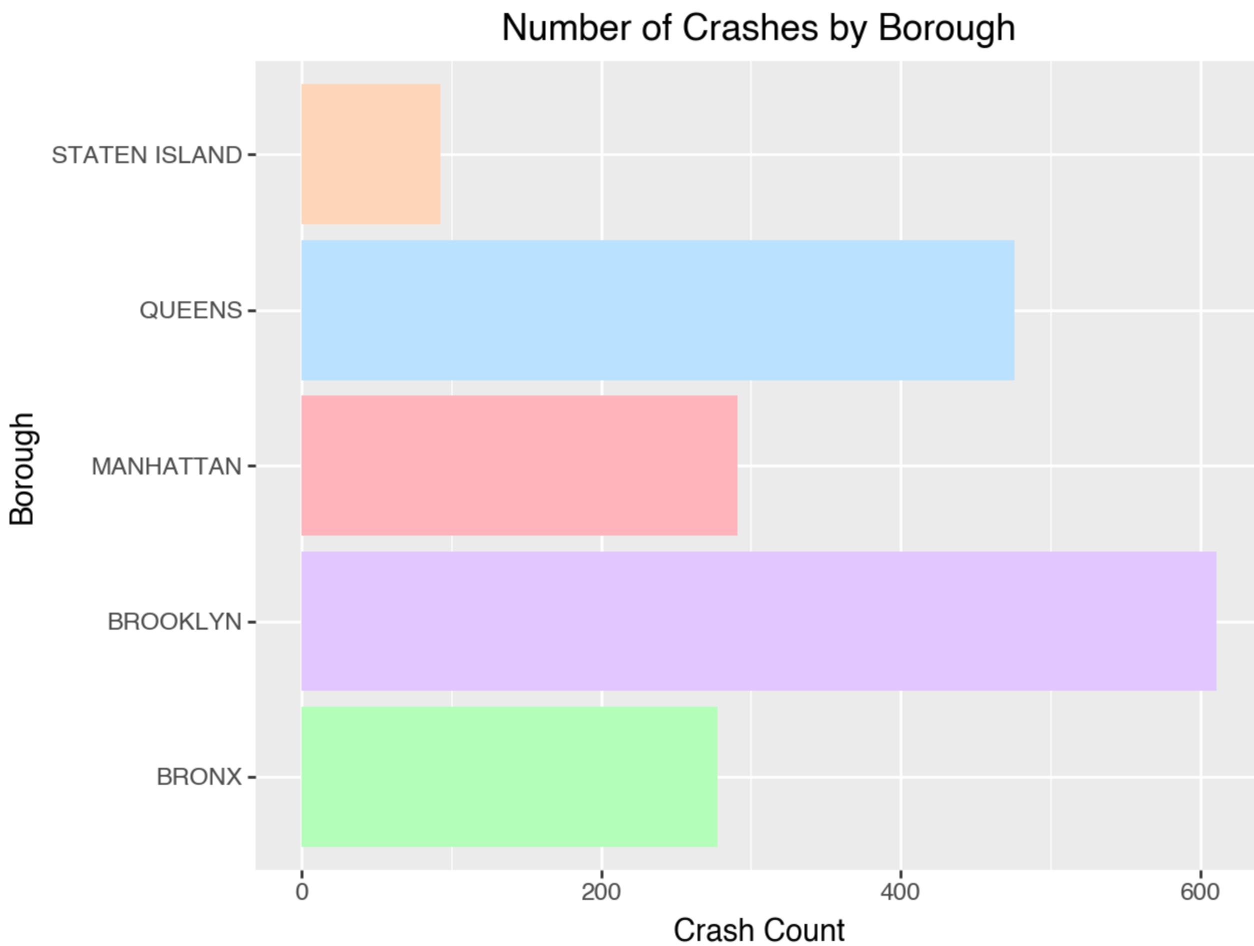



**Customizing your Bar Chart**

You can change up your bar chart a couple of different ways. You can handpick colors you want, designate it to variables, flip orientation, etc:

```
# Designate your preffered colors (pastel color codes)
colors = ['#B3FFBA', '#E1C6FF', '#FFB3BA', '#BAE1FF', '#FFD5BA']
```

```
# Adding fill= changes the color of bar according to variable
(ggplot(nyc_crash, aes(x='borough', fill = 'borough')) +
# Assigns your preffered colors
   geom_bar(fill = colors) +
# Flips orientation of the chart
   coord_flip() +
   labs(title='Number of Crashes by Borough',
        x='Borough',
        y='Crash Count'))
```

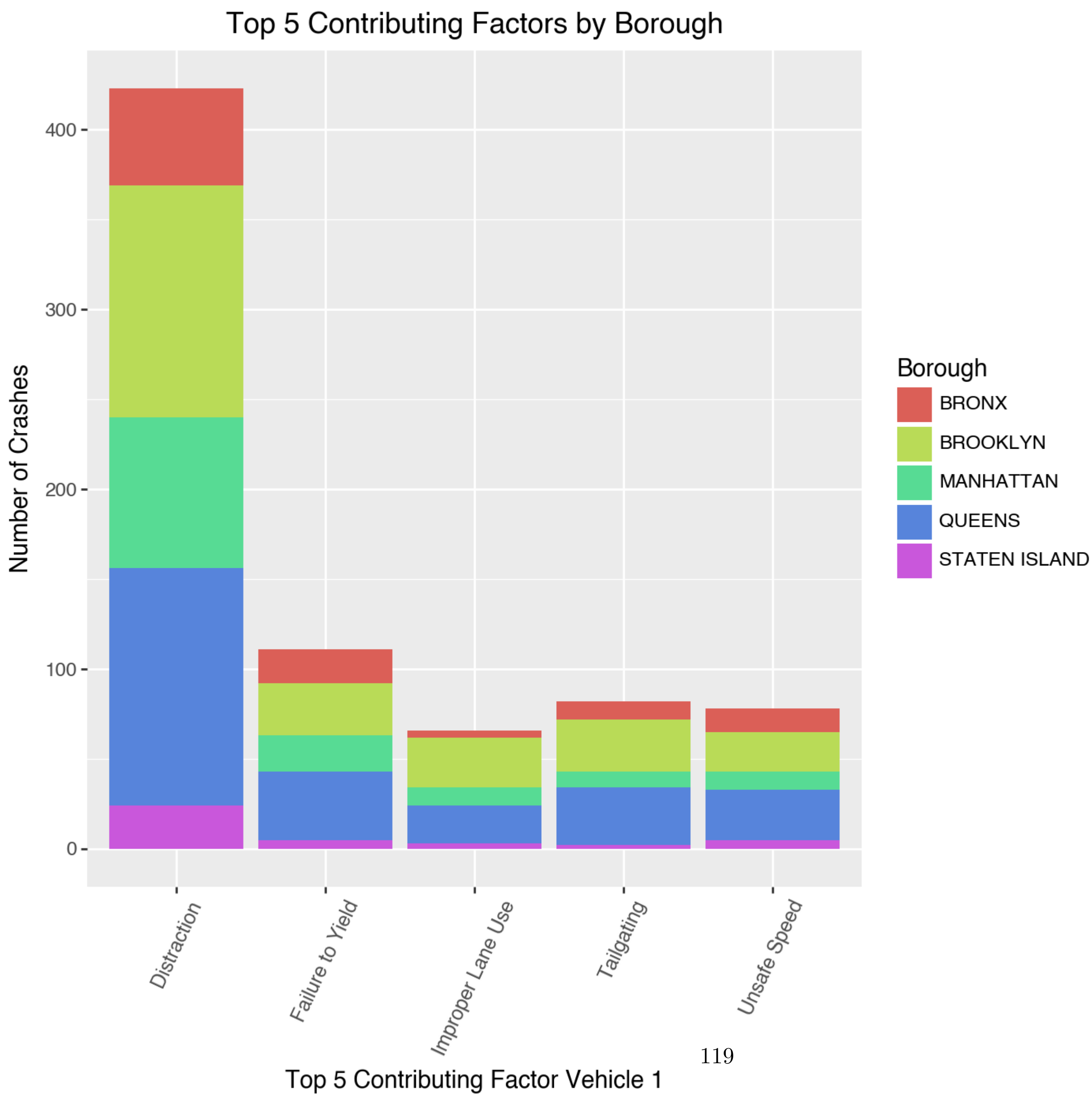



**Multivariable Bar Chart**

You can also split up a bar chart to make it visually easier to understand.

```
# Using 'confact' dataset again for better visualization
(ggplot(confact, aes(x='contributing_factor_vehicle_1', fill='borough')) +
    geom_bar() +
```

```
    labs(title='Top 5 Contributing Factors by Borough',
        x='Top 5 Contributing Factor Vehicle 1',
        y='Number of Crashes',
# Changes key name to "Borough"
        fill ='Borough') +
# size= creates smaller text
# angle= rotates x-axis text for readability
# figure_size= creates a larger image
    theme(axis_text_x=element_text(size=9, angle=65), figure_size= (7,7)))
```

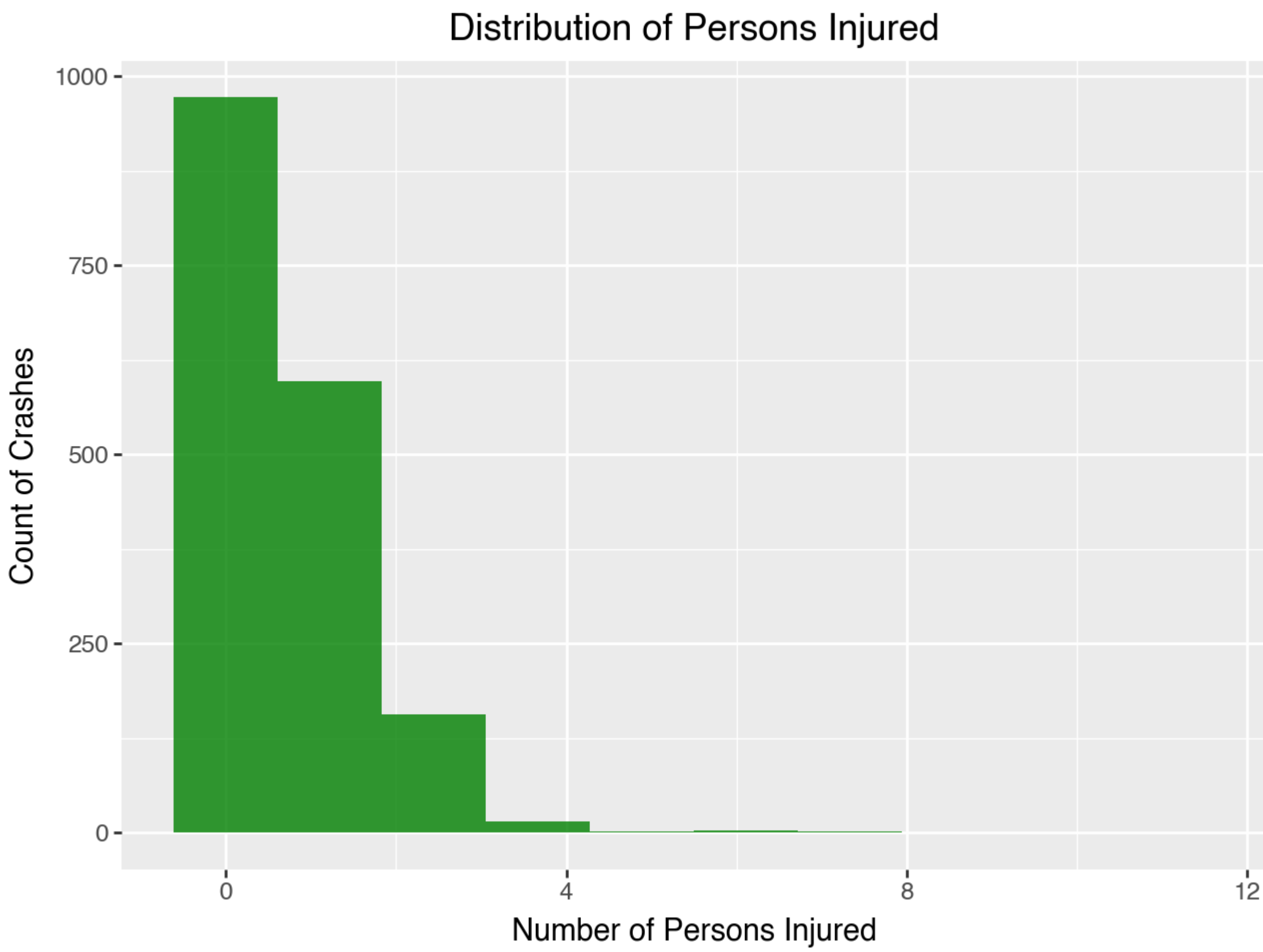

Top 5 Contributing Factors by Borough
Number of Crashes
0
100
200
300
400
Distraction
Failure to Yield
Improper Lane Use
Tailgating
Unsafe Speed
Top 5 Contributing Factor Vehicle 1
Borough
BRONX
BROOKLYN
MANHATTAN
QUEENS
STATEN ISLAND

### 7.1.6 Histogram

Another useful way to display data is a histogram. You can create one with `geom_hisogram()`. Using a histogram is very useful when displaying continuous data.

#### Basic Histogram

```
(ggplot(nyc_crash, aes(x='number_of_persons_injured')) +
# bins= sets the amount of bars in your histogram
    geom_histogram(bins=10, alpha=0.8, fill='green') +
    labs(title='Distribution of Persons Injured',
        x='Number of Persons Injured',
        y='Count of Crashes'))
```

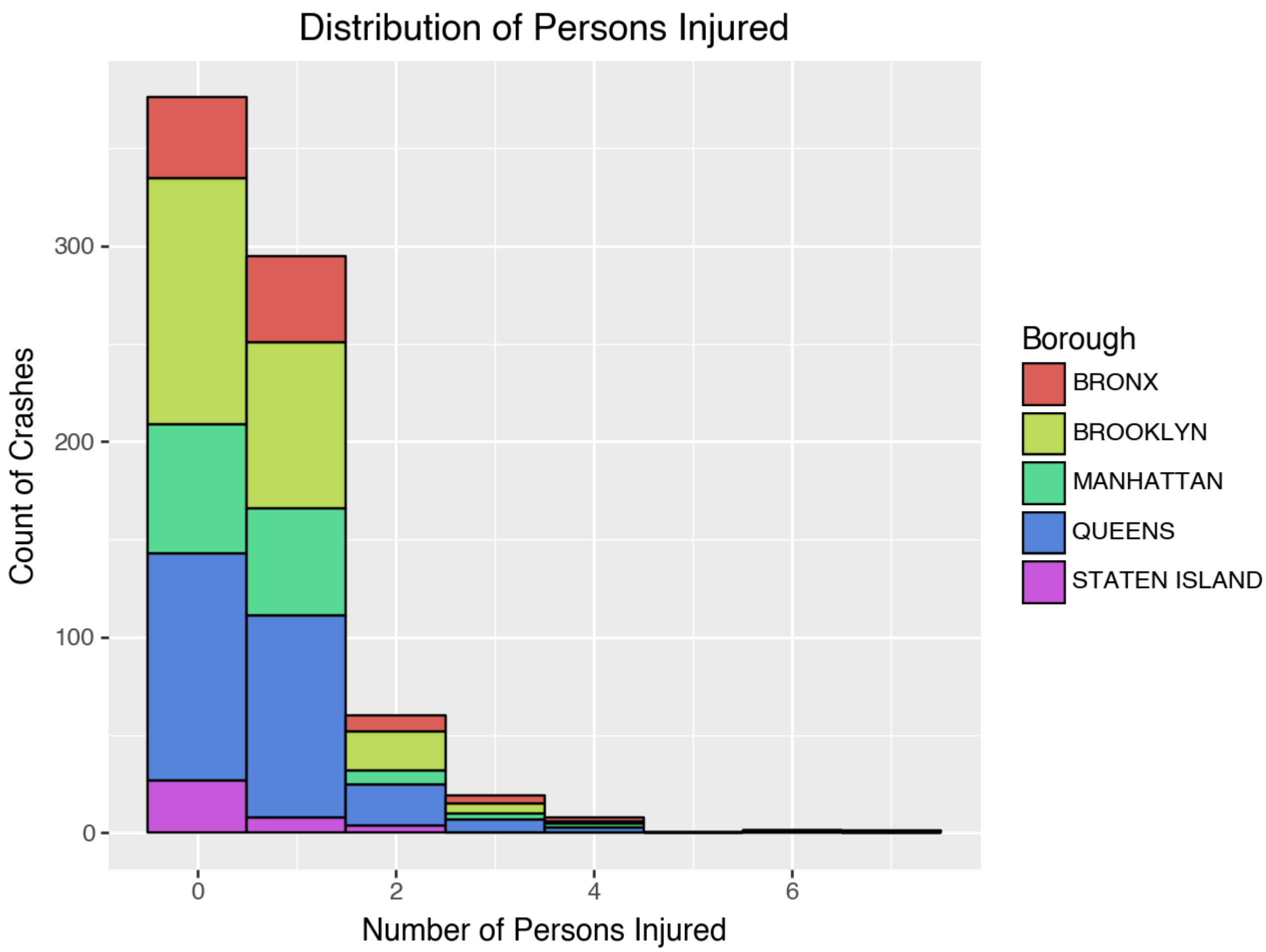



With a histogram it is very easy to understand trends for a dataset and you can see that our NYC crash data is positively skewed.

**Multivariable Histogram**

Similar to bar charts, you can make Histograms that display more than one variable.

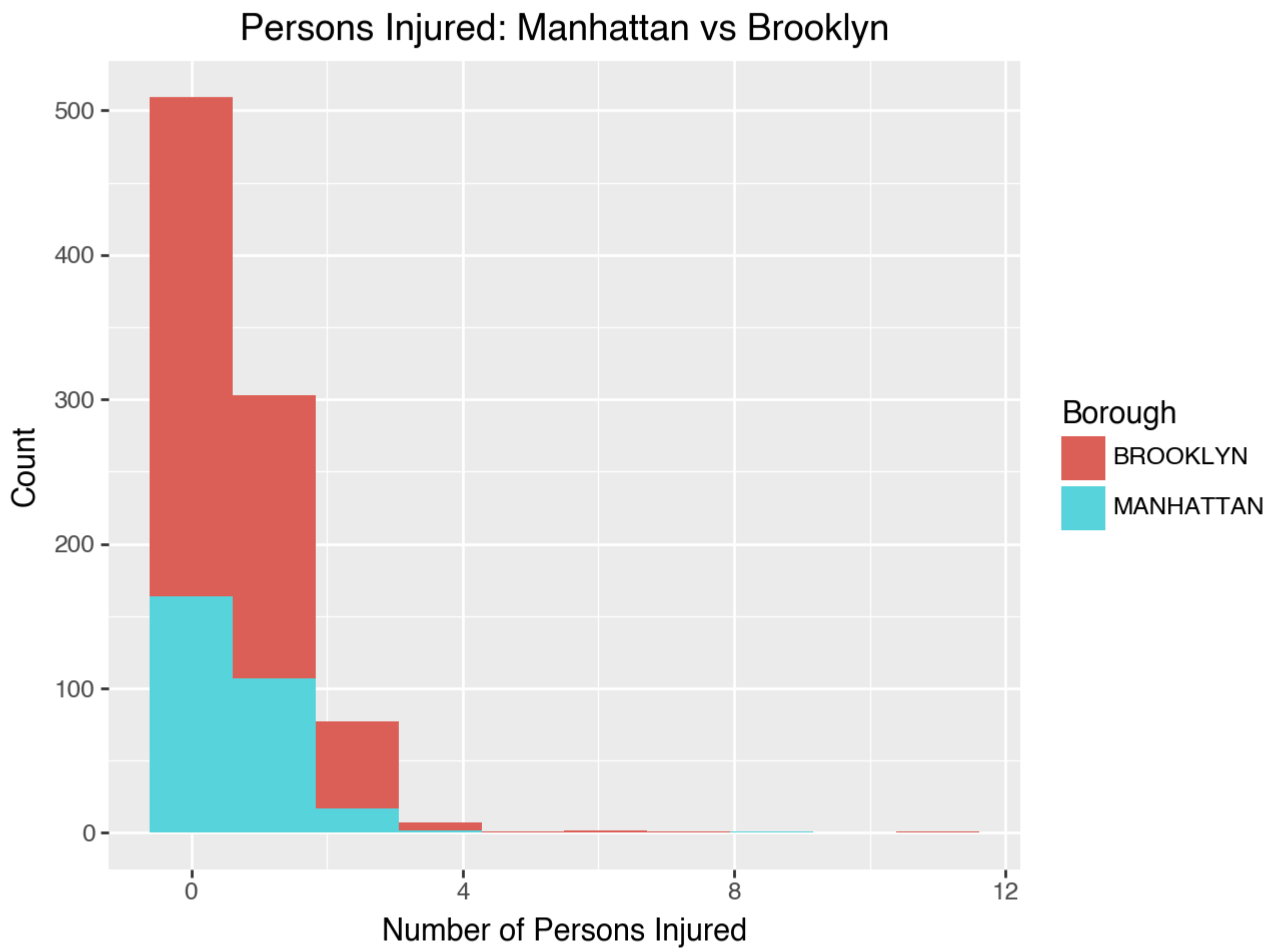
```
(ggplot(confact, aes(x='number_of_persons_injured', fill = 'borough')) +
# binwidth= changes width of your bars
# color= changes outline color for better visability
    geom_histogram(binwidth=1, color = 'black') +
    labs(title='Distribution of Persons Injured',
        x='Number of Persons Injured',
        y='Count of Crashes',
        fill = 'Borough'))
```




**Overlapping Histogram**

Histograms can also be useful when comparing multiple categories. Here we are comparing Manhattan and Brooklyn's number of persons injured with an overlapping histogram.

```
# Creating plot if crash is in 'MANHATTAN' or 'BROOKLYN'
(ggplot(nyc_crash[nyc_crash['borough'].isin(['MANHATTAN', 'BROOKLYN'])],
        aes(x='number_of_persons_injured', fill='borough')) +
    geom_histogram(bins=10) +
    labs(title='Persons Injured: Manhattan vs Brooklyn',
        x='Number of Persons Injured',
        y='Count',
        fill='Borough'))
```



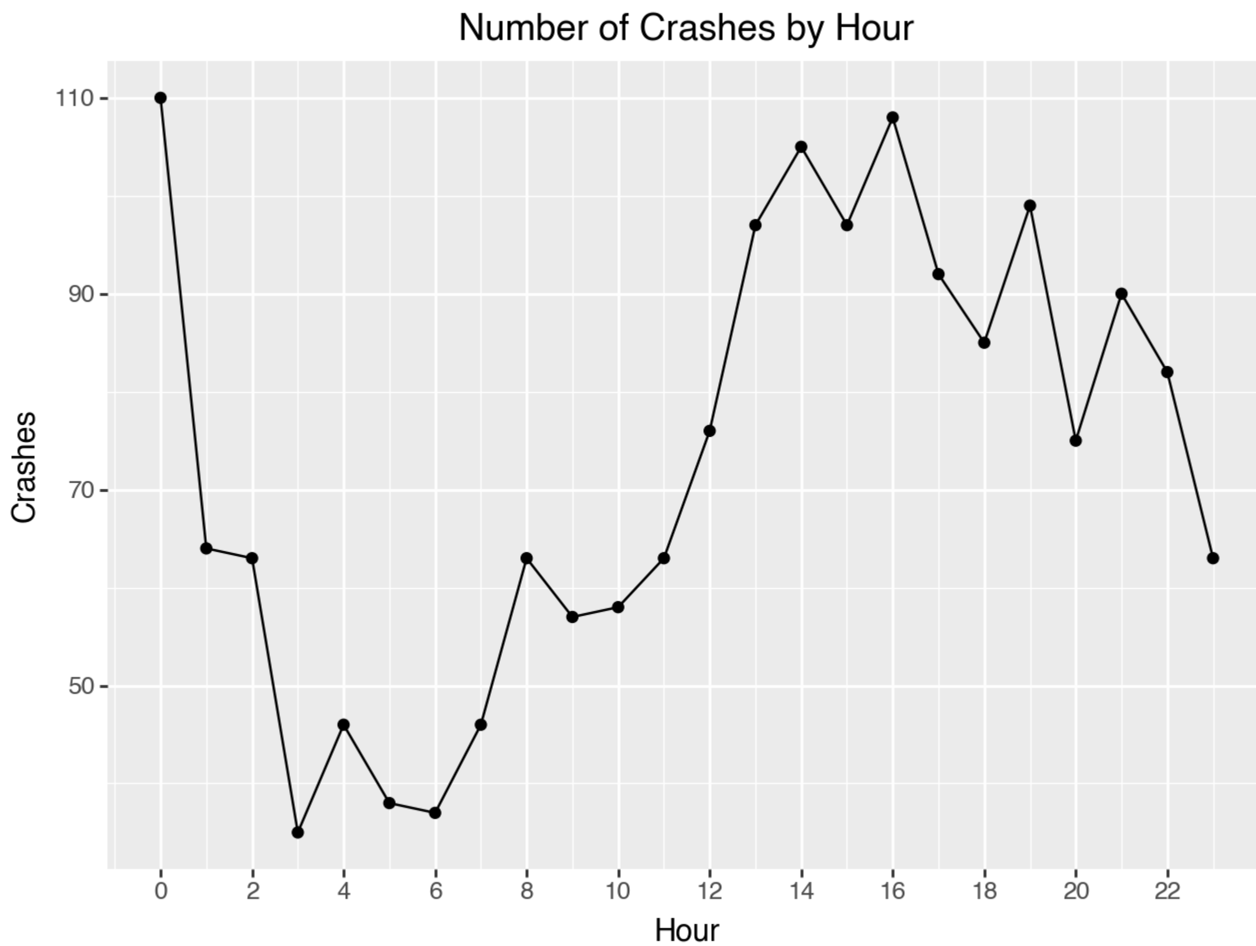

### 7.1.7 Line Chart

Line charts are great for time-series data and can be created with `geom_line()`. This type of chart is particularly useful for identifying patterns, fluctuations, and trends, making it easier to understand how a variable changes over a specified period. We will create one analyzing Number of Crashes by Hour.

### Basic Line Chart

```
# Finding crashes per hour
nyc_crash['crash_datetime'] = pd.to_datetime(nyc_crash['crash_datetime'])

# Extract hour
nyc_crash['crash_hour'] = nyc_crash['crash_datetime'].dt.hour

# Count crashes per hour
crash_counts = (nyc_crash.groupby(['crash_hour'])
                .size().reset_index(name='crash_count'))
```

```
# Plot crashes by hour
(ggplot(crash_counts, aes(x='crash_hour', y='crash_count')) +
# Creates the line chart
    geom_line() +
# Adds points for better visibility
    geom_point() +
    labs(title='Number of Crashes by Hour',
        x='Hour',
        y='Crashes') +
# Formats the x-axis to display ticks by every 2 hours
    scale_x_continuous(breaks=range(0, 24, 2)))
```

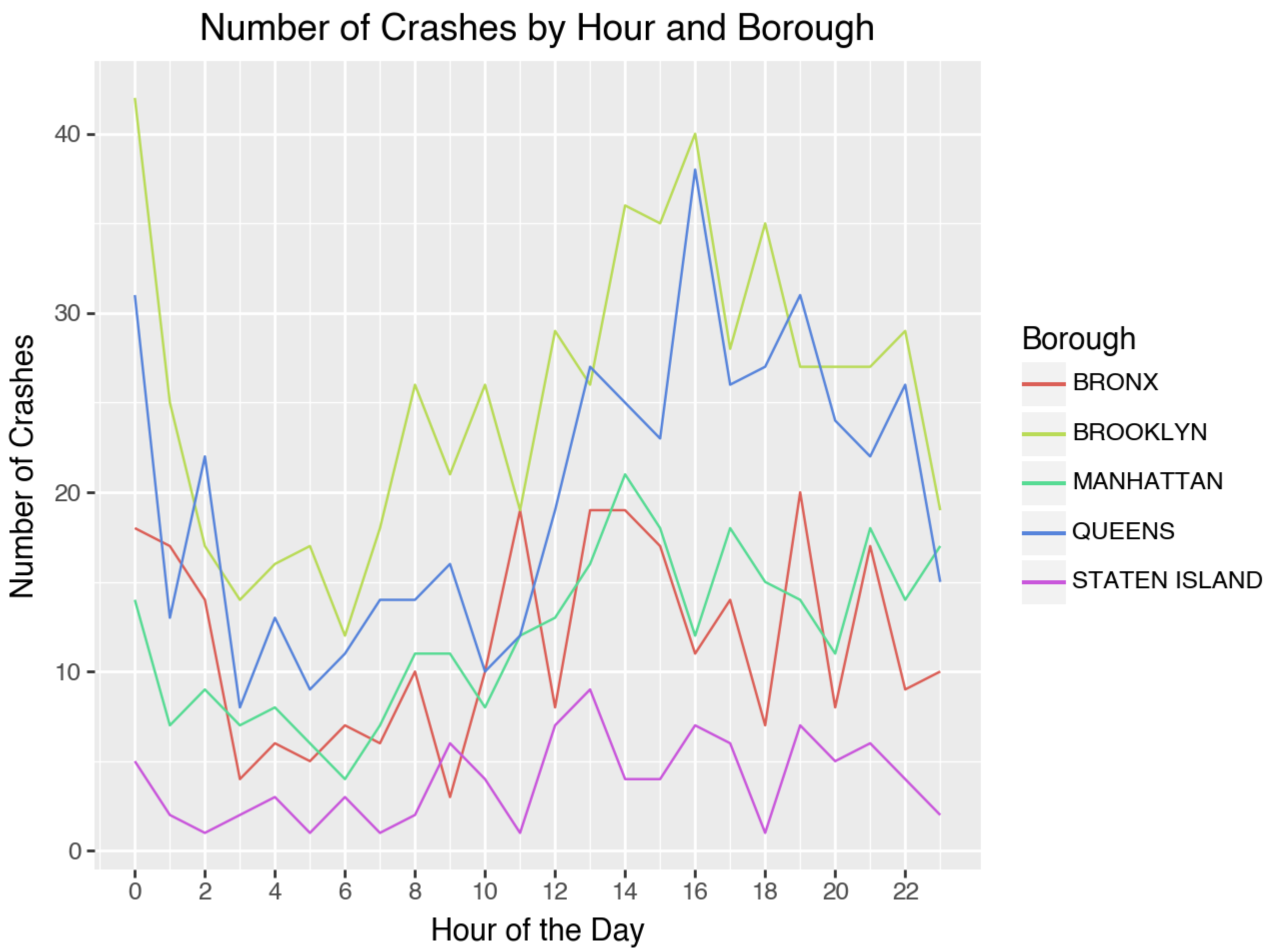



This example is excellent for understanding the grammar of graphics. As you can see, we use `geom_line()` to create the line chart, while also adding `geom_point()`, which is typically used for scatterplots, to make the figure clearer by layering additional details."

**Multivariable Line Chart**

Similarly to the other figures you can create a line chart with multiple variables. Now we will create a chart with number of crashes by borough.

```
# Setting crash counts to also include borough
crash_counts = nyc_crash.groupby(['crash_hour',
    'borough']).size().reset_index(name='crash_count')

# Plots crashes by hour with different lines for each borough
(ggplot(crash_counts, aes(x='crash_hour', y='crash_count',
    color='borough')) +
# size= changes the thinkness of the lines
    geom_line(size=0.5) +
    labs(title='Number of Crashes by Hour and Borough',
        x='Hour of the Day',
        y='Number of Crashes',
        color = 'Borough') +
    scale_x_continuous(breaks=range(0, 24, 2)))
```



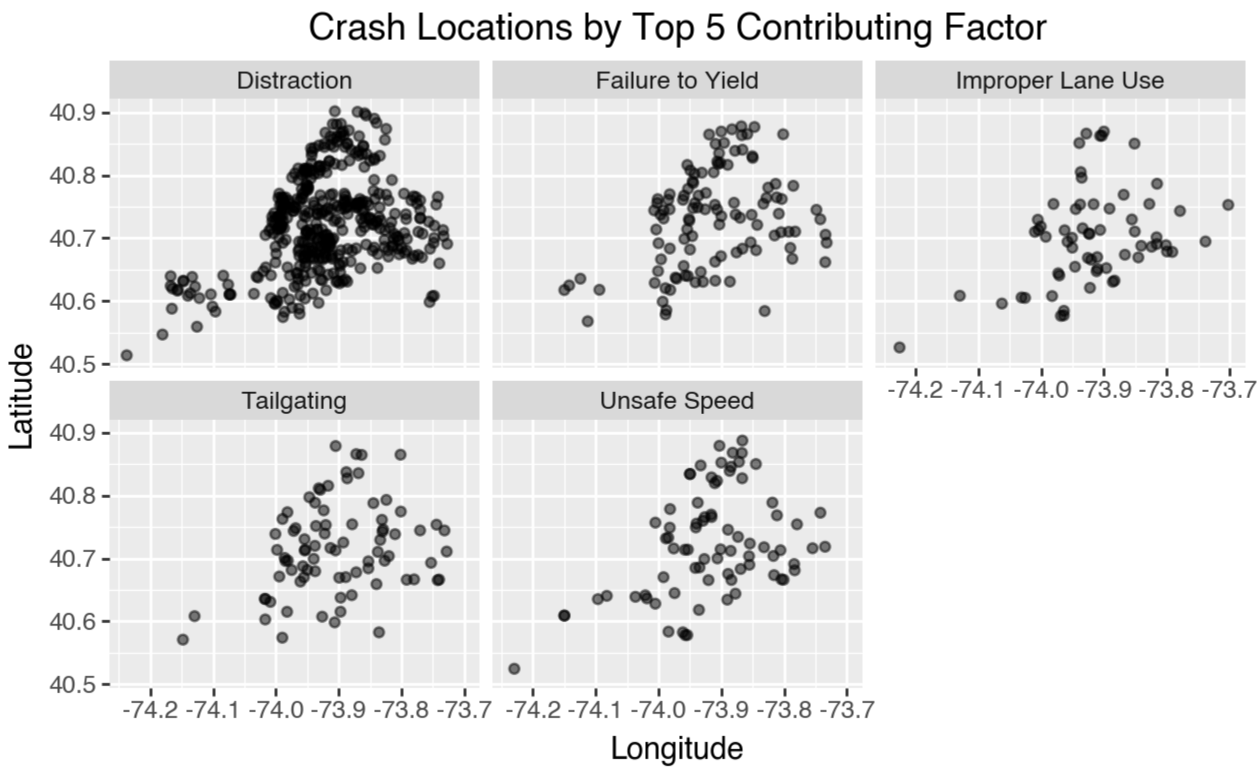

### 7.1.8 Faceting Your Plots

To organize your data in a way that enhances interpretability, you can utilize `facet_grid()` or `facet_wrap()`. This approach allows for the creation of separate plots based on categorical variables, making it easier to identify trends and patterns. You can facet any type of plots, scatterplots, bar charts, histograms, line charts, etc. using one or two variables.

**Scatterplots per Facet**

Scatterplot of Crash Locations by Contributing Factor with `facet_wrap()`:

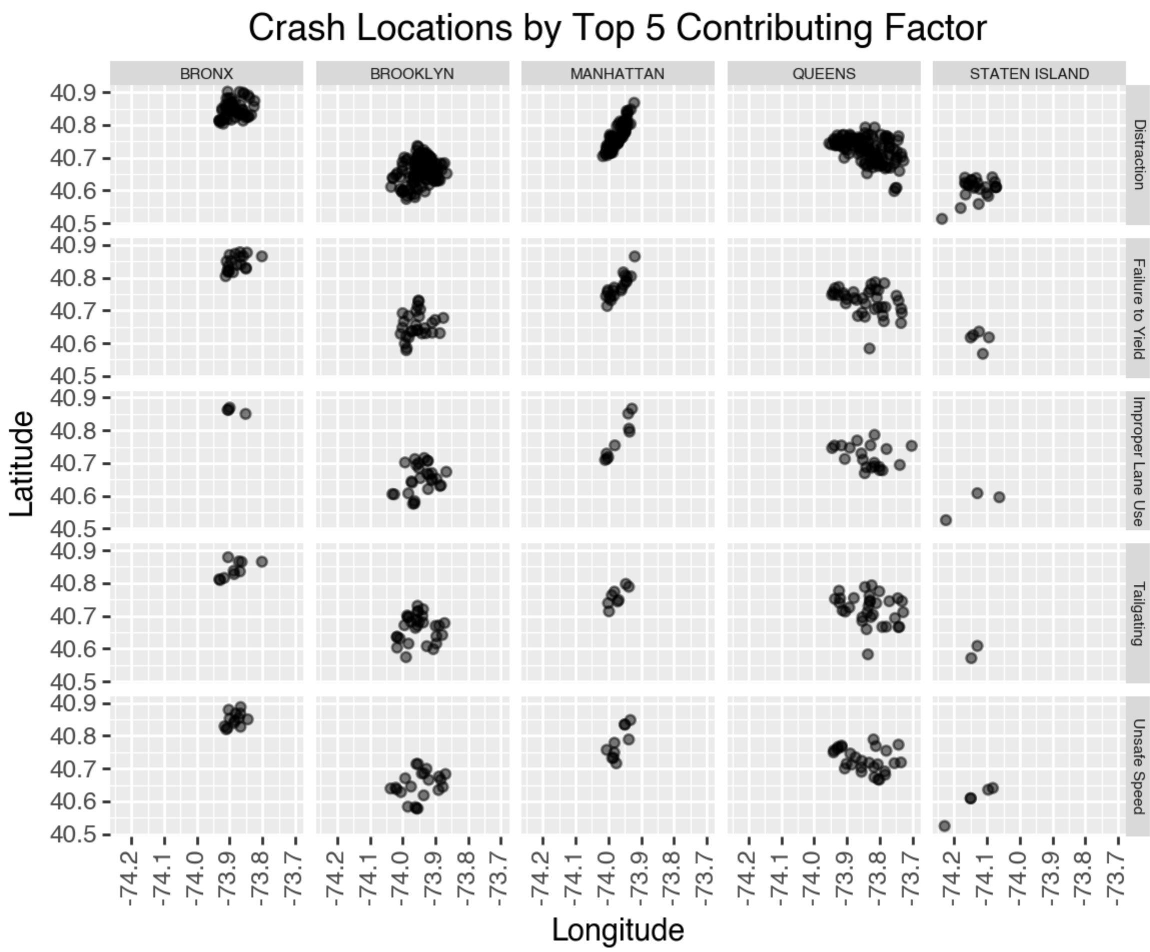
```
(ggplot(confact, aes(x='longitude', y='latitude')) +
    geom_point(alpha=0.5) +
# Creates separate plots for each contributing factor
    facet_wrap('contributing_factor_vehicle_1') +
    labs(title='Crash Locations by Top 5 Contributing Factor',
        x='Longitude',
        y='Latitude') +
    coord_fixed(ratio = 1))
```




Scatterplot of Two Variables, Crash Locations Contributing Factor and Borough with `facet_grid()`:

```
(ggplot(confact, aes(x='longitude', y='latitude')) +
    geom_point(alpha = 0.5) +
# Creates a grid of subplots based on the values of two variables
```

```
# ~'contributing_factor_vehicle_1' by 'borough'
    facet_grid('contributing_factor_vehicle_1 ~ borough') +
    labs(title='Crash Locations by Top 5 Contributing Factor',
        x='Longitude',
        y='Latitude') +
# Changes angle of text and size of the graphic
    theme(axis_text_x=element_text(angle=90),
# sprip_text=element_text changes text size of the facet titles
        strip_text=element_text(size=5.5)) +
    coord_fixed(ratio = 1))
```

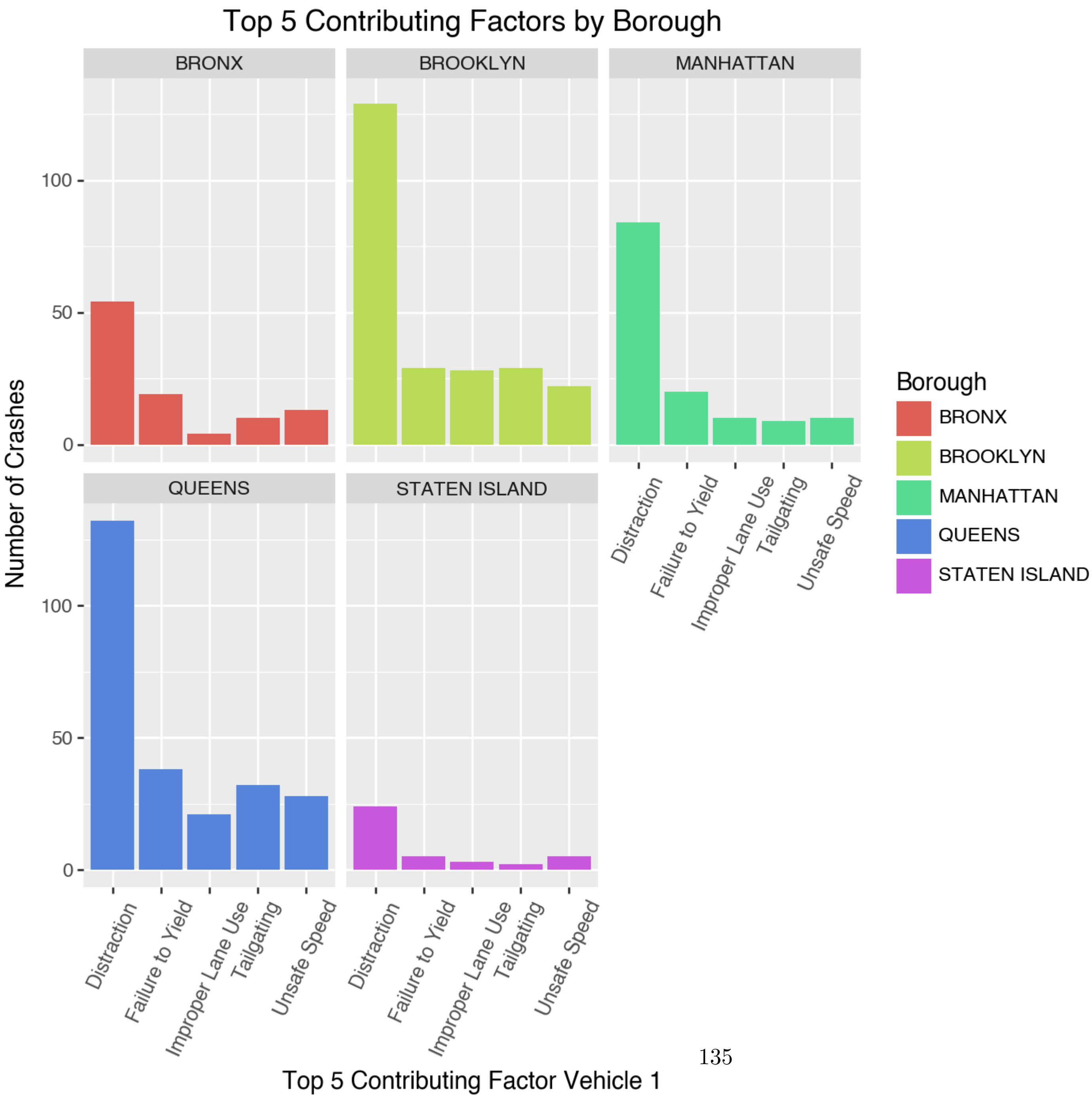



**Bar Chart per Facet**

Bar chart of Contributing Factors by Borough with `facet_wrap`:

```
(ggplot(confact, aes(x='contributing_factor_vehicle_1', fill='borough')) +
    geom_bar() +
    labs(title='Top 5 Contributing Factors by Borough',
```

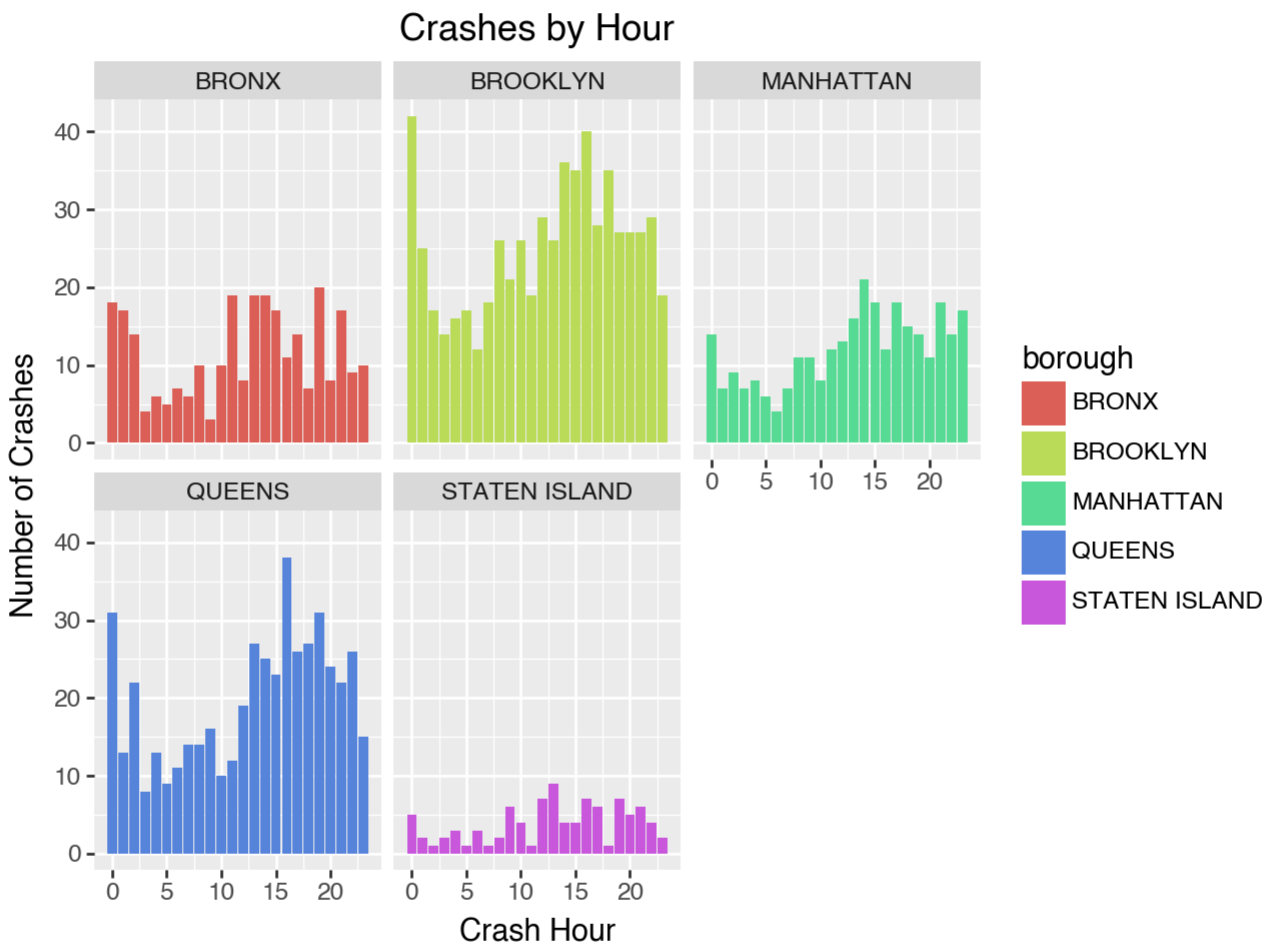
```
        x='Top 5 Contributing Factor Vehicle 1',
        y='Number of Crashes',
        fill = 'Borough') +
    facet_wrap('~ borough') +
    theme(axis_text_x=element_text(size=9, angle=65), figure_size= (7,7)))
```


Top 5 Contributing Factors by Borough
BRONX
BROOKLYN
MANHATTAN
QUEENS
STATEN ISLAND
Number of Crashes
0
50
100
Distraction
Failure to Yield
Improper Lane Use
Tailgating
Unsafe Speed
Top 5 Contributing Factor Vehicle 1
Borough
BRONX
BROOKLYN
MANHATTAN
QUEENS
STATEN ISLAND

#### Histograms per Facet

Histogram of Crashes per Hour by Borough with `facet_wrap`:

```
(ggplot(crash_counts, aes(x='crash_hour', y='crash_count', fill = 'borough'))
    geom_bar(stat='identity') +
    labs(x='Crash Hour', y='Number of Crashes', title = "Crashes by Hour") +
    facet_wrap('~ borough'))
```

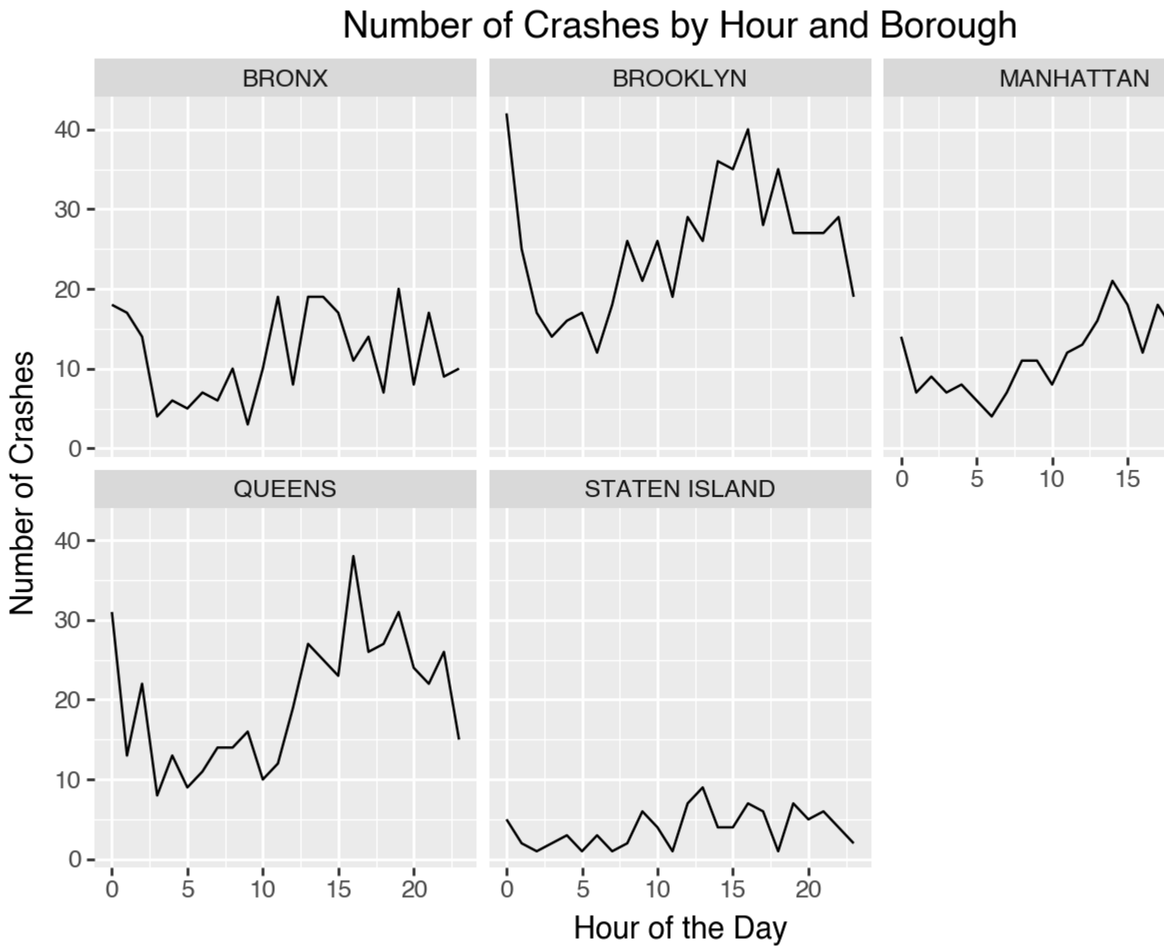



### Line Chart per Facet

You can use plot each variable by on separate panels with `facet_wrap()`.

```
(ggplot(crash_counts, aes(x='crash_hour', y='crash_count')) +
    geom_line() +
    # Breaks the figure up by borough
```

```
    facet_wrap("borough") +
    labs(title='Number of Crashes by Hour and Borough',
        x='Hour of the Day',
        y='Number of Crashes'))
```



### 7.1.9 Conclusion

`Plotnine` is a very powerful tool to make impactful and detailed graphics. The flexibility of its grammar of graphics approach means there are endless ways to modify, enhance, and be creative with your plots. You can layer geoms, adjust aesthetics, and apply scales, facets, and themes.

**Creating Specific Plots**

- Scatterplot `geom_point()`
- Boxplot `geom_box()`
- Histogram `geom_histogram()`
- Line Chart `geom_line()`
- Bar Chart `geom_bar()`
- Density Plot `geom_denisty()`

**Formatting and Customizing Your Figure:**

- `fill`: to change the color of the data
- `color`: to change the color of the borders
- `alpha`: to change the transparency
- `bins`: to change the number of bins
- `figure_size`: to change size of graphic
- `geom_smooth`: to add a smoothed line
- `facet`: plot each group on a separate panel
  - `facet_wrap()`: creates a series of plots arranged in a grid, wrapping into new rows or columns as needed
  - `facet_grid()`: allows you to create a grid layout based on two categorical variables, organizing plots in a matrix format
- `theme`: change overall theme

There are many other features and customizations you can do with Plotnine. For more information on how to leverage the full potential of this package for your data visualization needs check out Plotnine's Graph Gallery.

*Happy plotting!*

**Further Readings**

Python Graph Gallery. (2024). Plotnine: ggplot in python. Python Graph Gallery. https://python-graph-gallery.com/plotnine/

Sarker, D. (2018). A comprehensive guide to the grammar of graphics for effective visualization of multi-dimensional data. Towards Data Science. https://towardsdatascience.com/a-comprehensive-guide-to-the-grammar-of-graphics-for-effective-visualization-of-multi-dimensional-1f92b4ed4149

## 7.2 Spatial Data Visualization with Google Map

### 7.2.1 Subsection

# 8 Statistical Tests and Models

## 8.1 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, 5I
p-value = 0.02999
alternative hypothesis: two.sided
```

## 8.2 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`).

### 8.2.1 Installation of `statsmodels`

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

Using pip:

```
pip install statsmodels
```

### 8.2.2 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:** | Wed, 23 Oct 2024 | **Prob (F-statistic):** | 3.31e-14 |
| **Time:** | 10:01:10 | **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

```python
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.

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

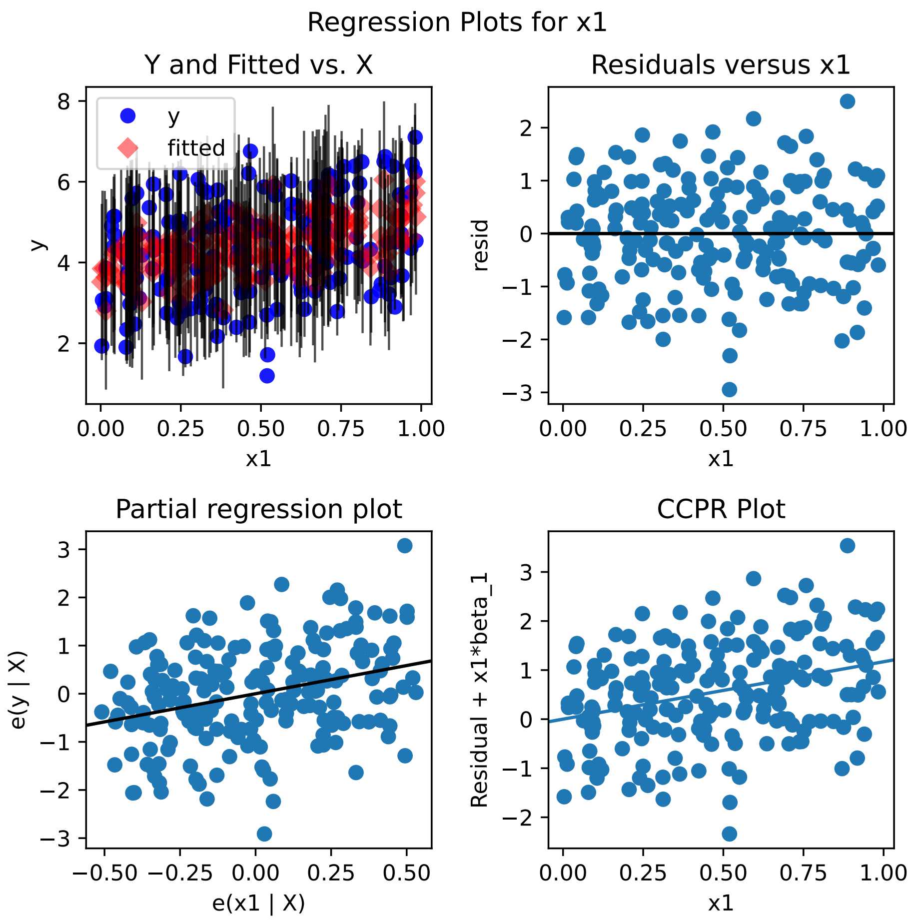
| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 200 |
| **Model:** | RLM | **Df Residuals:** | 194 |
| **Method:** | IRLS | **Df Model:** | 5 |
| **Norm:** | HuberT | | |
| **Scale Est.:** | mad | | |
| **Cov Type:** | H1 | | |
| **Date:** | Wed, 23 Oct 2024 | | |
| **Time:** | 10:01:10 | | |
| **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.

### 8.2.3 Generalized Linear Regression

A linear regression model cannot be applied to presence/absence or count data. Generalized Linear Models (GLM) extend the classical linear regression 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 $\beta$ 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 ($g(\mu) = \mu$) 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.

A logistic regression can be fit with `statsmodels.api.glm`.

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, 2, 2]) - 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'
```

```python
# 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:                -159.1
Date:                Wed, 23 Oct 2024   Deviance:                       318.3
Time:                        10:01:13   Pearson chi2:                 1.47e+0
No. Iterations:                     8   Pseudo R-squ. (CS):             0.419
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975
------------------------------------------------------------------------------
Intercept     -5.0186      0.392    -12.796      0.000      -5.787      -4.25
x1             1.9990      0.211      9.471      0.000       1.585       2.41
x2             2.1058      0.214      9.853      0.000       1.687       2.52
x3             1.9421      0.210      9.260      0.000       1.531       2.35
x4             2.1504      0.232      9.260      0.000       1.695       2.60
x5             2.0603      0.221      9.313      0.000       1.627       2.49
==============================================================================
```

## 8.3 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.

### 8.3.1 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) |

https://en.wikipedia.org/wiki/Confusion_matrix

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.

### 8.3.2 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.

### 8.3.3 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.

### 8.3.4 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.

### 8.3.5 F-beta Score

The F-beta score is a weighted harmonic mean of precision and recall, taking into account a $\beta$ parameter such that recall is considered $\beta$ 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 $\beta^2$ instead of just $\beta$.

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).

### 8.3.6 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:

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

### 8.3.7 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}")
```

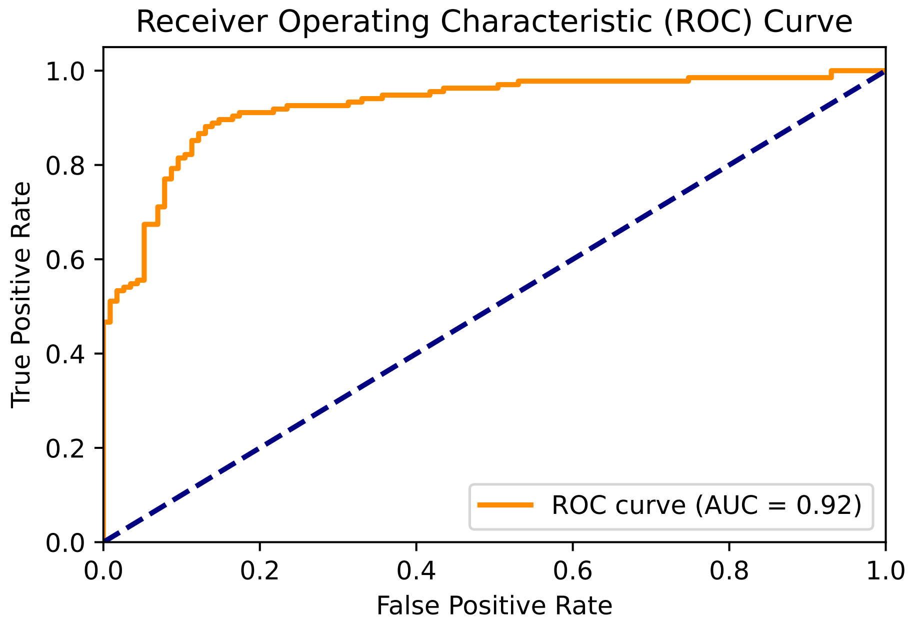
```
Confusion Matrix:
 [[104  11]
 [ 26 109]]
Accuracy: 0.85
Precision: 0.91
Recall: 0.81
```

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
```

## 8.4 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.

### 8.4.1 Theoretical Formulation of the Problem

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

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

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.
- $\beta$ is the vector of model coefficients (including the intercept).
- $\lambda$ is the regularization parameter that controls the trade-off between model fit and sparsity (higher $\lambda$) 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:

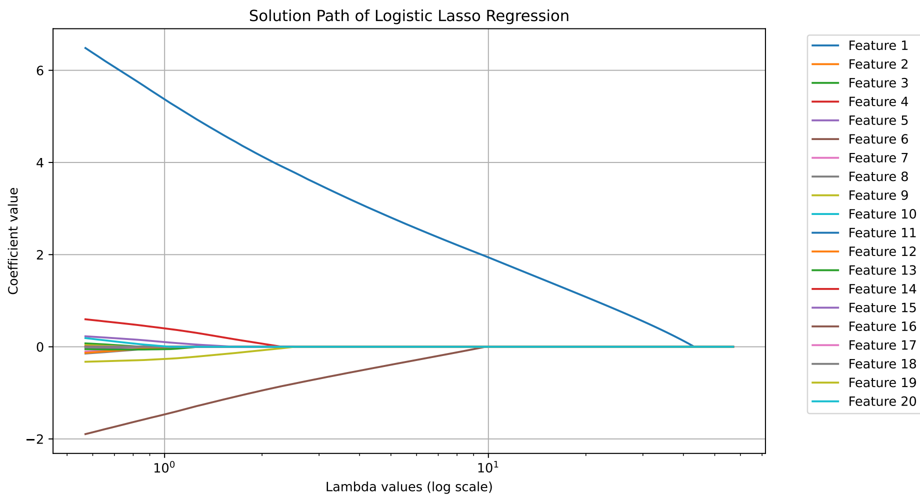
- Dimensionality Reduction: LASSO is particularly useful when the number of features $p$ is large, potentially even larger than the number of observations $n$, as it automatically reduces the number of features.
- Preventing Overfitting: The L1 penalty helps prevent overfitting by constraining the model, 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.

### 8.4.2 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 $\lambda$. This demonstration will use a simulated dataset to show how increasing $\lambda$ leads to more coefficients being set to zero.

```python
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 lam
for lam in lambda_values:
    model = LogisticRegression(penalty='l1', solver='liblinear', C=1/lam, max
    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()
```




### 8.4.3 Selection the Tuning Parameter

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

# 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.        ]]
```

### 8.4.4 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.

#### 8.4.4.1 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`.

#### 8.4.4.2 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.

#### 8.4.4.3 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)  # Ordi
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`).

# 9 Machine Learning: Overview

## 9.1 Introduction

Machine Learning (ML) is a branch of artificial intelligence that enables systems to learn from data and improve their performance over time without being explicitly programmed. At its core, machine learning algorithms aim to identify patterns in data and use those patterns to make decisions or predictions.

Machine learning can be categorized into three main types: supervised learning, unsupervised learning, and reinforcement learning. Each type differs in the data it uses and the learning tasks it performs, addressing addresses different tasks and problems. Supervised learning aims to predict outcomes based on labeled data, unsupervised learning focuses on discovering hidden patterns within the data, and reinforcement learning centers around learning optimal actions through interaction with an environment.

Let's define some notations to introduce them:

- $X$: A set of feature vectors representing the input data. Each element $X_i$ corresponds to a set of features or attributes that describe an instance of data.
- $Y$: A set of labels or rewards associated with outcomes. In supervised learning, $Y$ is used to evaluate the correctness of the model's predictions. In reinforcement learning, $Y$ represents the rewards that guide the learning process.

- $A$: A set of possible actions in a given context. In reinforcement learning, actions $A$ represent choices that can be made in response to a given situation, with the goal of maximizing a reward.

### 9.1.1 Supervised Learning

Supervised learning is the most widely used type of machine learning. In supervised learning, we have both feature vectors $X$ and their corresponding labels $Y$. The objective is to train a model that can predict $Y$ based on $X$. This model is trained on labeled examples, where the correct outcome is known, and it adjusts its internal parameters to minimize the error in its predictions, which occurs as part of the cross-validation process.

Key tasks in supervised learning include:

- Classification: Assigning data points to predefined categories or classes.
- Regression: Predicting a continuous value based on input data.

In supervised learning, the data consists of both feature vectors $X$ and labels $Y$, namely, $(X, Y)$.

### 9.1.2 Unsupervised Learning

Unsupervised learning involves learning patterns from data without any associated labels or outcomes. The objective is to explore and identify hidden structures in the feature vectors $X$. Since there are no ground-truth labels $Y$ to guide the learning process, the algorithm must discover patterns on its own. This is particularly useful when subject matter experts are unsure of common properties within a data set.

Common tasks in unsupervised learning include:

- Clustering: Grouping similar data points together based on certain features.
- Dimension Reduction: Simplifying the input data by reducing the number of features while preserving essential patterns.

In unsupervised learning, the data consists solely of feature vectors $X$.

### 9.1.3 Reinforcement Learning

Reinforcement learning involves learning how to make a sequence of decisions to maximize a cumulative reward. Unlike supervised learning, where the model learns from a static dataset of labeled examples, reinforcement learning involves an agent that interacts with an environment by taking actions $A$, receiving feedback in the form of rewards $Y$, and learning over time which actions lead to the highest cumulative reward.

The process in reinforcement learning involves:

- States: The context or environment the agent is in, represented by feature vectors $X$.
- Actions: The set of possible choices the agent can make in response to the current state, denoted as $A$.
- Rewards: Feedback the agent receives after taking an action, which guides the learning process.

In reinforcement learning, the data consists of feature vectors $X$, actions $A$, and rewards $Y$, namely, $(X, A, Y)$.

## 9.2 Bias-Variance Tradeoff

The variance-bias trade-off is a core concept in machine learning that explains the relationship between the complexity of a model, its performance on training data, and its ability to generalize to unseen data. It applies to both supervised and unsupervised learning, though it manifests differently in each.

### 9.2.1 Bias

Bias refers to the error introduced by approximating a complex real-world problem with a simplified model. A model with high bias makes strong assumptions about the data, leading to oversimplified patterns and poor performance on both the training data and new data. High bias results in underfitting, where the model fails to capture important trends in the data.

- Example (Supervised): In supervised learning, using a linear regression model to fit data that has a nonlinear relationship results in high bias because the model cannot capture the complexity of the data.
- Example (Unsupervised): In clustering (an unsupervised task), setting the number of clusters too low (e.g., forcing data into two clusters when more exist) leads to high bias, as the model oversimplifies the underlying structure.

### 9.2.2 Variance

Variance refers to the model's sensitivity to small changes in the training data. A model with high variance will adapt closely to the training data, potentially capturing noise or fluctuations that are not representative of the general data distribution. High variance leads to overfitting, where the model performs well on training data but poorly on new, unseen data.

- Example (Supervised): A decision tree with many branches can exhibit high variance. The model perfectly fits the training data but may perform poorly on test data because it overfits to specific idiosyncrasies in the training set.
- Example (Unsupervised): In clustering, setting the number of clusters too high or fitting overly flexible cluster shapes (e.g., in Gaussian Mixture Models) can lead to overfitting, where the model captures noise and splits data unnecessarily.

### 9.2.3 The Trade-Off

The bias-variance trade-off reflects the tension between bias and variance. As model complexity increases:

- Bias decreases: The model becomes more flexible and can capture more details of the data.
- Variance increases: The model becomes more sensitive to the particular training data, potentially capturing noise.

Conversely, a simpler model will:

- Have high bias: It may not capture all relevant patterns in the data.
- Have low variance: It will be less sensitive to fluctuations in the data and is more likely to generalize well to unseen data.

### 9.2.4 Bias-Variance in Supervised Learning

In supervised learning, the goal is to strike the right balance between bias and variance to minimize prediction error. This balance is critical for developing models that generalize well to new data. The total error of a model can be decomposed into:

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}.$$

- Bias: The error from using a model that is too simple.
- Variance: The error from using a model that is too complex and overfits the training data.
- Irreducible Error: This is the noise inherent in the data itself, which cannot be eliminated no matter how well the model is tuned.

### 9.2.5 Bias-Variance in Unsupervised Learning

In unsupervised learning, the bias-variance trade-off is less formally defined but still relevant. For example:

- In clustering, choosing the wrong number of clusters can lead to either high bias (too few clusters, oversimplifying the data) or high variance (too many clusters, overfitting the data).
- In dimensionality reduction, keeping too few components in Principal Component Analysis (PCA) increases bias by losing important information, while keeping too many components retains noise, increasing variance.

In unsupervised learning, balancing bias and variance typically involves tuning hyperparameters (e.g., number of clusters, number of components) to find the right complexity level.

### 9.2.6 Striking the Right Balance

To strike the balance between bias and variance in both supervised and unsupervised learning, techniques such as regularization, early stopping, cross-validation, and hyperparameter tuning are essential. These techniques help ensure the model is complex enough to capture patterns in the data but not so complex that it overfits to noise or irrelevant details.

## 9.3 Crossvalidation

Cross-validation is a technique used to evaluate machine learning models and tune hyperparameters by splitting the dataset into multiple subsets. This approach helps to avoid overfitting and provides a better estimate of the model's performance on unseen data. Cross-validation is especially useful for managing the bias-variance trade-off by allowing you to test how well the model generalizes without relying on a single train-test split.

### 9.3.1 K-Fold Cross-Validation

The most commonly used method is $k$-fold cross-validation:

- Split the data: The dataset is divided into $k$ equally-sized folds (subsets).
- Train on $k-1$ folds: The model is trained on $k-1$ folds, leaving one fold as a test set.
- Test on the remaining fold: The model's performance is evaluated on the fold that was left out.
- Repeat: This process is repeated $k$ times, with each fold used once as the test set.
- Average performance: The final cross-validation score is the average performance across all $k$ iterations.

By averaging the results across multiple test sets, cross-validation provides a more robust estimate of model performance and helps avoid overfitting or underfitting to any particular training-test split.

Leave-One-Out Cross-Validation (LOOCV) takes each observation as one fold. The dataset is split into $n$ subsets (where $n$ is the sample size), with each sample acting as a test set once. While this method provides the most exhaustive evaluation, it can be computationally expensive for large datasets.

### 9.3.2 Benefits of Cross-Validation

- Prevents overfitting: By testing the model on multiple subsets of data, cross-validation helps to identify if the model is too complex and overfits to the training data.
- Prevents underfitting: If the model performs poorly across all folds, it may indicate that the model is too simple (high bias).
- Better estimation: Cross-validation gives a better estimate of how the model will perform on unseen data compared to a single train-test split.

### 9.3.3 Cross-Validation in Unsupervised Learning

While cross-validation is most commonly used in supervised learning, it can also be applied to unsupervised learning through:

- Stability testing: Running unsupervised algorithms (e.g., clustering) on different data splits and measuring the stability of the results (e.g., using the silhouette score).

- Internal validation metrics: In clustering, internal metrics like the silhouette score or Davies-Bouldin index can be used to evaluate the quality of clustering across different data splits.

The bias-variance trade-off is a universal problem in machine learning, affecting both supervised and unsupervised models. Cross-validation is a powerful tool for controlling this trade-off by providing a reliable estimate of model performance and helping to fine-tune model complexity. By balancing bias and variance through careful model selection, regularization, and cross-validation, you can develop models that generalize well to unseen data without overfitting or underfitting.

### 9.3.4 A Curve-Fitting with Splines: An Example

Overfitting occurs when a model becomes overly complex and starts to capture not just the underlying patterns in the data but also the noise or random fluctuations. This can lead to poor generalization to new, unseen data. A clear sign of overfitting is when a model performs very well on the training data but performs poorly on test data, as it fails to generalize beyond the data it was trained on.

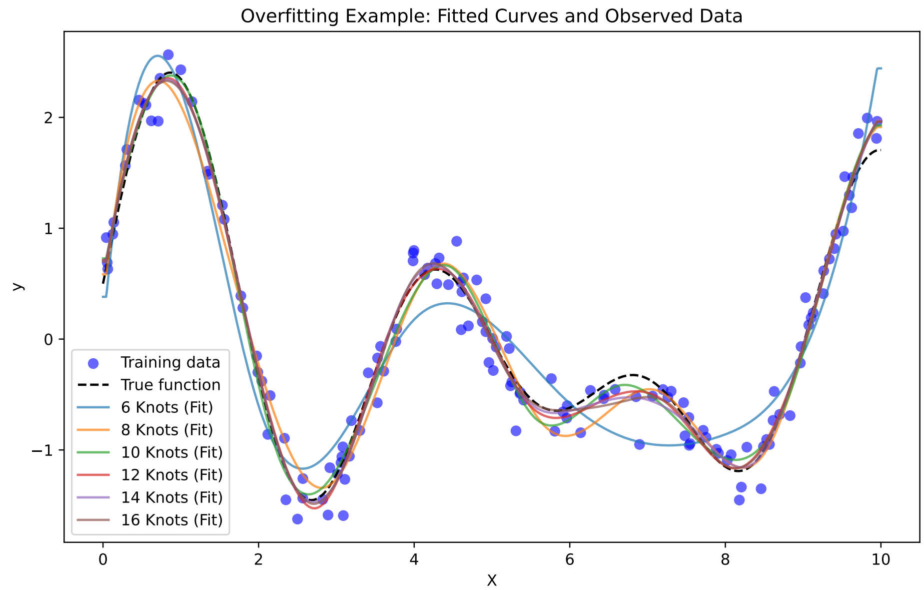
In this example, we illustrate overfitting using cubic spline regression with different numbers of knots. Splines are a flexible tool that allow for piecewise polynomial regression, with knots defining where the pieces of the polynomial meet. The more knots we use, the more flexible the model becomes, which can potentially lead to overfitting.

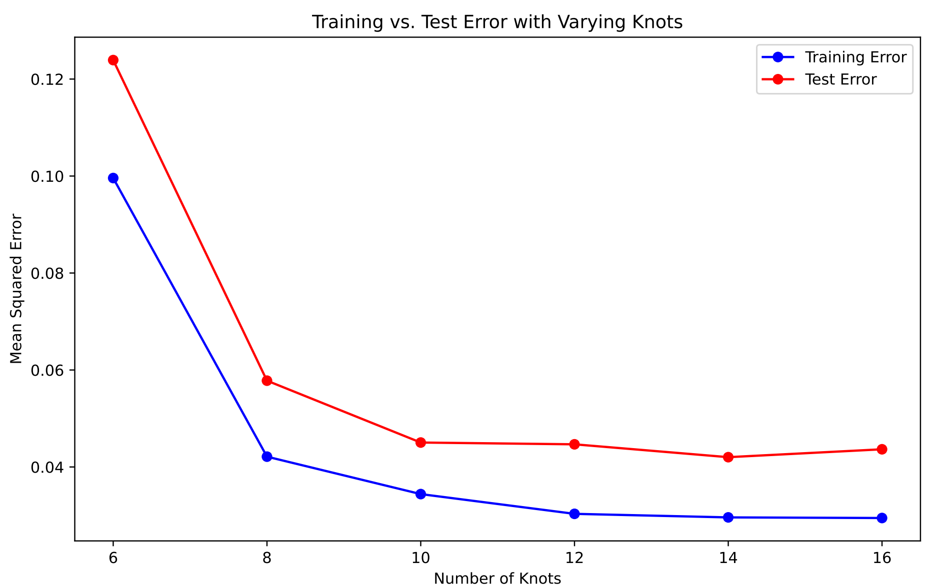
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import SplineTransformer
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error

def true_function(X):
    return np.sin(1.5 * X) + 0.5 * np.cos(0.5 * X) + np.sin(2 * X)
```

```
# Generate synthetic data using the more complex true function
X = np.sort(np.random.rand(200) * 10).reshape(-1, 1)
y = true_function(X).ravel() + np.random.normal(0, 0.2, X.shape[0])

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ranc

# Function to explore overfitting by plotting both errors and fitted curves
def overfitting_example_with_fitted_curves(X_train, y_train, X_test, y_test,
    train_errors = []
    test_errors = []

    # Generate fine grid for plotting the true curve and fitted models
    X_line = np.linspace(0, 10, 1000).reshape(-1, 1)
    y_true = true_function(X_line)

    # Plot the true function and observed data
    plt.figure(figsize=(10, 6))
    plt.scatter(X_train, y_train, color='blue', label='Training data', alpha=
    plt.plot(X_line, y_true, label='True function', color='black', linestyle=

    for n_knots in knots_list:
        # Create a spline model with fixed degree = 3 (cubic) and varying knc
        spline = SplineTransformer(degree=3, n_knots=n_knots, include_bias=Fa
        model = make_pipeline(spline, LinearRegression())

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

        # Predict on training and test data
        y_pred_train = model.predict(X_train)
        y_pred_test = model.predict(X_test)
        y_pred = model.predict(X_line)
```

```
        # Calculate training and test errors (mean squared error)
        train_errors.append(mean_squared_error(y_train, y_pred_train))
        test_errors.append(mean_squared_error(y_test, y_pred_test))

        # Plot the fitted curve
        plt.plot(X_line, y_pred, label=f'{n_knots} Knots (Fit)', alpha=0.7)

    print(train_errors, test_errors)

    plt.title('Overfitting Example: Fitted Curves and Observed Data')
    plt.xlabel('X')
    plt.ylabel('y')
    plt.legend()
    plt.show()

    # Plot training and test error separately
    plt.figure(figsize=(10, 6))
    plt.plot(knots_list, train_errors, label='Training Error', marker='o', color='blue')
    plt.plot(knots_list, test_errors, label='Test Error', marker='o', color='red')
    plt.title('Training vs. Test Error with Varying Knots')
    plt.xlabel('Number of Knots')
    plt.ylabel('Mean Squared Error')
    plt.legend()
    plt.show()

# Explore overfitting by varying the number of knots and overlaying the fitted curves
knots_list = [6, 8, 10, 12, 14, 16]
overfitting_example_with_fitted_curves(X_train, y_train, X_test, y_test, knots_list)
```

```
[np.float64(0.09958691622917616), np.float64(0.04212878051666032), np.float64(0.034406597679
```


Overfitting Example: Fitted Curves and Observed Data
Training data
True function
6 Knots (Fit)
8 Knots (Fit)
10 Knots (Fit)
12 Knots (Fit)
14 Knots (Fit)
16 Knots (Fit)
X
y


Training vs. Test Error with Varying Knots
Training Error
Test Error
Number of Knots
Mean Squared Error

In the first plot, we fit cubic splines with 2, 4, 6, 8, 10, and 12 knots to the data. As the number of knots increases, the model becomes more flexible and better able to fit the training data. With only 2 knots, the model is quite smooth and underfits the data, capturing only broad trends but missing the detailed structure of the true underlying function. With 4 or 6 knots, the model begins to capture more of the data's structure, balancing the bias-variance trade-off effectively. However, as we increase the number of knots to 10 and 12, the model becomes too flexible. It starts to fit the noise in the training data, producing a curve that adheres too closely to the data points. This is a classic case of overfitting: the model fits the training data very well, but it no longer generalizes to new data.

In the second plot, we observe the training error and test error as the number of knots increases. As expected, the training error consistently decreases as the number of knots increases, since a more complex model can fit the training data better. However, the test error tells a different story. Initially, the test error decreases as the model becomes more flexible, indicating that the model is learning meaningful patterns from the data. But after a certain point, the test error begins to increase, signaling overfitting.

This is a key insight into the bias-variance trade-off. While adding more complexity (in this case, more knots) reduces bias and improves fit on the training data, it also increases variance, making the model more sensitive to fluctuations and noise in the data. This results in worse performance on test data, as the model becomes too specialized to the training set.

The example clearly demonstrates how overfitting can occur when the model becomes too complex. In practice, it's important to find a balance between underfitting (high bias) and overfitting (high variance). Techniques such as cross-validation, regularization, or limiting model complexity (e.g., setting a reasonable number of knots in spline regression) can help manage this trade-off and produce models that generalize well to unseen data.

By tuning the number of knots or other model parameters, we can achieve a model that strikes the right balance, capturing the true patterns in the data without fitting the noise.

# 10 Supervised Learning

## 10.1 Decision Trees

Decision trees are widely used supervised learning models that predict the value of a target variable by iteratively splitting the dataset based on decision rules derived from input features. The model functions as a piecewise constant approximation of the target function, producing clear, interpretable rules that are easily visualized and analyzed (Breiman et al., 1984). Decision trees are fundamental in both classification and regression tasks, serving as the building blocks for more advanced ensemble models such as Random Forests and Gradient Boosting Machines.

### 10.1.1 Algorithm Formulation

The core mechanism of a decision tree algorithm is the identification of optimal splits that partition the data into subsets that are increasingly homogeneous with respect to the target variable. At any node $m$, the data subset is denoted as $Q_m$ with a sample size of $n_m$. The objective is to find a candidate split $\theta$, defined as a threshold for a given feature, that minimizes an impurity or loss measure $H$.

When a split is made at node $m$, the data is divided into two subsets: $Q_{m,l}$ (left node) with sample size $n_{m,l}$, and $Q_{m,r}$ (right node) with sample size $n_{m,r}$. The split quality, measured by $G(Q_m, \theta)$, is given by:

$$G(Q_m, \theta) = \frac{n_{m,l}}{n_m} H(Q_{m,l}(\theta)) + \frac{n_{m,r}}{n_m} H(Q_{m,r}(\theta)).$$

The algorithm aims to identify the split that minimizes the impurity:

$$\theta^* = \arg\min_{\theta} G(Q_m, \theta).$$

This process is applied recursively at each child node until a stopping condition is met.

- Stopping Criteria: The algorithm stops when the maximum tree depth is reached or when the node sample size falls below a preset threshold.
- Pruning: Reduce the complexity of the final tree by removing branches that add little predictive value. This reduces overfitting and improves the generalization accuracy of the model.

### 10.1.2 Search Space for Possible Splits

At each node in the decision tree, the search space for possible splits comprises all features in the dataset and potential thresholds derived from the values of each feature. For a given feature, the algorithm considers each unique value in the current node's subset as a possible split point. The potential thresholds are typically set as midpoints between consecutive unique values, ensuring the data is partitioned effectively.

Formally, let the feature set be $\{X_1, X_2, \dots, X_p\}$, where $p$ is the total number of features, and let the unique values of feature $ X_j $ at node $ m $ be denoted by ${v_{j,1}, v_{j,2}, ..., v_{j,k_j}} .Thesearchspaceatnode m$ includes:

- Feature candidates: $\{X_1, X_2, \dots, X_p\}$.

- Threshold candidates for $X_j$:

$$\left\{ \frac{v_{j,i} + v_{j,i+1}}{2} \mid 1 \leq i < k_j \right\}.$$

The search space therefore encompasses all combinations of features and their respective thresholds. While the complexity of this search can be substantial, particularly for high-dimensional data or features with numerous unique values, efficient algorithms use sorting and single-pass scanning techniques to mitigate the computational cost.

### 10.1.3 Metrics

#### 10.1.3.1 Classification

In decision tree classification, several criteria can be used to measure the quality of a split at each node. These criteria are based on how "pure" the resulting nodes are after the split. A pure node contains samples that predominantly belong to a single class. The goal is to minimize impurity, leading to nodes that are as homogeneous as possible.

- Gini Index: The Gini index measures the impurity of a node by calculating the probability of randomly choosing two different classes. A perfect split (all instances belong to one class) has a Gini index of 0. At node $m$, the Gini index is

  $$H(Q_m) = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}),$$

  where $p_{mk}$ is the proportion of samples of class $k$ at node $m$; and$K$ is the total number of classes The Gini index is often preferred for its speed and simplicity, and it's used by default in many implementations of decision trees, including `sklearn`.

- Entropy (Information Gain): Entropy is another measure of impurity, derived from information theory. It quantifies the "disorder" of the data at a node. Lower entropy means higher purity. At node $m$, it is defined as
$$H(Q_m) = -\sum_{k=1}^{K} p_{mk} \log p_{mk}$$
Entropy is commonly used in decision tree algorithms like ID3 and C4.5. The choice between Gini and entropy often depends on specific use cases, but both perform similarly in practice.

- Misclassification Error: Misclassification error focuses solely on the most frequent class in the node. It measures the proportion of samples that do not belong to the majority class. Although less sensitive than Gini and entropy, it can be useful for classification when simplicity is preferred. At node $m$, it is defined as
$$H(Q_m) = 1 - \max_k p_{mk},$$
where $\max_k p_{mk}$ is the largest proportion of samples belonging to any class $k$.

#### 10.1.3.2 Regression Criteria

In decision tree regression, different criteria are used to assess the quality of a split. The goal is to minimize the spread or variance of the target variable within each node.

- Mean Squared Error (MSE): Mean squared error is the most common criterion used in regression trees. It measures the average squared difference between the actual values and the predicted values (mean of the target in the node). The smaller the MSE, the better the fit. At node $m$, it is
$$H(Q_m) = \frac{1}{n_m} \sum_{i=1}^{n_m} (y_i - \bar{y}_m)^2,$$

where

- $y_i$ is the actual value for sample $i$;
- $\bar{y}_m$ is the mean value of the target at node $m$;
- $n_m$ is the number of samples at node $m$.

MSE works well when the target is continuous and normally distributed.

- Half Poisson Deviance (for count targets): When dealing with count data, the Poisson deviance is used to model the variance in the number of occurrences of an event. It is well-suited for target variables representing counts (e.g., number of occurrences of an event). At node $m$, it is

$$H(Q_m) = \sum_{i=1}^{n_m} \left( y_i \log \left( \frac{y_i}{\hat{y}_i} \right) - (y_i - \hat{y}_i) \right),$$

where $\hat{y}_i$ is the predicted count. This criterion is especially useful when the target variable represents discrete counts, such as predicting the number of occurrences of an event.

- Mean Absolute Error (MAE): Mean absolute error is another criterion that minimizes the absolute differences between actual and predicted values. While it is more robust to outliers than MSE, it is slower computationally due to the lack of a closed-form solution for minimization. At node $m$, it is

$$H(Q_m) = \frac{1}{n_m} \sum_{i=1}^{n_m} |y_i - \bar{y}_m|$$

MAE is useful when you want to minimize large deviations and can be more robust in cases where outliers are present in the data.

#### 10.1.3.3 Summary

In decision trees, the choice of splitting criterion depends on the type of task (classification or regression) and the nature of the data. For classification tasks, the Gini index and entropy are the most commonly used, with Gini offering simplicity and speed, and entropy providing a more theoretically grounded approach. Misclassification error can be used for simpler cases. For regression tasks, MSE is the most popular choice, but Poisson deviance and MAE are useful for specific use cases such as count data and robust models, respectively.

## 10.2 Naive Bayes

Naive Bayes is a probabilistic classification algorithm based on Bayes' Theorem, which is used for both binary and multiclass classification problems. It is particularly effective for high-dimensional datasets and is commonly applied in tasks like text classification, spam detection, and sentiment analysis. The algorithm is called "naive" because it assumes that all features are conditionally independent given the class label, an assumption that rarely holds in real-world data but still performs well in many cases.

### 10.2.1 Theoretical Foundations

The foundation of the Naive Bayes classifier is Bayes' Theorem, which is used to update the probability estimate of a hypothesis given new evidence. Mathematically, Bayes' Theorem is expressed as:

$$P(y \mid X) = \frac{P(X \mid y)\, P(y)}{P(X)},$$

where:

- $P(y \mid X)$: Posterior probability of class $y$ given the input features $X$.
- $P(X \mid y)$: Likelihood of observing $X$ given that the class is $y$.
- $P(y)$: Prior probability of the class $y$.
- $P(X)$: Marginal probability of the feature vector $X$.

#### 10.2.1.1 Naive Assumption and Likelihood Decomposition

The algorithm makes the simplifying assumption that features in $X$ are conditionally independent given the class $y$. This assumption enables the likelihood $P(X \mid y)$ to be decomposed as:

$$P(X \mid y) = \prod_{i=1}^{n} P(x_i \mid y),$$

where $X = \{x_1, x_2, \dots, x_n\}$ represents the feature vector with $n$ features, and $P(x_i \mid y)$ is the conditional probability of feature $x_i$ given the class $y$.

The model parameters are the prior probabilities $P(y)$ and the conditional probabilities $P(x_i \mid y)$. These are estimated from the training data using the maximum likelihood estimation (MLE):

1. Prior Estimation: The prior probability $P(y)$ is estimated as the proportion of training samples in class $y$:

   $$\hat{P}(y) = \frac{\text{count}(y)}{N},$$

   where $\text{count}(y)$ is the number of instances belonging to class $y$, and $N$ is the total number of training samples.

2. Conditional Probability Estimation:

- Categorical Features: For discrete or categorical features, the conditional probability $P(x_i \mid y)$ is estimated as:

$$\hat{P}(x_i \mid y) = \frac{\text{count}(x_i, y)}{\text{count}(y)},$$

  where $\text{count}(x_i, y)$ is the number of samples in class $y$ that have feature $x_i$.

- Continuous Features: For continuous features, Naive Bayes commonly assumes a Gaussian distribution. In this case, $P(x_i \mid y)$ is modeled using the Gaussian distribution with mean $\mu_{y,i}$ and variance $\sigma^2_{y,i}$:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma^2_{y,i}}} \exp\left(-\frac{(x_i - \mu_{y,i})^2}{2\sigma^2_{y,i}}\right).$$

  The parameters $\mu_{y,i}$ and $\sigma^2_{y,i}$ are estimated from the training data using the sample mean and variance for each feature in each class.

#### 10.2.1.2 Class Prediction

The goal of the Naive Bayes classifier is to predict the class $y$ that maximizes the posterior probability $P(y \mid X)$. After applying Bayes' Theorem and dropping the constant denominator $P(X)$, the decision rule becomes:

$$y^* = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y).$$

In practice, the log of the posterior is used to prevent numerical underflow:

$$\log P(y \mid X) = \log P(y) + \sum_{i=1}^{n} \log P(x_i \mid y).$$

The predicted class is the one that maximizes this expression.

#### 10.2.1.3 Surprisingly Good Performance

Although the assumption of conditional independence among features is often unrealistic, Naive Bayes still performs well for several reasons:

1. Robustness to Violations of Independence: Literature suggests that Naive Bayes can achieve good classification performance even when features are correlated, as long as the dependencies are consistent across classes (Domingos & Pazzani, 1997). This is because the decision boundaries produced by Naive Bayes are often well-aligned with the true boundaries, despite the imprecise probability estimates.

2. Decision Rule Effectiveness: Since Naive Bayes focuses on finding the class that maximizes the posterior probability, it is less sensitive to errors in individual probability estimates, as long as the relative ordering of probabilities remains correct (Rish, 2001).

3. Zero-One Loss Minimization: Naive Bayes aims to minimize the zero-one loss, i.e., the number of misclassifications. The method benefits from the fact that exact probability estimation is not essential for accurate classification, as the correct class can still be chosen even with approximate probabilities (Ng & Jordan, 2001).

4. High-Dimensional Settings: In high-dimensional settings, the conditional independence assumption can act as a form of implicit regularization, preventing overfitting by simplifying the probability model

(Rish, 2001). This makes Naive Bayes particularly well-suited for text classification and other sparse feature spaces.

#### 10.2.1.4 Advantages and Limitations

Advantages:

- Computationally efficient, with linear time complexity in terms of the number of features and data samples.
- Performs well on large datasets, especially when features are conditionally independent.
- Suitable for high-dimensional data, making it popular in text classification.

Limitations:

- Relies on the assumption of conditional independence, which may not hold in real-world datasets, potentially affecting performance.
- It is sensitive to zero probabilities; if a feature value never appears in the training set for a given class, its likelihood becomes zero. To address this, Laplace smoothing (or add-one smoothing) is often applied.

#### 10.2.1.5 Laplace Smoothing

Laplace smoothing is used to handle zero probabilities in the likelihood estimation. It adds a small constant $ $ (usually 1) to the count of each feature value, preventing the probability from becoming zero:

$$P(x_i \mid y) = \frac{\text{count}(x_i, y) + \alpha}{\sum_{x_i'} (\text{count}(x_i', y) + \alpha)}.$$

This adjustment ensures that even unseen features in the training data do not lead to zero probabilities, thus improving the model's robustness.

# 11 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.

    1. Install Quarto onto your computer following the instructions of Get Started. Document the obstacles you encountered and how you overcame them.
    2. 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.
    3. Render the homework into a pdf file and put the file into a release in your GitHub repo. Document any obstacles you have and how you overcome them.

2. **Git basics and GitHub setup** Learn the Git basics and set up an account on GitHub if you do not already have one. Practice the tips on Git in the notes. By going through the following tasks, ensure your repo has at least 10 commits, each with an informative message. Regularly check the status of your repo using `git status`. The specific tasks are:

    1. Clone the class notes repo to an appropriate folder on your computer.
    2. Add all the files to your designated homework repo from GitHub Classroom and work on that repo for the rest of the problem.
    3. Add your name and wishes to the Wishlist; commit.
    4. Remove the `Last, First` entry from the list; commit.

5. Create a new file called `add.qmd` containing a few lines of texts; commit.
6. Remove `add.qmd` (pretending that this is by accident); commit.
7. Recover the accidentally removed file `add.qmd`; add a long line (a paragraph without a hard break); add a short line (under 80 characters); commit.
8. Change one word in the long line and one word in the short line; use `git diff` to see the difference from the last commit; commit.
9. Play with other git operations and commit.

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 as if you are explaining them to someone who want to contribute too.

   1. Create a fork of the notes repo into your own GitHub account.
   2. Clone it to an appropriate folder on your computer.
   3. Render the classnotes on your computer; document the obstacles and solutions.
   4. Make a new branch (and name it appropriately) to experiment with your changes.
   5. Checkout your branch and add your wishes to the wish list; commit with an informative message; and push the changes to your GitHub account.
   6. 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** Write a function to demonstrate the Monty Hall problem through simulation. The function takes two arguments `ndoors` and `ntrials`, representing the number of doors in the experiment and the number of trails in a simulation, respectively. The function should return the proportion of wins for both the switch and

no-switch strategy. Apply your function with 3 doors and 5 doors, both with 1000 trials. Include sufficient text around the code to explain your them.

5. **Approximating $\pi$** Write a function to do a Monte Carlo approximation of $\pi$. The function takes a Monte Carlo sample size `n` as input, and returns a point estimate of $\pi$ 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 $\pi$. 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 form 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.

   1. List all the possible ways to group the four numbers.
   2. How many possibly ways are there to check for a solution?
   3. 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.

1. 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.
2. Clean up the variable names. Use lower cases and replace spaces with underscores.
3. Get the basic summaries of each variables: missing percentage; descriptive statistics for continuous variables; frequency tables for discrete variables.
4. Are their invalid `longitude` and `latitude` in the data? If so, replace them with `NA`.
5. Are there `zip_code` values that are not legit NYC zip codes? If so, replace them with `NA`.
6. Are there missing in `zip_code` and `borough`? Do they always co-occur?
7. 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.
8. Is it redundant to keep both `location` and the `longitude`/`latitude` at the NYC Open Data server?
9. Check the frequency of `crash_time` by hour. Is there a matter of bad luck at exactly midnight? How would you interpret this?
10. 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?
11. Print the whole frequency table of `contributing_factor_vehicle_1`. Convert lower cases to uppercases and check the frequencies again.
12. Provided an opportunity to meet the data provider, what suggestions 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.

    1. 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.
    2. 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.
    3. Overlay the locations of the crashes on a map of NYC. The map could be a static map or Google map.
    4. 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.
    5. Merge the crash data with the zip code database.
    6. Fit a logistic model with `severe` as the outcome variable and covariates that are available in the data or can be engineered from the data. For example, zip code level covariates can be obtained by merging with the zip code database; crash hour; number of vehicles involved.

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

    1. Set random seed to `1234`. Randomly select 20% of the crashes as testing data and leave the rest 80% as training data.
    2. 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.

3. Fit a logistic model on the training data with $L_1$ regularization. Select the tuning parameter with 5-fold cross-validation in F1 score
4. 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.

    1. Data cleaning.
        - Import the data, rename the columns with our preferred styles.
        - Summarize the missing information. Are there variables that are close to completely missing?
        - Are their redundant information in the data? Try storing the data using the Arrow format and comment on the efficiency gain.
        - Are there invalid NYC zipcode or borough? Justify and clean them if yes.
        - 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`.
        - Summarize your suggestions to the data curator in several bullet points.
    2. Data exploration.

- 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).
- Perform a formal hypothesis test to confirm the observations from your visualization. Formaly state your hypotheses and summary your conclusions in plain English.
- Create a binary variable `over2h` to indicate that a service request took two hours or longer to close.
- Does `over2h` depend on the complain type, borough, or weekday (vs weekend/holiday)? State your hypotheses and summarize your conclusions in plain English.

3. Data analysis.
   - The addresses of NYC police percincts are stored in `data/nypd_precincts.csv`. Use geocoding tools to find their geocode (longitude and lattitude) from the addresses.
   - Create a variable `dist2pp` which represent the distance from each request incidence to the nearest police precinct.
   - Create zip code level variables by merging with data from package `uszipcode`.
   - Randomly select 20% of the complaints as testing data with seeds `1234`. Build a logistic model to predict `over2h` for the noise complains 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.
   - 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


Breiman, L., Friedman, J. H., Olshen, R., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, *29*(2-3), 103–130.

Fayaz, O. (2022). *Principles of visual communication*.

Grillo, H. M., & Enesi, M. (2022). The impact, importance, types, and use of non-verbal communication in social relations. *Linguistics and Culture Review*, *6*, 291–307.

Ng, A., & Jordan, M. (2001). On discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, 841–848.

Prabavathi, R., & Nagasubramani, P. C. (2018). Effective oral and written communication. *Journal of Applied and Advanced Research*, *10*, 29–32.

Pragmatic. (2024). *Communication skills for data science*. https://www.pragmaticinstitute.com%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20/resources/articles/data/communication-skills-for-data-science/.

Radovilsky, Z., Hegde, V., Acharya, A., & Uma, U. (2018). Skills requirements of business data analytics and data science jobs: A comparative analysis. *Journal of Supply Chain and Operations Management*, *16*, 82–101.

Rish, I. (2001). An empirical study of the naive Bayes classifier. *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, *3*, 41–46.

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

Vandemeulebroecke, M., Baillie, M., Margolskee, A., & Magnusson, B. (2019). Effective visual communication for the quantitative scientist. *CPT Pharmacometrics Syst Pharmacol*, *10*, 705–719.

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

Wilkinson, L. (2012). *The grammar of graphics*. Springer.

Yer, S. (2018). Verbal communication as a two-way process in connecting people. *Social Science Research Network*.