

# **Introduction to Data Science**

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

Jun Yan and Students in Fall 2024:     Suha Akach  
Jack Bienveune     Zachary Blanchard     Sara Clokey  
                         Melanie Desrochesk     Thea Johnson  
Olivia Kashlapov     Rahul Manna     Julia Mazzola  
                         Mohammad Shahriyar Parvez     Deyu Xuo

2024-12-15



# Table of contents

<b>Preliminaries</b>	<b>1</b>
Sources at GitHub . . . . .	1
Compiling the Classnotes . . . . .	2
Midterm Project . . . . .	2
Final Project . . . . .	2
Adapting to Rapid Skill Acquisition . . . . .	3
Wishlist . . . . .	4
Presentation Orders . . . . .	7
Course Logistics . . . . .	9
Presentation Task Board . . . . .	9
Final Project Presentation Schedule . . . . .	12
Contributing to the Class Notes . . . . .	12
Homework Requirements . . . . .	13
Practical Tips . . . . .	13
Data analysis . . . . .	13
Presentation . . . . .	14
My Presentation Topic (Template) . . . . .	14
Introduction . . . . .	14
Sub Topic 1 . . . . .	14
Sub Topic 2 . . . . .	14
Sub Topic 3 . . . . .	15
Conclusion . . . . .	15
<b>1 Introduction</b>	<b>17</b>
1.1 What Is Data Science? . . . . .	17
1.2 Expectations from This Course . . . . .	18

*Table of contents*

1.3	Computing Environment . . . . .	19
1.3.1	Command Line Interface . . . . .	19
1.3.2	Python . . . . .	20
<b>2</b>	<b>Project Management</b>	<b>23</b>
2.1	Set Up Git/GitHub . . . . .	23
2.2	Most Frequently Used Git Commands . . . . .	24
2.3	Tips on using Git: . . . . .	25
2.4	Pull Request . . . . .	25
<b>3</b>	<b>Reproducible Data Science</b>	<b>27</b>
3.1	Introduction to Quarto . . . . .	27
3.2	Compiling the Classnotes . . . . .	28
3.2.1	Set up your Python Virtual Environment . . . . .	28
3.2.2	Clone the Repository . . . . .	29
3.2.3	Render the Classnotes . . . . .	30
<b>4</b>	<b>Python Refreshment</b>	<b>31</b>
4.1	Know Your Computer . . . . .	31
4.1.1	Operating System . . . . .	31
4.1.2	File System . . . . .	32
4.2	The Python World . . . . .	33
4.3	Standard Library . . . . .	33
4.4	Important Libraries . . . . .	35
4.5	Writing a Function . . . . .	36
4.5.1	Monty Hall . . . . .	37
4.6	Variables versus Objects . . . . .	38
4.7	Number Representation . . . . .	41
4.7.1	Integers . . . . .	41
4.7.2	Floating Number . . . . .	42
4.8	Virtual Environment . . . . .	44
<b>5</b>	<b>Communication in Data Science</b>	<b>47</b>
5.1	Introduction . . . . .	47

*Table of contents*

5.2	Importance & Application of Communication . . . . .	47
5.3	General Communication Skills . . . . .	48
5.3.1	Verbal Communication Skills . . . . .	48
5.3.2	Non-Verbal Communication Skills . . . . .	49
5.3.3	Visual Communication Skills . . . . .	49
5.3.4	Written Communication Skills . . . . .	49
5.4	Communication in Data Science . . . . .	50
5.4.1	Identify your Audience . . . . .	50
5.4.2	Utilize Data Visualization . . . . .	50
5.4.3	Focus on Data Communication Skills . . . . .	51
5.4.4	Give Space for Questions and Feedback . . . . .	51
5.5	Further Learning . . . . .	52
<b>6</b>	<b>Data Manipulation</b>	<b>53</b>
6.1	Introduction . . . . .	53
6.2	Import/Export Data . . . . .	54
6.2.1	Summary . . . . .	54
6.2.2	Package Pandas . . . . .	54
6.2.3	Export the data to a .csv file: . . . . .	55
6.2.4	Import files in common formats: .csv/.xlsx/.txt .	58
6.2.5	Import the data from other software . . . . .	64
6.2.6	View data information . . . . .	81
6.2.7	Find Null Values . . . . .	86
6.3	SQL . . . . .	90
6.3.1	Table of Contents . . . . .	90
6.3.2	What is a Database? . . . . .	91
6.3.3	Example Relational Database . . . . .	91
6.3.4	What is SQL? . . . . .	93
6.3.5	CRUD . . . . .	93
6.3.6	Implementing SQL Through Python to Create a Table	93
6.3.7	How to Insert into the SQL table? . . . . .	94
6.3.8	How to Update/Delete Using SQL? . . . . .	95
6.3.10	Using SQL to Work on a CSV file . . . . .	99
6.3.11	Queries . . . . .	100

*Table of contents*

6.3.12	Queries Output . . . . .	101
6.3.13	Conclusion . . . . .	103
6.3.14	Further reading . . . . .	103
6.4	NYC Crash Data . . . . .	103
6.5	Cross-platform Data Format <b>Arrow</b> . . . . .	107
6.6	Using the Census Data . . . . .	109
6.7	Extracting and Analyzing Census Tract Data . . . . .	113
6.7.1	Accessing the census Data . . . . .	113
6.7.2	Performing Dataframe Operations . . . . .	118
<b>7</b>	<b>Visualization</b>	<b>127</b>
7.1	Data Visualization with Plotnine . . . . .	127
7.1.1	Introduction . . . . .	127
7.1.2	What is Plotnine? . . . . .	127
7.1.3	Installing Plotnine . . . . .	128
7.1.4	Scatterplot . . . . .	129
7.1.5	Bar Chart . . . . .	140
7.1.6	Histogram . . . . .	146
7.1.7	Line Chart . . . . .	151
7.1.8	Faceting Your Plots . . . . .	155
7.1.9	Conclusion . . . . .	165
7.2	Spatial Data Visualization with Google Map . . . . .	166
7.2.1	Introducing Spatial Data . . . . .	167
7.2.2	Google & Spatial Data . . . . .	169
7.2.3	Getting Started in Google Maps Platform . . . . .	172
7.2.4	Using googlemaps and gmplot . . . . .	175
7.2.5	gmplot Heat Map Example . . . . .	179
7.2.6	Googlemaps . . . . .	179
7.2.7	Conclusion . . . . .	179
7.2.8	References . . . . .	180
7.3	Animating Plots and Maps . . . . .	180
7.3.1	Introduction . . . . .	180
7.3.2	Matplotlib . . . . .	181
7.3.3	Animating Plots . . . . .	185

*Table of contents*

7.3.4	Animating Maps . . . . .	193
7.3.5	Advance Example - Animating Maps . . . . .	200
7.3.6	Saving your Animation . . . . .	206
7.3.7	Conclusion . . . . .	207
<b>8</b>	<b>Statistical Tests and Models</b>	<b>209</b>
8.1	Tests for Exploratory Data Analysis . . . . .	209
8.2	Statistical Modeling . . . . .	211
8.2.1	Installation of <code>statsmodels</code> . . . . .	211
8.2.2	Linear Model . . . . .	211
8.2.3	Generalized Linear Regression . . . . .	216
8.3	Validating the Results of Logistic Regression . . . . .	220
8.3.1	Confusion Matrix . . . . .	221
8.3.2	Accuracy . . . . .	222
8.3.3	Precision . . . . .	223
8.3.4	Recall . . . . .	223
8.3.5	F-beta Score . . . . .	224
8.3.6	Receiver Operating Characteristic (ROC) Curve . .	225
8.3.7	Demonstration . . . . .	225
8.4	LASSO Logistic Models . . . . .	229
8.4.1	Theoretical Formulation of the Problem . . . . .	229
8.4.2	Solution Path . . . . .	230
8.4.3	Selection the Tuning Parameter . . . . .	233
8.4.4	Preparing for Logistic Regression Fitting . . . . .	235
<b>9</b>	<b>Machine Learning: Overview</b>	<b>243</b>
9.1	Introduction . . . . .	243
9.1.1	Supervised Learning . . . . .	244
9.1.2	Unsupervised Learning . . . . .	244
9.1.3	Reinforcement Learning . . . . .	245
9.2	Bias-Variance Tradeoff . . . . .	246
9.2.1	Bias . . . . .	246
9.2.2	Variance . . . . .	246
9.2.3	The Trade-Off . . . . .	247

*Table of contents*

9.2.4	Bias-Variance in Supervised Learning . . . . .	247
9.2.5	Bias-Variance in Unsupervised Learning . . . . .	248
9.2.6	Striking the Right Balance . . . . .	249
9.3	Crossvalidation . . . . .	249
9.3.1	K-Fold Cross-Validation . . . . .	249
9.3.2	Benefits of Cross-Validation . . . . .	250
9.3.3	Cross-Validation in Unsupervised Learning . . . . .	250
9.3.4	A Curve-Fitting with Splines: An Example . . . . .	251
<b>10</b>	<b>Supervised Learning</b>	<b>257</b>
10.1	Decision Trees: Foundation . . . . .	257
10.1.1	Algorithm Formulation . . . . .	257
10.1.2	Search Space for Possible Splits . . . . .	258
10.1.3	Metrics . . . . .	259
10.2	Boosted Trees . . . . .	262
10.2.1	Introduction . . . . .	262
10.2.2	Boosting Process . . . . .	263
10.2.3	Key Concepts . . . . .	265
10.2.4	Why Boosting Works . . . . .	265
10.2.5	Applications and Popular Implementations . . . . .	266
10.3	Naive Bayes . . . . .	266
10.3.1	Theoretical Foundations . . . . .	267
10.3.2	Types of Naive Bayes: . . . . .	271
10.3.3	Naive Bayes w/ NYC Crash Data . . . . .	272
10.4	Handling Imbalanced Data with SMOTE . . . . .	277
10.4.1	Introduction . . . . .	277
10.4.2	Class Imbalance . . . . .	278
10.4.3	Synthetic Model Oversampling Technique . . . . .	278
10.4.4	SMOTE versus Traditional Methods . . . . .	279
10.4.5	Installation and Setup . . . . .	280
10.4.6	Data Preparation . . . . .	280
10.4.7	Data Visualization . . . . .	282
10.4.8	Implementing SMOTE . . . . .	283
10.4.9	Visualization after SMOTE . . . . .	284

*Table of contents*

10.4.10 Model Training using SMOTE . . . . .	286
10.4.11 Model Evaluation . . . . .	287
10.4.12 Another Example Using NYC Crash Data . . . . .	289
10.4.13 NYC Crash Data Part 2 . . . . .	290
10.4.14 Limitations and Challenges of SMOTE . . . . .	292
10.4.15 Conclusion . . . . .	292
10.4.16 Further Readings . . . . .	293
<b>11 Unsupervised Learning</b>	<b>295</b>
11.1 Principle Component Analysis . . . . .	295
11.1.1 Theory . . . . .	295
11.1.2 Properties of PCA . . . . .	296
11.1.3 Interpreting PCA Results . . . . .	297
11.1.4 Example: PCA on 8x8 Digit Data . . . . .	298
11.1.5 Loading and Visualizing the Data . . . . .	299
11.2 K-Means Clustering . . . . .	310
11.2.1 Subsection . . . . .	310
<b>12 Advanced Topics</b>	<b>311</b>
12.1 Web Scraping . . . . .	311
12.1.1 What is Web-Scraping . . . . .	311
12.1.2 How to Web-Scrape with Python . . . . .	312
12.1.3 Examples using NYC Open Data . . . . .	315
12.1.4 A Note on Data Ethics . . . . .	325
<b>13 Exercises</b>	<b>327</b>
<b>References</b>	<b>335</b>



# Preliminaries

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

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

## Sources at GitHub

These lecture notes for STAT 3255/5255 in 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 GitHub repository at <https://github.com/statds/ids-f24>.

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

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

## **Compiling the Classnotes**

To reproduce the classnotes output on your own computer, here are the necessary steps:

- Clone the classnotes repository to an appropriate location on your computer.
- Set up a Python virtual environment in the root folder of the source.
- Install all the packages specified in `requirements.txt`.
- For some chapters that need to interact with Google map services, you need to save your API key in a file named `api_key.txt` in the root folder of the source.
- Render the book with `quarto render` from the root folder on a terminal; the rendered book will be stored under `_book`.

## **Midterm Project**

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

## *Adapting to Rapid Skill Acquisition*

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

## *Wishlist*

- Learn how to show my skills I have learned in this class during interviews
- Baptista, Stef
  - Develop a project/presentation suitable 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
  - Develop data science and machine learning skills

## *Preliminaries*

- 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 colabаратive 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.

### *Presentation Orders*

- 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 work-flows.
  - 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.

## Preliminaries

```
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']
```

## *Course Logistics*

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

## *Preliminaries*

- 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	Presentation with Quarto
09/16	Deyu Xu	Import/Export data
09/18	Sara Clokey	Communications in Data Science
09/23	Dorathea Johnson	Database with SQL
09/25	Xavier Febles	Statistical tests
09/30	Jack Bienvenue	Visualizing Spatial Data in a Google Map
10/02	Julia Mazzola	Data Visualization with Plotnine
10/07	Suha Akach	Naive Bayes classifier
10/09	Rahul Manna	Animation

### *Course Logistics*

Date	Presenter	Topic
10/23	Jaden Astle	Classification and Regression Trees
10/23	Olivia Kashalapov	Synthetic Minority Oversam- pling Technique (SMOTE)
10/28	Data science alumni panel	
10/30	Emily Borowski	Random Forest
10/30	Aditya Paricharak	Neural Networks
11/04	Melanie Desroches	Web Scraping
11/06	Qianruo Tan	Reinforcement Learning
11/11	Aansh Jha	K-means Clustering
11/11	Owen Babiec	Calling R from Python and Vice Versa
11/13	Stef Baptista	
11/13	Mohammad Parvez	Extracting and Analyzing Census Data

## *Preliminaries*

### **Final Project Presentation Schedule**

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

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.

## *Practical Tips*

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

Plagiarism is to be prevented. Remember that these class notes are publicly available online with your names attached. Here are some resources on how to avoid plagiarism. In particular, in our course, one convenient way to avoid plagiarism is to use our own data (e.g., NYC Open Data). Combined with your own explanation of the code chunks, it would be hard to plagiarize.

## **Homework 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).
  - Make at least 10 commits and form a style of frequent small commits.
- Use quarto source only. See Chapter 3.
- For the convenience of grading, add your standalone html or pdf output to a release in your repo.
- For standalone pdf output, you will need to have LaTeX installed.

## **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

## *Preliminaries*

- 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:

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

## **Sub Topic 2**

Put materials on topic 2 here.

*My Presentation Topic (Template)*

**Sub Topic 3**

Put materials on topic 3 here.

**Conclusion**

Put summaries 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

## *1 Introduction*

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

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

## **1.2 Expectations from This Course**

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

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

### *1.3 Computing Environment*

- **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>

## *1 Introduction*

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

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

Here are several commonly used shell commands:

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

### *1.3 Computing Environment*

2. “Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython.” Third Edition, by Wes McKinney, 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 behind or ahead a remote branch.
- `git add`:
  - Adds new or modified files to the Git staging area.
  - Gives the option to select which files are to be sent to the remote repository
- `git rm`:
  - Used to remove files from the staging index or the local repository.
- `git commit`:
  - Commits changes made to the local repository and saves it like a snapshot.
  - A message is recommended with every commit to keep track of changes made.
- `git push`:
  - Used to send commits made on local repository to the remote repository.

### *2.3 Tips on using Git:*

## **2.3 Tips on using Git:**

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

## **2.4 Pull Request**

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

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



# 3 Reproducible Data Science

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

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

## 3.1 Introduction to Quarto

To get started with Quarto, see documentation at Quarto.

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

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

### *3 Reproducible Data Science*

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

- Markdown basics
- Using Python
- Presentations

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

## **3.2 Compiling the Classnotes**

The sources of the classnotes are at <https://github.com/statds/ids-f24>. This is also the source tree that you will contribute to this semester. I expect that you clone the repository to your own computer, update it frequently, and compile the latest version on your computer (reproducibility).

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

### **3.2.1 Set up your Python Virtual Environment**

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

To create a Python virtual environment for our classnotes:

### *3.2 Compiling the Classnotes*

```
python3 -m venv .ids-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 (folder), 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

## *4 Python Refreshment*

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,

## *4.2 The Python World*

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 library.

## **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.

4 Python Refreshment

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

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

The constant is only represented by a given double precision.

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

2.71828182845904509080

Now use package decimal to export with an arbitrary precision.

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

'7182818284590452353602874713526624977572470936999595749669676277240766303539

## 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([ 5.44494482, -5.32588498, 2.73379488, 7.36824826, -1.02063687,  
 5.35369198, -5.33732664, 6.72444362, 0.27352833, 0.57009663])

The pdf and cdf can be evaluated:

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

array([0.06883129, 0.01864129, 0.09807139, 0.04052654, 0.07499262,
      0.07017877, 0.01854381, 0.04965062, 0.09086504, 0.09356233])
```

## 4.5 Writing a Function

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

The first is a recursive solution.

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

%timeit fib_rs(10)
```

8.23  $\mu$ s  $\pm$  875 ns per loop (mean  $\pm$  std. dev. of 7 runs, 100,000 loops each)

The second uses dynamic programming memoization.

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

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

## 4.5 Writing a Function

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

```
2.23 µs ± 397 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)
```

```
71.9 µs ± 5.13 µs 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 Monty Hall

Here is a function that performs the Monty Hall experiments.

```
import numpy as np

def montyhall(ndoors, ntrials):
    doors = np.arange(1, ndoors + 1) / 10
```

## 4 Python Refreshment

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

Test it out:

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

```
{'noswitch': 245, 'switch': 380}
```

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!
```

## 4.6 Variables versus Objects

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

Now check their addresses

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

```
4937134464
4937134464
```

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
```

```
4449218808
4558470672
```

## 4 Python Refreshment

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

Numerals 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)

## 4 Python Refreshment

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

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

```
9223372036854775808000
```

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

### 4.7.2 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](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
```

## 4.7 Number Representation

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

False

What is really going on?

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

```
Decimal('0.100000000000000055511151231257827021181583404541015625')
```

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

```
Decimal('23.9999999999989341858963598497211933135986328125')
```

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
```

## 4 Python Refreshment

`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.

## 4.8 Virtual Environment

- 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

## *5 Communication in Data Science*

- 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 General Communication Skills*

#### **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

## *5 Communication in Data Science*

- 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”; 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

## *5.4 Communication in Data Science*

- 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

## *5 Communication in Data Science*

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

Pragmatic. (2024). Communication skills for data science. <https://www.pragmaticinstitute.com/resources/articles/data/communication-skills-for-data-science/>.

# 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 first 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 version.

```
## 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
```

## 6.2 Import/Export Data

Different versions mean there are differences 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

Firstly, 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 evaluating 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 os
## Import the cleaned crashes data
## Choose the function, read_feather from Pandas
## Add the relative address of the data file to let your computer deal with the code smoothly
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 function "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	

## 6 Data Manipulation

```

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    473674
1           None                  None    4736768
2           None                  None    473706
3           None                  None    473751
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

```

## 6.2 Import/Export Data

```
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 function `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 function 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("foreign/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 Data Manipulation

### 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("foreign/nyc_crashes_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_id \
0                  NaN                      NaN        4736740
1                  NaN                      NaN        4736768
```

## 6.2 Import/Export Data

```
        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 function `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("foreign/nyc_crashes_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 we use is `read_excel` no matter what Pandas version is.

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

## 6 Data Manipulation

```
import pandas as pd
## Choose the function "read_excel"
## Add the command "engine" to read the file smoothly
df_excel = pd.read_excel("foreign/nyc_crashes_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_id
0                  NaN           NaN           NaN      473674
1                  NaN           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 30 columns]
```

## 6.2 Import/Export Data

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

```
df_excel = pd.read_excel("foreign/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 generating 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("foreign/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("foreign/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.

## 6 Data Manipulation

```
import chardet
## Use "with" statement
with open("foreign/nyc_crashes_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("foreign/nyc_crashes_cleaned.txt", encoding = "utf-8")
## The default 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
1	1	2024-06-30 00:32:00		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				
1				
2				
3				
4				

## 6.2 Import/Export Data

```
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
```

## 6 Data Manipulation

```
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

In this section, all of the specific files generated by the code I provided are stored in the folder which name is “foreign” and can be accessed according to the relative path.

#### 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.

```
import pandas as pd
df = pd.read_csv("foreign/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("foreign/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.

## 6.2 Import/Export Data

```
import pandas as pd
import pyreadstat
df_without_unnamed = pd.read_csv("foreign/Without_Unamed.csv")
sas_file = "foreign/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("foreign/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
```

## 6 Data Manipulation

```
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
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 0x10f31ba10>
```

### 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
```

## 6.2 Import/Export Data

```
from rpy2.robj import pandas2ri
## Activate conversion between Pandas and R
pandas2ri.activate()
df_csv = pd.read_csv("foreign/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 = 'foreign/nyccrashes_cleaned.Rdata')")
print("The CSV file has been transferred to a .Rdata file successfully.")
```

- The error means : 1.Type conversion failure of columns. Some columns were not converted to the correct data type in R as expected, and were instead coerced to strings. 2.The data is still saved, but there are potential data type issues. 3.These errors will not influence importing the .Rdata file.
- If you want to perform necessary type conversions, 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.robj as ro
## Load the .Rdata file
r_file_path = "foreign/nyccrashes_cleaned.Rdata"
ro.r["load"](r_file_path)
```

## 6 Data Manipulation

```
## View loaded variables
loaded_objects = ro.r("ls()")
## Show the loaded vatiabels
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)
```

Loading custom .Rprofile  
Loaded R objects: [1] "R"

	Unnamed: 0	crash_datetime	borough	zip_code	latitude	longitude	location	on_street_name
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

## 6.2 Import/Export Data

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
0		cross_street_name	off_street_name \
1		None	GOLD STREET ...
2		None	None ...
3		None	OCEAN PARKWAY ...
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 ...
0		contributing_factor_vehicle_2	contributing_factor_vehicle_3 \
1		Unspecified	None
2		Unspecified	None
3		None	None
4		None	None
...		...	...
1870		Unspecified	None
1871		Unspecified	None
1872		Unspecified	None
1873		Unspecified	None
1874		None	None

## 6 Data Manipulation

```

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

```

## 6.2 Import/Export Data

```
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 function `to_stata` to save the .dta file.

```
import pandas as pd
df_csv = pd.read_csv("foreign/nyccrashes_cleaned.csv")
## Export the data to the default working directory
df_csv.to_stata("foreign/stata.dta")
```

## 6 Data Manipulation

### 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("foreign/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_id
0
1                               4736740
                               4736768

  vehicle_type_code_1      vehicle_type_code_2
0                 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 Import/Export Data

### 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` from `pyreadstat`. We are supposed to use the CSV file which is without space. We can use the following code.

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

#### 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.

```
import pandas as pd
import pyreadstat
df_4, meta = pyreadstat.read_sav("foreign/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
```

## 6 Data Manipulation

```
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 0x11786e150>
```

### 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.

## 6.2 Import/Export Data

```
import pandas as pd
from scipy.io import savemat
df_csv = pd.read_csv("foreign/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
output_file = os.path.join("foreign", "MATLAB.mat")
## Export the data to the default working directory
savemat(output_file, 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.

```
import pandas as pd
from scipy.io import loadmat
df_csv = pd.read_csv("foreign/nyccrashes_cleaned.csv")
df_5 = loadmat("foreign/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', '2024-06-30 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', 'BROOKLYN', 'QUEENS'])]
```

## 6 Data Manipulation

```
'BRONX',
'BRONX',
        ',',
        ''],
        dtype='<U32'), array([[  nan,
40.861084]]), array([[      nan,         nan, -73.96744, ..., -73
-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
        ', ...,
        '120 AVENUE
        ',
        'nan
        ',
        'nan
        '],
        dtype='<U32'), array(['GOLD ST
        ',
        'nan
        ',
        '2797      OCEAN PARKWAY
        ',
        'nan
        ',
        'nan
        '],
        dtype='<U35'), array([[0, 0
        ,
        'Unspecified
        '],
        dtype='<U53'), array(['Unspecified
        ',
        'Unspecified
        ',
        'nan
        ',
        'Unspecified
        ',
        'Unspecified
        '],
        ', ...,
        'Passing or Lane Usage Improper
        ',
        'Driver Inexperience
        ',
        'Unspecified
        '],
        ', ...,
        'Unspecified
        ',
        'Unspecified
        ',
        'Unspecified
        '],
```

## 6.2 Import/Export Data

```
'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, 4736768, 47370
'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                                ', dtype='<U35'), array(['nan
'nan                                ',
'nan                                ', ...,
```

## 6 Data Manipulation

```
'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 achieve the goal. The code is following.

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

#### 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("foreign/HDF5.h5", key = "data")  
print(df_6.head(2))
```

## 6.2 Import/Export Data

```
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`

## 6 Data Manipulation

```
## 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 specific 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("foreign/*.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 me  
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  \\\n0        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 ...  \\\n0  (0.0, 0.0)                 NaN                  NaN  GOLD STREET ...
```

## 6.2 Import/Export Data

```
1      NaN  BELT PARKWAY RAMP          NaN          NaN  ...
0      contributing_factor_vehicle_2  contributing_factor_vehicle_3  \
0          Unspecified                  NaN
1          Unspecified                  NaN

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

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

0      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)

## 6 Data Manipulation

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

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

```
## 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

## 6.2 Import/Export Data

```
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
```

## 6 Data Manipulation

```
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_injured
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

## 6.2 Import/Export Data

Unnamed: 0	zip_code	latitude	longitude	number_of_persons_injured	number_o
------------	----------	----------	-----------	---------------------------	----------

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, -73.92750)
1874	1874	2024-07-07 14:12:00	BRONX	10468.0	40.861084	-73.91149	(40.861084, -73.91149)

## 6 Data Manipulation

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.

## 6.2 Import/Export Data

- 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

## 6 Data Manipulation

```
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 selection 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)]
```

## 6.2 Import/Export Data

	Unnamed: 0	crash_datetime	borough	zip_code	latitude	longitude	location
0	0	2024-06-30 17:30:00	NaN	NaN	NaN	NaN	(0.0, 0.0)
1	1	2024-06-30 00:32:00	NaN	NaN	NaN	NaN	NaN
2	2	2024-06-30 07:05:00	BROOKLYN	11235.0	40.581060	-73.967440	(40.58106, -73.96744)
3	3	2024-06-30 20:47:00	MANHATTAN	10021.0	40.763630	-73.953300	(40.76363, -73.95330)
4	4	2024-06-30 10:14:00	BROOKLYN	11222.0	40.730460	-73.951490	(40.73046, -73.95149)
...	...	...	...	...	...	...	...
1870	1870	2024-07-07 21:25:00	BRONX	10457.0	40.852520	-73.900020	(40.85252, -73.90002)
1871	1871	2024-07-07 10:31:00	BRONX	10460.0	40.843945	-73.885800	(40.843945, -73.88580)
1872	1872	2024-07-07 20:15:00	QUEENS	11436.0	40.677982	-73.791214	(40.677982, -73.791214)
1873	1873	2024-07-07 14:45:00	BRONX	10452.0	40.843822	-73.927500	(40.843822, -73.927500)
1874	1874	2024-07-07 14:12:00	BRONX	10468.0	40.861084	-73.911490	(40.861084, -73.911490)

### 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

## *6 Data Manipulation*

```
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 Table of Contents**

- What is a Database?
- What is SQL
- CRUD Model
- Creating Tables with SQL

## 6.3 SQL

- Inserting into SQL Tables
- Updating/Deleting SQL Tables
- Using SQL to work with NYC Open Data
- Queries

### 6.3.2 What is a Database?

- Collection of related information
- Ex: Phone books, grocery list, student records
- Relational and non-relational
- Relational databases are structured into columns and rows
- Each row represents an observation and each column represents an attribute
- A key uniquely identifies each row
- Trying to input repeat keys causes an error

### 6.3.3 Example Relational Database

```
import sqlite3
import pandas as pd

# creates a database file called phonebook.db and lets you connect
connection = sqlite3.connect("phonebook.db")

# created a cursor object lets you use the cursor function from SQuite3 module
cursor = connection.cursor()

cursor.execute("DROP TABLE IF EXISTS phonebook")

# Execute function allows you to send commands in the SQL language as strings
cursor.execute("""
```

## 6 Data Manipulation

```
CREATE TABLE IF NOT EXISTS phonebook (
    phonebook_id INTEGER PRIMARY KEY,
    name TEXT,
    phone_num TEXT UNIQUE,
    address TEXT
)
"""

cursor.execute("INSERT INTO phonebook VALUES (1, 'Greta Colic', '2035452367')
cursor.execute("INSERT INTO phonebook VALUES(2, 'Carlos Alavarez', '91456527
cursor.execute("INSERT INTO phonebook VALUES(3, 'Marin Yanko', '5753917568',
cursor.execute("INSERT INTO phonebook VALUES(4, 'Mira Watson', '9146522761',
cursor.execute("INSERT INTO phonebook VALUES(5, 'Harry Smith', '2036658279',

connection.commit()

# alternative way of displaying output
"""cursor.execute("SELECT * FROM phonebook")
rows = cursor.fetchall()
for row in rows:
    print(row)"""

output = pd.read_sql_query("SELECT * FROM phonebook", connection)
print()
print()
print(output)

connection.close()
```

phonebook_id	name	phone_num	address
--------------	------	-----------	---------

### 6.3 SQL

0	1	Greta Colic	2035452367	1 Hartsdale road
1	2	Carlos Alavarez	9145652761	13 Porter street
2	3	Marin Yanko	5753917568	110 Ocean avenue
3	4	Mira Watson	9146522761	12 Hindmarsh avenue
4	5	Harry Smith	2036658279	180 Wallace road

#### 6.3.4 What is SQL?

- Structured Query Language
- Allows users to interact with databases to store and retrieve data
- The essential operations follow the CRUD acronym

#### 6.3.5 CRUD

- Create, read, update and delete
- Essential operations for SQL to manage a database
- Create: adds a new record (row) to a database with unique attributes
- Read: Returns records based on specified search criteria
- Update: Allows you to change attribute(s) of the record
- Delete: Allows you to remove records from the database

#### 6.3.6 Implementing SQL Through Python to Create a Table

```
import sqlite3

# creates a database file called phonebook.db and lets you connect
connection = sqlite3.connect("phonebook.db")

# creates a cursor object using the cursor function from SQLite3 module
cursor = connection.cursor()
```

## 6 Data Manipulation

```
"""
CREATE TABLE tablename
(
    attribute1 datatype PRIMARY KEY,
    attribute2 datatype,
    attribute3 datatype
);
"""
# You MUST include a primary key to uniquely identify entries

# Execute function allows you to send commands in the SQL language as strings
cursor.execute("""
CREATE TABLE IF NOT EXISTS phonebook (
    phonebook_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    phone_num TEXT UNIQUE,
    address TEXT
)
""")
connection.commit()
connection.close()
```

### 6.3.7 How to Insert into the SQL table?

```
import sqlite3
import pandas as pd

# Connects to the previously created phonebook.db
connection = sqlite3.connect("phonebook.db")

# creates a cursor object using the cursor function from SQLite3 module
```

### 6.3 SQL

```
cursor = connection.cursor()

"""
INSERT INTO database VALUES("value1", "value2", "value3");

"""

cursor.execute("INSERT INTO phonebook VALUES (1, 'Greta Colic', '2035452367', '1 Hartsdale rd');
cursor.execute("INSERT INTO phonebook VALUES(2, 'Carlos Alavarez', '9145652761', '13 Porter st');
cursor.execute("INSERT INTO phonebook VALUES(3, 'Marin Yanko', '5753917568', '110 Ocean ave');
cursor.execute("INSERT INTO phonebook VALUES(4, 'Mira Watson', '9146522761', '12 Hindmarsh ave');
cursor.execute("INSERT INTO phonebook VALUES(5, 'Harry Smith', '2036658279', '180 Wallace road');

# How to input data if there's a missing value?
"""
INSERT INTO database(attribute1, attribute2) VALUES(val1, val2);
"""

# only works if the missing value is not a primary key
cursor.execute("INSERT INTO phonebook(phonebook_id, name, phone_num) VALUES(6, 'Stacy Yang', '9145652762');

connection.commit()

# Allows you to see the created table
output = pd.read_sql_query("SELECT * FROM phonebook", connection)
print(output)
connection.close()
```

#### 6.3.8 How to Update/Delete Using SQL?

## 6 Data Manipulation

```
# Updating an attribute (WHERE statement is optional)
connection = sqlite3.connect("phonebook.db")

# created a cursor object lets you use the cursor function from SQlite3 module
cursor = connection.cursor()

# Updates Greta's number
cursor.execute("UPDATE phonebook SET phone_num = '2035151234' WHERE name = 'Greta')

# Deletes Harry Smith from the phonebook
cursor.execute("DELETE FROM phonebook WHERE name = 'Harry Smith';")

# Changes Carlos's last name
cursor.execute("UPDATE phonebook SET name = 'Carlos Ramos' WHERE name = 'Carlo')

# Updating multiple columns
update_multiple_query = """
UPDATE phonebook
SET phone_num = '7777777777', address = '45 Main St'
WHERE name = 'Marin Yanko';
"""

cursor.execute(update_multiple_query)

# deleting a table
cursor.execute("DROP TABLE phonebook;")

connection.commit()
connection.close()
```

### 6.3.9

```

import sqlite3
import pandas as pd

# creates a database file called phonebook.db and lets you connect
connection = sqlite3.connect("phonebook.db")

# created a cursor object lets you use the cursor function from SQuite3 module
cursor = connection.cursor()

cursor.execute("DROP TABLE IF EXISTS phonebook;")
# Execute function allows you to send commands in the SQL language as strings
cursor.execute("""
CREATE TABLE IF NOT EXISTS phonebook (
    phonebook_id INT,
    name VARCHAR(30),
    phone_num VARCHAR(15) PRIMARY KEY,
    address VARCHAR(30)
)
""")

cursor.execute("INSERT INTO phonebook VALUES (1, 'Greta Colic', '2035452367', '1 Hartsdale rd')
cursor.execute("INSERT INTO phonebook VALUES(2, 'Carlos Alavarez', '9145652761', '13 Porter ave')
cursor.execute("INSERT INTO phonebook VALUES(3, 'Marin Yanko', '5753917568', '110 Ocean ave')
cursor.execute("INSERT INTO phonebook VALUES(4, 'Mira Watson', '9146522761', '12 Hindmarsh ave')
cursor.execute("INSERT INTO phonebook VALUES(5, 'Harry Smith', '2036658279', '180 Wallace road')
cursor.execute("INSERT INTO phonebook(phonebook_id, name, phone_num) VALUES(6, 'Stacy Yang', '9145652762')

connection.commit()

# alternative way of displaying output

```

## 6 Data Manipulation

```
"""cursor.execute("SELECT * FROM phonebook")
rows = cursor.fetchall()
for row in rows:
    print(row)"""

output = pd.read_sql_query("SELECT * FROM phonebook", connection)
print(output)
print(" ")
print(" ")

cursor.execute("UPDATE phonebook SET phone_num = '2035151234' WHERE name = 'Oscar'
# Deletes Harry Smith from the phonebook
cursor.execute("DELETE FROM phonebook WHERE name = 'Harry Smith';")

# Updating multiple columns
update_multiple_query = """
UPDATE phonebook
SET phone_num = '7777777777', address = '45 Main St'
WHERE name = 'Marin Yanko';
"""

cursor.execute(update_multiple_query)

cursor.execute("UPDATE phonebook SET name = 'Carlos Ramos' WHERE name = 'Carlos
connection.commit()
output = pd.read_sql_query("SELECT * FROM phonebook", connection)
print(output)

connection.close()
```

	phonebook_id	name	phone_num	address
0	1	Greta Colic	2035452367	1 Hartsdale road

### 6.3 SQL

1	2	Carlos Alavarez	9145652761	13 Porter street
2	3	Marin Yanko	5753917568	110 Ocean avenue
3	4	Mira Watson	9146522761	12 Hindmarsh avenue
4	5	Harry Smith	2036658279	180 Wallace road
5	6	Stacy Yang	9178852765	None

	phonebook_id	name	phone_num	address
0	1	Greta Colic	2035151234	1 Hartsdale road
1	2	Carlos Ramos	9145652761	13 Porter street
2	3	Marin Yanko	7777777777	45 Main St
3	4	Mira Watson	9146522761	12 Hindmarsh avenue
4	6	Stacy Yang	9178852765	None

#### 6.3.10 Using SQL to Work on a CSV file

```
import pandas as pd
import sqlite3

# Creates an SQL Database
conn = sqlite3.connect("nyc.db")

# Reads the CSV file using the path
data = pd.read_csv(r"data/nyccrashes_2024w0630_by20240916.csv")

cursor = conn.cursor()

# automatically converts data to an SQL table
data.to_sql('nyccrashes', conn, if_exists='replace', index=False)
```

## 6 Data Manipulation

```
import pandas as pd
import sqlite3

# Creates an SQL Database
conn = sqlite3.connect("nyc.db")

# Reads the CSV file using the path
data = pd.read_csv(r"data/nyccrashes_2024w0630_by20240916.csv")

cursor = conn.cursor()

# automatically converts data to an SQL table
data.to_sql('nyccrashes', conn, if_exists='replace', index=False)
```

1875

### 6.3.11 Queries

- Commands used to pull needed data out of a database

```
# Query for everythin in the table
query1 = pd.read_sql_query("SELECT * FROM nyccrashes;", conn)
print(query1.head(3))

# Query to count total crashes
total_crashes = pd.read_sql_query("SELECT COUNT(*) FROM nyccrashes;", conn)
print(total_crashes)

# Query to only retrieve fixed attributes
specific_columns = pd.read_sql_query("SELECT \"ZIP CODE\", \"CRASH TIME\" FROM nyccrashes")
print(specific_columns.head(3))
```

100

### 6.3 SQL

```
# Groups Crashes by borough
crashes_by_borough = pd.read_sql_query("""
    SELECT BOROUGH, COUNT(*) AS Total_Crashes
    FROM nyccrashes
    GROUP BY BOROUGH;
""", conn)
print(crashes_by_borough)

# Query to show the fatal crashes
fatal_crashes = pd.read_sql_query("SELECT * FROM nyccrashes WHERE \"NUMBER OF PERSONS KILLED\" > 0")
print(fatal_crashes.head())
```

#### 6.3.12 Queries Output

```
# Query for everythin in the table
# query1 = pd.read_sql_query("SELECT * FROM nyccrashes;", conn)
# print(query1.head(3))

# Query to count total crashes
print("Here's the output for the count query.")
total_crashes = pd.read_sql_query("SELECT COUNT(*) FROM nyccrashes;", conn)
print(total_crashes)
print()

# Groups Crashes by borough
print("Here's the output for the query to group crashes by borough.")
crashes_by_borough = pd.read_sql_query("""
    SELECT BOROUGH, COUNT(*) AS Total_Crashes
    FROM nyccrashes
    GROUP BY BOROUGH;
""", conn)
```

## 6 Data Manipulation

```
print(crashes_by_borough)
print()
# Query to only retrieve fixed attributes
print("Here's the output for the specific columns query.")
specific_columns = pd.read_sql_query("SELECT \"ZIP CODE\", \"CRASH TIME\" FROM nyccrashes")
print(specific_columns.head(5))

# Query to show the fatal crashes
# fatal_crashes = pd.read_sql_query("SELECT * FROM nyccrashes WHERE \"NUMBER OF VICTIMS\" >= 3")
# print(fatal_crashes.head())
conn.commit()
```

Here's the output for the count query.

```
COUNT(*)
0      1875
```

Here's the output for the query to group crashes by borough.

	BOROUGH	Total_Crashes
0	None	541
1	BRONX	213
2	BROOKLYN	462
3	MANHATTAN	228
4	QUEENS	381
5	STATEN ISLAND	50

Here's the output for the specific columns query.

	ZIP CODE	CRASH TIME
0	Nan	17:30
1	Nan	0:32
2	11235.0	7:05
3	Nan	20:47
4	11222.0	10:14

## 6.4 NYC Crash Data

### 6.3.13 Conclusion

- SQL works with relational databases
- SQL performs the CRUD functions: create, read, update, and delete to work with databases
- A query is a request from a database for information
- SQL can be used to manipulate data from various formats including CSV files

### 6.3.14 Further reading

- Python Software Foundation. (n.d.). sqlite3 — DB-API 2.0 interface for SQLite databases. Python Documentation. <https://docs.python.org/3/library/sqlite3.html>

## 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/nyc_crashes_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(' ', '_')
```

## 6 Data Manipulation

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

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

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 column
df['crash_datetime'] = pd.to_datetime(df['crash_date'] + ' ' +
                                       df['crash_time'], format='%m/%d/%Y %H:%M', errors='coerce')

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

## 6.4 NYC Crash Data

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")
```

## 6 Data Manipulation

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

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

Let's try it out:

```
# Example usage
latitude = 40.730610
longitude = -73.935242
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 `borough` 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

## 6 Data Manipulation

dataset is available upfront and can be stored in a file system for repeated reads and writes.

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

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

```
pip install pyarrow
```

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

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

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

# File paths
csv_file = 'data/nyc_crashes_2024w0630_by20240916.csv'
feather_file = 'data/nyc_crashes_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)
```

## 6.6 Using the Census Data

```
# 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:

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

## *6 Data Manipulation*

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.

## 6.6 Using the Census Data

Install the `uszipcode` package into the current virtual environment, if it has not been installed 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 directly 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
```

## 6 Data Manipulation

```
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.to_feather('data/nyccrashes_merged.feather')

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, -73.96744)
1	2024-06-30 20:47:00	MANHATTAN	10021	40.76363	-73.95330	(40.76363, -73.95330)
2	2024-06-30 10:14:00	BROOKLYN	11222	40.73046	-73.95149	(40.73046, -73.95149)
3	2024-06-30 15:52:00	BRONX	10468	40.86685	-73.89597	(40.86685, -73.89597)
4	2024-06-30 16:30:00	BROOKLYN	11226	40.63969	-73.95321	(40.63969, -73.95321)

## 6.7 Extracting and Analyzing Census Tract Data

### 6.7 Extracting and Analyzing Census Tract Data

This section was authored by **Mohammad Shahriyar Parvez**, a Ph.D. scholar in the Department of Geography, Sustainability, and Urban Studies at the University of Connecticut. For any questions or difficulties in executing the content of this section, please feel free to reach out at shahriyar@uconn.edu.

This section explains how to access data from the US Decennial Census and the American Community Survey (ACS). Basic operations will be applied to calculate and map poverty rates in Storrs, CT. Data will be retrieved from the US Census Bureau's 2022 American Community Survey (ACS) (refer to this page for the dataset).

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

#### 6.7.1 Accessing the census Data

##### 6.7.1.1 Import census data

Importing census data into Python requires a Census API key. A key can be obtained from Census API Key. You have to provide your Institution name and email address. A unique 40 digit text string API will be sent to you. You have to keep track of this number and store it in a safe place.

## 6 Data Manipulation

```
# Set up the API key for accessing Census data
with open("CensusAPI.txt") as f:
    api_key = f.read().strip()
c = Census(api_key)
```

Once the Census API key is configured, access census data at the tract level for Connecticut using the 2022 ACS. Focus on these variables: total income-to-poverty ratio (C17002\_001E), income less than 0.50 of the poverty level (C17002\_002E), income between 0.50 and 0.99 of the poverty level (C17002\_003E), and total population (B01003\_001E). For details on why these variables are relevant to poverty analysis, refer to the Census Bureau's guidelines on poverty measurement and the ACS variable documentation.

Use the `census` package's convenience methods to pull data for different geographic levels. The FIPS code for Connecticut is 09, but if necessary, use the `us` library to identify FIPS codes for other states.

```
# Obtain Census variables from the ACS at the tract level for Storrs, CT
ct_census = c.acs5.state_county_tract(
    fields=('NAME', 'C17002_001E', 'C17002_002E', 'C17002_003E', 'B01003_001E',
            state_fips=states.CT.fips,
            county_fips="*",
            tract="*",
            year=2022
)
```

With the data retrieved and assigned to a variable, load it into a DataFrame using the `pandas` library. This will allow for easy manipulation and analysis of the data.

```
# Create a dataframe from the census data
ct_df = pd.DataFrame(ct_census)
```

## 6.7 Extracting and Analyzing Census Tract Data

```
# Show the dataframe
print(ct_df.head(20))
print('Shape: ', ct_df.shape)
```

	NAME	C17002_001E	\
0	Census Tract 4001.01; Capitol Planning Region;...	2916.0	
1	Census Tract 4001.02; Capitol Planning Region;...	4266.0	
2	Census Tract 4002; Capitol Planning Region; Co...	6213.0	
3	Census Tract 4003; Capitol Planning Region; Co...	6677.0	
4	Census Tract 4153; Capitol Planning Region; Co...	2615.0	
5	Census Tract 4154; Capitol Planning Region; Co...	6046.0	
6	Census Tract 4155; Capitol Planning Region; Co...	3245.0	
7	Census Tract 4156; Capitol Planning Region; Co...	4201.0	
8	Census Tract 4157; Capitol Planning Region; Co...	3284.0	
9	Census Tract 4158; Capitol Planning Region; Co...	2606.0	
10	Census Tract 4159; Capitol Planning Region; Co...	1427.0	
11	Census Tract 4160; Capitol Planning Region; Co...	4738.0	
12	Census Tract 4161; Capitol Planning Region; Co...	4995.0	
13	Census Tract 4162; Capitol Planning Region; Co...	2839.0	
14	Census Tract 4163; Capitol Planning Region; Co...	4533.0	
15	Census Tract 4164; Capitol Planning Region; Co...	3180.0	
16	Census Tract 4165; Capitol Planning Region; Co...	4364.0	
17	Census Tract 4166; Capitol Planning Region; Co...	3110.0	
18	Census Tract 4167; Capitol Planning Region; Co...	6364.0	
19	Census Tract 4168; Capitol Planning Region; Co...	3394.0	

	C17002_002E	C17002_003E	B01003_001E	state	county	tract
0	41.0	22.0	2945.0	09	110	400101
1	73.0	305.0	4266.0	09	110	400102
2	55.0	136.0	6250.0	09	110	400200
3	135.0	197.0	6688.0	09	110	400300
4	465.0	360.0	2621.0	09	110	415300

## 6 Data Manipulation

```
5      431.0      318.0     6046.0    09     110  415400
6      408.0      353.0     3245.0    09     110  415500
7      300.0      667.0     4303.0    09     110  415600
8      134.0      169.0     3290.0    09     110  415700
9      459.0      313.0     2775.0    09     110  415800
10     131.0      405.0     1437.0    09     110  415900
11     382.0      646.0     4738.0    09     110  416000
12     509.0      992.0     5017.0    09     110  416100
13     133.0      633.0     2911.0    09     110  416200
14     373.0      796.0     4533.0    09     110  416300
15      50.0      110.0     3185.0    09     110  416400
16     350.0      180.0     4404.0    09     110  416500
17     403.0      196.0     3171.0    09     110  416600
18     424.0     1109.0     6481.0    09     110  416700
19     401.0       13.0     3394.0    09     110  416800
Shape: (884, 8)
```

Displaying the DataFrame reveals 884 rows, indicating 884 census tracts, along with 8 columns. This structure provides a comprehensive view of the data across all selected tracts.

### 6.7.1.2 Import Shapefile

Read the Connecticut census tracts shapefile into Python and reproject it to the UTM Zone 18N projection. The shapefile is available for download from the Census Bureau's website, specifically on the Cartographic Boundary Files page or the TIGER/Line Shapefiles page.

```
# Access shapefile of Connecticut census tracts
ct_tract = gpd.read_file("https://www2.census.gov/geo/tiger/TIGER2024/TRACT/
                           "tl_2024_09_tract.zip")

# Reproject shapefile to UTM Zone 18N
```

## 6.7 Extracting and Analyzing Census Tract Data

```
ct_tract = ct_tract.to_crs(epsg=32618)

# Print GeoDataFrame of shapefile
print(ct_tract.head(2))
print('Shape: ', ct_tract.shape)
print("\nThe shapefile projection is: {}".format(ct_tract.crs))
```

```
STATEFP COUNTYFP TRACTCE           GEOID           GEOIDFQ  NAME  \
0      09        110  535200  09110535200  1400000US09110535200  5352
1      09        110  881200  09110881200  1400000US09110881200  8812

          NAMELSAD  MTFCC FUNCSTAT     ALAND    AWATER    INTPTLAT  \
0  Census Tract 5352   G5020           S  43206583  1355239  +41.9294855
1  Census Tract 8812   G5020           S  2273066   28584  +41.8089384

    INTPTLON                               geometry
0 -072.4062911  POLYGON ((710429.8 4638466.647, 710437.611 463...
1 -072.2516171  POLYGON ((727399.183 4633086.018, 727416.326 4...
Shape: (884, 14)
```

The shapefile projection is: EPSG:32618

Printing the shapefile confirms it also contains 884 rows, representing 884 census tracts. This matches the number of census records, which is a good start!

However, there's a potential issue: while we have the census data and the shapefile of corresponding census tracts, they are currently stored in two separate variables (`ct_df` and `ct_tract`). This separation complicates mapping, as the datasets are not yet linked.

## 6 Data Manipulation

### 6.7.2 Performing Dataframe Operations

#### 6.7.2.1 Create new column from old columns

To address this issue, join the two DataFrames using a shared field, or “key,” that is common to both datasets.

After reviewing the two datasets, the `GEOID` column in `ct_tract` and the combination of the `state`, `county`, and `tract` columns in `ct_df` can serve as a unique key for joining. Currently, these columns are not in a format that can be directly matched, so the `state`, `county`, and `tract` columns from `ct_df` will need to be merged into a single column to align with the `GEOID` column in `ct_tract`.

To create this new column—or access any existing column in a DataFrame—use indexing with `[]` and the column name as a string. (Alternatively, columns can be accessed by index position; see the pandas documentation on indexing and selecting data for more details.)

```
# Combine state, county, and tract columns together
ct_df["GEOID"] = ct_df["state"] + ct_df["county"] + ct_df["tract"]

# Print head of dataframe
ct_df.head(10)
```

	NAME	C17002_001E	C17002_002E	C
0	Census Tract 4001.01; Capitol Planning Region;...	2916.0	41.0	22
1	Census Tract 4001.02; Capitol Planning Region;...	4266.0	73.0	30
2	Census Tract 4002; Capitol Planning Region; Co...	6213.0	55.0	13
3	Census Tract 4003; Capitol Planning Region; Co...	6677.0	135.0	19
4	Census Tract 4153; Capitol Planning Region; Co...	2615.0	465.0	36
5	Census Tract 4154; Capitol Planning Region; Co...	6046.0	431.0	33
6	Census Tract 4155; Capitol Planning Region; Co...	3245.0	408.0	35
7	Census Tract 4156; Capitol Planning Region; Co...	4201.0	300.0	60

## 6.7 Extracting and Analyzing Census Tract Data

	NAME	C17002_001E	C17002_002E	C17002_003E	B01
8	Census Tract 4157; Capitol Planning Region; Co...	3284.0	134.0	169.0	329
9	Census Tract 4158; Capitol Planning Region; Co...	2606.0	459.0	313.0	277

By printing the first few rows of the DataFrame, the newly created `GEOID` column is visible, showing the combined values from the `state`, `county`, and `tract` columns. This new column now matches the format of the `GEOID` column in `ct_tract`, setting up the data for a successful join.

### 6.7.2.2 Remove dataframe columns that are no longer needed

To minimize clutter, delete the `state`, `county`, and `tract` columns from `ct_df`, as they're no longer needed after creating the `GEOID` column. When modifying a DataFrame, reassign the modified DataFrame back to the original variable (or a new variable if preferred) to save the changes. Alternatively, use `inplace=True` within the `drop` function to apply changes directly without reassignment. For additional details, refer to the pandas documentation on `drop`.

```
# Remove columns
ct_df = ct_df.drop(columns = ["state", "county", "tract"])

# Show updated dataframe
ct_df.head(2)
```

	NAME	C17002_001E	C17002_002E	C17002_003E	B01
0	Census Tract 4001.01; Capitol Planning Region;...	2916.0	41.0	22.0	2945
1	Census Tract 4001.02; Capitol Planning Region;...	4266.0	73.0	305.0	4266

## 6 Data Manipulation

### 6.7.2.3 Check column data types

Ensure that the key column (GEOID) in both DataFrames is of the same data type to allow for a successful join. Start by checking the data type of the GEOID column in each DataFrame. If the data types differ, adjust one or both columns as needed to make them consistent.

```
# Check column data types for census data
print(ct_df.dtypes)

# Check column data types for census shapefile
print(ct_tract.dtypes)
```

```
NAME          object
C17002_001E   float64
C17002_002E   float64
C17002_003E   float64
B01003_001E   float64
GEOID         object
dtype: object
STATEFP        object
COUNTYFP       object
TRACTCE        object
GEOID          object
GEOIDFQ        object
NAME           object
NAMELSAD       object
MTFCC          object
FUNCSTAT       object
ALAND          int64
AWATER          int64
INTPTLAT      object
INTPTLON      object
```

## 6.7 Extracting and Analyzing Census Tract Data

```
geometry    geometry
dtype: object
```

### 6.7.2.4 Merge dataframes

Now it's time to merge the two DataFrames using the `GEOID` columns as the primary key. Use the `merge` method in GeoPandas, applying it to the `va_tract` shapefile dataset to combine the data based on the `GEOID` column.

```
# Join the attributes of the dataframes together
ct_merge = ct_tract.merge(ct_df, on = "GEOID")

# Show result
print(ct_merge.head(2))
print('Shape: ', ct_merge.shape)
```

```
STATEFP COUNTYFP TRACTCE           GEOID           GEOIDFQ NAME_x \
0      09        110  535200  09110535200  1400000US09110535200   5352
1      09        110  881200  09110881200  1400000US09110881200   8812

          NAMELSAD  MTFCC FUNCSTAT      ALAND     AWATER      INTPTLAT \
0  Census Tract 5352  G5020           S  43206583  1355239  +41.9294855
1  Census Tract 8812  G5020           S  2273066   28584  +41.8089384

      INTPTLON                               geometry \
0 -072.4062911  POLYGON ((710429.8 4638466.647, 710437.611 463...
1 -072.2516171  POLYGON ((727399.183 4633086.018, 727416.326 4...

          NAME_y  C17002_001E \
0  Census Tract 5352; Capitol Planning Region; Co...       6650.0
1  Census Tract 8812; Capitol Planning Region; Co...       502.0
```

## 6 Data Manipulation

```
C17002_002E  C17002_003E  B01003_001E  
0            102.0        213.0      6664.0  
1            318.0         15.0     10590.0  
Shape:  (884, 19)
```

The merged DataFrame still has 884 rows, indicating that all (or nearly all) rows were matched correctly. The census data has now been appended to the shapefile data within the DataFrame.

### *Additional Notes on Joining DataFrames*

- The columns used as keys do not need to have the same name, as long as they contain matching data.
- This join was a **one-to-one relationship**, where each attribute in one DataFrame matched exactly one attribute in the other. Joins with **many-to-one**, **one-to-many**, or **many-to-many** relationships are also possible, though they may require additional considerations. For more details, refer to the Esri ArcGIS documentation on joins and relates.

#### 6.7.2.5 Subset dataframe

With the DataFrames merged, further clean up the data by removing unnecessary columns. Instead of using the `drop` method, select only the columns needed and create a new DataFrame containing just those selected columns. This approach streamlines the DataFrame and keeps only the relevant data for analysis.

```
# Create new dataframe from select columns  
ct_poverty_tract = ct_merge[["STATEFP", "COUNTYFP", "TRACTCE",  
                           "GEOID", "geometry", "C17002_001E",  
                           "C17002_002E", "C17002_003E",
```

## 6.7 Extracting and Analyzing Census Tract Data

```
"B01003_001E"]]

# Show dataframe
print(ct_poverty_tract.head(2))
print('Shape: ', ct_poverty_tract.shape)

      STATEFP COUNTYFP TRACTCE      GEOID \
0          09        110  535200  09110535200
1          09        110  881200  09110881200

                                         geometry  C17002_001E \
0  POLYGON ((710429.8 4638466.647, 710437.611 463...       6650.0
1  POLYGON ((727399.183 4633086.018, 727416.326 4...       502.0

      C17002_002E  C17002_003E  B01003_001E
0          102.0        213.0       6664.0
1          318.0         15.0      10590.0
Shape: (884, 9)
```

The number of columns has reduced from 19 to 9, streamlining the data for analysis.

### 6.7.2.6 Calculate Poverty Rates Using Column Math

To estimate the poverty rate, divide the sum of C17002\_002E (ratio of income to poverty in the past 12 months, < 0.50) and C17002\_003E (ratio of income to poverty in the past 12 months, 0.50 - 0.99) by B01003\_001E (total population). This calculation provides the proportion of the population below the poverty line.

**Note:** C17002\_001E (ratio of income to poverty in the past 12 months, total) should theoretically represent the entire population, but it does not

## 6 Data Manipulation

exactly match B01003\_001E (total population). This discrepancy is minor, so it will be disregarded for the purpose of this calculation.

```
# Get poverty rate and store ctues in new column
ct_poverty_tract["Poverty_Rate"] = (
    ct_poverty_tract["C17002_002E"] + ct_poverty_tract["C17002_003E"])
    / ct_poverty_tract["B01003_001E"] * 100
)

# Show dataframe
ct_poverty_tract.head(2)
```

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

A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/>

	STATEFP	COUNTYFP	TRACTCE	GEOID	geometry
0	09	110	535200	09110535200	POLYGON ((710429.8 463846
1	09	110	881200	09110881200	POLYGON ((727399.183 4633

### 6.7.2.7 Plotting the Results

With the spatial component linked to the census data, the results can now be visualized on a map. Plot the data to display poverty rates across counties, taking advantage of the merged spatial information for a clear geographic representation of poverty distribution.

## 6.7 Extracting and Analyzing Census Tract Data

```
# Create subplots
fig, ax = plt.subplots(1, 1, figsize = (12, 8))

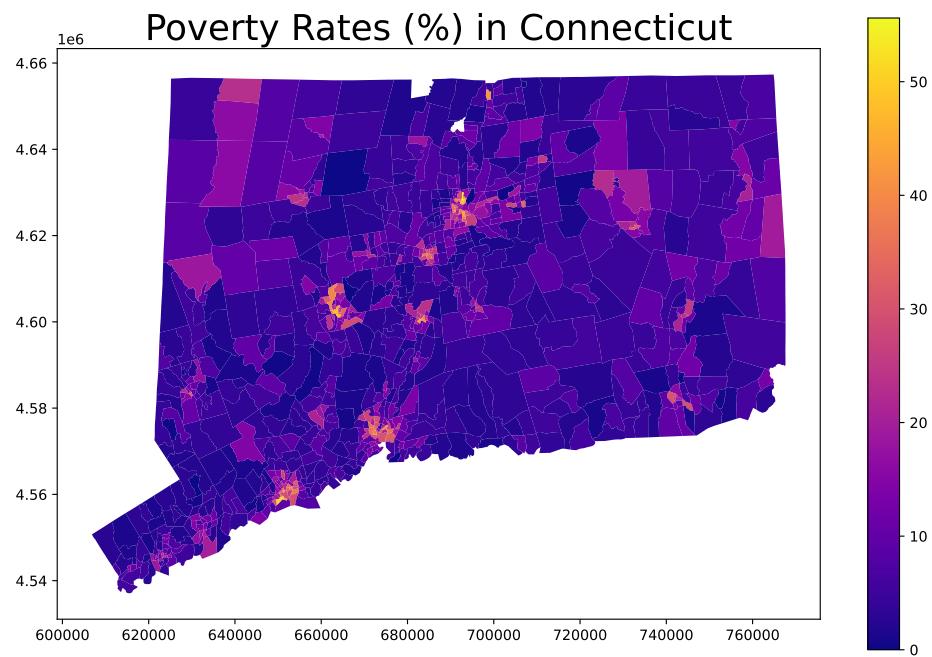
# Plot data
ct_poverty_tract.plot(column = "Poverty_Rate",
                      ax = ax,
                      cmap = "plasma",
                      legend = True)

# Stylize plots
plt.style.use('bmh')

# Set title
ax.set_title('Poverty Rates (%) in Connecticut', fontdict = {
    'fontsize': '25', 'fontweight' : '3'})
```

Text(0.5, 1.0, 'Poverty Rates (%) in Connecticut')

## 6 Data Manipulation



# 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 grammar 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 grammar, we can layer our graphics to create complex and detailed visualizations.

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

## 7 Visualization

- **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
```

## 7.1 Data Visualization with 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/nyc_crashes_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.

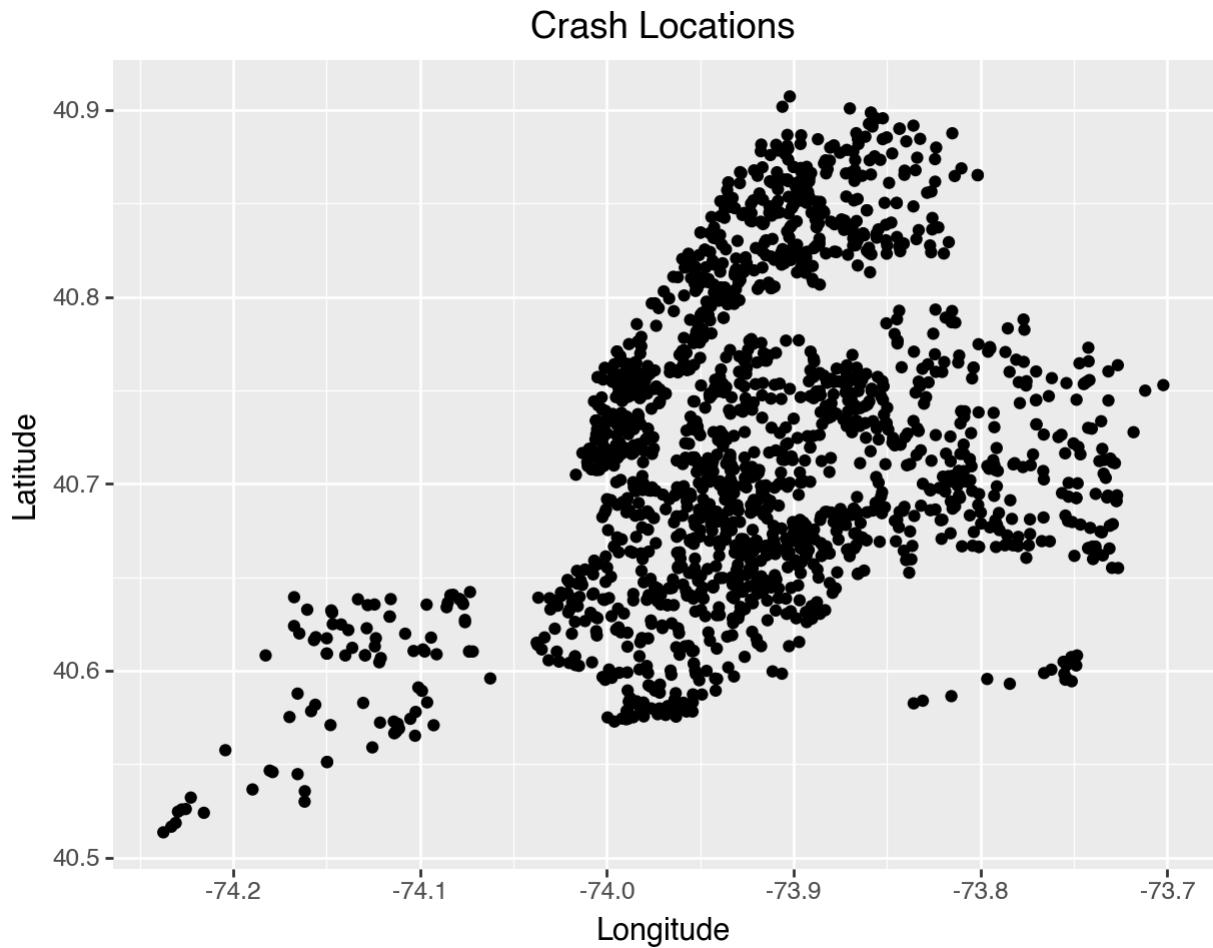
#### Creating a Basic Scatterplot

## 7 Visualization

```
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))
```

## 7.1 Data Visualization with Plotnine



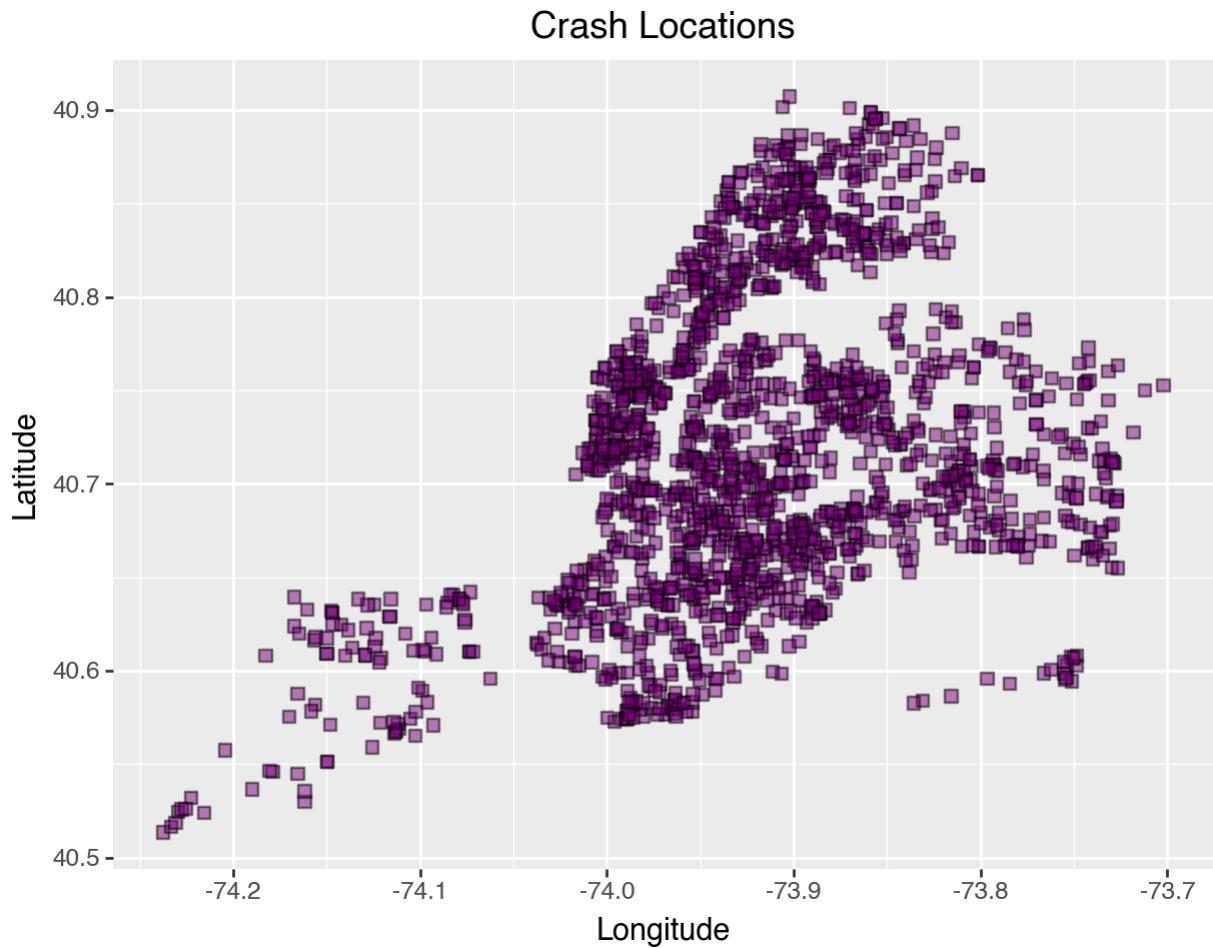
### Customizing a Scatterplot

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()`.

## 7 Visualization

```
(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= changes 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))
```

## 7.1 Data Visualization with Plotnine



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`:

## 7 Visualization

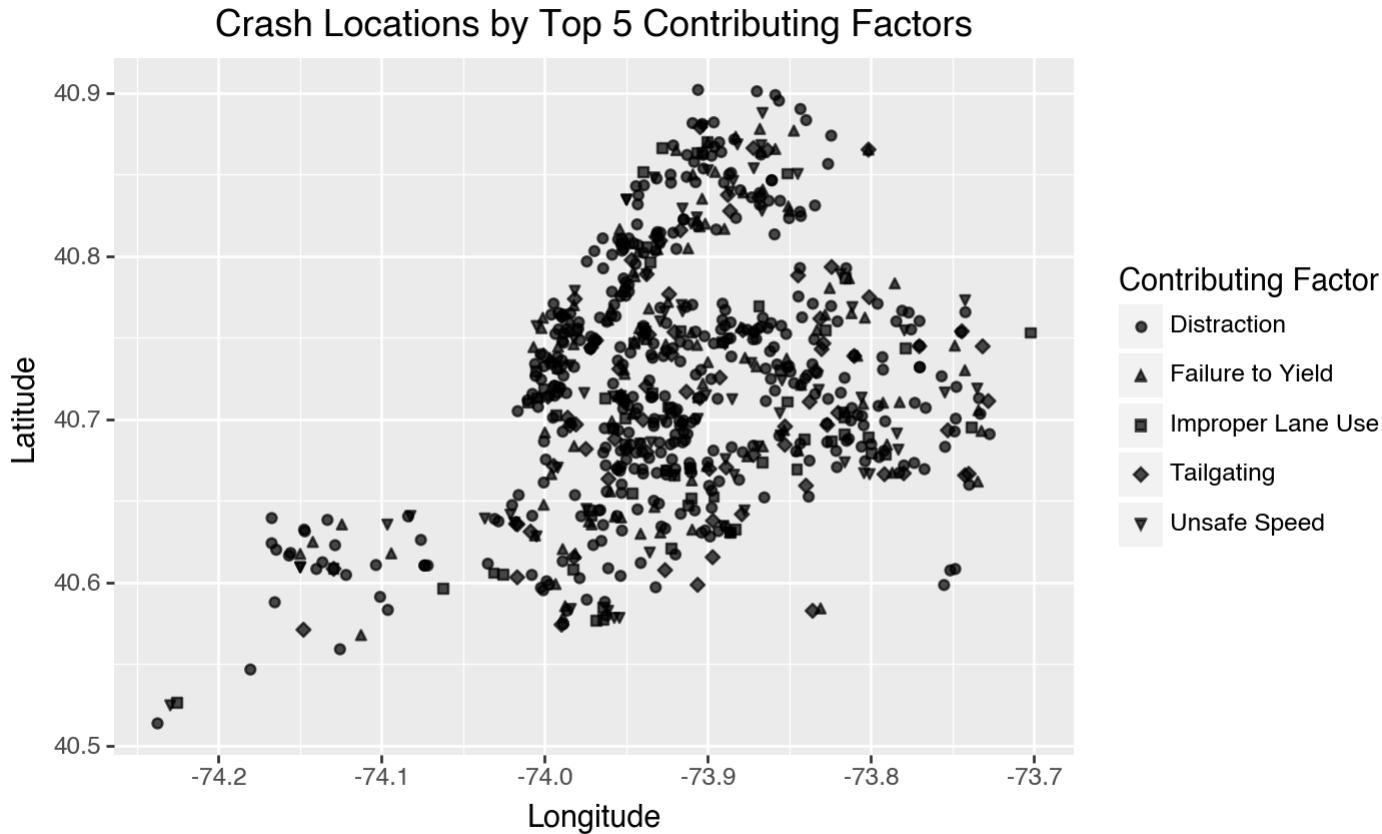
```
# List of top 5 reasons for the contributing factor
# Abbreviating names for clarity
factor1 = {"Driver Inattention/Distraction": "Distraction",
           "Failure to Yield Right-of-Way": "Failure to Yield",
           "Following Too Closely": "Tailgating",
           "Unsafe Speed": "Unsafe Speed",
           "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(factor1)]

# Change to shortened names for better visibility
confact.loc[:, 'contributing_factor_vehicle_1'] = confact[
    'contributing_factor_vehicle_1'].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)))
```

## 7.1 Data Visualization with Plotnine



### 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

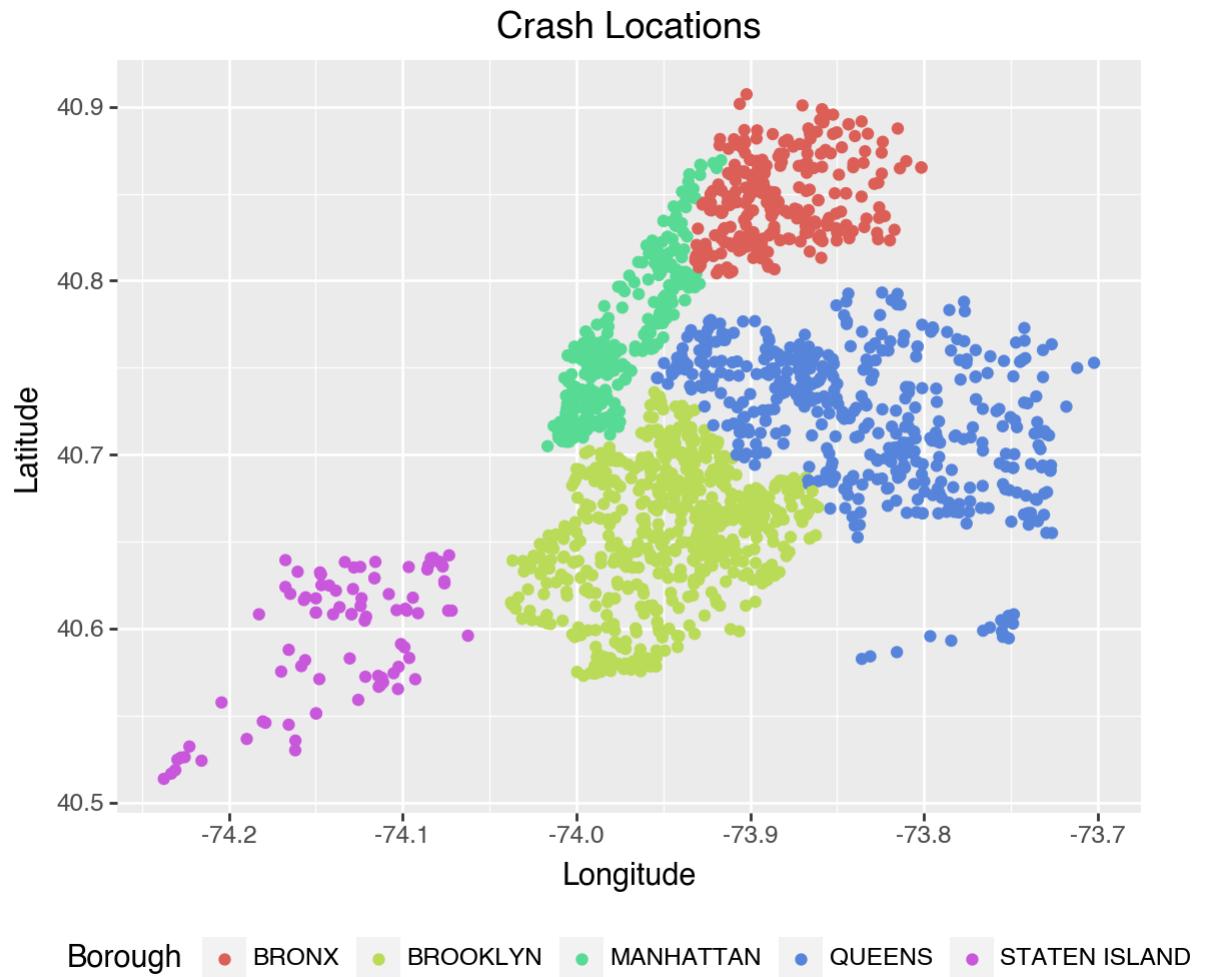
## 7 Visualization

plot.

Changing color of point according to borough:

```
# color= changes 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) +  
# legend_position= changes where the legend is located  
  theme(figure_size = (7,5), legend_position='bottom'))
```

## 7.1 Data Visualization with Plotnine

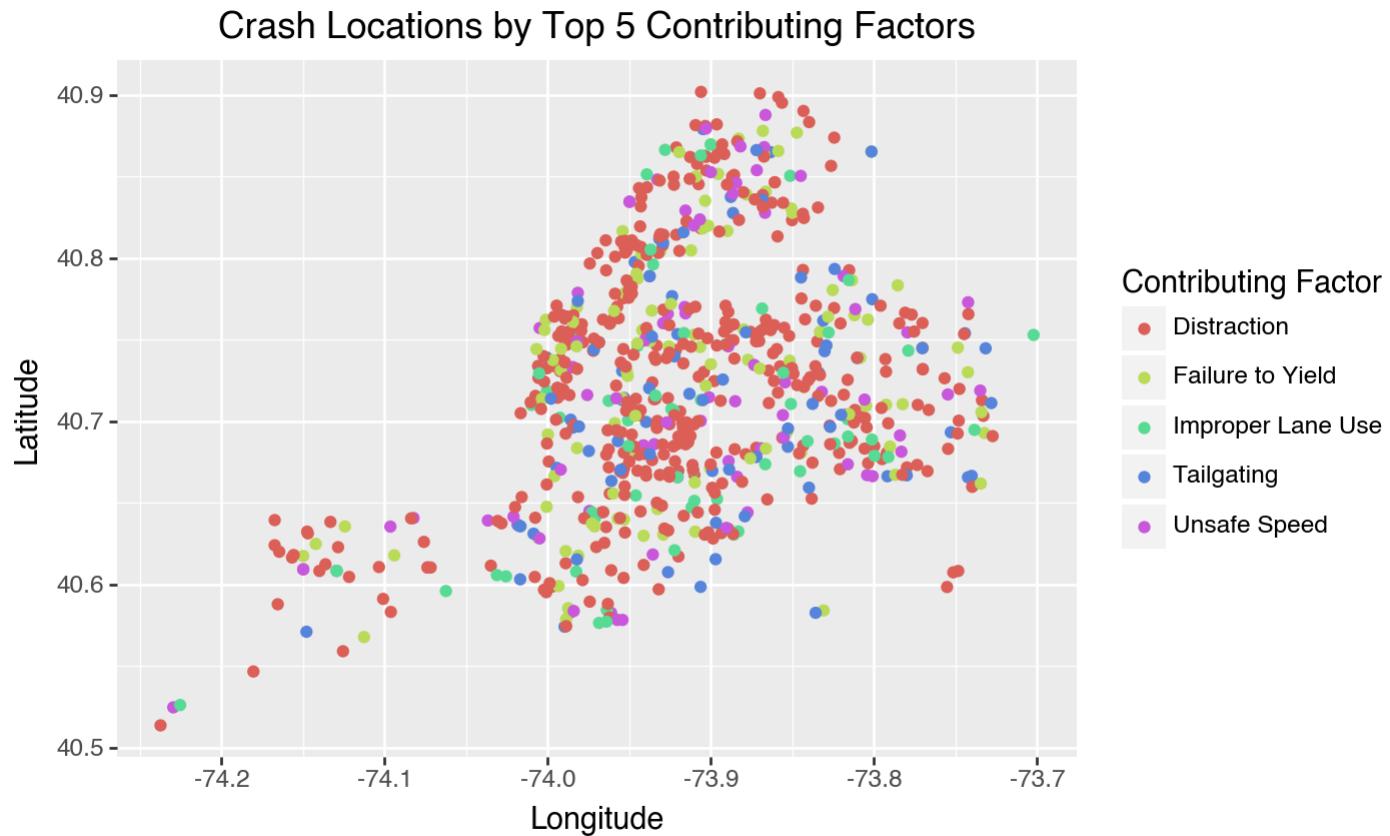


As you can see, each borough is represented by its own color, allowing the audience to easily identify which borough the crash occurred in.

Changing color of points by `contributing_factor_vehicle_1`:

## 7 Visualization

```
# 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

## 7 Visualization

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 0x11af63710>
```

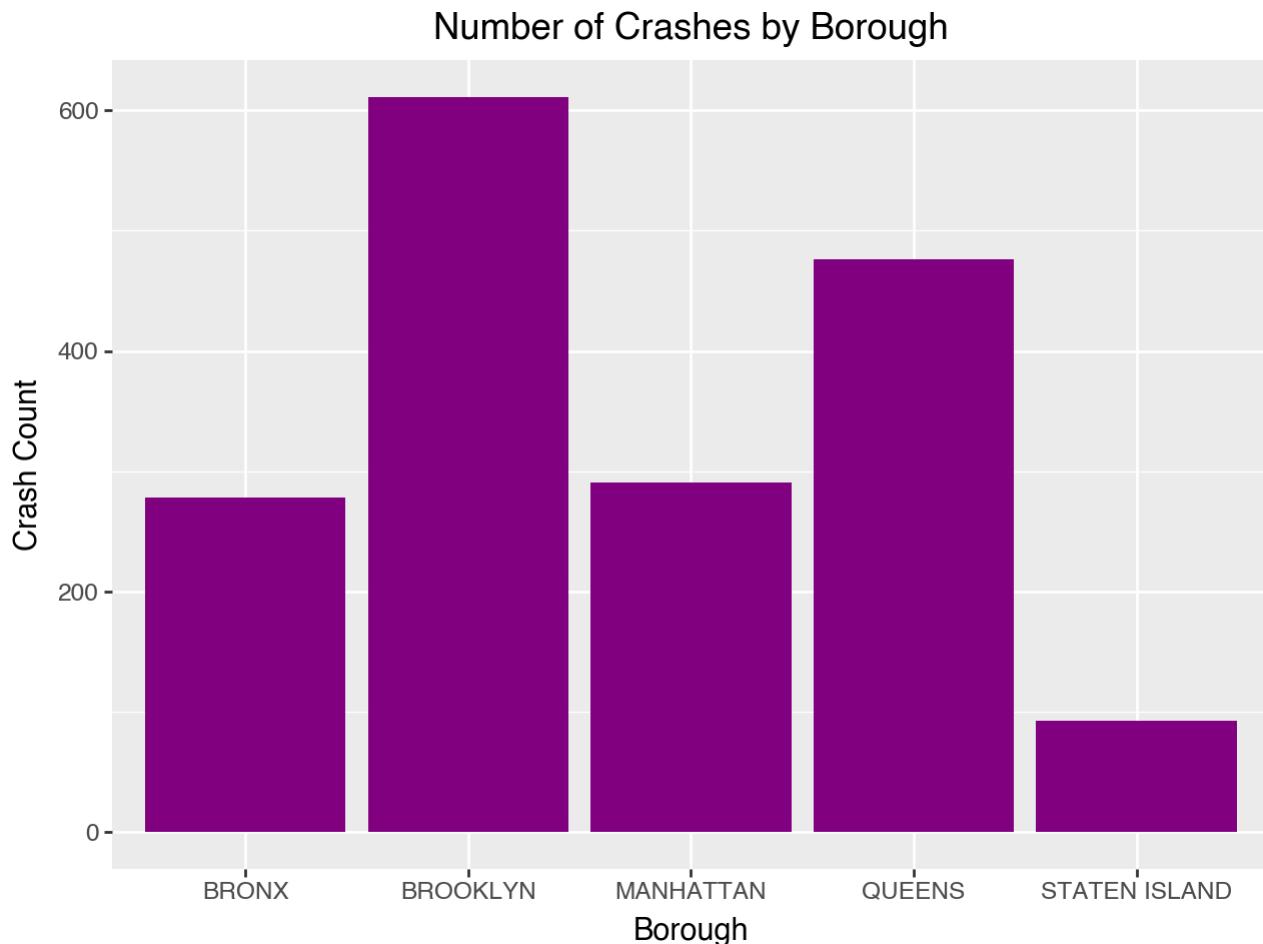
### 7.1.5 Bar Chart

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'))
```

## 7.1 Data Visualization with Plotnine



#### Customizing your Bar Chart

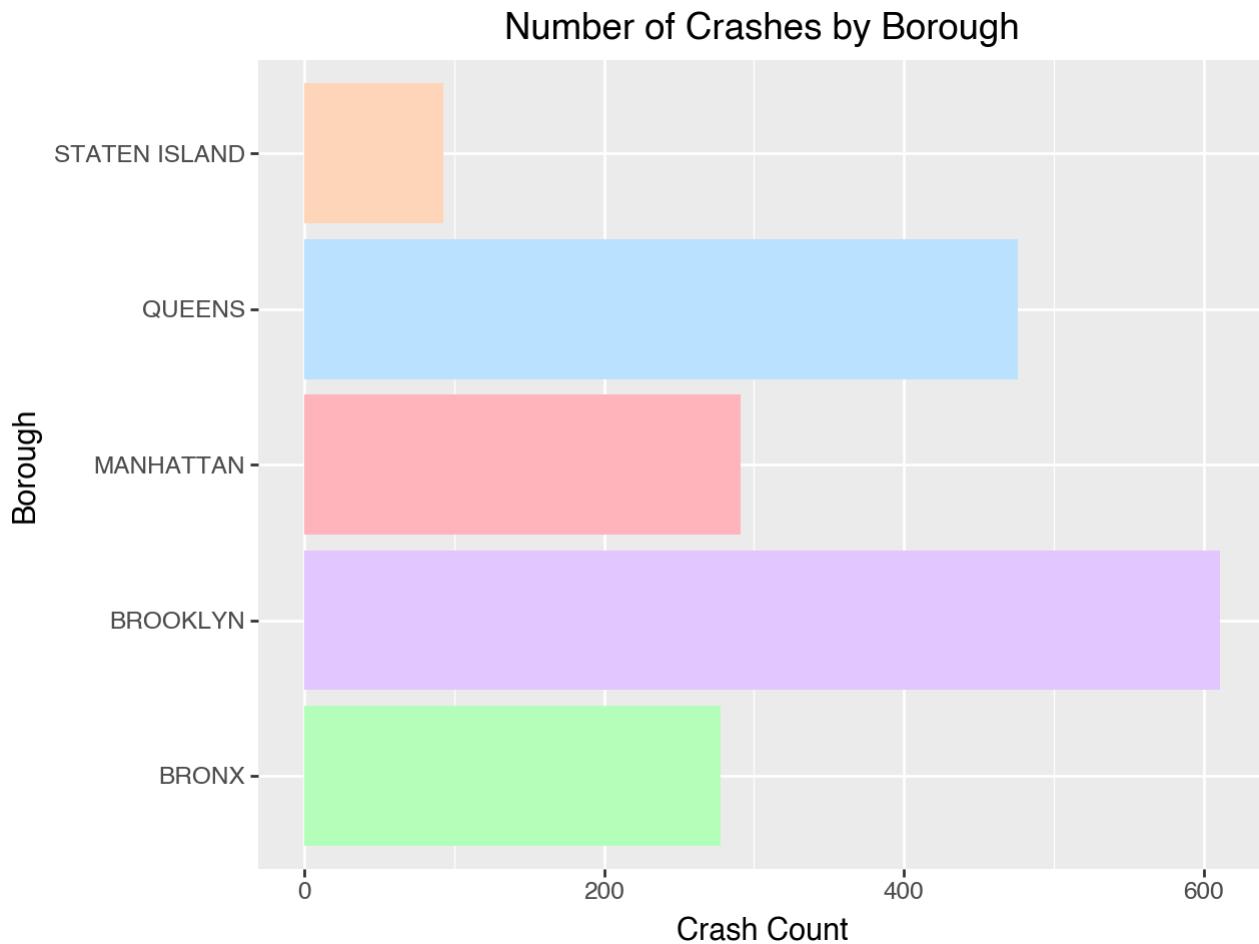
You can change up your bar chart a couple of different ways. You can handpick colors you want, designate it to variables, flip orientation, etc:

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

## 7 Visualization

```
# 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'))
```

## 7.1 Data Visualization with Plotnine



### Multivariable Bar Chart

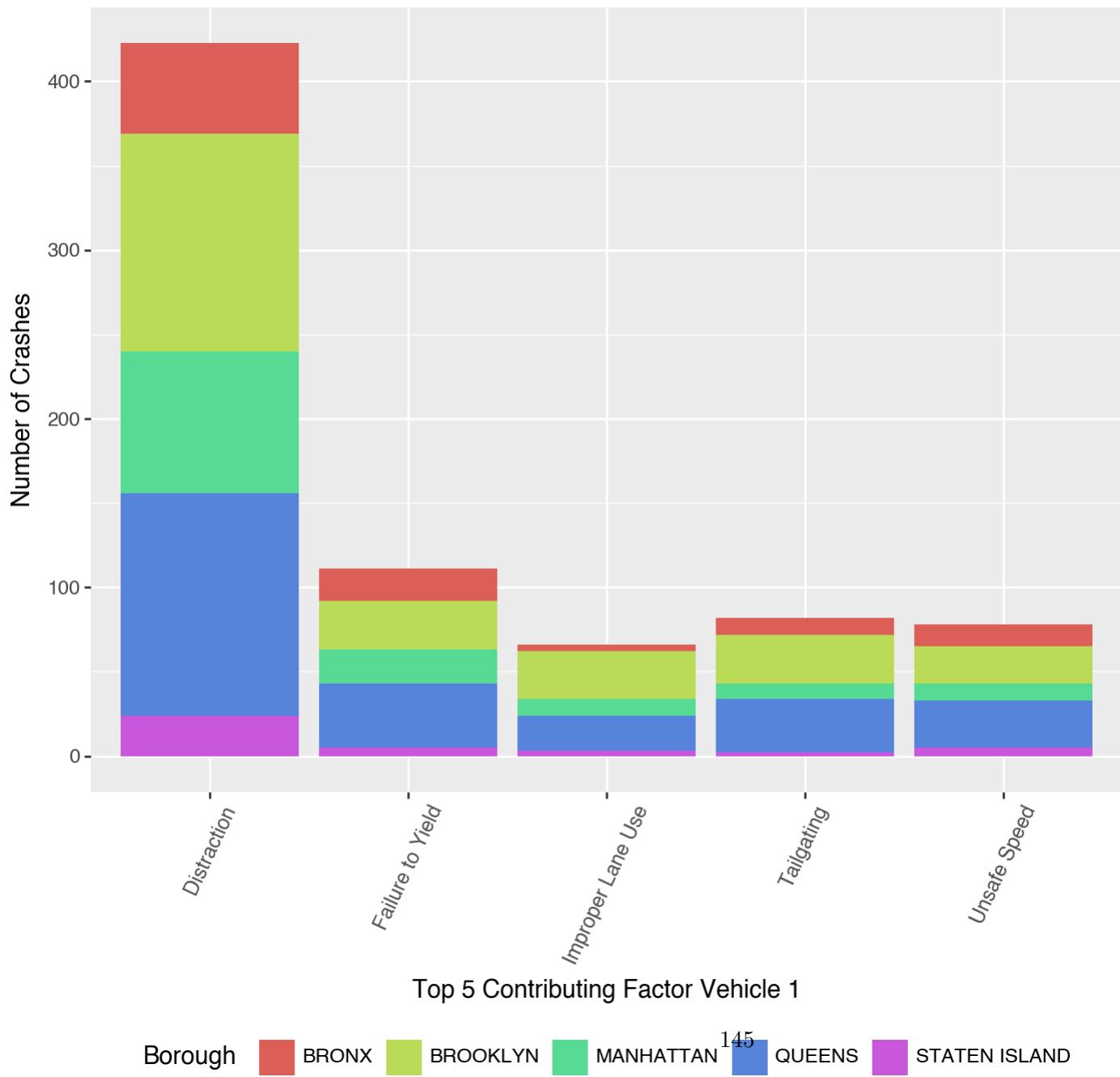
You can also split up a bar chart to make it visually easier to understand.

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

## 7 Visualization

```
  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), legend_position='bottom'))
```

### Top 5 Contributing Factors by Borough



### 7.1.6 Histogram

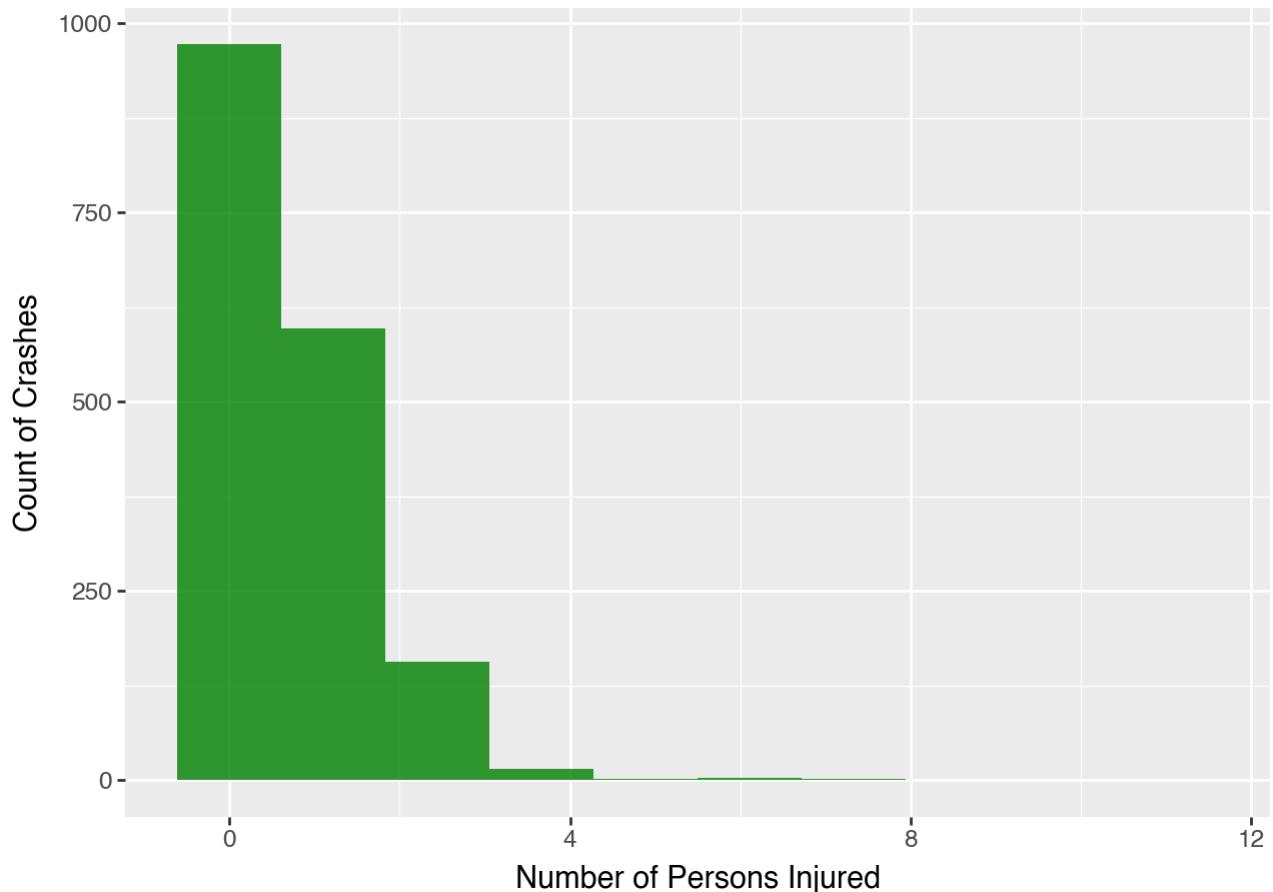
Another useful way to display data is a histogram. You can create one with `geom_histogram()`. Using a histogram is very useful when displaying continuous data.

#### 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'))
```

## 7.1 Data Visualization with Plotnine

### Distribution of Persons Injured



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.

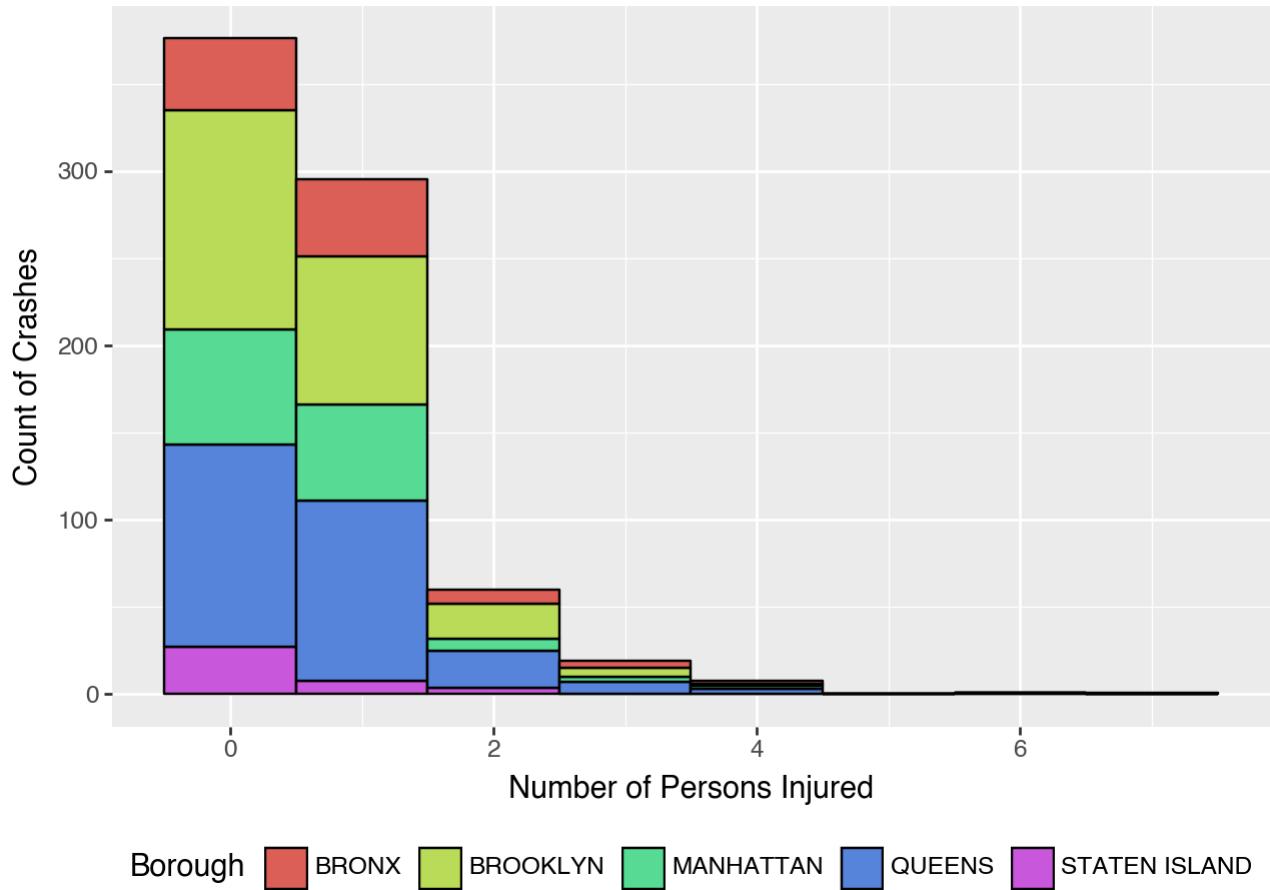
### Multivariable Histogram

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

## 7 Visualization

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

### Distribution of Persons Injured



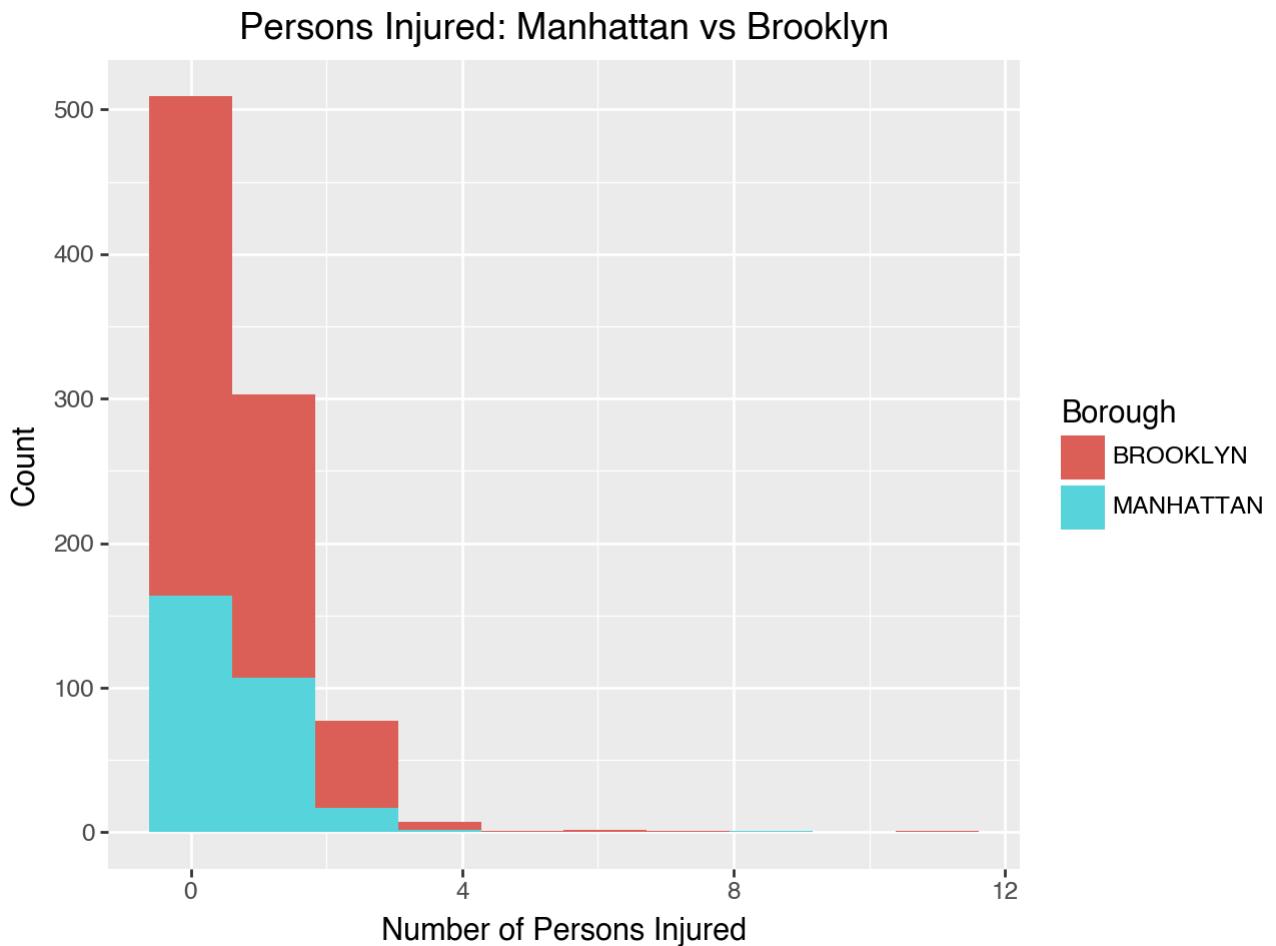
### Overlapping Histogram

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.

## 7 Visualization

```
# 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 Data Visualization with Plotnine



#### 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

## 7 Visualization

a variable changes over a specified period. We will create one analyzing Number of Crashes by Hour.

### Basic Line Chart

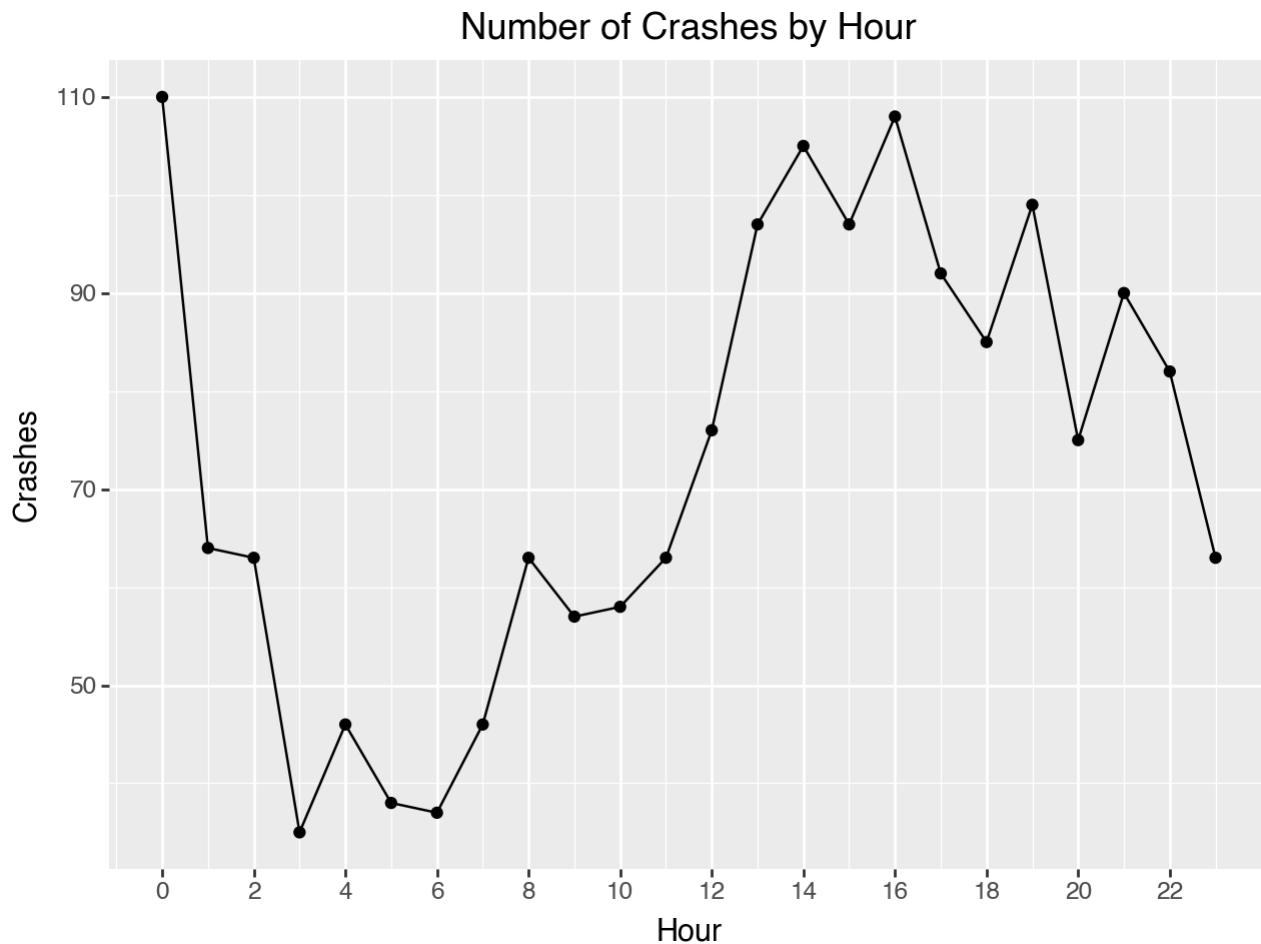
```
# Finding crashes per hour
nyc_crash['crash_datetime'] = pd.to_datetime(nyc_crash['crash_datetime'])

# 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)))
```

## 7.1 Data Visualization with Plotnine



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.

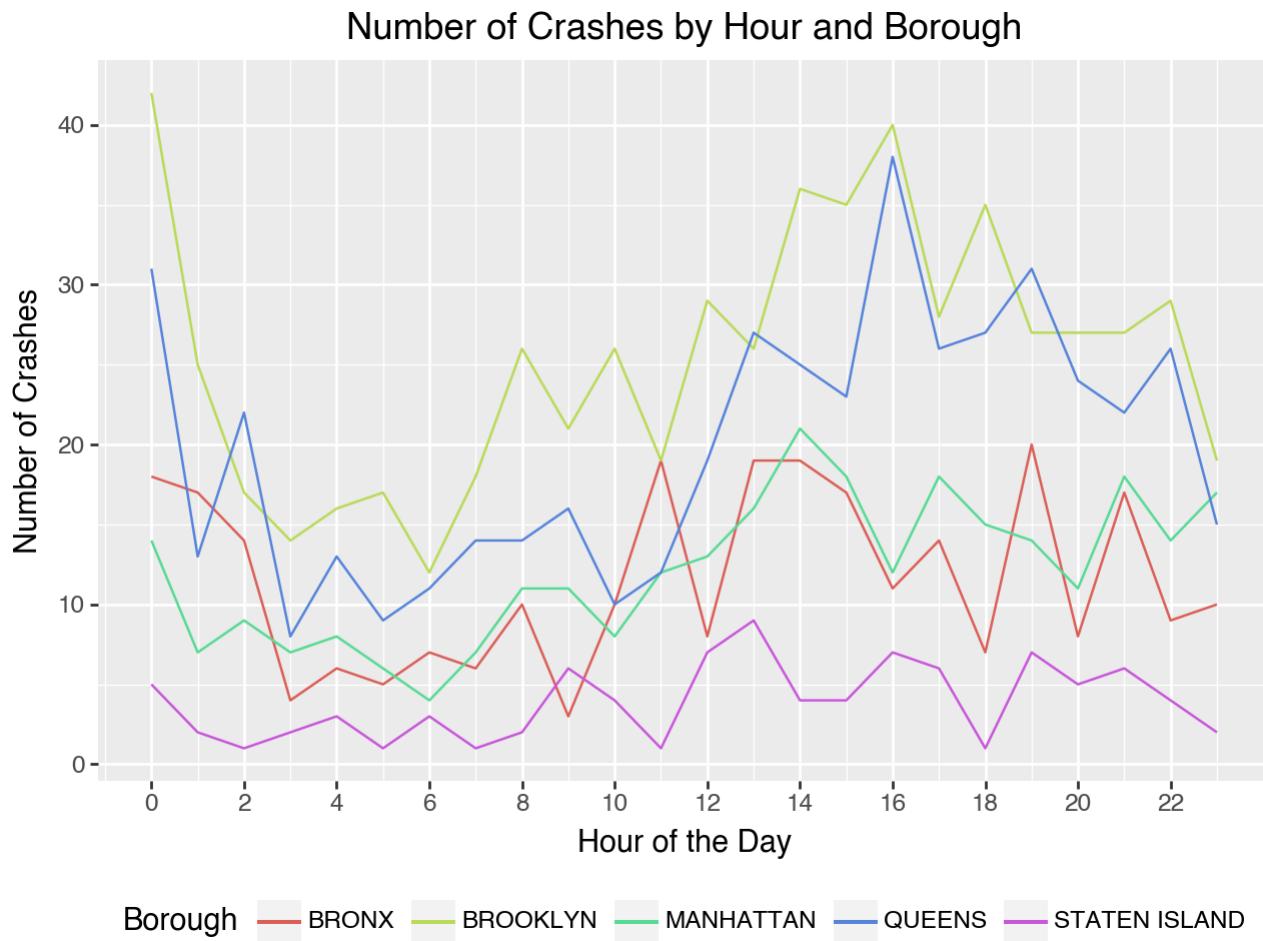
### Multivariable Line Chart

## 7 Visualization

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)) +
  theme(legend_position='bottom'))
```



#### 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

## 7 Visualization

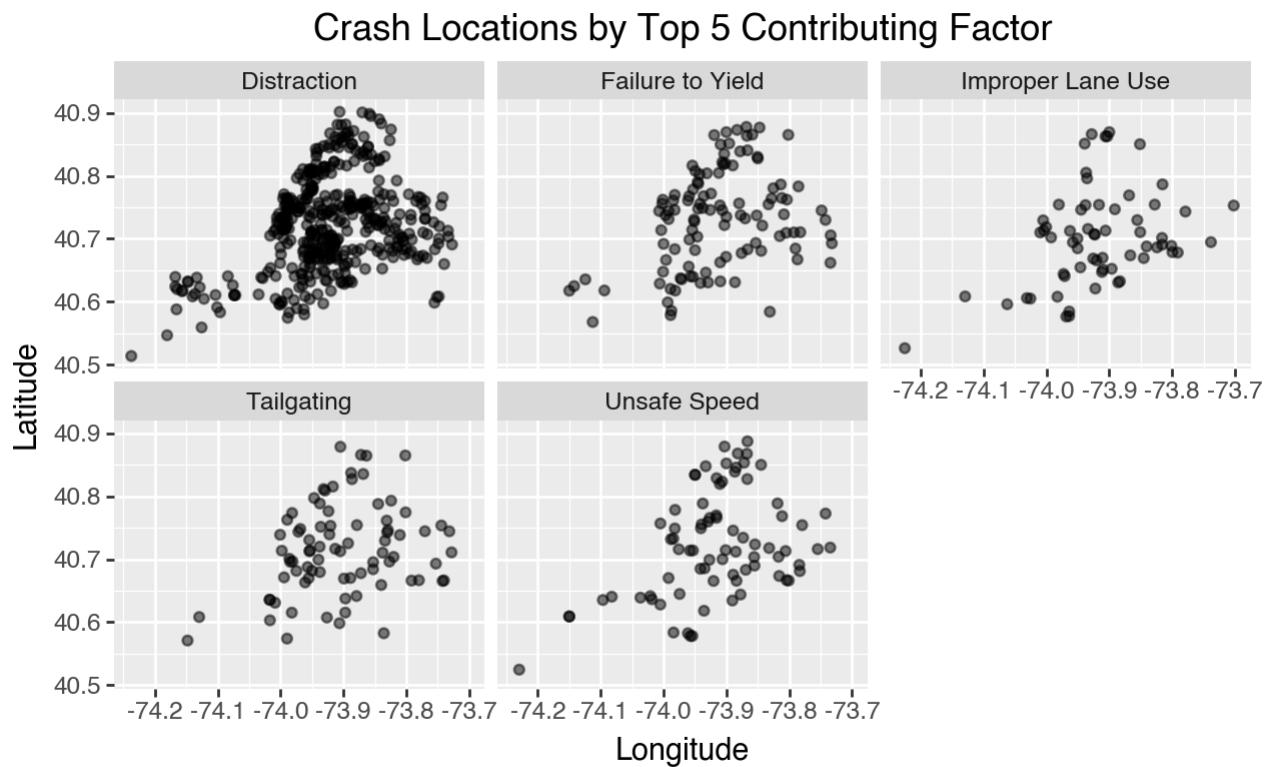
identify trends and patterns. You can facet any type of plots, scatterplots, bar charts, histograms, line charts, etc. using one or two variables.

### 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))
```

## 7.1 Data Visualization with Plotnine



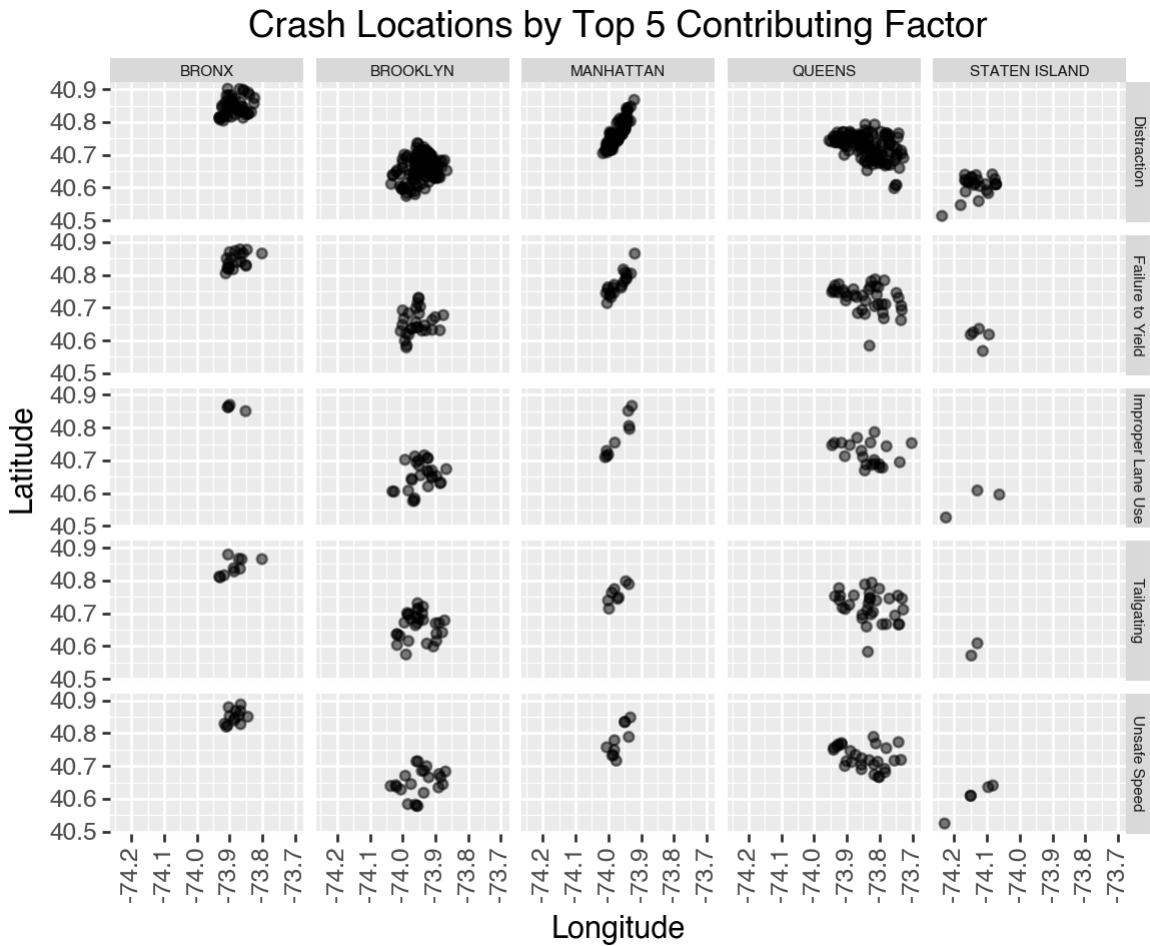
Scatterplot of Two Variables, Crash Locations Contributing Factor and Borough with `facet_grid()`:

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

## 7 Visualization

```
# ~'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),
# strip_text=element_text changes text size of the facet titles
       strip_text=element_text(size=5.5)) +
  coord_fixed(ratio = 1))
```

## 7.1 Data Visualization with Plotnine



### Bar Chart per Facet

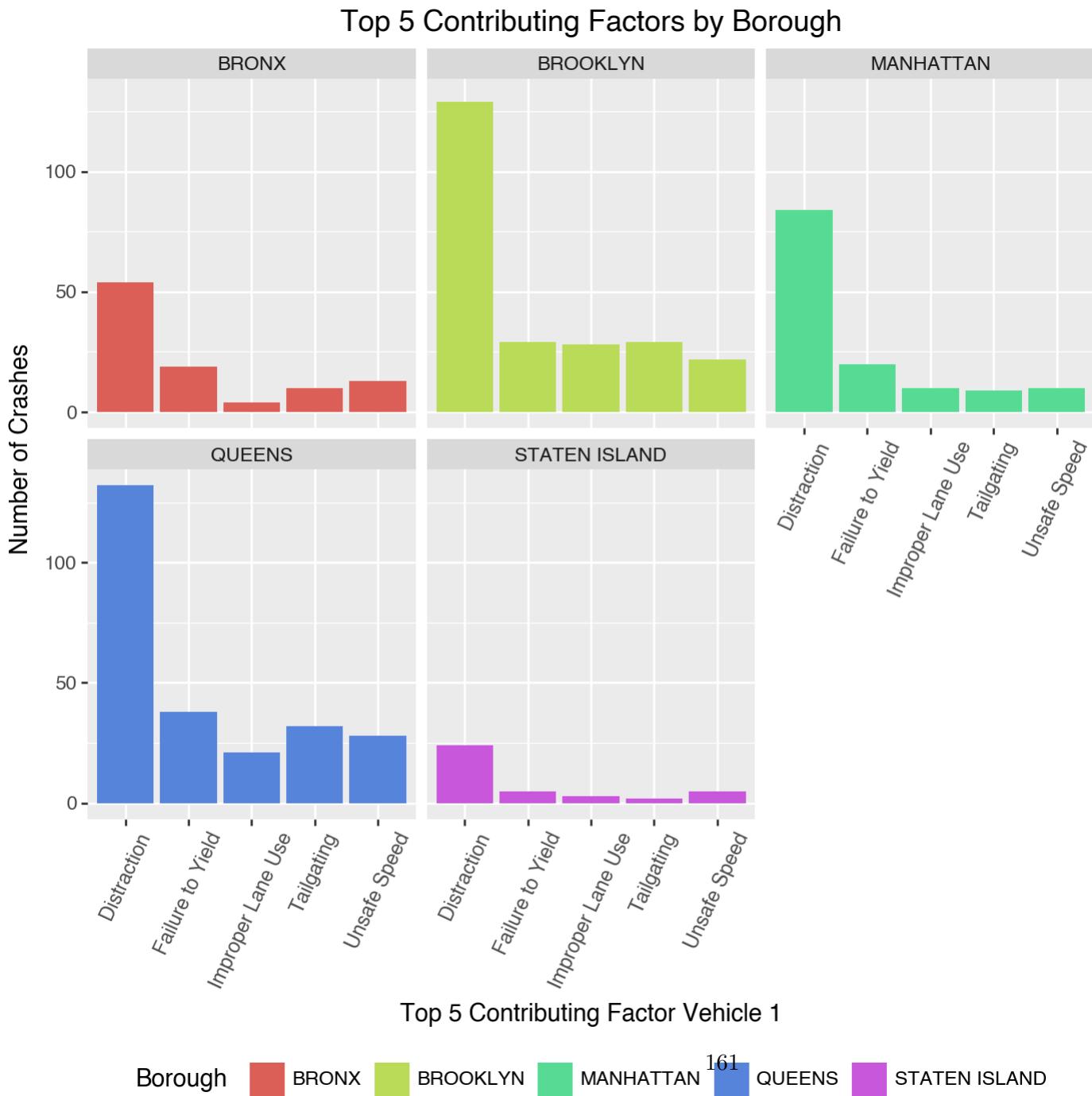
Bar chart of Contributing Factors by Borough with `facet_wrap`:

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

## 7 Visualization

```
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), legend_position='bottom'))
```

## 7.1 Data Visualization with Plotnine



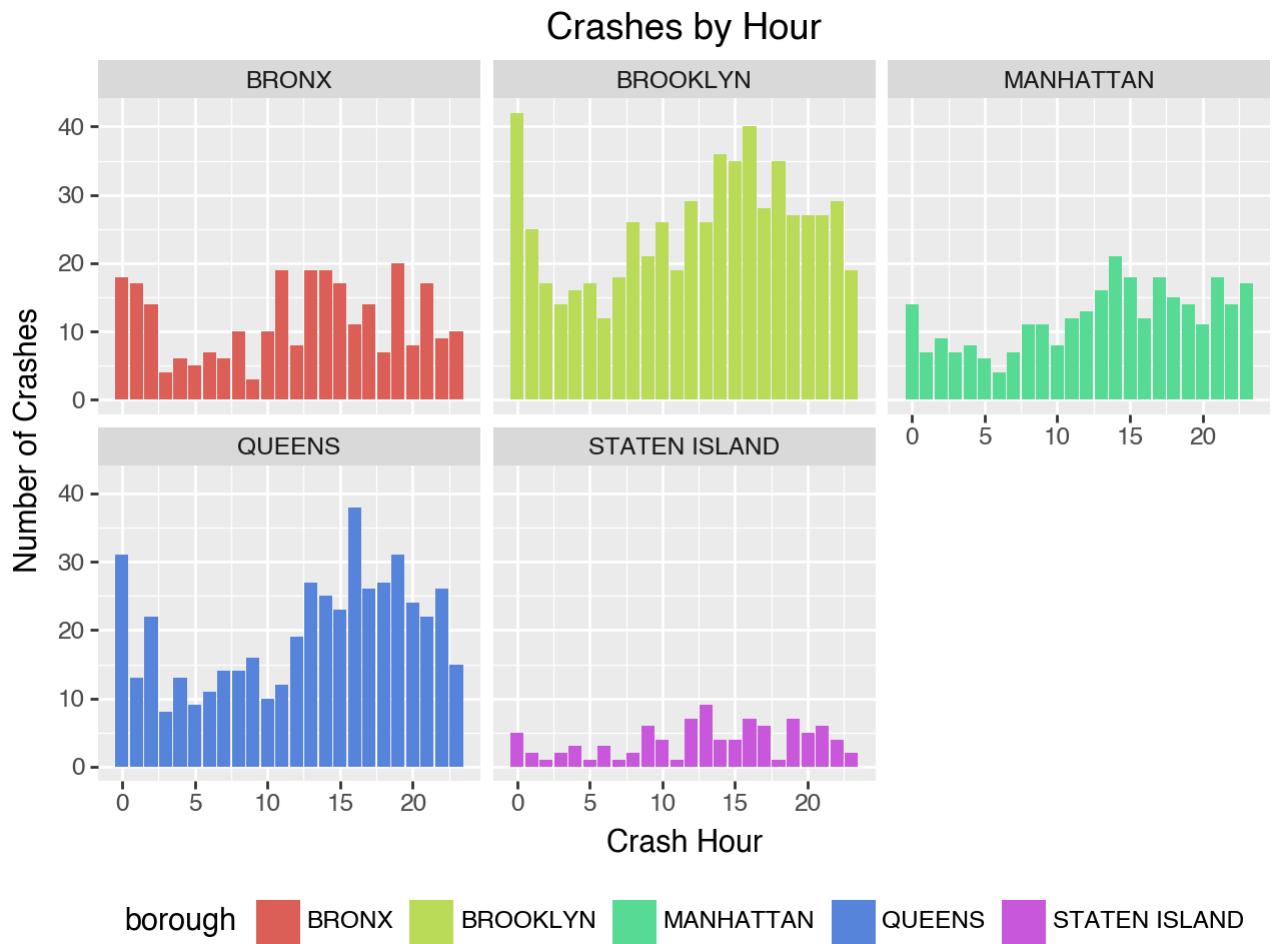
## 7 Visualization

### 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") +  
  theme(legend_position='bottom') +  
  facet_wrap(~ borough))
```

## 7.1 Data Visualization with Plotnine



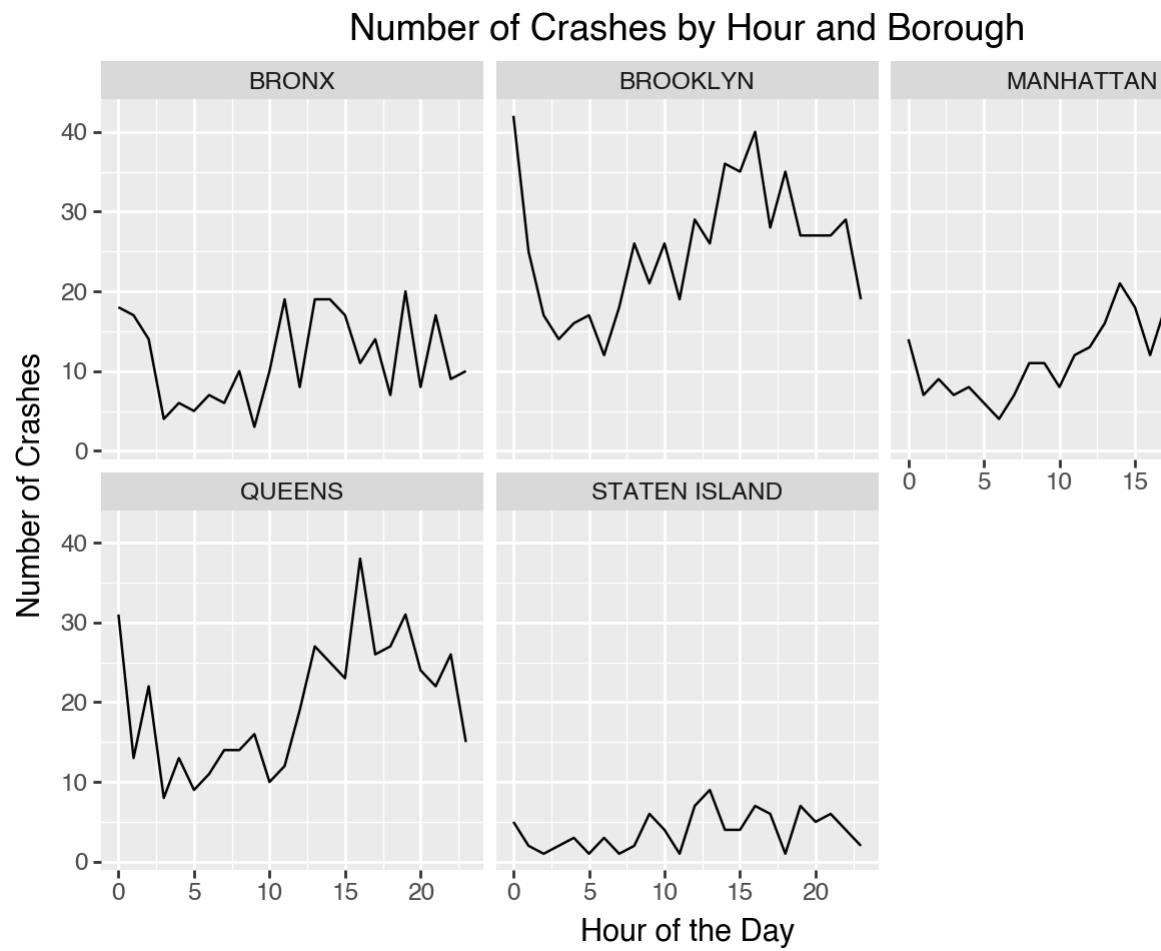
### Line Chart per Facet

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
```

## 7 Visualization

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



## 7.1 Data Visualization with Plotnine

### 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_density()`

#### 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

## 7 Visualization

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

Hi! My name is Jack Bienvenue, a senior in Statistical Data Science.

In this section, I'll walk you through how to begin with spatial data visualizations.

Specifically, in this section we'll talk about using *Google Maps* as a tool to visualize spatial data.

We will start by talking about Google and their spatial data products, but work our way into something more aligned with our needs, which is integrating code into our Python scripts to generate Google Maps versions of our outputs.

This section will be capped off with some example code of using Google Maps to create a heatmap of crash data in New York City.

*Skip to proceed to purely technical information by clicking here*

Enjoy and keep learning!

## *7.2 Spatial Data Visualization with Google Map*

### **7.2.1 Introducing Spatial Data**

#### **7.2.1.1 Having fun with spatial data**

Spatial data handling and visualization introduces new challenges, softwares, and packages.

Spatial data operations can be time-consuming and the pathway to your desired results is often unclear.

**Stay patient, have fun**

## 7 Visualization

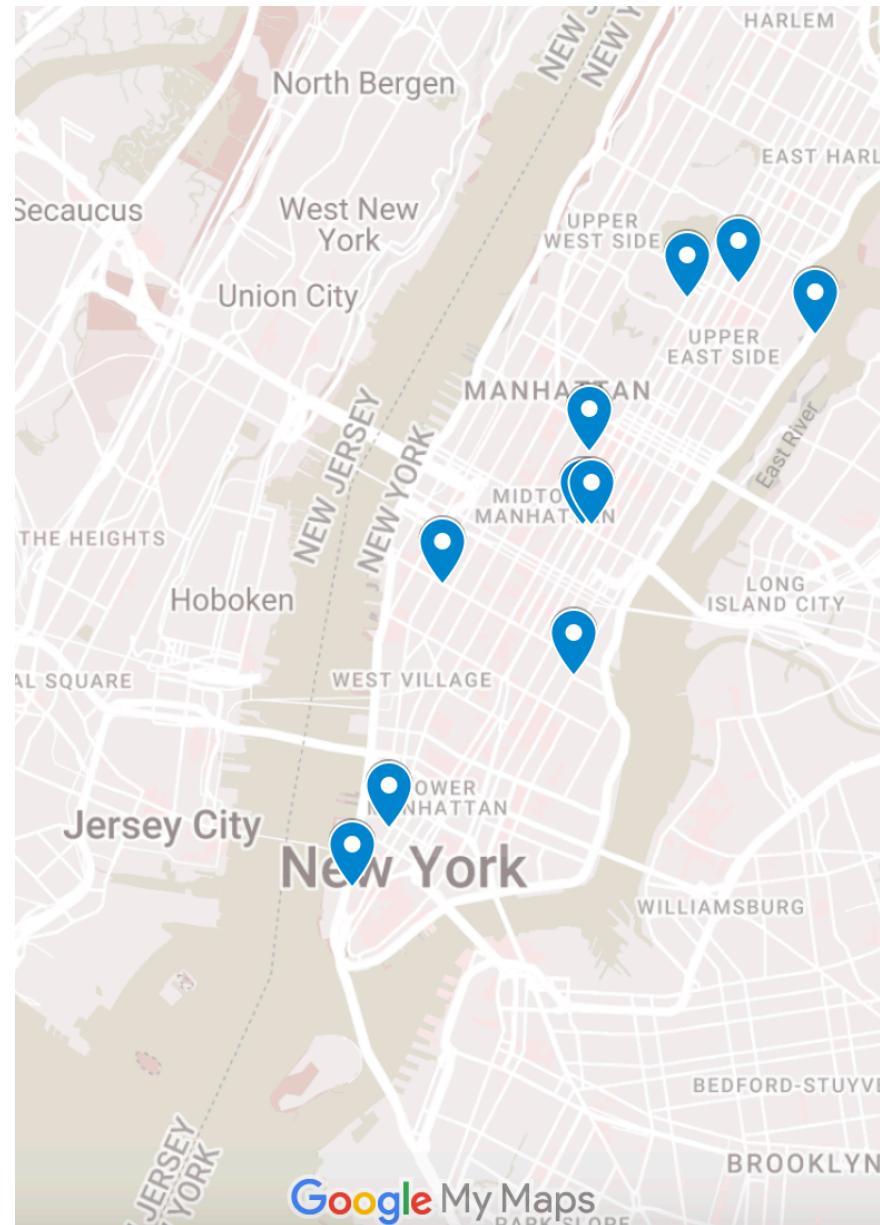


Figure 7.1: Map of all the locations in NYC where I have eaten a bagel

## *7.2 Spatial Data Visualization with Google Map*

Above is an example of me having fun with spatial data.

There are so many fun things to do with data visualization. Remember, you have the skillset to do things that almost nobody can! Make the most of it for yourself and for the world.

### **7.2.1.2 Why Spatial Data?**

**Spatial trends matter.** They allow us to do things like:

- Understand geographic localizations of trends
  - Unveil hidden trends
- Evaluate how social factors may influence trends
  - Create information to address disparities
- Provide recommendations to relevant local authorities

### **7.2.2 Google & Spatial Data**

#### **7.2.2.1 Google Spatial Data Products:**

Google offers several products for performing Geographic Information Science (GIS) tasks. These include:

- Google MyMaps
  - A simple, no-code way to visualize data
- Google Earth Engine
  - Enhanced computing capabilities for analytic tasks
- Google Maps Platform
  - Comprehensive map construction, focused on visualization

## 7 Visualization

### 7.2.2.2 Google MyMaps

Google MyMaps allows for visualization of line and point data.

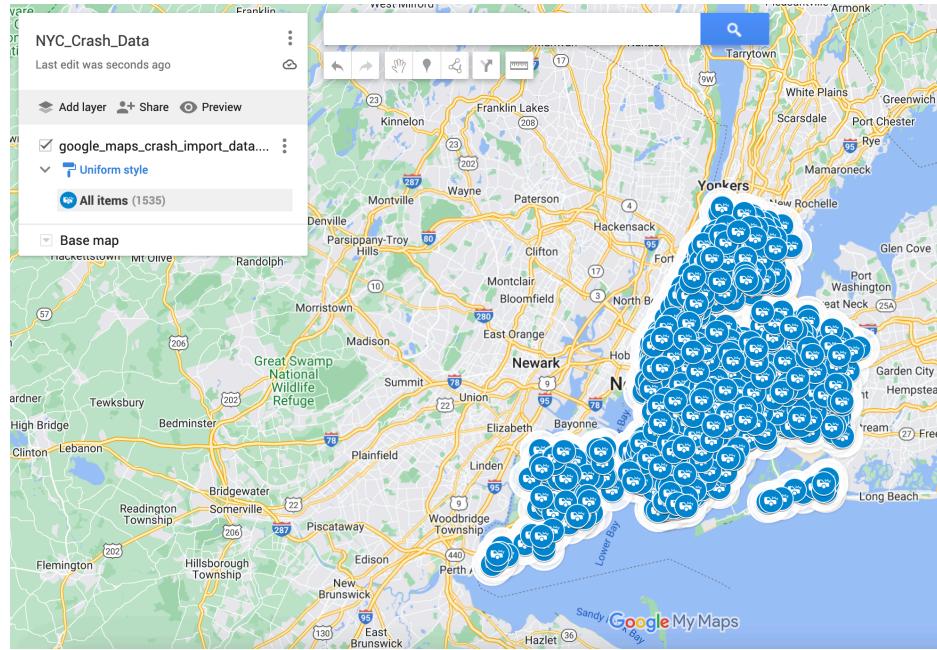


Figure 7.2: Image of MyMaps UI. As we can see, there is a very small extent of functionality.

### 7.2.2.3 Google Earth Engine

Google Earth Engine is used for **remote sensing** applications.

It uses JavaScript.

It pairs well with landcover classification to aid in tasks like:

- Landcover Classification

## 7.2 Spatial Data Visualization with Google Map

- Tracking deforestation
- Habitat Monitoring
- **Solar Panel Identification**

### 7.2.2.4 Google Earth Engine Example

Here's an example of Google Earth Engine being used to identify commercial solar properties to fill in missing data:



Figure 7.3: Example of Landcover classification model in GEE

### 7.2.2.5 Google Maps Platform

Google Maps Platform significantly broadens capabilities.

It allows us to perform tasks such as:

## *7 Visualization*

- Coalescing nearby points into clusters for zooming
- Creating heat maps from point data
- Embedding maps into websites
- Automatically translating a map into a user's language
- 3D geospatial visualization
- and more!

### **7.2.3 Getting Started in Google Maps Platform**

#### **7.2.3.1 Why Google Maps Platform?**

We want to be able to include maps right inside of our Python code, so Google Maps will be our platform of choice.

Google Maps Platform uses Google Cloud computing.

We must log in and do some setup there first.

#### **7.2.3.2 Google Cloud Platform**

Google Cloud Platform is accessible here.

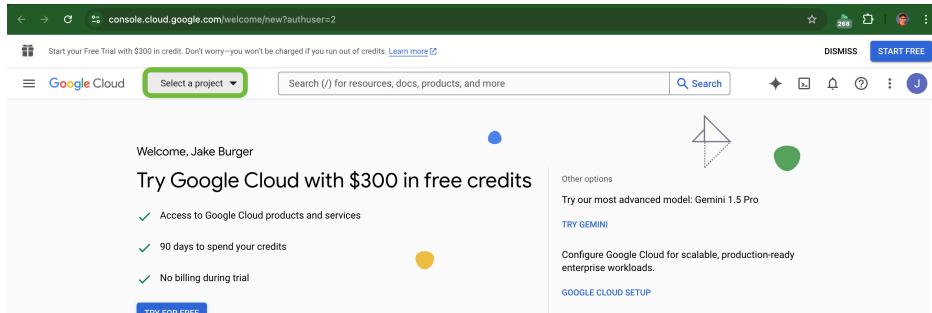
UConn terminated Google accounts in 2024, so to access Google Cloud Console, sign in with your **personal email**.

#### **7.2.3.3 Accessing the console**

To access Google Maps Platform, we must enter our Google Cloud Console.

Navigate to [console.cloud.google.com](https://console.cloud.google.com) to view your console. It should look like this. Press “select project” and create your new project.

## 7.2 Spatial Data Visualization with Google Map



### 7.2.3.4 Building Project

- “Select a Project”
- “New Project”
- Name your project, set location as ‘No Organization’
- “Create”
- Select newly created project
- Next, we’ll set up an API key in the following slides

### 7.2.3.5 API: What is an API?

API stands for “Application Programming Interface.”

**Google Maps API** is a set of tools allowing us to perform tasks like:

- Visualizing data and embedding it into websites
- Route finding
- 3D map viewing
  - Street view, etc.
- And more!

## 7 Visualization

### 7.2.3.6 API: What are API keys?

An API Key allows us to access our projects **exclusively**

Generate an API key for your Maps project in Google Cloud here:

The screenshot shows the Google Cloud Platform interface for managing credentials. The URL in the address bar is `console.cloud.google.com/google/maps-apis/credentials?hl=en&project=nyc-open-data-crashes`. The left sidebar has a tree view with nodes like Overview, APIs & Services (which is expanded), Metrics, Quotas, Keys & Credentials (selected and highlighted with a green border), Support, Solution Library, Map Management, Map Styles, and Datasets. The main content area is titled "Keys & Credentials" and shows a sub-section "All Google Maps Platform APIs". It includes a note about restricting API keys, a link to "Credentials in APIs & Services", and a "Help me choose" section. Below this is a table titled "API Keys" with one entry: "Maps Platform API Key" created on Sep 18, 2024. A callout box highlights the "API key" option, which is described as identifying the project using a simple API key to check quota and access.

Figure 7.4: Where to navigate to generate API key

### 7.2.3.7 API: Enabling necessary APIs

In this example, we will visualize our data as a heat map. For this we will need to enable the following APIs:

- Maps JavaScript API
- Maps Datasets API
- Places API

*(Select 'APIs and Services' from the left sidebar to access)*

## 7.2 Spatial Data Visualization with Google Map

### 7.2.3.8 Python and Google Maps

Let's begin by downloading our packages in terminal or an equivalent command shell:

```
% pip install googlemaps  
% pip install gmplot
```

Now we can start working with Google Maps in Python!

- googlemaps
  - Provides access to Google Maps Platform services in a Python client
- gmplot
  - Primarily for plotting

### 7.2.4 Using googlemaps and gmplot

#### 7.2.4.1 gmplot: Initialization

We'll use the standardized nyccrashes\_cleaned.feather file provided in the coursenotes for our examples.

```
% pip install pyarrow
```

```
import pandas as pd  
import gmplot  
import os  
df = pd.read_feather('data/nyccrashes_cleaned.feather')
```

### 7.2.4.2 gmplot

Here's some example code for gmplot using the cleaned NYC Crashes Dataset:

```

import os
import pandas as pd
import gmplot

## Load the dataset using pd.read_feather()
df = pd.read_feather('data/nyc_crashes_cleaned.feather')

#NaN values prevent plotting, let's remove them:
df = df.dropna(subset=['latitude', 'longitude'])

## Insert API key (substitute with your own!)
api_key_file_path = "api_key.txt"
with open(file_path, 'r') as file:
    api_key = file.read().strip()

## Center the map on NYC using the mean center
center_lat = df['latitude'].mean()
center_lon = df['longitude'].mean()

## Create gmplot object (third argument is the zoom level)
gmap = gmplot.GoogleMapPlotter(center_lat, center_lon, 11, apikey=api_key)

## Extract latitudes and longitudes directly
latitudes = df['latitude'].tolist()
longitudes = df['longitude'].tolist()

## Plot all points at once
gmap.scatter(latitudes, longitudes, marker=True, size=20, color='red')

```

## 7.2 Spatial Data Visualization with Google Map

```
## Define output path
output_dir = "map_outputs"
output_path = os.path.join(output_dir, "nyc_crashes_map.html")

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

## Save the map
gmap.draw(output_path)
```

### 7.2.4.3 gmplot example code output:

You would retrieve this as an html generated in the specified directory, but for this presentation I'll show this as an image:

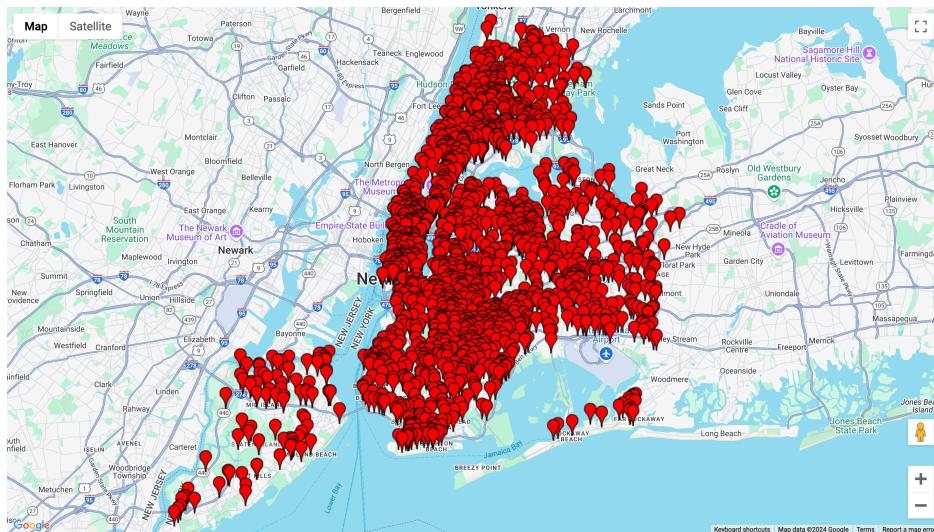


Figure 7.5: Output of example code

#### 7.2.4.4 gmplot: Heat Maps

Here's some example code for a heat map:

```
## Now create a heatmap

## Create a new gmplot object for the heatmap
heatmap = gmplot.GoogleMapPlotter(center_lat, center_lon, 11,
                                   apikey=api_key)

## Plot the heatmap
heatmap.heatmap(latitudes, longitudes)

## Define output path for heatmap
heatmap_output_path = os.path.join(output_dir, "nyccrashes_heatmap.html")

## Save the heatmap
heatmap.draw(heatmap_output_path)
```

(Woods 2017)

## 7.2 Spatial Data Visualization with Google Map

### 7.2.5 gplot Heat Map Example

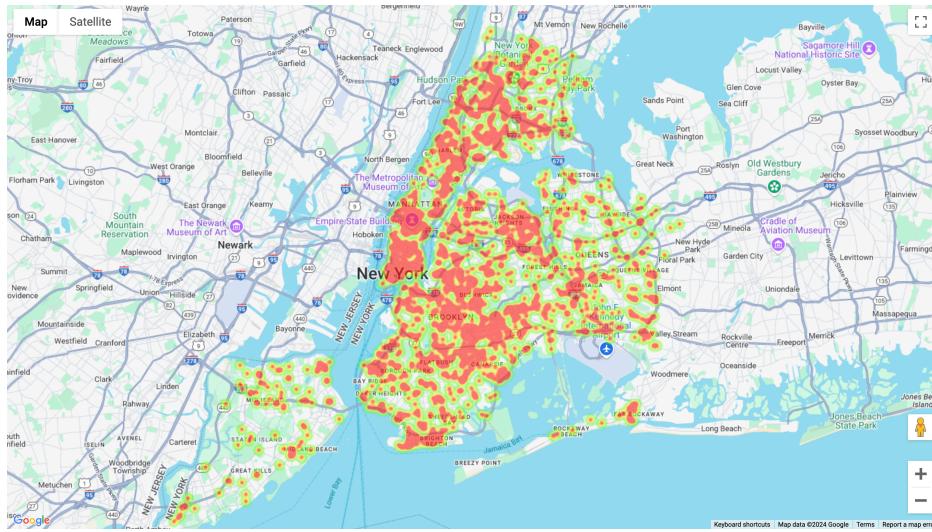


Figure 7.6: Heat map output; *Remark: multiple gplot html outputs might not reliably work for a single script*

### 7.2.6 Googlemaps

Googlemaps as a package is valuable in allowing us to access advanced functionality, beyond the scope of this write-up. Access more information here. (Google 2023)

### 7.2.7 Conclusion

#### 7.2.7.1 Geospatial Visualization

There are a range of different geospatial visualization platforms.

## 7 Visualization

But, using Python, you can do all the spatial visualization you need right in your code!

### 7.2.7.2 Thank you!

Best of luck with your spatial data visualization! Keep learning.

### 7.2.8 References

- Woods, M. (2017). gmplot: Plotting data on Google Maps, the easy (stupid) way. URL: <Https://Github.Com/Gmplot/Gmplot>.
- Google. (2023). googlemaps (v4.7.3) [Python package]. PyPI. <https://pypi.org/project/googlemaps/>

## 7.3 Animating Plots and Maps

### 7.3.1 Introduction

Hello everyone! My name is Rahul Manna. I am an undergraduate junior doing a dual degree in Statistical Data Science and Mechanical Engineering. I will be showing you how you can animate plots and maps using Matplotlib's `FuncAnimation` function.

Animated maps and plots are valuable for showing changes over time or across locations, making trends and patterns easier to see. They're useful in fields like public health, where animated maps can show how a disease spreads, or in economics, where plots can track market trends. Environmental scientists also use animated weather maps to illustrate seasonal shifts. These visuals make complex data clearer and help in understanding and decision-making.

### 7.3 Animating Plots and Maps

#### Overview

1. Matplotlib Review
2. Animating Plots
3. Animating Maps
4. Saving your Animation

#### 7.3.2 Matplotlib

- Developed by John D. Hunter in 2003 as a tool for scientific computing in Python.
- Initially aimed at providing MATLAB-like plotting capabilities but has evolved into one of the most widely-used plotting libraries.

Source: Hunter & Matplotlib Development Team (2023a)

##### Fun Fact

Matplotlib was used for data visualization during the 2008 landing of the Phoenix spacecraft on Mars and in generating the first image of a black hole (Source: Collaboration (2019)).

##### 7.3.2.1 Installation:

To install `matplotlib`, you can use either of the following lines in your terminal or conda prompt respectively.

```
pip install matplotlib # pip users  
conda install -c conda-forge matplotlib # conda users
```

### 7.3.2.2 Basic Matplotlib Commands

Majority of plotting with `matplotlib` is done with the `pyplot` module which can be imported with the following code.

```
import matplotlib.pyplot as plt
```

These are some of the most common `matplotlib.pyplot` commands.

- `plt.plot()` : Plot y versus x as lines and/or markers.
- `plt.scatter()` : Create a scatter plot of points.
- `plt.bar()` : Create bar charts.
- `plt.hist()` : Create histograms.
- `plt.xlabel()` : Set the label for the x-axis.
- `plt.ylabel()` : Set the label for the y-axis.
- `plt.title()` : Set the title of the plot.
- `plt.legend()` : Display a legend for the plot.
- `plt.subplots()` : Create a figure and a grid of subplots.
- `plt.show()` : Display all open figures.

Source: Hunter & Matplotlib Development Team (2023a)

### 7.3.2.3 Some Examples

Here are some examples of plots made using `matplotlib.pyplot`.

#### 7.3.2.3.1 Coin Toss

Let's create a line plot to show how the proportion of heads in coin tosses changes as the number of tosses increases.

### 7.3 Animating Plots and Maps

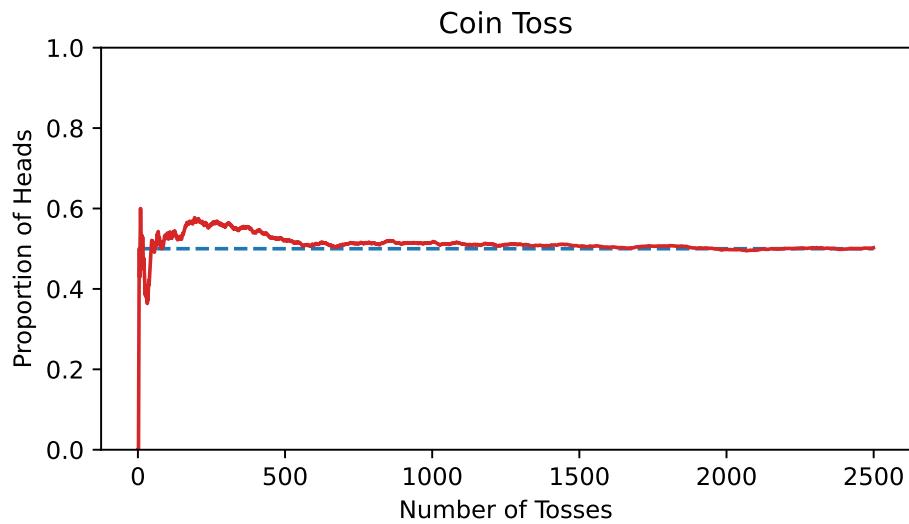
```
import random
import matplotlib.pyplot as plt

random.seed(3255)

def prob_heads(trials):
    result = []
    prop_heads = []
    for i in range(trials):
        toss = random.randint(0,1)
        result.append(toss)
        prop_heads.append(sum(result)/len(result))
    return prop_heads

plt.figure(figsize=(6,3))
plt.hlines(0.5,0,2500,linestyles='dashed')
plt.plot(prob_heads(2500),color='tab:red')
plt.ylim(0,1)
plt.title("Coin Toss")
plt.ylabel('Proportion of Heads')
plt.xlabel('Number of Tosses')
plt.show()
```

## 7 Visualization



### 7.3.2.3.2 Bar Chart

Let's create a simple bar plot of the crashes in New York City by borough in the week of July 30, 2024.

```
import matplotlib.pyplot as plt
import pandas as pd

crash_data = pd.read_feather("data/nyccrashes_cleaned.feather")

crash_data = crash_data.groupby('borough')

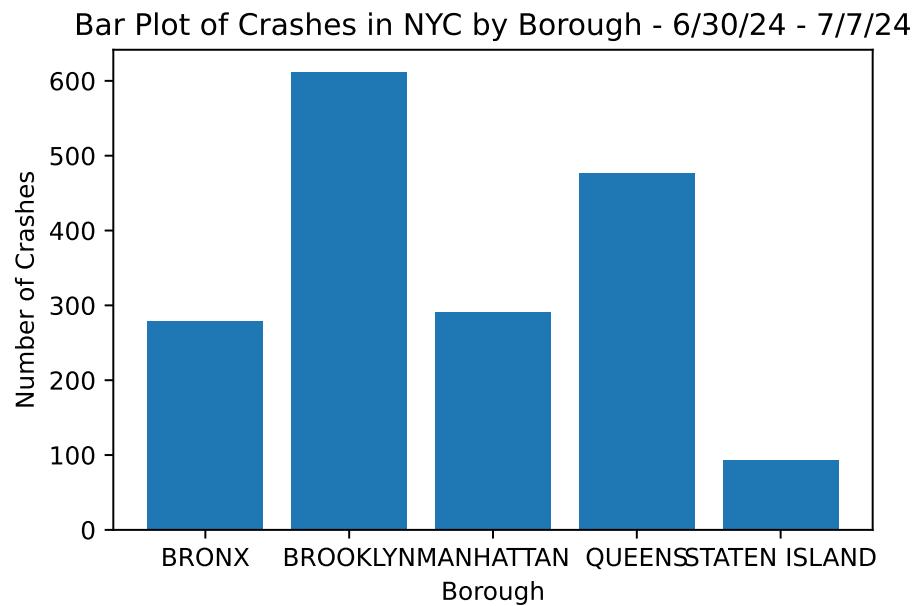
boroughs = crash_data.groups.keys()

heights = [len(crash_data.get_group(x)) for x in boroughs]

plt.bar(boroughs,height=heights)
```

### 7.3 Animating Plots and Maps

```
plt.title('Bar Plot of Crashes in NYC by Borough - 6/30/24 - 7/7/24')
plt.xlabel('Borough')
plt.ylabel('Number of Crashes')
plt.show()
```



#### 7.3.3 Animating Plots

##### 7.3.3.1 Matplotlib's FuncAnimation

```
from matplotlib.animation import FuncAnimation
```

## 7 Visualization

`FuncAnimation` is used to create animations in Matplotlib by repeatedly calling a user-defined function.

```
anim = FuncAnimation(fig,func,frames,interval,repeat,repeat_delay)
```

### Key Inputs

- `fig`: Matplotlib figure object.
- `func`: The update function for each frame.
- `frames`: Sequence or number of frames.
- `interval`: Time interval between frames (ms).
- `repeat`: Whether to repeat animation (True/False).
- `repeat_delay`: Delay before repeating (ms).

Source: Hunter & Matplotlib Development Team (2023b)

#### 7.3.3.2 Coin Toss Animation

Using `FuncAnimation`, we can animate the coin toss plot we previously made.

```
prop_heads = prob_heads(2500)

frames = range(len(prop_heads))

fig, ax = plt.subplots(figsize=(12,6))

def update(frame):
    # Clear previous frame
    ax.clear()

    # Add title, and labels
    ax.set_title('Coin Toss')
    ax.set_ylabel('Proportion of Heads')
```

### 7.3 Animating Plots and Maps

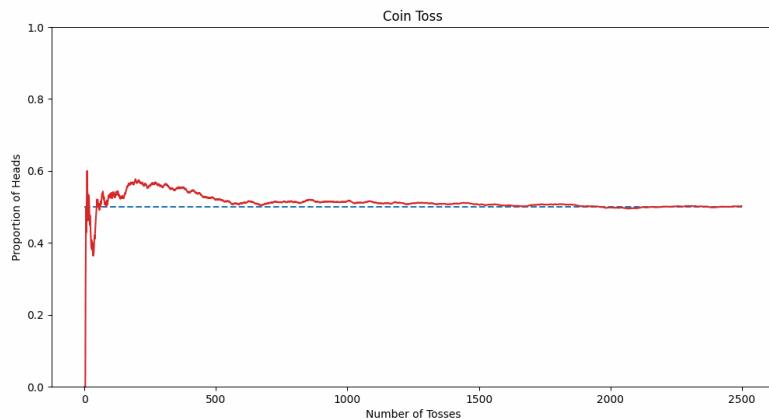
```
ax.set_xlabel('Number of Tosses')
ax.set_xlim(0,1)

# Plot data
ax.hlines(0.5,0,frame+1,linestyles='dashed')
ax.plot(range(1,frame+1),prop_heads[:frame],color='tab:red')

anim = FuncAnimation(fig,update,frames=frames,repeat=False)

anim.save('coin_toss.gif',writer='Pillow',fps=50)

plt.show()
```



#### 7.3.3.3 A Step Further - Coin Toss Animation

We can take this a step further by labeling the current proportion value for each frame.

## 7 Visualization

```
prop_heads = prob_heads(2500)

frames = range(len(prop_heads))

fig, ax = plt.subplots(figsize=(12,6))

def update(frame):
    ax.clear()
    ax.set_title('Coin Toss')
    ax.set_ylabel('Proportion of Heads')
    ax.set_xlabel('Number of Tosses')
    ax.hlines(0.5,0,frame+1,linestyles='dashed')
    ax.set_ylim(0,1)

    # Add text
    ax.text(frame+1,prop_heads[frame]*1.05,f'{prop_heads[frame]:.3f}',weight='bold')

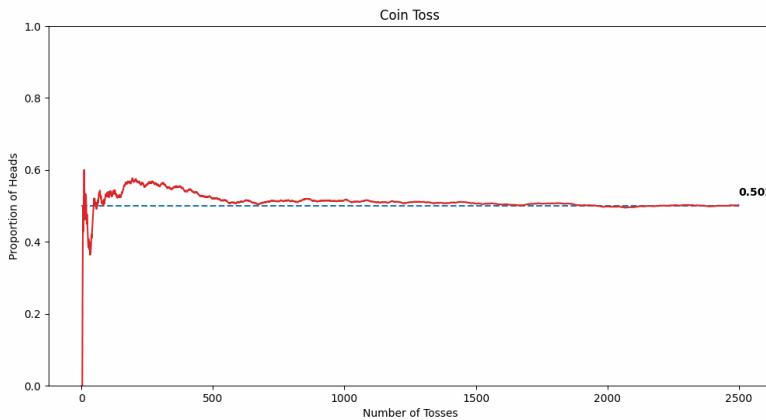
    ax.plot(range(1,frame+1),prop_heads[:frame],color='tab:red')

anim = FuncAnimation(fig,update,frames=frames)

anim.save('coin_toss_with_txt.gif',writer='Pillow',fps=50)

plt.show()
```

### 7.3 Animating Plots and Maps



#### 7.3.3.4 Bar Chart Animation

Let's animate the bar chart we created in section 8.1.5. Here I am using `plt.barch` to create a horizontal bar chart.

```
crash_data['crash_datetime'] = pd.to_datetime(crash_data['crash_datetime'],
format='%Y-%m-%d %H:%M:%S')

fig, ax = plt.subplots(figsize=(12,6))

def update(frame):
    ax.clear()
    current_data = crash_data[crash_data['crash_datetime'] <= frame]

    # Group data by borough and count the number of crashes
    grouped_data = current_data.groupby('borough').size().reset_index(
        name='num_crashes')

    # Sort by number of crashes for consistent bar ordering
```

## 7 Visualization

```
grouped_data = grouped_data.sort_values(by='num_crashes', ascending=True)

# Create horizontal bar chart
bars = ax.barh(grouped_data['borough'], grouped_data['num_crashes'])

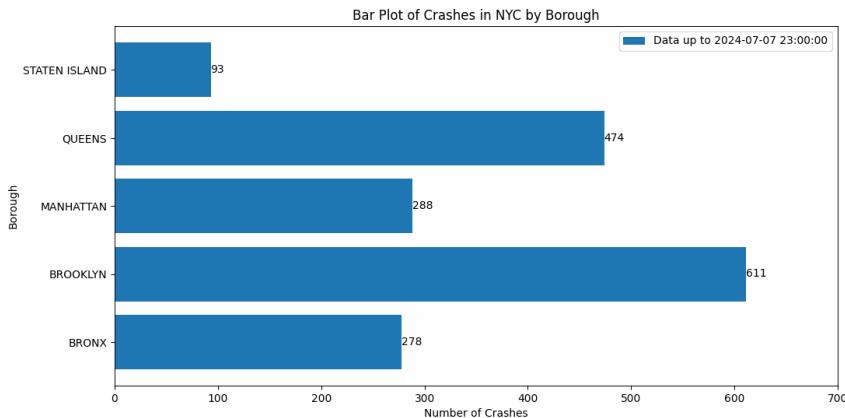
# Set titles and labels
ax.set_title('Bar Plot of Crashes in NYC by Borough')
ax.set_xlabel('Number of Crashes')
ax.set_ylabel('Borough')
ax.legend([f'Data up to {frame}'], prop={'size': 'large'})

# Annotate bars with crash numbers
for i, bar in enumerate(bars):
    ax.text(bar.get_width(), bar.get_y() + bar.get_height() / 2,
            f'{grouped_data["num_crashes"].iloc[i]}', va='center',
            ha='left', color='black')

anim = FuncAnimation(fig, update, frames=pd.date_range(
    start=crash_data['crash_datetime'].min(),
    end=crash_data['crash_datetime'].max(), freq='h'))

anim.save('bar_plot_animation.gif', writer='Pillow', fps=15)
plt.show()
```

### 7.3 Animating Plots and Maps



#### 7.3.3.5 Relative Bar Plot Animation

Similarly, we can plot and animate the relative bar plot of crashes by borough.

```
fig, ax = plt.subplots(figsize=(12,6))

def update(frame):
    ax.clear()
    current_data = crash_data[crash_data['crash_datetime'] <= frame]

    total_crashes = len(current_data)
    grouped_data = current_data.groupby('borough')

    boroughs = sorted(grouped_data.groups.keys())
    height = [len(grouped_data.get_group(x))/total_crashes for x in boroughs]

    # Create horizontal bar chart
    bars = ax.bar(boroughs,height)
```

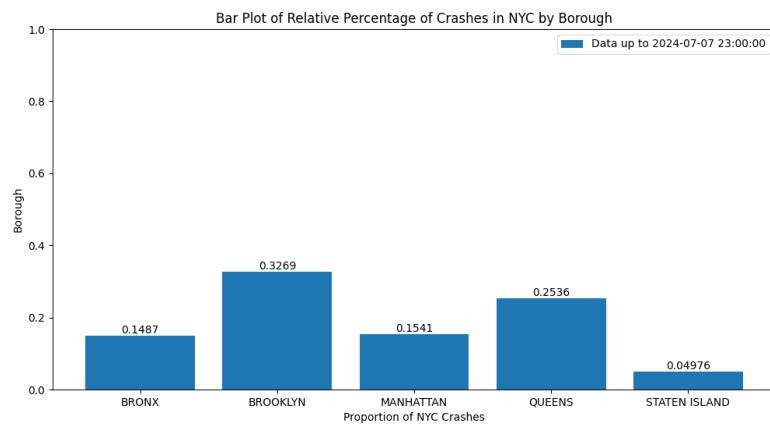
## 7 Visualization

```
# Set titles and labels
ax.set_title('Bar Plot of Relative Percentage of Crashes in NYC by Borough')
ax.set_xlabel('Proportion of NYC Crashes')
ax.set_ylabel('Borough')
ax.legend([f'Data up to {frame}'])
ax.set_ylim(0,1)

# Annotate bars with crash numbers
for i, bar in enumerate(bars):
    ax.text(bar.get_x() + bar.get_width() / 2, bar.get_height(),
            f'{height[i]:.4}', va='bottom', ha='center', color='black')

anim = FuncAnimation(fig, update, frames=pd.date_range(start=
                                         crash_data['crash_datetime'].min(),
                                         end=crash_data['crash_datetime'].max(), freq='h'))

plt.show()
```



## 7.3 Animating Plots and Maps

### 7.3.4 Animating Maps

The same `FuncAnimation` function can be used to animate maps.

#### 7.3.4.1 Basic Idea

1. Process data as a Geo-dataframe with Geopandas
2. Obtain map using libraries like `Contextily`, `Cartopy`, or `Basemap`.
3. Create a frame function that:
  1. Clears previous plot
  2. Plots geo-spatial data
  3. Add a basemap background
4. Animate the map.

#### 7.3.4.2 More Packages

Two more packages are needed to create an animated map.

**Geopandas (Source: Team (2024))** {.smaller} - Extends Pandas to support spatial (geometric) data operations. - Handles GeoDataFrames that store geometries like points, lines, and polygons. - Provides support for reading and writing geospatial data formats - Integrates easily with Matplotlib for plotting spatial data.

**Contextily (Source: Kharlamov (2023))** {.smaller} - Adds basemaps to Matplotlib and Geopandas plots. - Fetches tile-based maps from popular online sources (like OpenStreetMap).

#### Installation

```
pip install geopandas contextily # pip users  
conda install -c conda-forge geopandas contextily # conda users
```

## 7 Visualization

```
import geopandas as gpd
import contextily as ctx
```

### 7.3.4.3 Step 1: Process Data as GeoDataFrame

```
# Read Data
crash_data = pd.read_feather("data/nyccrashes_cleaned.feather")

# Make Date Time Column
crash_data['crash_datetime'] = pd.to_datetime(crash_data['crash_datetime'],
                                              format='%Y-%m-%d %H:%M:%S')

# Create Hour and Date columns
crash_data['date'] = crash_data['crash_datetime'].dt.date
crash_data['hour'] = crash_data['crash_datetime'].dt.hour

# Create GeoPandas Data frame
crash_gdf = gpd.GeoDataFrame(crash_data,
                             geometry=gpd.points_from_xy(crash_data['longitude'],
                                                          crash_data['latitude']), crs="EPSG:4326")

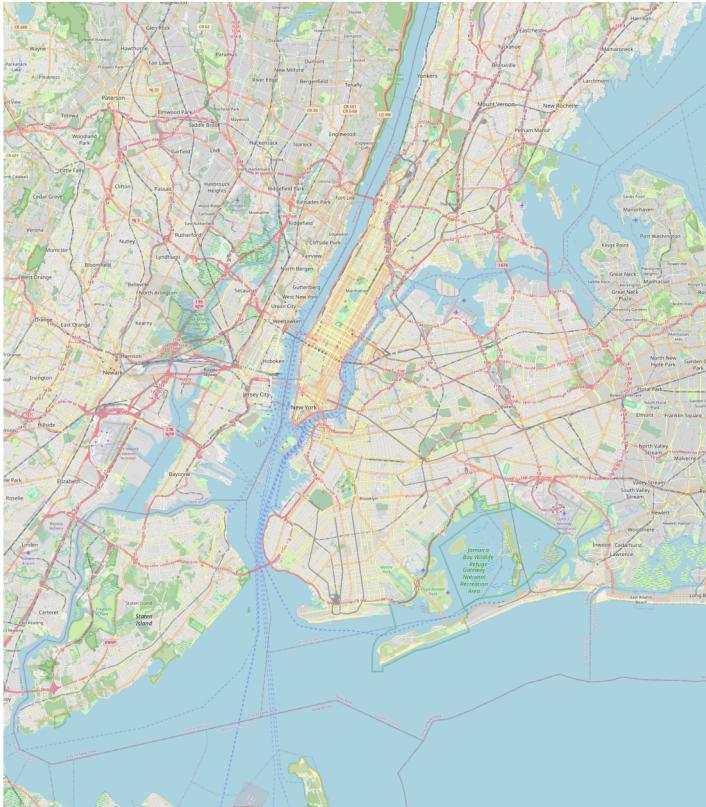
# Transform from EPSG 4326 to EPSG 3857
crash_gdf = crash_gdf.to_crs(epsg=3857)

# Group crash_gdf by date and then hour
crash_grouped = crash_gdf.groupby(['date', 'hour'])
```

## 7.3 Animating Plots and Maps

### 7.3.4.4 Step 2: Get Basemap

```
newyork = ctx.Place("New York", source=ctx.providers.OpenStreetMap.Mapnik, zoom=12)
ny_img = newyork.im # Get Map Image
ny_bbox = newyork.bbox_map # Get Coordinates EPSG 3857
```



#### 7.3.4.5 Step 3: Frame Function

```

crash_grouped = crash_gdf.groupby(['date','hour'])

keys = [key for key in grouped.groups.keys()] # frames

fig, ax = plt.subplots(figsize=(6,7))

def update(frame):
    # Extract date, hr from input
    date, hr = frame

    # Get group
    df = crash_grouped.get_group((date,hr))

    ax.clear() # Clear previous frame

    # Plot using Geopandas df.plot
    df.plot(ax=ax,color='red',edgecolor='k',label=f'Date: {date}\nHour: {hr}')

    ax.imshow(ny_img,extent=ny_bbox) # add basemap
    ax.legend(loc='lower right')
    ax.set_title("Crashes by Hour in NYC - Week of 6/30/24 to 7/7/24")
    ax.axis('off')

```

#### 7.3.4.6 Step 3: Animate the Map

```
fig.subplots_adjust(left=0,right=1,top=0.95,bottom=0)
```

```
anim = FuncAnimation(fig,update,frames=keys)
anim.save("crash_maps_nyc_24-6-30_24-7-7_300dpi.gif",writer='Pillow',fps=2,dpi=300)
```

### 7.3 Animating Plots and Maps

```
plt.show()
```

Crashes by Hour in NYC - Week of 6/30/24 to 7/7/24



#### 7.3.4.7 Other Basemaps

The Contextily has several other basemaps available. Here are some of there most popular options.

- `ctx.providers.Stamen.Toner`: Black-and-white map, minimal details.
- `ctx.providers.Stamen.Terrain`: Topographic map with terrain shading.
- `ctx.providers.OpenStreetMap.Mapnik`: Standard OpenStreetMap, detailed streets.
- `ctx.providers.CartoDB.Positron`: Light map with subtle details.
- `ctx.providers.CartoDB.Voyager`: Colorful and detailed street map.
- `ctx.providers.Esri.WorldImagery`: High-res satellite imagery.

Source: Kharlamov (2023)

#### 7.3.4.8 Satellite Example

This is an example of the same map animated with a satellite basemap.

### 7.3 Animating Plots and Maps

Satellite Map of Crashes by Hour in NYC - Week of 6/30/24 to 7/7/24



### 7.3.5 Advance Example - Animating Maps

The NYC Open Data contains traffic speed data that is updated several times a day. We will be plotting this data on the map we previously made.

Link to data: [DOT Traffic Speeds NBE](#)

These are some of the variables from the dataset:

- DATA\_AS\_OF: Timestamp when the data was recorded.
- SPEED: Recorded speed in miles per hour.
- LINK\_ID: Identifier for road segments.
- LINK\_POINTS: List of geographic coordinates for the road segment.
- ENCODED\_POLY\_LINE: Encoded string representing the polyline of the segment.
- BOROUGH: NYC borough where data is collected (e.g., Manhattan, Brooklyn).

#### 7.3.5.1 Traffic Speed Data

Let's process the DATA\_AS\_OF column as a `datetime` column.

Since the data only contains the speed which is different for different roads, we can create a normalized speed column. This column contains the traffic speed at the time divided by the average speed over that week for each 'LINK\_ID' or unique section of the road.

```
speed_data = pd.read_feather("data/DOT_Traffic_Speeds_NBE_20241005.feather")

# Create Date Column
speed_data['DATA_AS_OF'] = pd.to_datetime(speed_data['DATA_AS_OF'],
                                           format='%m/%d/%Y %I:%M:%S %p')
speed_data['hour'] = speed_data['DATA_AS_OF'].dt.hour
```

### 7.3 Animating Plots and Maps

```
speed_data['date'] = speed_data['DATA_AS_OF'].dt.date

group_link_id = speed_data.groupby(['LINK_ID'])

# Function to Normalize speed
def normalize_speed(row):
    group = group_link_id.get_group(row['LINK_ID'])
    if row['SPEED']:
        row['normalized_speed'] = row['SPEED'] / group['SPEED'].mean()
    else:
        row['normalized_speed'] = None
    return row

speed_data = speed_data.apply(normalize_speed, axis=1)
```

#### 7.3.5.2 Extract Polyline

We can extract the polyline to plot on the map.

```
from shapely.geometry import LineString
import polyline

# Example encoded-poly-line
# {q{vFrzrcMta@kl@hDiInBiIr@}F\mH0sR

def poly(x):
    try:
        # Decode Polyline
        decoded = polyline.decode(x)

        # Create a box for NYC to filter invalid coordinates
        valid_lat_range = (40.47, 40.91)
```

## 7 Visualization

```
valid_lon_range = (-74.25, -73.7)

# Filter coordinates
cleaned_coords = [(lon, lat) for lat, lon in decoded
                   if valid_lat_range[0] <= lat <= valid_lat_range[1] and
                   valid_lon_range[0] <= lon <= valid_lon_range[1]]

# Return coordinates
return cleaned_coords if cleaned_coords else None

except (IndexError, TypeError):
    return None

# Decode Polyline
speed_data['decoded_polyline'] = speed_data['ENCODED_POLY_LINE'].apply(poly)

# Geometry for Geo data frame by processing polylines as linestring
geometry = speed_data['decoded_polyline'].apply(lambda x: LineString(x)
                                                 if x else None)

# Create Geo data frame and convert to EPSG 3857
speed_gdf = gpd.GeoDataFrame(speed_data, geometry=geometry, crs='EPSG:4326')
speed_gdf = speed_gdf.to_crs(epsg=3857)

# Obtain data for July 2
grouped_speed_gdf = speed_gdf.groupby(['date'])
grouped_speed_gdf = grouped_speed_gdf.get_group(datetime.date(2024, 7, 2))

# Group Dataframe by hour
grouped_speed_gdf = grouped_speed_gdf.groupby(['hour'])
```

### 7.3 Animating Plots and Maps

#### 7.3.5.3 Updated Frame Function

```
from matplotlib.colors import Normalize

# Obtain Crash data for July 2 grouped by hour
crash_grouped = crash_gdf.groupby(['date'])
crash_grouped = crash_grouped.get_group(datetime.date(2024, 7, 2))
crash_grouped = crash_grouped.groupby(['hour'])

# Get keys to generate frames
keys = [key for key in crash_grouped.groups.keys()]

# Normalize range to make color bar
norm = Normalize(vmin=speed_gdf['normalized_speed'].min(),
                  vmax=speed_gdf['normalized_speed'].max())

fig, ax = plt.subplots(figsize=(6, 7))

def update(frame):
    hr = frame

    # Get Crash and Traffic Data
    df = crash_grouped.get_group((hr))
    speed_df = grouped_speed_gdf.get_group((hr))

    speed_df = speed_df.drop_duplicates(subset='LINK_ID',
                                         keep='last').reset_index(drop=True)

    ax.clear()

    df.plot(ax=ax, color='red', edgecolor='k',
            label=f'Date: 2 July, 2024\nHour: {hr}' )
```

## 7 Visualization

```
speed_df.plot(ax=ax,column='normalized_speed',cmap='plasma',norm=norm,
              label='Normalized Traffic Speed')

ax.imshow(ny_img,extent=ny_bbox)
ax.legend(loc='lower right')
ax.set_title("Crash and Traffic Data by Hour - 2 July, 2024")
ax.axis('off')

# Add Colorbar
sm = cm.ScalarMappable(cmap='plasma', norm=norm)
fig.colorbar(sm, ax=ax,fraction=0.02, pad=0.03, shrink=0.6,aspect=25)

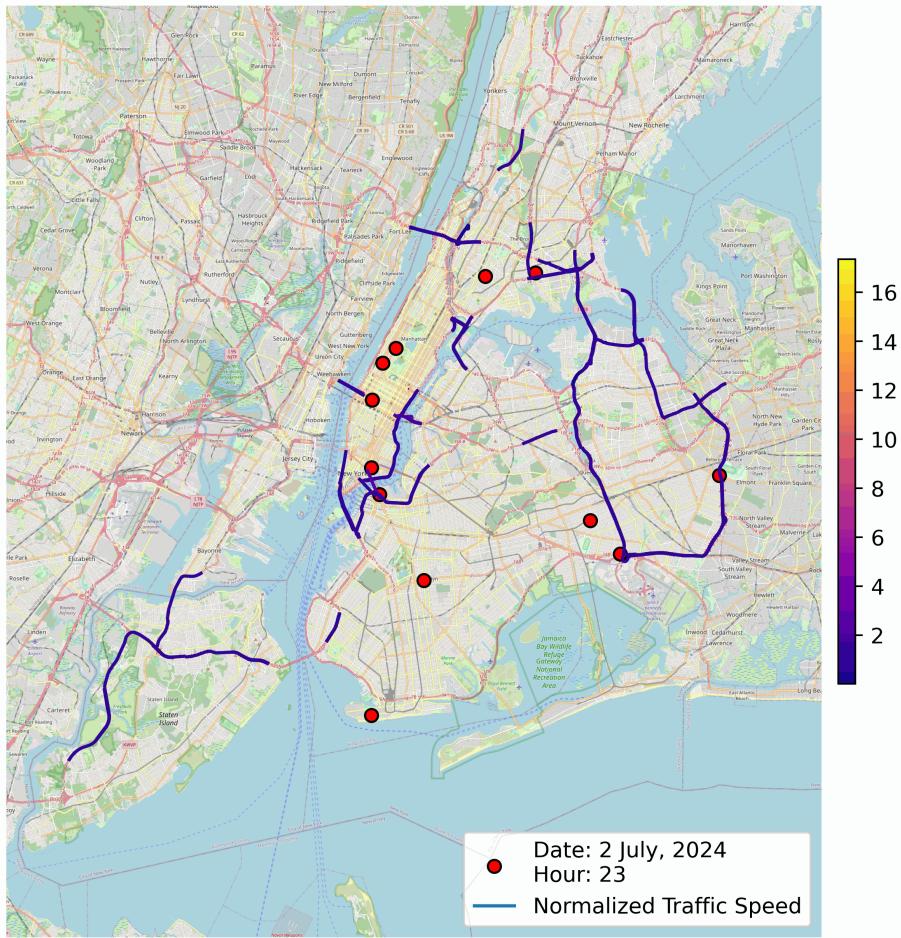
# Remove Whitespace
fig.subplots_adjust(left=0.04,right=0.95,top=1,bottom=0)

anim = FuncAnimation(fig,update,frames=keys)
anim.save("crash_traffic_7_1_24.gif", writer='Pillow', fps=2)

plt.show()
```

### 7.3 Animating Plots and Maps

Crash and Traffic Data by Hour - 2 July, 2024



### 7.3.6 Saving your Animation

#### 7.3.6.1 GIF

To save your animation as a **GIF**:

- **Writer:** Pillow
- **Command:** Use `anim.save()` with the `writer='Pillow'` option.

```
pip install pillow # pip users
conda install -c conda-forge pillow # conda users
```

**Example:**

```
anim.save('animation.gif', writer='Pillow', fps=30, dpi=200)
```

#### 7.3.6.2 MP4

To save your animation as **MP4**:

- **Writer:** ffmpeg
- **Command:** Use `anim.save()` with the `writer='ffmpeg'` option.

```
conda install -c conda-forge ffmpeg # conda users
```

**Pip Users:**

1. Download from [ffmpeg.org](http://ffmpeg.org)
2. Extract the folder
3. Add the bins folder path to your system variables.

**Example:**

### 7.3 Animating Plots and Maps

```
anim.save('animation.mp4', writer='ffmpeg', fps=30, dpi=300)
```

#### 7.3.7 Conclusion

Here is what was covered in this section of the book.

- Animating Plots and Maps using `FuncAnimation`
  - Creating a figure and axes object with `plt.subplots()`
  - Creating an update function
  - Generating Animation
  - Obtaining and adding basemap background for map animations
  - Processing geospatial data
  - Saving animations

If you have more questions or would like to try it:

- Matplotlib's FunctionAnimation Documentation
- Matplotlib Documentation
- Contextily Documentation
- GeoPandas Documentation



# 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 contingency 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

## 8 Statistical Tests and Models

- `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.robjobjects.numpy2ri
from rpy2.robjobjects.packages import importr
rpy2.robjobjects.numpy2ri.activate()

stats = importr('stats')

w0630 = pd.read_feather("data/nyc_crashes_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 .Rprofile
borough    BRONX   BROOKLYN  MANHATTAN   QUEENS   STATEN ISLAND
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)
```

## *8.2 Statistical Modeling*

```
data: structure(c(149L, 129L, 345L, 266L, 164L, 127L, 249L, 227L, 65L, 28L), dim = c(2L, 5L)
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.

## 8 Statistical Tests and Models

```
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()
```

## 8.2 Statistical Modeling

---

<b>Dep. Variable:</b>	y	<b>R-squared:</b>	0.309			
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.292			
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	17.38			
<b>Date:</b>	Wed, 23 Oct 2024	<b>Prob (F-statistic):</b>	3.31e-14			
<b>Time:</b>	10:01:10	<b>Log-Likelihood:</b>	-272.91			
<b>No. Observations:</b>	200	<b>AIC:</b>	557.8			
<b>Df Residuals:</b>	194	<b>BIC:</b>	577.6			
<b>Df Model:</b>	5					
<b>Covariance Type:</b>	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
<b>const</b>	1.8754	0.282	6.656	0.000	1.320	2.431
<b>x1</b>	1.1703	0.248	4.723	0.000	0.682	1.659
<b>x2</b>	0.8988	0.235	3.825	0.000	0.435	1.362
<b>x3</b>	0.9784	0.238	4.114	0.000	0.509	1.448
<b>x4</b>	1.3418	0.250	5.367	0.000	0.849	1.835
<b>x5</b>	0.6027	0.239	2.519	0.013	0.131	1.075
<b>Omnibus:</b>	0.810				<b>Durbin-Watson:</b>	1.978
<b>Prob(Omnibus):</b>	0.667				<b>Jarque-Bera (JB):</b>	0.903
<b>Skew:</b>	-0.144				<b>Prob(JB):</b>	0.637
<b>Kurtosis:</b>	2.839				<b>Cond. No.</b>	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

## 8 Statistical Tests and Models

```
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 (`r1m`) instead of OLS. Note that the model specification and the function interface is similar to R.

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

## 8.2 Statistical Modeling

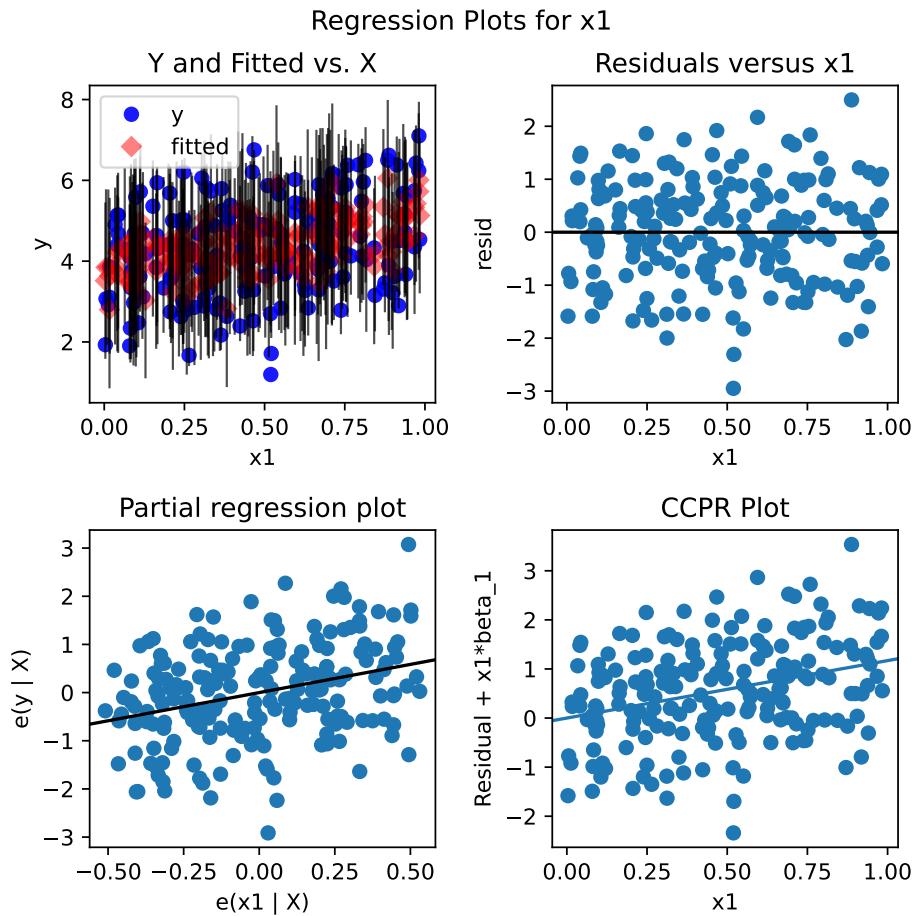
<b>Dep. Variable:</b>	y	<b>No. Observations:</b>	200			
<b>Model:</b>	RLM	<b>Df Residuals:</b>	194			
<b>Method:</b>	IRLS	<b>Df Model:</b>	5			
<b>Norm:</b>	HuberT					
<b>Scale Est.:</b>	mad					
<b>Cov Type:</b>	H1					
<b>Date:</b>	Wed, 23 Oct 2024					
<b>Time:</b>	10:01:10					
<b>No. Iterations:</b>	16					
	coef	std err	z	P> z	[0.025	0.975]
<b>Intercept</b>	1.8353	0.294	6.246	0.000	1.259	2.411
<b>x1</b>	1.1254	0.258	4.355	0.000	0.619	1.632
<b>x2</b>	0.9664	0.245	3.944	0.000	0.486	1.447
<b>x3</b>	0.9995	0.248	4.029	0.000	0.513	1.486
<b>x4</b>	1.3275	0.261	5.091	0.000	0.816	1.839
<b>x5</b>	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 regres-

## 8.2 Statistical Modeling

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

- Random Component: This specifies the distribution of the response variable  $Y$ . It is assumed to be from the exponential family of distributions, such as Binomial for binary data and Poisson for count data.
- Systematic Component: This consists of the linear predictor, a linear combination of unknown parameters and explanatory variables. It is denoted as  $\eta = X\beta$ , where  $X$  represents the explanatory variables, and  $\beta$  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.

## 8 Statistical Tests and Models

- 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
```

## 8.2 Statistical Modeling

```
# 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'
```

## 8 Statistical Tests and Models

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

# 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+00		
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	-4.285	
x1	1.9990	0.211	9.471	0.000	1.585	2.414	
x2	2.1058	0.214	9.853	0.000	1.687	2.520	
x3	1.9421	0.210	9.260	0.000	1.531	2.337	
x4	2.1504	0.232	9.260	0.000	1.695	2.600	
x5	2.0603	0.221	9.313	0.000	1.627	2.449	

### 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

### 8.3 Validating the Results of Logistic Regression

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](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:

## 8 Statistical Tests and Models

- 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.

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

- 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.

### *8.3 Validating the Results of Logistic Regression*

- 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.

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

- 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).

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

- 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.

## 8 Statistical Tests and Models

- 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 Validating the Results of Logistic Regression

#### 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:

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

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

#### 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,

## 8 Statistical Tests and Models

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, ran

# 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)
```

### 8.3 Validating the Results of Logistic Regression

```
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)

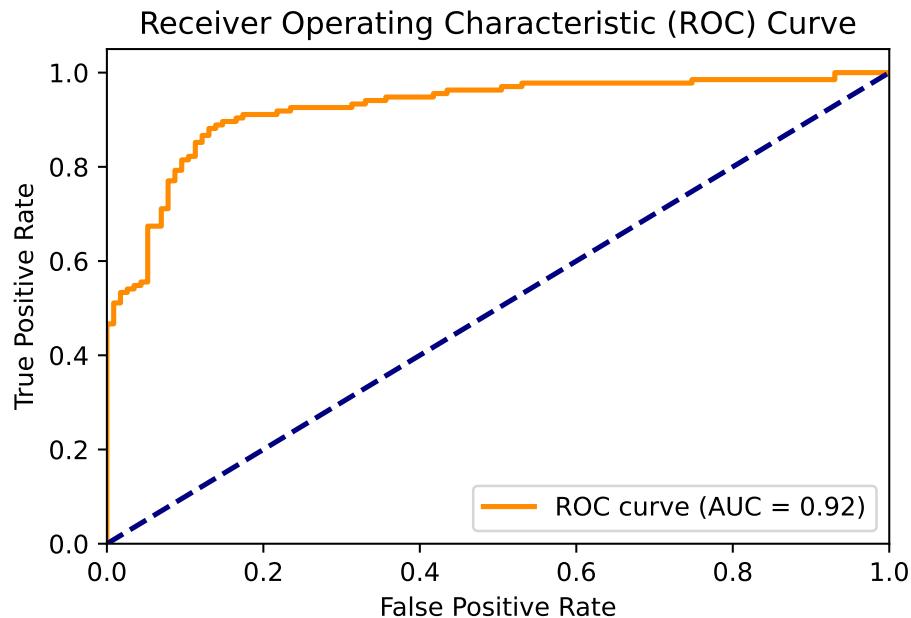
# Print AUC
print(f"AUC: {roc_auc:.2f}")

# 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 classifier)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

## 8 Statistical Tests and Models

```
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

AUC: 0.92



We could pick the best threshold that optimizes 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 get binary predictions
    f1 = f1_score(y_test, y_pred_thresh)
    f1_scores.append(f1)
```

## 8.4 LASSO Logistic Models

```
# 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,

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

where:

## 8 Statistical Tests and Models

- $\hat{p}_i = \frac{1}{1+e^{-X_i\beta}}$  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.

## 8.4 LASSO Logistic Models

```
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_11(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_11(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 = []
```

## 8 Statistical Tests and Models

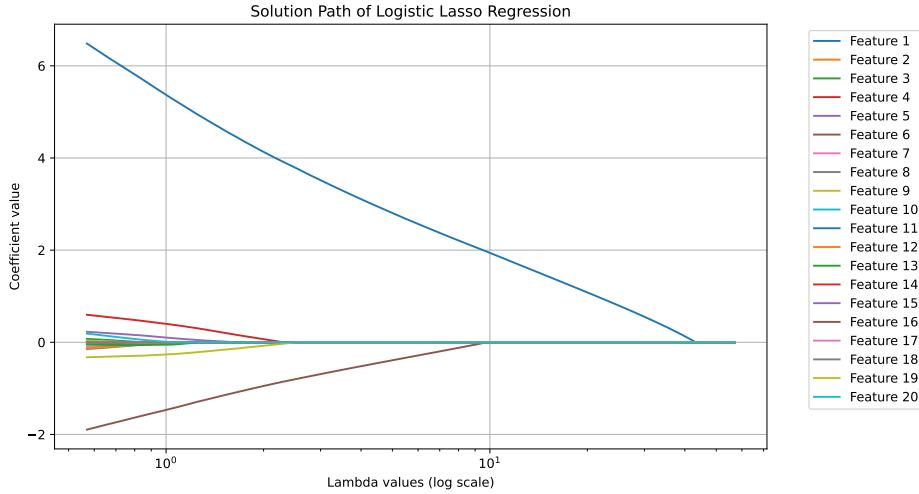
```
# Step 6: Fit logistic regression with L1 regularization (Lasso) for each lambda value
for lam in lambda_values:
    model = LogisticRegression(penalty='l1', solver='liblinear', C=1/lam, max_iter=1000)
    model.fit(X_scaled, y)
    coefficients.append(model.coef_.flatten())

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

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

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

## 8.4 LASSO Logistic Models



### 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:

## 8 Statistical Tests and Models

- 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-validation
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
)
```

## 8.4 LASSO Logistic Models

```
# 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 standardize them so that they have mean zero and standard deviation one. There are multi-

## *8 Statistical Tests and Models*

ple 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

## 8.4 LASSO Logistic Models

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 need to be coded into numerical 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
```

## 8 Statistical Tests and Models

```
from sklearn.compose import ColumnTransformer
import numpy as np
from scipy.special import expit # Sigmoid function

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

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

## 8.4 LASSO Logistic Models

```
'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)
    ]
)
```

## 8 Statistical Tests and Models

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.

## 8.4 LASSO Logistic Models

```
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

## *8 Statistical Tests and Models*

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 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.

## 9 Machine Learning: Overview

- $A$ : 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:

## 9.1 Introduction

- 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

## 9.2 Bias-Variance Tradeoff

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

## *9 Machine Learning: Overview*

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.

## *9 Machine Learning: Overview*

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.

### 9.3 Crossvalidation

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)
```

## 9 Machine Learning: Overview

```
# 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, random_state=42)

# 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=0.5)
    plt.plot(X_line, y_true, label='True function', color='black', linestyle='solid')

    for n_knots in knots_list:
        # Create a spline model with fixed degree = 3 (cubic) and varying knots
        spline = SplineTransformer(degree=3, n_knots=n_knots, include_bias=False)
        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)

        # Compute and store errors
        train_errors.append(np.mean((y_train - y_pred_train) ** 2))
        test_errors.append(np.mean((y_test - y_pred_test) ** 2))
```

### 9.3 Crossvalidation

```
# 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)

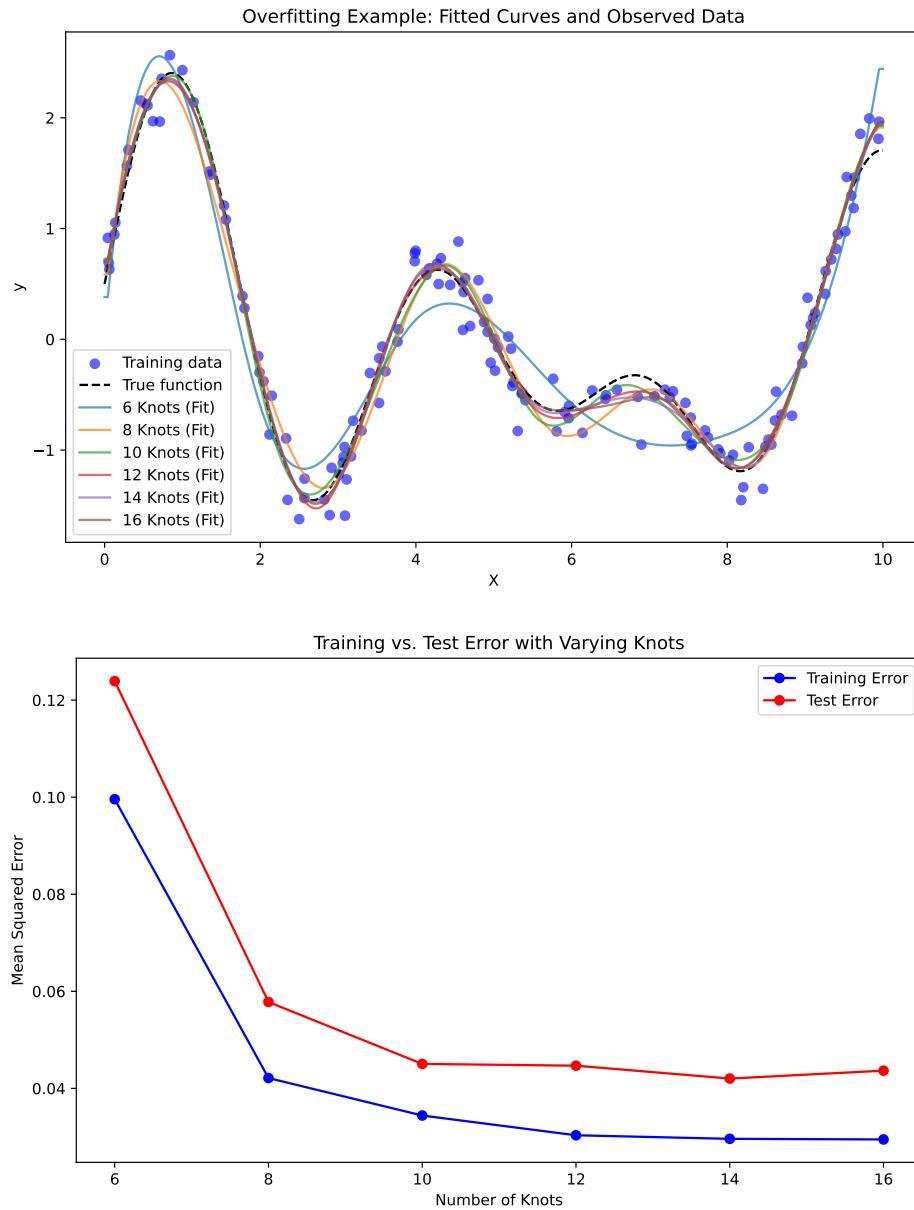
plt.title('Overfitting Example: Fitted Curves and Observed Data')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()

# 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]

## 9 Machine Learning: Overview



### 9.3 Crossvalidation

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.

## *9 Machine Learning: Overview*

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: Foundation

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:

## 10 Supervised Learning

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

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  $\{v_{j,1}, v_{j,2}, \dots, v_{j,k_j}\}$ . The search space at node  $m$  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`.

## 10 Supervised Learning

- 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

$$H(Q_m) = \frac{1}{n_m} \sum_{i=1}^{n_m} (y_i - \bar{y}_m)^2,$$

## 10.1 Decision Trees: Foundation

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 Boosted Trees

Boosted trees are a powerful ensemble technique in machine learning that combine multiple weak learners, typically decision trees, into a strong learner. Unlike bagging methods, which train trees independently, boosting fits models sequentially, with each new model correcting the errors of the previous ensemble. Gradient boosting, one of the most popular variants, optimizes a loss function by iteratively adding trees that reduce the residual errors of the current ensemble.

### 10.2.1 Introduction

Boosted trees build on the general concept of boosting, which aims to create a strong predictor from a series of weak predictors. In boosted trees, the weak learners are shallow decision trees, often referred to as “stumps,” and they are added sequentially to the model. At each step, a new tree focuses on the training instances that are hardest to predict, improving overall accuracy. This iterative focus on “hard-to- predict” instances is the defining characteristic of boosting.

The effectiveness of boosted trees has made them popular for various tasks, including classification, regression, and ranking. They also form the foundation for algorithms like XGBoost, LightGBM, and CatBoost, known for their speed and scalability.

### 10.2.2 Boosting Process

The boosting process in gradient boosted trees builds an ensemble by adding trees iteratively, each designed to minimize the residual errors from the combined predictions of the previous trees. This iterative approach allows the model to refine its predictions by optimizing a loss function, denoted as  $L(y, F(x))$ , where  $y$  is the true value and  $F(x)$  is the model's prediction.

#### 10.2.2.1 Model Iteration

The boosting process can be delineated as follows:

1. **Initialization:** Start with a base model  $F_0(x)$ , which is usually the mean of the target variable in regression or the log odds in classification:
  - For regression:  $F_0(x) = \text{mean}(y_i)$ .
  - For classification:  $F_0(x) = \log\left(\frac{P(y=1)}{1-P(y=1)}\right)$ .
2. **Iterative Boosting:**

At each iteration  $m$ :

- Compute the pseudo-residuals, representing the negative gradient of the loss function with respect to the current model predictions. The residuals at iteration  $m$  are defined as:

$$r_i^{(m)} = - \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \Big|_{F(x)=F_{m-1}(x)}.$$

The residuals guide the next tree to focus on reducing the largest errors from the previous iteration.

- Fit a new tree  $h_m(x)$  to the pseudo-residuals. The new tree is trained to predict the residuals of the current ensemble model, identifying where the model needs the most improvement.
- Update the model as the sum of the previous model and the newly added tree, scaled by a learning rate  $\eta$ :

$$F_m(x) = F_{m-1}(x) + \eta h_m(x).$$

The learning rate, a small positive number (e.g., 0.01 to 0.1), controls the contribution of each tree, ensuring incremental improvements and reducing the risk of overfitting.

### 3. Final Model:

After  $M$  iterations, the ensemble model is given by:

$$F_M(x) = F_0(x) + \sum_{m=1}^M \eta h_m(x).$$

The final model  $F_M(x)$  represents the sum of the initial model and the incremental improvements made by each of the  $M$  trees, with each tree trained to correct the residuals of the ensemble up to that point.

### 10.2.3 Key Concepts

1. **Loss Function:** The loss function measures the discrepancy between the actual and predicted values. It guides the model updates. Common choices include:
  - Squared error for regression:  $L(y, F(x)) = \frac{1}{2}(y - F(x))^2$ .
  - Logistic loss for binary classification:  $L(y, F(x)) = \log(1 + \exp(-y F(x)))$ .
2. **Learning Rate:** The learning rate scales the contribution of each tree and helps control the speed of learning. A smaller learning rate typically requires more trees but results in a more robust model with better generalization.
3. **Regularization:** Boosted trees incorporate regularization to avoid overfitting, including:
  - Tree depth: Limits the maximum depth of each tree, reducing model complexity.
  - L1/L2 penalties: Regularize the weights of the trees, similar to Lasso and Ridge regression.
  - Subsampling: Uses a fraction of the training data at each iteration, making the model more robust to overfitting and improving generalization.

### 10.2.4 Why Boosting Works

The iterative approach of boosting, focusing on correcting the errors of the ensemble at each step, distinguishes gradient boosting from other ensemble methods like bagging or random forests. Key reasons for its effectiveness include:

1. **Error Correction:** By focusing on the hardest-to-predict instances, boosting gradually improves model accuracy, leading to better performance than models trained independently.
2. **Weighted Learning:** Boosting adjusts the weights of training samples based on errors, ensuring that the model learns disproportionately from difficult cases, reducing bias.
3. **Flexibility:** Boosted trees can handle various loss functions, making them suitable for different types of tasks, including regression, classification, and ranking.

#### 10.2.5 Applications and Popular Implementations

Boosted trees are widely used in real-world applications, ranging from financial risk modeling to predictive maintenance. They are also favored in machine learning competitions due to their interpretability and robustness. Popular implementations include:

- XGBoost: Known for its speed and performance, with features like regularization, column sampling, and advanced tree pruning.
- LightGBM: Optimized for speed and scalability, using histogram-based algorithms to handle large datasets efficiently.
- CatBoost: Effective with categorical features, using advanced encoding techniques and built-in support for categorical variables.

### 10.3 Naive Bayes

This section was contributed by Suha Akach.

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.

### 10.3.1 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:

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

where:

- $P(y | X)$ : Posterior probability of class  $y$  given the input features  $X$ .
- $P(X | y)$ : 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  $X$ .

#### 10.3.1.1 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  $P(X | y)$  to be decomposed as:

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

## 10 Supervised Learning

where  $X = \{x_1, x_2, \dots, x_n\}$  represents the feature vector with  $n$  features, and  $P(x_i | y)$  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  $P(x_i | y)$ . 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  $P(x_i | y)$  is estimated as:

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

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,  $P(x_i | y)$  is modeled using the Gaussian distribution with mean  $\mu_{y,i}$  and variance  $\sigma_{y,i}^2$ :

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

### 10.3 Naive Bayes

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

#### 10.3.1.2 Class Prediction

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

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

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

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

The predicted class is the one that maximizes this expression.

#### 10.3.1.3 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

## 10 Supervised Learning

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.

### 10.3.1.4 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.

#### 10.3.1.5 Laplace Smoothing

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

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

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

#### 10.3.2 Types of Naive Bayes:

There are 5 types of Naive Bayes classifiers:

- **Gaussian Naive Bayes:** This type of Naive Bayes is used when the dataset consists of numerical features. It assumes that the features follow a Gaussian (normal) distribution. This model is fitted by finding the mean and standard deviation of each class (IBM, 2024).
- **Categorical Naive Bayes:** When the dataset contains categorical features, we use Categorical Naive Bayes. It assumes that each feature follows a categorical distribution.

- **Bernoulli Naive Bayes:** Bernoulli Naive Bayes is applied when the features are binary or follow a Bernoulli distribution. That is, variables with two values, such as True and False or 1 and 0. (IBM, 2024).
- **Multinomial Naive Bayes:** Multinomial Naive Bayes is commonly used for text classification tasks. It assumes that features represent the frequencies or occurrences of different words in the text.
- **Complement Naive Bayes:** Complement Naive Bayes is a variation of Naive Bayes that is designed to address imbalanced datasets. It is particularly useful when the majority class overwhelms the minority class in the dataset. It aims to correct the imbalance by considering the complement of each class when making predictions (GeeksforGeeks, 2023).

Each type of Naive Bayes classifier is suitable for different types of datasets based on the nature of the features and their distribution. By selecting the appropriate Naive Bayes algorithm, we can effectively model and classify data based on the given features.

### 10.3.3 Naive Bayes w/ NYC Crash Data

Since we have an imbalanced dataset where there are more non severe crashes than severe, we will use Complement Naive Bayes classifier to predict severe crashes based on our predictors.

Our assumed independent predictors after feature engineering are: `borough`, `location`, `household_median_income`, `crash_date`, `crash_time`, `time_category`, `contributing_factor_vehicle_1`, `vehicle_type_code_1`.

We assume a crash is severe if there are more than 0 persons killed and/or injured.

### 10.3 Naive Bayes

```
import pandas as pd
import numpy as np
import warnings
import uszipcode as us

# Disable warnings
warnings.filterwarnings("ignore")

# Load the dataset
df = pd.read_feather('data/nyc_crashes_cleaned.feather')

# 1. Separate crash_datetime into date and time (convert datetime into numeric features)
df['crash_datetime'] = pd.to_datetime(df['crash_datetime'])
df['crash_date'] = df['crash_datetime'].dt.date
df['crash_time'] = df['crash_datetime'].dt.time

# Extract relevant features from datetime (for example: hour)
df['hour'] = df['crash_datetime'].dt.hour

# 2. Create time_category column with updated time intervals
def categorize_time(hour):
    if 0 <= hour < 6:
        return 'midnight' # 12:00 AM to 5:59 AM
    elif 6 <= hour < 12:
        return 'morning' # 6:00 AM to 11:59 AM
    elif 12 <= hour < 18:
        return 'afternoon' # 12:00 PM to 5:59 PM
    elif 18 <= hour < 21:
        return 'evening' # 6:00 PM to 8:59 PM
    else:
        return 'night' # 9:00 PM to 11:59 PM

df['time_category'] = df['hour'].apply(categorize_time)
```

## 10 Supervised Learning

```
# 3. Add median household income for each zip code using the uszip package
def get_median_income(zipcode):
    try:
        z = us.search.by_zipcode(str(zipcode))
        if z:
            return z.median_income
        else:
            return np.nan
    except:
        return np.nan

df['household_median_income'] = df['zip_code'].apply(get_median_income)
```

### 10.3.3.1 Defining predictors and target variable.

```
from imblearn.over_sampling import SMOTE
from sklearn.naive_bayes import ComplementNB # Complement Naive Bayes
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

# Target variable for severe crashes based on number of persons killed and injured
df['severe_crash'] = (df['number_of_persons_killed'] > 0) | (df['number_of_persons_injured'] > 0)

# Selected predictors
predictors = ['borough', 'location', 'household_median_income',
              'crash_date', 'crash_time', 'time_category',
              'contributing_factor_vehicle_1', 'vehicle_type_code_1']

# Convert categorical columns into dummy variables
X = pd.get_dummies(df[predictors], drop_first=True)
```

```
# Handle NaN or missing values in numeric columns if necessary
X.fillna(0, inplace=True)

# Target variable
y = df['severe_crash']
```

### 10.3.3.2 Fitting Our Model

After accounting for imbalanced data using SMOTE, we find the k nearest neighbors in the minority class to generate synthetic points between the chosen point and its neighbors. In this case, 7 nearest neighbors are considered to generate synthetic samples. We also use alpha 0.5 for our laplace smoothing to apply an equal level of smoothing across all feature probabilities. Finally, we apply a threshold to check if the probability for class 1 (positive class) is greater than 0.4, and if so, assigns the sample to class 1, otherwise to class 0.

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3255)

# Resample to balance the classes (SMOTE)
smote = SMOTE(random_state=3255, k_neighbors=7)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Initialize the Complement Naive Bayes model with Laplace smoothing
model = ComplementNB(alpha=0.5)

# Fit the model
model.fit(X_train_res, y_train_res)

# Get the probabilities for each class
y_prob = model.predict_proba(X_test)
```

## 10 Supervised Learning

```
# Apply threshold to the predicted probabilities
y_pred = (y_prob[:, 1] > 0.4).astype(int)

# Evaluate the model with the new threshold
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(report)
```

Confusion Matrix:

```
[[111  96]
 [ 64 104]]
```

Classification Report:

	precision	recall	f1-score	support
False	0.63	0.54	0.58	207
True	0.52	0.62	0.57	168
accuracy			0.57	375
macro avg	0.58	0.58	0.57	375
weighted avg	0.58	0.57	0.57	375

### 10.3.3.3 Interpreting our Results:

Recall vs. Precision: Our model is better at identifying severe crashes (higher recall of 0.62) but is less accurate when it does so (lower precision of 0.52). This means that while the model catches a good portion of the

## *10.4 Handling Imbalanced Data with SMOTE*

severe crashes, it also misidentifies a fair number of non-severe crashes as severe.

F1-Score: The F1-score of 0.57 for severe crashes is a balanced measure, showing that our model is moderately effective at distinguishing severe crashes from non-severe ones.

Improvement Opportunities: The overall accuracy of 58% is moderate, so there's potential for further optimization.

In conclusion, our CNB model is fairly good at predicting severe crashes with an accuracy of 58% but may benefit from more tuning to improve precision and reduce false positives, especially in the context of an imbalanced dataset where we don't have many instances of people injured or killed.

## **10.4 Handling Imbalanced Data with SMOTE**

This section is presented by Olivia Kashalapov.

### **10.4.1 Introduction**

This presentation on SMOTE will cover the following topics:

- What is class imbalance and why is it important?
- What is SMOTE?
- Why do we use it and how does it work?
- Why is SMOTE better than other traditional methods for handling class imbalance?
- Using SMOTE in data visualization, analysis, model training, and evaluation.
- The downsides and limitations of SMOTE.

### 10.4.2 Class Imbalance

**Before we can learn about SMOTE, we have to understand class imbalance and why it is important.** - Class imbalance is a common issue in many datasets in which the distribution of examples within the dataset are either biased or skewed. - Let's say there is a dataset for a rare medical diagnosis and there are two classes, with disease and without disease. The data can be taken to explore if there is a certain variable that makes it more likely for an individual to be diagnosed with this rare disease. Since the disease is rare, the class of people with the disease is going to be significantly smaller than the class of those without. In this case, the data will be skewed towards the class of people without the disease and this may skew the findings of the predictive model. - Addressing class imbalance improves the performance of models and increases model accuracy. - Unfortunately, "most of the machine learning algorithms used for classification were designed around the assumption of an equal number of examples for each class." – this is where SMOTE comes in!

### 10.4.3 Synthetic Model Oversampling Technique

- Synthetic Model Oversampling Technique, better known as SMOTE, is an algorithm that focuses on the feature space (already existing data points) in the minority class to generate new data points to create balance between the majority and minority classes.
- Here is how it works:
  1. It identifies imbalance in the dataset and recognizes the minority class.
  2. Using an existing data point in the minority class, it takes the difference between the point and a nearest neighbor.
  3. It then multiplies the difference by random number between 1 and 0.

## *10.4 Handling Imbalanced Data with SMOTE*

4. This difference is then added to the sample to generate new synthetic example in the featured space (minority class).
  5. This continues with next nearest neighbor up to user-defined number ( $k$ ), In other words, there are  $k$  synthetic points created between the selected existing point and its  $k$  nearest neighbors.
  6. This is repeated for all data points within the minority class
- In simpler terms, it creates synthetic data points in the minority class by creating points that lie in between pre-existing ones
  - SMOTE works well because it attempts to remove bias on skewed distributions and balances the data using pre-existing data points within the dataset. It uses the data already being used to create realistic randomized data points.

### **10.4.4 SMOTE versus Traditional Methods**

- There are other ways to handle class imbalance within data sets, other than SMOTE
- One method of this is random under sampling the majority class where random points are chosen in the majority class to be discarded. This often leaves out too much data which could be important for training the predictive model and there are chances that the remaining sample ends up being biased.
- Another option is random oversampling the minority class which is done by randomly duplicating points within the minority class. Although this fixes the class imbalance, it could also lead to overfitting of the model, making it less accurate to the true population.
- SMOTE mitigates the problems of random oversampling and under sampling since the generated data points are not replications of already occurring instances and the majority class keeps all of its existing instances. It is much more unlikely for there to be a case of overfitting the model and no useful information will be left out of the model either.

#### 10.4.5 Installation and Setup

To install SMOTE, you can type one of two commands into your terminal:

```
pip install imbalanced-learn
# OR
conda install imbalanced-learn
```

To import SMOTE on Python, you use this command:

```
from imblearn.over_sampling import SMOTE
```

Just like that, you are ready to use SMOTE!

#### 10.4.6 Data Preparation

Here I am creating a simple data set in which there is an extremely apparent class imbalance. I am doing this rather than using past data sets so that you can truly see the work of SMOTE without other factors that can make the process confusing.

```
import pandas as pd
import numpy as np

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

# Generate synthetic dataset
data_size = 1000
class_0_size = int(data_size * 0.9) # 90% for class 0
class_1_size = data_size - class_0_size # 10% for class 1
```

## 10.4 Handling Imbalanced Data with SMOTE

```
# Class 0: Majority class
feature1_class_0 = np.random.normal(0, 1, class_0_size)
feature2_class_0 = np.random.normal(1, 1, class_0_size)
target_class_0 = np.zeros(class_0_size)

# Class 1: Minority class
feature1_class_1 = np.random.normal(2, 1, class_1_size)
feature2_class_1 = np.random.normal(2, 1, class_1_size)
target_class_1 = np.ones(class_1_size)

# Combine the majority and minority class
feature1 = np.concatenate([feature1_class_0, feature1_class_1])
feature2 = np.concatenate([feature2_class_0, feature2_class_1])
target = np.concatenate([target_class_0, target_class_1])

# Create a DataFrame
data = pd.DataFrame({
    'feature1': feature1,
    'feature2': feature2,
    'target': target
})

# Display the first few rows
print(data.head())

# Save the dataset as CSV for further use
data.to_csv('synthetic_class_imbalance_dataset.csv', index=False)
```

	feature1	feature2	target
0	0.496714	1.368673	0.0
1	-0.138264	0.606661	0.0
2	0.647689	1.028745	0.0

## 10 Supervised Learning

```
3  1.523030  2.278452      0.0
4 -0.234153  1.191099      0.0
```

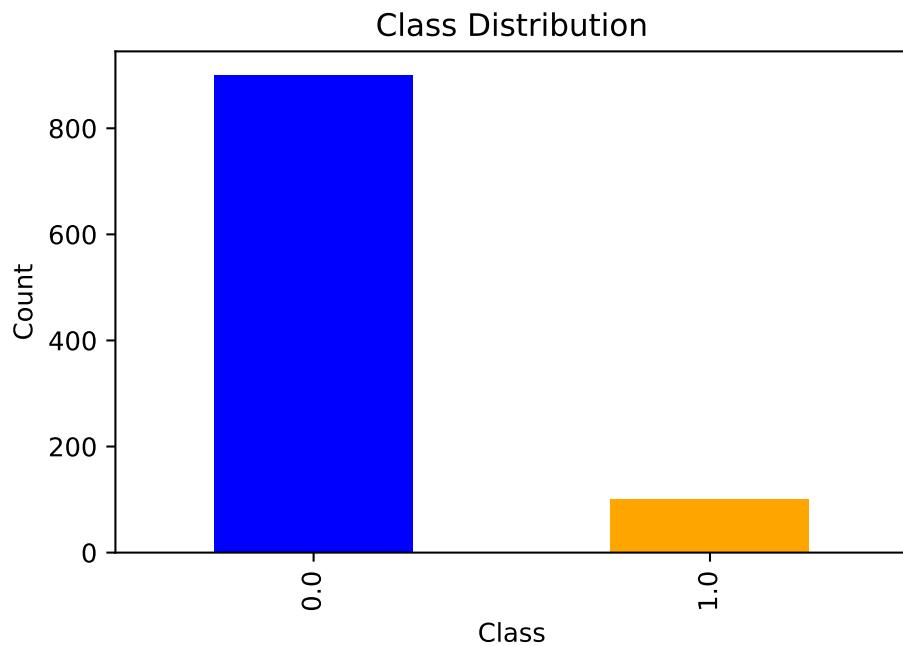
### 10.4.7 Data Visualization

Before using SMOTE to balance the classes, we can view the distribution of the minority and majority classes. It is quite evident that class 0 has many more instances when compared to class 1. This means any predictive models made with this exact data are likely to be skewed towards class 0.

```
import matplotlib.pyplot as plt

# Visualize the class distribution
data['target'].value_counts().plot(kind='bar', color=['blue', 'orange'])
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```

## 10.4 Handling Imbalanced Data with SMOTE



### 10.4.8 Implementing SMOTE

Now we can use SMOTE on the dataset I created to handle the imbalance between the classes.

```
# Split the dataset into features (X) and target (y)
X = data[['feature1', 'feature2']]
y = data['target']

# Check the initial class distribution
print(y.value_counts())

from imblearn.over_sampling import SMOTE
```

## 10 Supervised Learning

```
# Initialize SMOTE with custom parameters
smote = SMOTE(
    sampling_strategy=1,      # Resample minority to 100% of majority class
    k_neighbors=5,            # Use 5 nearest neighbors to generate synthetic samples
    random_state=42,          # Set a random state for reproducibility
)

# Apply SMOTE to the dataset
X_resampled, y_resampled = smote.fit_resample(X, y)
print(pd.Series(y_resampled).value_counts())
```

```
target
0.0    900
1.0    100
Name: count, dtype: int64
target
0.0    900
1.0    900
Name: count, dtype: int64
```

In this example of SMOTE application, I am utilizing multiple customized parameters. Without these specifications, the SMOTE() command will resample the minority to have the same number of instances as the majority class and utilize the 5 nearest neighbors to generate these samples. **Without a specified random state, SMOTE will choose one so it is recommended to include that parameter for reproducibility.**

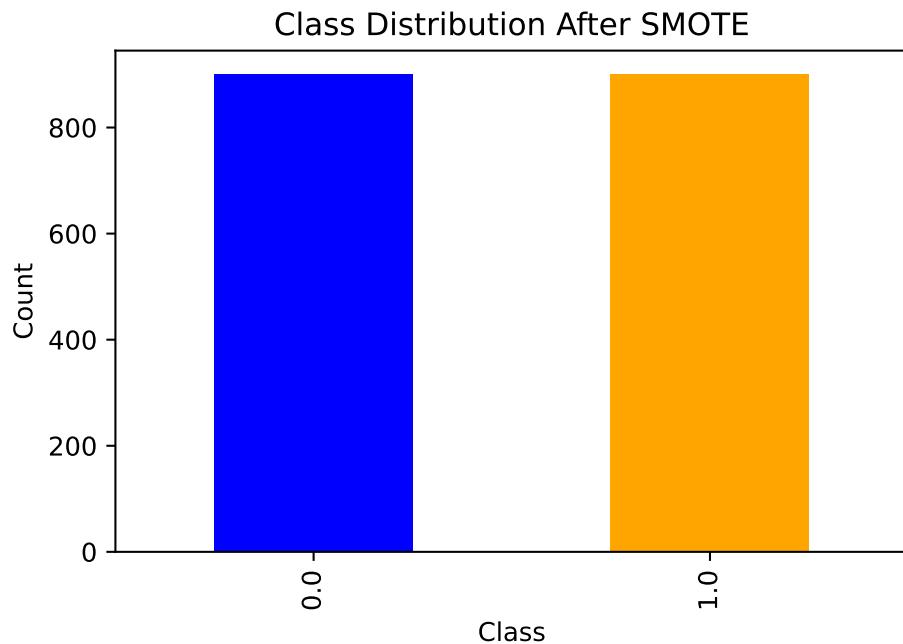
### 10.4.9 Visualization after SMOTE

*Keep in mind, the minority class will remain smaller than the majority due to the sampling\_strategy parameter included in the previous slide.*

## 10.4 Handling Imbalanced Data with SMOTE

```
import matplotlib.pyplot as plt

# Visualize the class distribution after SMOTE
pd.Series(y_resampled).value_counts().plot(kind='bar', color=['blue', 'orange'])
plt.title('Class Distribution After SMOTE')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



Much better!

### 10.4.10 Model Training using SMOTE

Now that the dataset is balanced, we can train the machine learning model. In this case, I am using logistic regression, which works well in many binary cases.

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Split the resampled data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled)

# Initialize and train a Logistic Regression model
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test)

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Print the classification report (includes precision, recall, F1-score)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Confusion Matrix:

```
[[227  40]
 [ 29 244]]
```

## 10.4 Handling Imbalanced Data with SMOTE

Classification Report:

	precision	recall	f1-score	support
0.0	0.89	0.85	0.87	267
1.0	0.86	0.89	0.88	273
accuracy			0.87	540
macro avg	0.87	0.87	0.87	540
weighted avg	0.87	0.87	0.87	540

There were 227 cases in which the model correctly predicted class 1 (true positive) and 244 cases in which the model correctly predicted class 0 (true negative). There were 40 cases in which class 1 was predicted, but it was class 0. Lastly, there were 29 cases in which class 0 was predicted, but it was class 1.

The accuracy of this model is 87%, which means it correctly predicts the class 87% of the time.

### 10.4.11 Model Evaluation

So how good is this model actually? Here I am going to use the ROC curve and AUC, since the last slide already touched on accuracy and confusion matrix results.

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

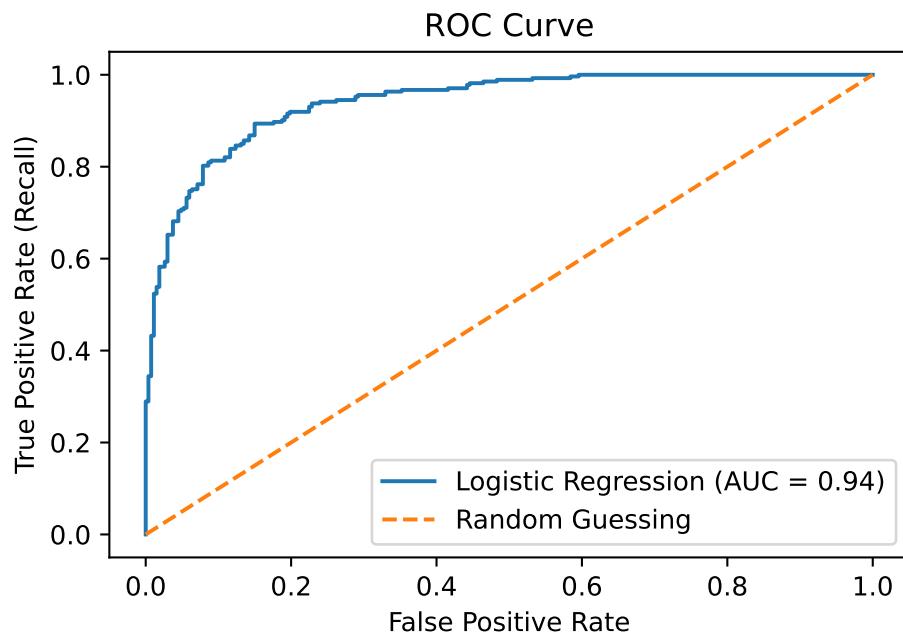
# Compute predicted probabilities for ROC curve
y_prob = log_reg.predict_proba(X_test)[:, 1]

# Generate ROC curve values
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
```

## 10 Supervised Learning

```
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (AUC = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], linestyle='--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('ROC Curve')
plt.legend()
plt.show()

# Compute AUC
auc = roc_auc_score(y_test, y_prob)
print("AUC:", auc)
```



## 10.4 Handling Imbalanced Data with SMOTE

AUC: 0.9407197047646485

The area under the curve (AUC) determines how good a model is at distinguishing between the positive and negative classes, with a score of 1 being perfect. In this case, with an AUC of 0.941, the model is extremely good at making these distinctions.

### 10.4.12 Another Example Using NYC Crash Data

For the NYC Crash Severity Prediction homework we did, SMOTE came in helpful when it came to creating synthetic data in our model predictors. Classes like ‘contributing\_factor\_vehicle\_4’ and ‘vehicle\_type\_code\_5’ were missing a lot of data, making our prediction models very skewed.

Here is how I used SMOTE to fix this issue.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, f1_score
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE

# Identify merged data
merged_df = pd.read_feather("data/nyc_crashes_merged.feather")

merged_df['Severe Crash'] = ((merged_df['number_of_persons_killed'] > 0) |
                             (merged_df['number_of_persons_injured'] > 1)).astype(int)

# Select predictors
predictors = ['borough', 'on_street_name', 'cross_street_name',
              'off_street_name', 'contributing_factor_vehicle_1',
```

## 10 Supervised Learning

```
'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',
'median_home_value', 'median_household_income']

# Initialize data
X = pd.get_dummies(merged_df[predictors], drop_first=True)
y = merged_df['Severe Crash']

# Impute any missing values
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_imputed,
                                                    y, test_size=0.2,
                                                    random_state=1234)

# Apply SMOTE to the training data
smote = SMOTE(random_state=1234)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

### 10.4.13 NYC Crash Data Part 2

Now we can continue on to logistic regression modeling and evaluating the accuracy of our predictive model with a balanced dataset!

## 10.4 Handling Imbalanced Data with SMOTE

```
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

# Fit the logistic regression model
model = LogisticRegression()
model.fit(X_train_resampled, y_train_resampled)

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

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

# 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:

```
[[228  86]
 [ 20  16]]
```

Accuracy: 0.70

Precision: 0.16

Recall: 0.44

This predictive model correctly determined whether a NYC car crash was severe or not 83% of the time, which is decently accurate. Without the use of SMOTE, this number would have been much lower.

#### 10.4.14 Limitations and Challenges of SMOTE

- It does not take into consideration other neighboring classes, which can cause overlap between classes
- It is also not the most effective for high-dimensional data (another reason I made a sample dataset rather than using one with many predictors)
- There is no consideration of the quality of the synthetic samples
- It is only suitable for continuous variables
- Your choice of k can severely impact the quality of the synthetic data

#### 10.4.15 Conclusion

- SMOTE is a tool used to handle class imbalance in datasets
- It creates synthetic data points utilizing instances already in the minority class
- This creates a balanced data set which can improve model prediction and be used for a variety of machine learning applications

## *10.4 Handling Imbalanced Data with SMOTE*

### **10.4.16 Further Readings**

- Brownlee, J. (2020). A gentle introduction to imbalanced classification
- Brownlee, J. (2021). Smote for imbalanced classification with python
- Galli, S. (2023). Overcoming class imbalance with SMOTE: How to tackle imbalanced datasets in Machine Learning
- Imbalanced data: How to handle imbalanced classification problems. (2023)
- Maklin, C. (2022). Synthetic minority over-sampling technique (smote)
- Or, D. B. (2024). Solving the class imbalance problem
- Smote, Package Imbalanced Learning Manual



# 11 Unsupervised Learning

## 11.1 Principle Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms a dataset with potentially correlated features into a set of uncorrelated components. These components are ordered by the amount of variance each one captures, allowing PCA to simplify the data structure while retaining the most informative features. This approach is widely used in unsupervised learning, particularly for data compression and noise reduction.

### 11.1.1 Theory

PCA works by identifying directions, or “principal components,” along which the variance of the data is maximized. Let  $X$  be a dataset with  $n$  observations and  $p$  features, represented as an  $n \times p$  matrix. The principal components are derived from the eigenvectors of the data’s covariance matrix, indicating the directions in which the data exhibits the most variation.

1. Standardization: To ensure that each feature contributes equally to the analysis, the features in  $X$  are often standardized to have zero mean and unit variance. This prevents features with larger scales from dominating the principal components.

## 11 Unsupervised Learning

2. Covariance Matrix: The covariance matrix  $S$  of the dataset is computed as:

$$S = \frac{1}{n-1} X^T X.$$

This matrix captures the pairwise covariances between features, showing how they vary together.

3. Eigenvalue Decomposition: PCA proceeds by calculating the eigenvalues and eigenvectors of the covariance matrix  $S$ . The eigenvectors represent the principal components, and the eigenvalues measure the amount of variance each component captures.
4. Dimensionality Reduction: By selecting the top  $k$  eigenvectors (those with the largest eigenvalues), the data can be projected into a lower-dimensional space that retains the  $k$  most significant components:

$$X_{\text{reduced}} = XW_k,$$

where  $W_k$  is the matrix containing the top  $k$  eigenvectors.

### 11.1.2 Properties of PCA

PCA has several important properties that make it valuable for unsupervised learning:

1. Variance Maximization: The first principal component is the direction that maximizes variance in the data. Each subsequent component maximizes variance under the constraint of being orthogonal to previous components.

### *11.1 Principle Component Analysis*

2. Orthogonality: Principal components are orthogonal to each other, ensuring that each captures unique information. This property transforms the data into an uncorrelated space, simplifying further analysis.
3. Dimensionality Reduction: By selecting only components with the largest eigenvalues, PCA enables dimensionality reduction while preserving most of the data's variability. This is especially useful for large datasets.
4. Reconstruction: If all components are retained, the original data can be perfectly reconstructed. When fewer components are used, the reconstruction is approximate but retains the essential structure of the data.
5. Sensitivity to Scaling: PCA is sensitive to the scale of input data, so standardization is often necessary to ensure that each feature contributes equally to the analysis.

#### **11.1.3 Interpreting PCA Results**

The output of PCA provides several insights into the data:

1. Principal Components: Each principal component represents a linear combination of the original features. The loadings (or weights) for each feature indicate the contribution of that feature to the component. Large weights (positive or negative) suggest that the corresponding feature strongly influences the principal component.
2. Explained Variance: Each principal component captures a specific amount of variance in the data. The proportion of variance explained by each component helps determine how many components are needed to retain the key information in the data. For example, if the first two components explain 90% of the variance, then these

## 11 Unsupervised Learning

two components are likely sufficient to represent the majority of the data's structure.

3. Selecting the Number of Components: The cumulative explained variance plot indicates the total variance captured as more components are included. A common approach is to choose the number of components such that the cumulative variance reaches an acceptable threshold (e.g., 95%). This helps in balancing dimensionality reduction with information retention.
4. Interpretation of Component Scores: The transformed data points, or “scores,” in the principal component space represent each original observation as a combination of the selected principal components. Observations close together in this space have similar values on the selected components and may indicate similar patterns.
5. Identifying Patterns and Clusters: By visualizing the data in the reduced space, patterns and clusters may become more apparent, especially in cases where there are inherent groupings in the data. These patterns can provide insights into underlying relationships between observations.

PCA thus offers a powerful tool for both reducing data complexity and enhancing interpretability by transforming data into a simplified structure, with minimal loss of information.

### 11.1.4 Example: PCA on 8x8 Digit Data

The 8x8 digit dataset contains grayscale images of handwritten digits (0 through 9), with each image represented by an 8x8 grid of pixel intensities. Each pixel intensity is a feature, so each image has 64 features in total.

### 11.1.5 Loading and Visualizing the Data

Let's start by loading the data and plotting some sample images to get a sense of the dataset.

```
# Import required libraries
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

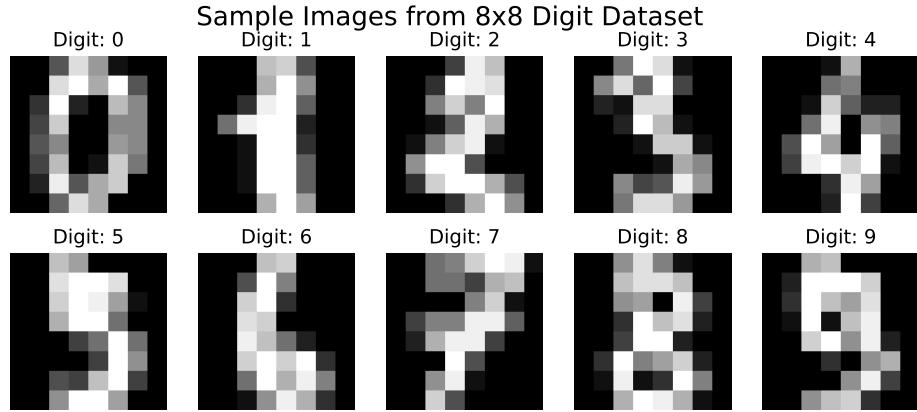
# Load the 8x8 digit dataset
digits = load_digits()
X = digits.data # feature matrix with 64 features (8x8 pixel intensities)
y = digits.target # target labels (0-9 digit classes)

# Display the shape of the data
print("Feature matrix shape:", X.shape)
print("Target vector shape:", y.shape)

# Plot some sample images from the dataset
fig, axes = plt.subplots(2, 5, figsize=(10, 4))
for i, ax in enumerate(axes.flat):
    ax.imshow(X[i].reshape(8, 8), cmap='gray')
    ax.set_title(f"Digit: {y[i]}")
    ax.axis('off')
plt.suptitle("Sample Images from 8x8 Digit Dataset", fontsize=16)
plt.show()
```

```
Feature matrix shape: (1797, 64)
Target vector shape: (1797,)
```

## 11 Unsupervised Learning



After loading and visualizing the data, we observe the following:

- Each digit is represented by a grid of 8x8 pixels, resulting in a 64-dimensional feature space.
- Since each pixel represents a separate feature, the dataset has high dimensionality relative to its visual simplicity.

Given the high dimensionality of the data, we may want to address the following research questions using PCA:

- Can we reduce the dimensionality of the dataset while preserving the essential structure of each digit? By reducing dimensions, we aim to simplify the data representation, which can aid in visualization and computational efficiency.
- How many principal components are necessary to capture most of the variance in the data? Identifying this will help us understand how many features are truly informative in distinguishing the digits.
- Are there distinct clusters in the reduced space? Visualizing the data in two or three dimensions could reveal any inherent groupings or patterns related to the different digit classes.

## 11.1 Principle Component Analysis

### 11.1.5.1 Performing PCA and Plotting Variance Contribution

Let's proceed by applying PCA to the digits data and plotting the explained variance to understand how much variance each principal component captures. This will help determine the optimal number of components to retain for a good balance between dimensionality reduction and information preservation.

The primary goal here is to identify the number of components that capture most of the variance. We'll use a cumulative explained variance plot to visualize how much total variance is captured as we include more principal components.

```
# Import the PCA module
from sklearn.decomposition import PCA
import numpy as np

# Initialize PCA without specifying the number of components
pca = PCA()
X_pca = pca.fit_transform(X)

# Calculate the explained variance ratio for each component
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)

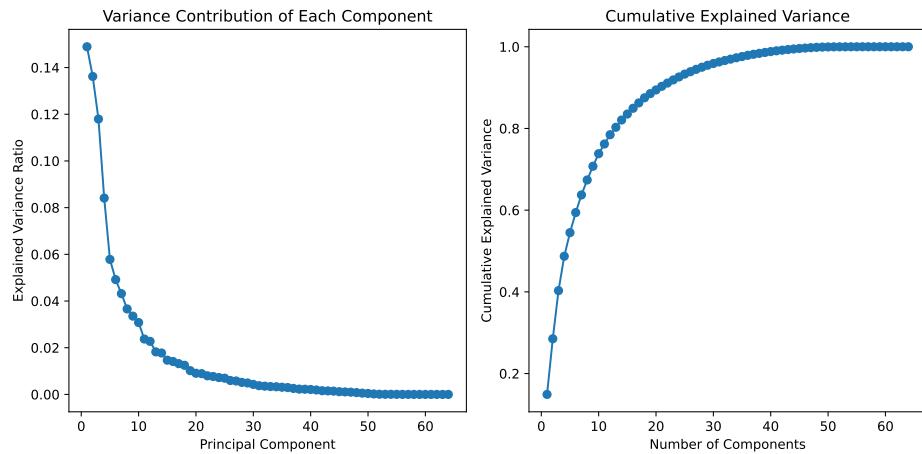
# Plot the explained variance and cumulative variance
plt.figure(figsize=(10, 5))

# Plot individual explained variance
plt.subplot(1, 2, 1)
plt.plot(np.arange(1, len(explained_variance) + 1), explained_variance, marker='o')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Variance Contribution of Each Component')
```

## 11 Unsupervised Learning

```
# Plot cumulative explained variance
plt.subplot(1, 2, 2)
plt.plot(np.arange(1, len(cumulative_variance) + 1), cumulative_variance, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance')

plt.tight_layout()
plt.show()
```



Variance Contribution of Each Component: The left plot shows the amount of variance explained by each individual component. This can help identify which components contribute significantly to capturing the variance in the data.

Cumulative Explained Variance: The right plot displays the cumulative explained variance as the number of components increases. This plot is useful for determining the number of components to retain. Generally, we look for a “knee” or “elbow” point where the cumulative variance starts to level off.

## 11.1 Principle Component Analysis

To select the number of components:

- Set a Variance Threshold: A typical approach is to select enough components to capture a certain percentage of the total variance (e.g., 90% or 95%).
- Elbow Method: Identify a point on the cumulative variance plot where additional components contribute minimally to the variance. This “elbow” point represents an efficient number of components.

In this example, we see that the first 10 components contain approximately 75% of the variance; around 50 components are needed to describe close to 100% of the variance.

### 11.1.5.2 PCA in Dimension Reduction

Let’s continue by projecting the digit data onto the first two and first three principal components, allowing us to visualize the data in a lower-dimensional space. This will help us see how well PCA captures the structure of the data and whether distinct clusters form in the reduced space.

```
# Apply PCA to reduce data to the first two and three components
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X)

pca_3d = PCA(n_components=3)
X_pca_3d = pca_3d.fit_transform(X)

# Plotting the 2D projection
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca_2d[:, 0], X_pca_2d[:, 1], c=y, cmap='tab10', s=15, alpha=0.7)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("2D PCA Projection of Digit Data")
```

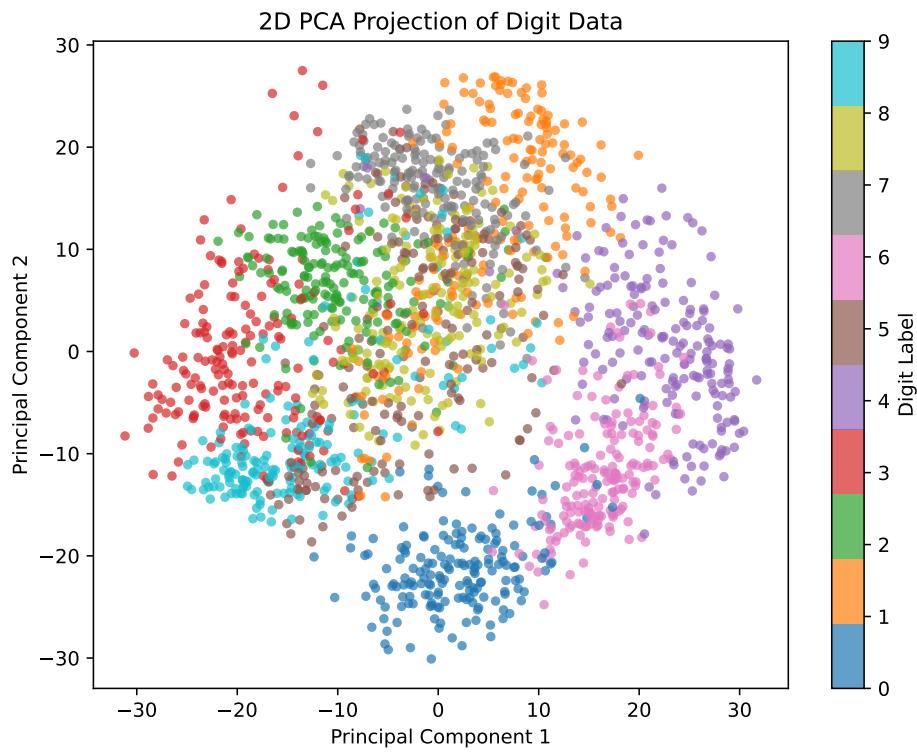
## 11 Unsupervised Learning

```
plt.colorbar(scatter, label='Digit Label')
plt.show()

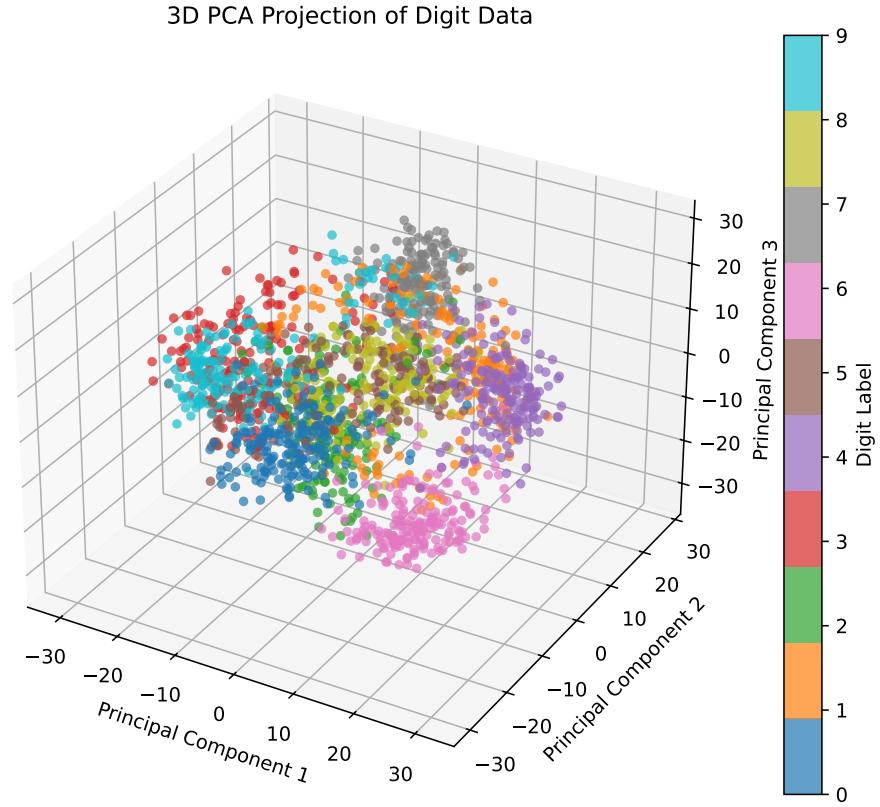
# Plotting the 3D projection
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca_3d[:, 0], X_pca_3d[:, 1], X_pca_3d[:, 2],
                     c=y, cmap='tab10', s=15, alpha=0.7)
ax.set_xlabel("Principal Component 1")
ax.set_ylabel("Principal Component 2")
ax.set_zlabel("Principal Component 3")
ax.set_title("3D PCA Projection of Digit Data")
fig.colorbar(scatter, ax=ax, label='Digit Label')
plt.show()
```

## 11.1 Principle Component Analysis



## 11 Unsupervised Learning



This 3D projection of the MNIST digit data shows each image's position in the space defined by the first three principal components. Here are some key observations:

1. **Cluster Formation:** Distinct clusters of points represent different digits. Digits with similar shapes, such as “1” and “7” (both often vertical), may appear closer to each other in this reduced space. This clustering suggests that PCA effectively captures structural features, even when reducing dimensions.
2. **Effectiveness of Dimensionality Reduction:** Despite reducing

## 11.1 Principle Component Analysis

from 64 dimensions to only three, PCA retains essential variance, allowing for distinction between different digits. This demonstrates PCA's utility in data compression, providing a simplified representation without losing significant information.

3. **Exploring Further Dimensions:** Additional components could capture more variance, if required. However, the first three components often capture most of the meaningful variance, balancing dimensionality reduction with information retention.

This PCA projection shows that the MNIST digit data has underlying patterns well-represented by the first few components. These findings highlight PCA's usefulness in compressing high-dimensional data while preserving its structure, making it a valuable tool for visualization, noise reduction, and as a pre-processing step in machine learning tasks.

### 11.1.5.3 PCA in Noise Filtering

To demonstrate PCA's use in noise filtering, we'll follow these steps:

1. Add Random Noise: Add random noise to the original digit images.
2. Fit PCA to Noisy Data: Apply PCA to the noisy data, selecting enough components to retain 50% of the variance.
3. Reconstruct the Digits: Use PCA's inverse transform to reconstruct the digits from the reduced components, effectively filtering out the noise.
4. Display the Results: Show a side-by-side comparison of the original, perturbed, and reconstructed images for visual assessment.

```
def plot_digits(datasets, titles):
    """
    Plots a 2x5 grid of images for each dataset in datasets,
    using a compact and uniform layout.
    """
```

## 11 Unsupervised Learning

```
Parameters:
- datasets: list of 2D numpy arrays, each array with shape (n_samples, 64)
  representing different versions of the digit data
  (e.g., original, noisy, reconstructed).
- titles: list of strings, titles for each dataset (e.g., ["Original", "Noisy",
  "Reconstructed"])
fig, axes = plt.subplots(len(datasets) * 2, 5, figsize=(5, 6),
                       subplot_kw={'xticks':[], 'yticks':[]},
                       gridspec_kw=dict(hspace=0.1, wspace=0.1))

for row, (data, title) in enumerate(zip(datasets, titles)):
    for i, ax in enumerate(axes[row * 2: row * 2 + 2].flat):
        ax.imshow(data[i].reshape(8, 8), cmap='binary', interpolation='nearest')
        axes[row * 2, 0].set_ylabel(title, rotation=0, labelpad=30, fontsize=10)

plt.suptitle("PCA Noise Filtering: Original, Noisy, and Reconstructed Digits")
# plt.tight_layout()
plt.show()

# Applying the function to the original, noisy, and reconstructed datasets
# Load the 8x8 digit dataset
digits = load_digits()
X = digits.data # Original digit data

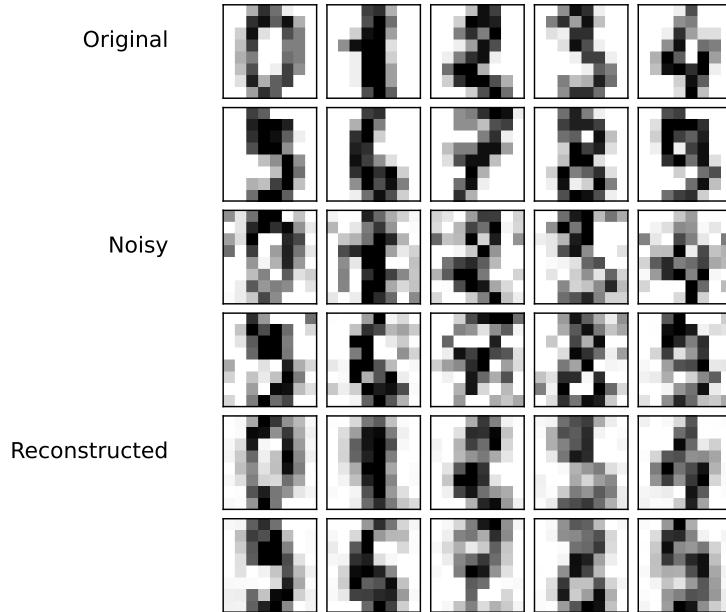
# Add random noise to the data
np.random.seed(0)
noise = np.random.normal(0, 4, X.shape)
X_noisy = X + noise

# Fit PCA to retain 50% of the variance
pca_50 = PCA(0.50)
X_pca_50 = pca_50.fit_transform(X_noisy)
X_reconstructed_50 = pca_50.inverse_transform(X_pca_50)
```

## 11.1 Principle Component Analysis

```
# Plot the original, noisy, and reconstructed digits using the function  
plot_digits([X, X_noisy, X_reconstructed_50], ["Original", "Noisy", "Reconstructed"]))
```

PCA Noise Filtering: Original, Noisy, and Reconstructed Digits



This visualization demonstrates the noise filtering effect of PCA:

- Original vs. Noisy Images: The second row shows the effect of added random noise, making the digits less recognizable.
- Reconstructed Images: In the third row, PCA has filtered out much of the random noise, reconstructing cleaner versions of the digits while preserving important structural features. This illustrates PCA's effectiveness in noise reduction by retaining only the principal components that capture meaningful variance.

## **11.2 K-Means Clustering**

This section is presented by .....

### **11.2.1 Subsection**

# **12 Advanced Topics**

## **12.1 Web Scraping**

This section was written by Melanie Desroches, a senior majoring in statistics and minoring in computer science. The goal of this section is to introduce web-scraping so that it can be utilized for data science. This will include what web-scraping is, how to web-scrape with Python using examples, and how to web-scrape ethically.

### **12.1.1 What is Web-Scraping**

As data scientists, we often want to collect data from a variety of sources. In the age of the internet, a lot of the data we may want to collect is available on a website. However, this data is often times not available in an easily downloadable format. This is where web-scraping becomes valuable. Web-scraping is an automated process used to gather data from websites. This allows us to access and collect large amounts of data directly from web pages if the information is not available for download.

Websites are primarily structured with HTML (Hypertext Markup Language), which organizes and displays content. Web scrapers parse through this HTML code to identify and extract relevant information. Therefore, it is important to have a basic understanding of HTML in order to identify what part of the website you are trying to scrape. The contents of a web page are broken up and identified by elements. Here are some examples of common elements that are important for web-scraping:

## *12 Advanced Topics*

- <body> : identifies the website body
- <table> : identifies a table
- <tbody> : identifies the body of the table
- <tr> : identifies the row of a table

### **12.1.2 How to Web-Scrape with Python**

There are many ways to web-scrape with Python. We will cover the two main packages, BeautifulSoup and Selenium.

#### **12.1.2.1 BeautifulSoup**

The BeautifulSoup Python Library simplifies the process of parsing and navigating HTML and XML documents, making it easier to extract data from websites. BeautifulSoup is ideal for scraping data from static websites. Static websites do not change based on user actions or require server-side interactions to update content dynamically. Basically, what you see is what you get. Static websites tend to be pretty simple so scraping from them is relatively easy.

Beautiful Soup can be installed by running

```
pip install beautifulsoup4
```

in your terminal.

#### **12.1.2.2 Selenium**

Selenium is used for web browser automation and dynamic websites. Dynamic sites often use backend programming to pull data from a database, customize it, and render it in real time based on user requests. This

## 12.1 Web Scraping

makes Selenium great at performing web-scraping tasks that involve multiple pages or performing actions within those pages. Because dynamic websites tend to be a bit more complex, you need to use a package like Selenium that is more equipped for the complex structure.

Selenium can be installed by running

```
pip install selenium
```

in your terminal.

### 12.1.2.3 Web Driver

To control a web browser, Selenium also requires a WebDriver. We recommend Chrome Driver because it is cross-platform; follow the instructions for developers to set up your Chrome Driver.

After setting up Chrome Driver, you can check its availability from a terminal:

```
chromedriver --version
```

A commonly seen error on Mac is

```
Error: “chromedriver” cannot be opened because the developer  
cannot be verified. Unable to launch the chrome browser
```

This can be fixed by running:

```
xattr -d com.apple.quarantine $(which chromedriver)
```

See explanation from StackOverflow.

#### **12.1.2.4 Beautiful Soup vs Selenium**

Both Beautiful Soup and Selenium are helpful tools in web-scraping. But they both have their strengths and weaknesses. Beautiful Soup is lightweight, easy to learn, and perfect for working with static HTML content. However, Beautiful Soup is more limited when it comes to dynamic websites, which are much more common nowadays. Selenium is better for interacting with dynamic web content that loads JavaScript or requires actions like clicking, scrolling, or filling forms. That said, Selenium can be slower and more resource-intensive since it opens a browser window to simulate real user actions.

#### **12.1.2.5 A Step-by Step Guide to Web-Scraping**

1. Find the website URL with the information you want to select
2. Send an HTTP request to the URL and confirm you have access to the page. Generally, 200-299 means the request has been granted and 400-499 means that your request is not allowed.
3. Use the “Inspect” tool in your browser to identify the tags, classes, or elements associated with the data you want to extract. This can be done by right-clicking on the web page and pressing select. If you hover your clicker over the different sections of HTML, the parts of the website that section is associated with will become highlighted. Use this to find the element that is associated with the data that you want to scrape.
4. Use a parsing library like Beautiful Soup or Selenium to process the HTML response. Beautiful Soup requires the use of the requests package in order to send a request. Selenium uses the webdriver to send the request.
5. Clean and store the relevant infomation.

## 12.1 Web Scraping

### 12.1.3 Examples using NYC Open Data

Since this class has used the NYC Open Data, let's build on this data set in order to get some additional information that is not already available.

#### 12.1.3.1 Beautiful Soup and NYPD Precincts

Say you want to get the addresses of all of the NYPD Precincts in New York City. This information is available in table format on the NYPD website. Since the NYPD Precincts aren't changed, the website is static, making Beautiful Soup the best package to use to scrape this website.

Start by making sure you have Beautiful Soup and Requests installed. The requests package can be installed using

```
pip install requests
```

Import the `requests` package, `BeautifulSoup` from `bs4`, and `pandas` (to create a new data frame). We have already identified the url that will be scraping data from. In the code below, there is a dictionary called `headers`. This is optional. Headers can help make your requests look more like a browser. If you choose to use a header, include it when you send your request to the url. Otherwise, the request can be sent to the url using `requests.get()`.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# URL of the NYPD precincts page
url = "https://www.nyc.gov/site/nypd/bureaus/patrol/precincts-landing.page"

# Send a GET request to the page
```

## 12 Advanced Topics

```
headers = {
    "User-Agent": "Mozilla/5.0 Chrome/87.0"
}
response = requests.get(url, headers=headers)

# Check if the request was successful
print(response.status_code)
if response.status_code != 200:
    print(f"Failed to retrieve page: Status code {response.status_code}")
```

200

Since the response to the request was 200, which means the request was successful, we are clear to move onto the next step which is parsing the table.

To start parsing, you have to call Beautiful Soup. When you pass `response.text` into Beautiful Soup, it takes the raw HTML of the webpage as a string. `html.parser` specifies the parsing engine used by Beautiful Soup to process the HTML. It is a built-in Python HTML parser that is fast and works well for most cases.

To identify which parts of the website you want to webscrape, you can right click on the website and click inspect. This will show you the HTML of the page. The table can be found under the `<table>` element with the class `rt`. Using this information, have Beautiful Soup find the table using `.find()`. Within the table, the rows are identified by `<tr>` within the HTML. In each row, the name of the precinct and address is found in the `<td>` element with the data labels `Precinct` and `Address` respectively. From this, use `.find_all('tr')` to find all the rows in the table and then within each row, extract the precinct and address.

316

## 12.1 Web Scraping

```
# Parse the HTML content
soup = BeautifulSoup(response.text, 'html.parser')

# Find the table with class "rt" which holds the precinct data
table = soup.find("table", {"class": "rt"})

# Lists to hold the extracted data
precinct_names = []
addresses = []

# Extract each row of the table (each row corresponds to one precinct)
for row in table.find_all("tr"):
    # Find the "Precinct" and "Address" columns by data-label attribute
    precinct_cell = row.find("td", {"data-label": "Precinct"})
    address_cell = row.find("td", {"data-label": "Address"})

    # If both cells are found, store their text content
    if precinct_cell and address_cell:
        precinct_names.append(precinct_cell.get_text(strip=True))
        addresses.append(address_cell.get_text(strip=True))
```

The extracted information can be stripped so that only the relevant text is included and then added to their relevant list. Now that the data has been collected and cleaned, a new dataframe can be created.

```
# Create a DataFrame with the extracted data
precincts_df = pd.DataFrame({
    "Precinct": precinct_names,
    "Address": addresses
})

# Display the DataFrame
print(precincts_df)
```

## 12 Advanced Topics

	Precinct	Address
0	1st Precinct	16 Ericsson Place
1	5th Precinct	19 Elizabeth Street
2	6th Precinct	233 West 10 Street
3	7th Precinct	19 1/2 Pitt Street
4	9th Precinct	321 East 5 Street
..	...	...
72	115th Precinct	92-15 Northern Boulevard
73	120th Precinct	78 Richmond Terrace
74	121st Precinct	970 Richmond Avenue
75	122nd Precinct	2320 Hylan Boulevard
76	123rd Precinct	116 Main Street

[77 rows x 2 columns]

### 12.1.3.2 Selenium and Weather Data

Say you want to see if the weather makes an impact of the number or severity of crashes in New York City. Weather data in New York City can be found on Wundergroud. Since information on weather is always being monitored and collected, the data that we want for a specific time period is being held in the websites database. Therefore, the website is dynamic and Selenium can be used for web scraping.

The first step is to set up Selenium and the WebDriver. In this example, I use Chrome Driver. Options can be initialized with `chrome_options = Options()` for the Chrome browser. The options I used were `--headless` (which allows the browser to run without a visible window) and `--disable-gpu` (which can improve performance in headless mode).

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
```

## 12.1 Web Scraping

```
from selenium.webdriver.chrome.options import Options

# Set up ChromeDriver
chrome_options = Options()
chrome_options.add_argument("--headless") # Run in headless mode (no browser UI)
chrome_options.add_argument("--disable-gpu") # Disable GPU acceleration
chrome_options.add_argument("--no-sandbox") # Required for some environments
```

Next we need find the path of the Chrome Driver. The following code is a cross-platform solution.

```
# Path to your ChromeDriver executable
# config_file_path = "config.txt"
# with open(config_file_path, 'r') as file:
#     chrome_driver_path = file.read().strip()

import os

def find_application_path(app_name):
    for path in os.environ["PATH"].split(os.pathsep):
        full_path = os.path.join(path, app_name)
        if os.path.isfile(full_path) and os.access(full_path, os.X_OK):
            return full_path
    return None

chrome_driver_path = find_application_path("chromedriver")
```

The driver then can be initialized with the path and driver options.

```
service = Service(chrome_driver_path)

# Initialize the ChromeDriver
driver = webdriver.Chrome(service=service, options=chrome_options)
```

## 12 Advanced Topics

Once Selenium and the webdriver is set up, go to the page and find the target data. Same as with the Beautiful Soup example, go to the url and identify the table that you want to webscrape. In this case, I want the table at the bottom of the page that lists the daily observations of the temperature, dew point, humidity, wind speed, pressure, and precipitation. The table is identified as `<table>` with the class `<days>`. In Selenium, `driver.get(url)` opens the webpage in the Edge browser. Once the table has loaded, `(By.CSS_SELECTOR, "table.days")` selects the main data table by its CSS selector “`table.days`”, ensuring we’re targeting the right element.

```
# Define the target URL
url = f"https://www.wunderground.com/history/weekly/us/ny/new-york-city/KLGA"

# Load the page
driver.get(url)

from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Wait for table to load
wait = WebDriverWait(driver, 15)
table = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "table.days")))
```

Within the table, the rows are indentified by `tr` in `tbody` and the columns are in `td`.

```
# Initialize lists for each data type
dates = []
max_temps = []
min_temps = []
humidity_values = []
```

## 12.1 Web Scraping

```
wind_speeds = []
pressure_values = []
precip_values = []

# Get all rows
rows = table.find_elements(By.CSS_SELECTOR, "tbody tr")
for row in rows:
    # Get all 'td' elements in the row
    columns = row.find_elements(By.TAG_NAME, "td")
    # Extract text from each column
    row_data = [col.text.strip() for col in columns]
    # Print the content of the row
    print("Row Data:", row_data) # This will print the content of each row
```

```
Row Data: ['Jun\n30\n1\n2\n3\n4\n5\n6', 'Jun', '30', '1', '2', '3', '4', '5', '6', 'Max Avg']
Row Data: ['Jun']
Row Data: ['30']
Row Data: ['1']
Row Data: ['2']
Row Data: ['3']
Row Data: ['4']
Row Data: ['5']
Row Data: ['6']
Row Data: ['Max', 'Avg', 'Min']
Row Data: ['101', '79.7', '73']
Row Data: ['79', '72.7', '65']
Row Data: ['83', '75.1', '67']
Row Data: ['83', '75.8', '68']
Row Data: ['85', '77.3', '72']
Row Data: ['90', '81.0', '74']
Row Data: ['92', '78.7', '72']
Row Data: ['Max', 'Avg', 'Min']
Row Data: ['74', '70.3', '59']
```

## 12 Advanced Topics

```
Row Data: ['56', '53.1', '46']
Row Data: ['56', '49.5', '45']
Row Data: ['58', '51.5', '47']
Row Data: ['69', '63.0', '52']
Row Data: ['73', '70.8', '69']
Row Data: ['73', '70.9', '69']
Row Data: ['Max', 'Avg', 'Min']
Row Data: ['91', '74.9', '36']
Row Data: ['65', '50.9', '38']
Row Data: ['55', '41.3', '29']
Row Data: ['70', '43.9', '28']
Row Data: ['85', '61.8', '48']
Row Data: ['87', '72.1', '55']
Row Data: ['91', '78.0', '54']
Row Data: ['Max', 'Avg', 'Min']
Row Data: ['20', '10.2', '5']
Row Data: ['23', '14.5', '8']
Row Data: ['12', '8.7', '6']
Row Data: ['18', '10.3', '0']
Row Data: ['15', '9.3', '3']
Row Data: ['14', '7.4', '0']
Row Data: ['17', '7.6', '0']
Row Data: ['Max', 'Avg', 'Min']
Row Data: ['30.0', '29.9', '29.8']
Row Data: ['30.1', '30.0', '29.9']
Row Data: ['30.2', '30.2', '30.1']
Row Data: ['30.2', '30.1', '30.0']
Row Data: ['30.0', '29.9', '29.8']
Row Data: ['29.8', '29.8', '29.7']
Row Data: ['29.9', '29.8', '29.8']
Row Data: ['Total']
Row Data: ['0.06']
Row Data: ['0.11']
Row Data: ['0.00']
```

## 12.1 Web Scraping

```
Row Data: ['0.00']
Row Data: ['0.00']
Row Data: ['0.03']
Row Data: ['0.11']
```

As you can see the output is pretty messy. From this step, we need to find the important parts and strip it of the text. This can be done by identifying the indicies of the rows that we want, using `find_elements` to find the corresponding tag, and then stripping the text to add it to the relevant list.

```
# Process the first row which contains all the dates
date_row = rows[0].text.split('\n')
dates = [date for date in date_row if date.isdigit()][:7] # Get first 7 dates

# Find temperature values (rows 10-16 contain the actual temperature data)
temp_rows = rows[10:17] # Get rows 10-16
for row in temp_rows:
    cells = row.find_elements(By.TAG_NAME, "td")
    if len(cells) >= 3:
        max_temps.append(cells[0].text.strip())
        min_temps.append(cells[2].text.strip())

# Find humidity values (rows 18-24)
humidity_rows = rows[18:25]
for row in humidity_rows:
    cells = row.find_elements(By.TAG_NAME, "td")
    if len(cells) >= 2:
        humidity_values.append(cells[1].text.strip())

# Find wind speed values (rows 26-32)
wind_rows = rows[26:33]
for row in wind_rows:
```

## 12 Advanced Topics

```
cells = row.find_elements(By.TAG_NAME, "td")
if len(cells) >= 1:
    wind_speeds.append(cells[0].text.strip())

# Find pressure values (rows 42-48)
pressure_rows = rows[42:49]
for row in pressure_rows:
    cells = row.find_elements(By.TAG_NAME, "td")
    if len(cells) >= 1:
        pressure_values.append(cells[0].text.strip())

# Find precipitation values (rows 50-56)
precip_rows = rows[50:57]
for row in precip_rows:
    cells = row.find_elements(By.TAG_NAME, "td")
    if len(cells) >= 1:
        precip_values.append(cells[0].text.strip())
```

Once all the relevant data has been collected and cleaned, it can be added to a new dataframe.

```
import pandas as pd

# Create DataFrame
weather_data = pd.DataFrame({
    'Date': dates,
    'Max Temperature (°F)': max_temps,
    'Min Temperature (°F)': min_temps,
    'Humidity (%)': humidity_values,
    'Wind Speed (mph)': wind_speeds,
    'Pressure (in)': pressure_values,
    'Precipitation (in)': precip_values
})
```

## 12.1 Web Scraping

```
print(weather_data)
driver.quit()
```

	Date	Max Temperature (°F)	Min Temperature (°F)	Humidity (%)	\
0	30	101	73	70.3	
1	1	79	65	53.1	
2	2	83	67	49.5	
3	3	83	68	51.5	
4	4	85	72	63.0	
5	5	90	74	70.8	
6	6	92	72	70.9	

	Wind Speed (mph)	Pressure (in)	Precipitation (in)
0	91	30.0	0.06
1	65	30.1	0.11
2	55	30.2	0.00
3	70	30.2	0.00
4	85	30.0	0.00
5	87	29.8	0.03
6	91	29.9	0.11

Lastly, `driver.quit()` closes the browser.

### 12.1.4 A Note on Data Ethics

While web scraping is not explicitly illegal, it can get you in hot water if you are not careful. Web scraping is a powerful tool and it should be treated as such. Just because you can web scrape doesn't always mean you should.

#### **12.1.4.1 Why Web-Scraping can be un-ethical**

There are several reasons that web scraping may be deemed unethical.

- The website you are trying to web scrape may not allow it.
- The information being scraped is considered private information or intellectual property.
- Sending too many requests at once can overwhelm the server and crash the website.

#### **12.1.4.2 Some Tips to Help You Scrape Ethically**

- You can check if a website allows web scraping in either the terms of use section of the website or by checking the websites `.robots.txt` to see who is allowed to use the website and what parts are available for scraping.
- Always be mindful of what kind of information you are trying to collect and if it is private information/intellectual property
- Never scrape from a website that requires login or payment
- Spread out the time of the requests in order to prevent the website from crashing. If using Selenium, use `WebDriverWait` from `selenium.webdriver.support.ui` to wait for the page to load. Otherwise, use the `time` package to space out the requests.

# 13 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.
  - a. Install Quarto onto your computer following the instructions of Get Started. Document the obstacles you encountered and how you overcame them.
  - b. 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.
  - c. 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:
  - a. Clone the class notes repo to an appropriate folder on your computer.
  - b. Add all the files to your designated homework repo from GitHub Classroom and work on that repo for the rest of the problem.
  - c. Add your name and wishes to the Wishlist; commit.
  - d. Remove the `Last, First` entry from the list; commit.

## 13 Exercises

- e. Create a new file called `add.qmd` containing a few lines of texts; commit.
  - f. Remove `add.qmd` (pretending that this is by accident); commit.
  - g. 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.
  - h. 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.
  - i. 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.
    - a. Create a fork of the notes repo into your own GitHub account.
    - b. Clone it to an appropriate folder on your computer.
    - c. Render the classnotes on your computer; document the obstacles and solutions.
    - d. Make a new branch (and name it appropriately) to experiment with your changes.
    - e. Checkout your branch and add your wishes to the wish list; commit with an informative message; and push the changes to your GitHub account.
    - f. Make a pull request to class notes repo from your fork at GitHub. Make sure you have clear messages to document the changes.
  4. **Monty Hall** Consider a generalized Monty Hall experiment. Suppose that the game start with  $n$  doors; after you pick one, the host opens  $m \leq n - 2$  doors, that show no award. Include sufficient text around the code chunks to explain them.
    - a. Write a function to simulate the experiment once. The function takes two arguments `ndoors` and `nempty`, which represent the

number of doors and the number of empty doors showed by the host, respectively. It returns the result of two strategies, switch and no-switch, from playing this game.

- b. Play this game with 3 doors and 1 empty a few times.
- c. Play this game with 10 doors and 8 empty a few times.
- d. Write a function to play this game `ntrial` times and return the proportion of wins for both strategies.
- e. Apply your function to play this game 1000 times, with 3 doors and 10 doors, and summarize your results.

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 trials 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

1. **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.
2. **Google Billboard Ad** Find the first 10-digit prime number occurring in consecutive digits of  $e$ . This was a Google recruiting ad.
3. **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  $\circ$  represent one of the four operators. For example,

$$(\square \circ \square) \circ (\square \circ \square)$$

## 13 Exercises

is one way to group the the operations.

- a. List all the possible ways to group the four numbers.
  - b. How many possible ways are there to check for a solution?
  - c. Write a function to solve the problem in a brutal force way. The inputs of the function are four numbers. The function returns a list of solutions. Some of the solutions will be equivalent, but let us not worry about that for now.
4. **NYC Crash Data Cleaning** The NYC motor vehicle collisions data with documentation is available from NYC Open Data. The raw data needs some cleaning.
- a. Use the filter from the website to download the crash data of the week of June 30, 2024 in CSV format; save it under a directory `data` with an informative name (e.g., `nyccrashes_2024w0630_by20240916.csv`); read the data into a Panda data frame with careful handling of the date time variables.
  - b. Clean up the variable names. Use lower cases and replace spaces with underscores.
  - c. Get the basic summaries of each variables: missing percentage; descriptive statistics for continuous variables; frequency tables for discrete variables.
  - d. Are their invalid `longitude` and `latitude` in the data? If so, replace them with `NA`.
  - e. Are there `zip_code` values that are not legit NYC zip codes? If so, replace them with `NA`.
  - f. Are there missing in `zip_code` and `borough`? Do they always co-occur?
  - g. 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.
  - h. Is it redundant to keep both `location` and the `longitude/latitude` at the NYC Open Data server?

- i. Check the frequency of `crash_time` by hour. Is there a matter of bad luck at exactly midnight? How would you interpret this?
  - j. 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?
  - k. Print the whole frequency table of `contributing_factor_vehicle_1`. Convert lower cases to uppercases and check the frequencies again.
  - l. Provided an opportunity to meet the data provider, what suggestions would you make based on your data exploration experience?
5. **NYC Crash Data Exploration** Except for the first question, use the cleaned crash data in feather format.
- a. Construct a contingency table for missing in geocode (latitude and longitude) by borough. Is the missing pattern the same across boroughs? Formulate a hypothesis and test it.
  - b. Construct a `hour` variable with integer values from 0 to 23. Plot the histogram of the number of crashes by `hour`. Plot it by borough.
  - c. Overlay the locations of the crashes on a map of NYC. The map could be a static map or Google map.
  - d. Create a new variable `severe` which is one if the number of persons injured or deaths is 1 or more; and zero otherwise. Construct a cross table for `severe` versus borough. Is the severity of the crashes the same across boroughs? Test the null hypothesis that the two variables are not associated with an appropriate test.
  - e. Merge the crash data with the zip code database.
  - f. Fit a logistic model with `severe` as the outcome variable and covariates that are available in the data or can be engineered from the data. For example, zip code level covariates can be

## 13 Exercises

obtained by merging with the zip code database; crash hour; number of vehicles involved.

6. **NYC Crash severity modeling** Using the cleaned NYC crash data, merged with zipcode level information, predict `severe` of a crash.
  - a. Set random seed to `1234`. Randomly select 20% of the crashes as testing data and leave the rest 80% as training data.
  - b. Fit a logistic model on the training data and validate the performance on the testing data. Explain the confusion matrix result from the testing data. Compute the F1 score.
  - c. Fit a logistic model on the training data with  $L_1$  regularization. Select the tuning parameter with 5-fold cross-validation in F1 score
  - d. Apply the regularized logistic regression to predict the severity of the crashes in the testing data. Compare the performance of the two logistic models in terms of accuracy, precision, recall, F1-score, and AUC.
7. **Midterm project: Noise complaints in NYC** The NYC Open Data of 311 Service Requests contains all requests from 2010 to present. We consider a subset of it with requests to NYPD on noise complaints that are created between 00:00:00 06/30/2024 and 24:00:00 07/06/2024. The subset is available in CSV format as `data/nypd311w063024noise_by100724.csv`. Read the data dictionary online to understand the meaning of the variables.
  - a. Data cleaning.
    - i. Import the data, rename the columns with our preferred styles.
    - ii. Summarize the missing information. Are there variables that are close to completely missing?

- iii. Are there redundant information in the data? Try storing the data using the Arrow format and comment on the efficiency gain.
  - iv. Are there invalid NYC zipcode or borough? Justify and clean them if yes.
  - v. Are there date errors? Examples are earlier `closed_date` than `created_date`; `closed_date` and `created_date` matching to the second; dates exactly at midnight or noon to the second; `action_update_date` after `closed_date`.
  - vi. Summarize your suggestions to the data curator in several bullet points.
- b. Data exploration.
- i. If we suspect that response time may depend on the time of day when a complaint is made, we can compare the response times for complaints submitted during nighttime and daytime. To do this, we can visualize the comparison by complaint type, borough, and weekday (vs weekend/holiday).
  - ii. Perform a formal hypothesis test to confirm the observations from your visualization. Formally state your hypotheses and summarize your conclusions in plain English.
  - iii. Create a binary variable `over2h` to indicate that a service request took two hours or longer to close.
  - iv. Does `over2h` depend on the complaint type, borough, or weekday (vs weekend/holiday)? State your hypotheses and summarize your conclusions in plain English.
- c. Data analysis.
- i. The addresses of NYC police precincts are stored in `data/nypd_precincts.csv`. Use geocoding tools to find their geocode (longitude and latitude) from the addresses.
  - ii. Create a variable `dist2pp` which represent the distance from each request incidence to the nearest police precinct.
  - iii. Create zip code level variables by merging with data from

## 13 Exercises

- ```
package uszipcode.
```
- iv. Randomly select 20% of the complaints as testing data with seeds 1234. Build a logistic model to predict `over2h` for the noise complaints with the training data, using all the variables you can engineer from the available data. If you have tuning parameters, justify how they were selected.
  - v. Assess the performance of your model in terms of commonly used metrics. Summarize your results to a New Yorker who is not data science savvy.

# References

- Breiman, L., Friedman, J. H., Olshen, R., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth.
- Collaboration, E. H. T. (2019). First M87 event horizon telescope results. I. The shadow of the supermassive black hole. *The Astrophysical Journal Letters*, 875(1), L1. <https://doi.org/10.3847/2041-8213/ab0c57>
- 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*.
- GeeksforGeeks. (2023). *Complement naive bayes (CNB) algorithm*. <https://www.geeksforgeeks.org/complement-naive-bayes-cnb-algorithm/>
- 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.
- Hunter, J. D., & Matplotlib Development Team, the. (2023a). *Matplotlib: Visualization with python*. <https://matplotlib.org/stable/index.html>
- Hunter, J. D., & Matplotlib Development Team, the. (2023b). *Matplotlib.animation.FuncAnimation*. [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.animation.FuncAnimation.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.animation.FuncAnimation.html)
- IBM. (2024). *What are naïve bayes classifiers?* <https://www.ibm.com/topics/naive-bayes>
- Kharlamov, M. (2023). *Contextily: Adding basemaps to geospatial data*. [https://contextily.readthedocs.io/en/latest/intro\\_guide.html](https://contextily.readthedocs.io/en/latest/intro_guide.html)
- Ng, A., & Jordan, M. (2001). On discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. *Proceedings of*

## References

- 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.
- 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.
- Team, G. D. (2024). *GeoPandas documentation*. <https://geopandas.org/en/stable/index.html>
- 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*.