Introduction to Data Science

STAT 3255/5255 @ UConn, Fall 2024

Jun Yan and Students in Fall 2024

2024-09-04

Table of contents

Preliminaries

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

Sources at GitHub

These lecture notes for STAT 3255/5255 in Fall 2024 represent a collaborative effort between Professor Jun Yan and the students enrolled in the course. This cooperative approach to education was facilitated through the use of GitHub, a platform that encourages collaborative coding and content development. To view these contributions and the lecture notes in their entirety, please visit our Spring 2024 repository at https://github.com/statds/ids-f24.

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

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

Midterm Project

TBA

Final Project

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

- ASA Data Challenge Expo
- Kaggle
- DrivenData
- Top 10 Data Science Competitions in 2024

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

Adapting to Rapid Skill Acquisition

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

> When a technologically-minded person is asked to help a friend, family member, or colleague with a computer problem, most of the time it's less a matter of knowing the answer as much as knowing how to quickly find an unknown answer. In data science it's the same: searchable web resources such as online documentation, mailing-list threads, and StackOverflow answers contain a wealth of information, even (especially?) if

> it is a topic you've found yourself searching before. Being an effective practitioner of data science is less about memorizing the tool or command you should use for every possible situation, and more about learning to effectively find the information you don't know, whether through a web search engine or another means.

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

Wishlist

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

- Yan, Jun
  - Make data science more 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
- Astle, Jaden
- Babiec, Owen
- Baptista, Stef
- Bienvenue, Jack
- Blanchard, Zachary
- Borowski, Emily

- Clokey, Sara
- Desroches, Melanie
- Fang, Zetong
- Febles, Xavier
- Freed, Brent
- Jha, Aansh
- Johnson, Dorothea
- Kashalapov, Olivia
- Klinowski, Amalia
- Lee, Seunghyeon
- Manna, Rahul
- Mazzola, Julia
- Paricharak, Aditya
- Parvez, Mohammad Shahriyar
- Tan, Qianruo
- Xu, Deyu

Presentation Orders

The topic presentation order is set up in class.

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

import random
## seed jointly set by the class
random.seed(4737 + 8852 + 3196 + 2344 + 47)
random.sample(presenters, len(presenters))
## random.shuffle(presenters) # This would shuffle the list in place
```

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

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

Course Logistics

Presentation Task Board

Here are some example tasks:

- Making presentations with Quarto
- Data science ethics
- Data science communication skills
- Import/Export data
- Arrow as a cross-platform data format
- Database operation with Structured query language (SQL)
- Grammer of graphics
- Handling spatial data
- Visualize spatial data in a Google map
- Animation
- Classification and regression trees
- Support vector machine
- Random forest
- Naive Bayes
- Bagging vs boosting
- Neural networks
- Deep learning
- TensorFlow

- Autoencoders
- Reinforcement learning
- Calling C/C++ from Python
- Calling R from Python and vice versa
- Developing a Python package

Please use this Google Sheet to sign up.

Date	Presenter	Topic

Final Project Presentation Schedule

We use the same order as the topic presentation for undergraduate final presentation.

Date	Presenter

Contributing to the Class Notes

Contribution to the class notes is through a 'pull request'.

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

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

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

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

Homework Requirements

- Use the repo from Git Classroom to submit your work. See Section Chapter 2.
  - Keep the repo clean (no tracking generated files).
    - Never "Upload" your files; use the git command lines.
    - Make commit message informative (think about the readers).
- Use `quarto` source only. See Chapter 3.
- For the convenience of grading, add your html output to a release in your repo.
- For standalone pdf output, you will need to have LaTeX installed.

Practical Tips

Data analysis

- Use an IDE so you can play with the data interactively

Preliminaries

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

Presentation

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

My Presentation Topic (Template)

Introduction

Put an overview here. Use Markdown syntax.

Sub Topic 1

Put materials on topic 1 here

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

```
import pandas as pd

# do something
```

Sub Topic 2

Put materials on topic 2 here.

Sub Topic 3

Put matreials on topic 3 here.

Conclusion

Put sumaries here.

1 Introduction

1.1 What Is Data Science?

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

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

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

where

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

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

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

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

1.2 Expectations from This Course

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

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

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

1.3 Computing Environment

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

1.3.1 Command Line Interface

On Linux or MacOS, simply open a terminal.

On Windows, several options can be considered.

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

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

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

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 IPython and Jupyter Notebook in class.

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

Online books on Python for data science:

- "Python Data Science Handbook: Essential Tools for Working with Data," First Edition, by Jake VanderPlas, O'Reilly Media, 2016.

2. "Python for Data Analysis: Data Wrangling with Pan- das, NumPy, and IPython." Third Edition, by Wes McK- inney, O'Reilly Media, 2022.

2 Project Management

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

There are tools that make learning Git easy.

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

2.1 Set Up Git/GitHub

Download Git if you don't have it already.

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

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

See how to get started with GitHub account.

2.2 Most Frequently Used Git Commands

- `git clone`
- `git pull`
- `git status`
- `git add`
- `git remove`
- `git commit`
- `git push`

2.3 Tips on using Git:

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

2.4 Pull Request

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

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

3 Reproducibile Data Science

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

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

3.1 Introduction to Quarto

To get started with Quarto, see documentation at Quarto.

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

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

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

- Markdown basics
- Using Python
- Presentations

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

3.2 Compiling the Classnotes

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

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

3.2.1 Set up your Python Virtual Environment

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

To create a Python virtual environment for our classnotes:

```
python3 -m venv .ids-f24-venv
```

Here `.ids-f24-venv` is the name of the virtual environment to be created. Choose an informative name. This only needs to be set up once.

To activate this virtual environment:

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

After activating the virtual environment, you will see `(.ids-f24-venv)` at the beginning of your shell prompt. Then, the Python interpreter and packages needed will be the local versions in this virtual environment without interfering your system-wide installation or other virtual environments.

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

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

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

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

3.2.2 Clone the Repository

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

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

3.2.3 Render the Classnotes

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

```
cd ids-f24
quarto render
```

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

4 Python Refreshment

4.1 Know Your Computer

4.1.1 Operating System

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

- **Linux:** Linux distributions have a variety of package managers depending on the distribution. For instance, Ubuntu uses APT (Advanced Package Tool), Fedora uses DNF (Dandified Yum), and Arch Linux uses Pacman. These package managers are integral to the Linux experience, allowing users to install, update, and manage software packages easily from repositories.
- **macOS:** macOS uses Homebrew as its primary package manager. Homebrew simplifies the installation of software and tools that aren't included in the standard macOS installation, using simple commands in the terminal.
- **Windows:** Windows users often rely on the Microsoft Store for apps and software. For more developer-focused package management, tools like Chocolatey and Windows Package Manager (Winget) are used. Additionally, recent versions of Windows have introduced the Windows Subsystem for Linux (WSL). WSL allows Windows users to run a Linux environment directly on Windows, unifying Windows

and Linux applications and tools. This is particularly useful for developers and data scientists who need to run Linux-specific software or scripts. It saves a lot of trouble Windows users used to have before its time.

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

4.1.2 File System

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

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

along with others like `pwd` (print working directory), which displays your current directory, equips you with the basics of moving around the file system in the command line, an indispensable skill for a wide range of computing tasks in Unix-like systems.

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

4.2 The Python World

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

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

4.3 Standard Library

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

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

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

The constant is only represented by a given double precision.

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

```
2.71828182845904509080
2.71828182845904509079559829842764884233474731445312500000000000000000000000
```

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

```
import decimal  # for what?

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

```
'718281828459045235360287471352662497757247093699959574966967627724076630353
```

4.4 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([ 6.63812599, -0.47118277, -1.9186411 ,  0.82291147,  3.02948168,
        4.49290595,  3.10187312,  3.61424788,  2.42517462,  0.15059873])
```

The pdf and cdf can be evaluated:

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

```
array([0.05092051, 0.08240834, 0.06172292, 0.09550937, 0.09648645,
       0.0821311 , 0.09602235, 0.09193587, 0.09917374, 0.08962537])
```

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)
```

```
9.81  s ± 588 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

The second uses dynamic programming memoization.

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

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

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

```
1.99 s ± 30.2 ns per loop (mean ± std. dev. of 7 runs, 100,000 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)
```

```
75.3 s ± 1.02 s per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

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

4.5.1 Monte Hall

Here is a function that performs the Monte Hall experiments.

```
import numpy as np

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

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

Test it out:

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

```
{'noswitch': np.int64(245), 'switch': np.int64(357)}
```

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

4.6 Variables versus Objects

In Python, variables and the objects they point to actually live in two different places in the computer memory. Think of variables as pointers to the objects they're associated with, rather than being those objects. This matters when multiple variables point to the same object.

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

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

Now check their addresses

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

```
4603785024
4603785024
```

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
```

```
4481708296
4547258832
```

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

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

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

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

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

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

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

append_42(x)
```

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

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

4.7 Number Representation

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

4.7.1 Integers

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

```
import numpy as np

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

```
array(9223372036854775807)
```

What if we increment it by 1?

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

```
array(-9223372036854775808)
```

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

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

```
9223372036854775808000
```

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

4.7.2 Floating Number

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

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

```
False
```

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

```
False
```

What is really going on?

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

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

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

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

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

```
True
```

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

```
True
```

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

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

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

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

4.8 Virtual Environment

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

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

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

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

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

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

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

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

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

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

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

5 Exercises

1. **Quarto and Git setup** Quarto and Git are two important tools for data science. Get familiar with them through the following tasks. Please use the `templates/hw.qmd` template.

    1. Install Quarto onto your computer following the instructions of Get Started. Document the obstacles you encountered and how you overcame them.
    2. Pick a tool of your choice (e.g., VS Code, Jupyter Notebook, Emacs, etc.), follow the instructions to reproduce the example of line plot on polar axis.
    3. Render the homework into a pdf file and put the file into a release in your GitHub repo. Document any obstacles you have and how you overcome them.

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

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

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

3. **Contributing to the Class Notes** To contribute to the classnotes, you need to have a working copy of the sources on your computer. Document the following steps in a `qmd` file as if you are explaining them to someone who want to contribute too.

    1. Create a fork of the notes repo into your own GitHub account.
    2. Clone it to an appropriate folder on your computer.
    3. Render the classnotes on your computer; document the obstacles and solutions.
    4. Make a new branch (and name it appropriately) to experiment with your changes.
    5. Checkout your branch and add your wishes to the wish list; commit with an informative message; and push the changes to your GitHub account.
    6. Make a pull request to class notes repo from your fork at GitHub. Make sure you have clear messages to document the changes.

4. **Monty Hall** Write a function to demonstrate the Monty Hall problem through simulation. The function takes two arguments `ndoors` and `ntrials`, representing the number of doors in the experiment and the number of trails in a simulation, respectively. The function should return the proportion of wins for both the switch and

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

5. **Approximating** π Write a function to do a Monte Carlo approximation of π. The function takes a Monte Carlo sample size `n` as input, and returns a point estimate of π and a 95% confidence interval. Apply your function with sample size 1000, 2000, 4000, and 8000. Repeat the experiment 1000 times for each sample size and check the empirical probability that the confidence intervals cover the true value of π. Comment on the results.

6. **Google Billboard Ad** Find the first 10-digit prime number occurring in consecutive digits of e. This was a Google recruiting ad.

7. **Game 24** The math game 24 is one of the addictive games among number lovers. With four randomly selected cards form a deck of poker cards, use all four values and elementary arithmetic operations $(+ - \times /)$ to come up with 24. Let $\square$ be one of the four numbers. Let $\bigcirc$ represent one of the four operators. For example,

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

is one way to group the the operations.

   1. List all the possible ways to group the four numbers.
   2. How many possibly ways are there to check for a solution?
   3. Write a function to solve the problem in a brutal force way. The inputs of the function are four numbers. The function returns a list of solutions. Some of the solutions will be equivalent, but let us not worry about that for now.

8. The NYC motor vehicle collisions data with documentation is available from NYC Open Data. The raw data needs some cleaning. (JY: Add variable name cleaning next year.)

1. Use the filter from the website to download the crash data of January 2023; save it under a directory `data` with an informative name (e.g., `nyc_crashes_202301.csv`).
2. Get basic summaries of each variable: missing percentage; descriptive statistics for continuous variables; frequency tables for discrete variables.
3. Are the `LATITUDE` and `LONGITIDE` values all look legitimate? If not (e.g., zeroes), code them as missing values.
4. If `OFF STREET NAME` is not missing, are there any missing `LATITUDE` and `LONGITUDE`? If so, geocode the addresses.
5. (Optional) Are the missing patterns of `ON STREET NAME` and `LATITUDE` the same? Summarize the missing patterns by a cross table. If `ON STREET NAME` and `CROSS STREET NAME` are available, use geocoding by intersection to fill the `LATITUDE` and `LONGITUDE`.
6. Are `ZIP CODE` and `BOROUGH` always missing together? If `LATITUDE` and `LONGITUDE` are available, use reverse geocoding to fill the `ZIP CODE` and `BOROUGH`.
7. Print the whole frequency table of `CONTRIBUTING FACTOR VEHICLE 1`. Convert lower cases to uppercases and check the frequencies again.
8. Provided an opportunity to meet the data provider, what suggestions do you have to make the data better based on your data exploration experience?

9. Except the first problem, use the cleaned data set with missing geocode imputed (`data/nyc_crashes_202301_cleaned.csv`).

    1. Construct a contigency table for missing in geocode (latitude and longitude) by borough. Is the missing pattern the same across borough? Formulate a hypothesis and test it.
    2. Construct a `hour` variable with integer values from 0 to 23. Plot the histogram of the number of crashes by `hour`. Plot it by borough.

3. Overlay the locations of the crashes on a map of NYC. The map could be a static map or Google map.
4. Create a new variable `injury` which is one if the number of persons injured is 1 or more; and zero otherwise. Construct a cross table for `injury` versus borough. Test the null hypothesis that the two variables are not associated.
5. Merge the crash data with the zip code database.
6. Fit a logistic model with `injury` as the outcome variable and covariates that are available in the data or can be engineered from the data. For example, zip code level covariates can be obtained by merging with the zip code database.

10. Using the cleaned NYC crash data, perform classification of `injury` with support vector machine and compare the results with the benchmark from regularized logistic regression. Use the last week's data as testing data.

    1. Explain the parameters you used in your fitting for each method.
    2. Explain the confusion matrix result from each fit.
    3. Compare the performance of the two approaches in terms of accuracy, precision, recall, F1-score, and AUC.

11. The NYC Open Data of 311 Service Requests contains all requests from 2010 to present. We consider a subset of it with request time between 00:00:00 01/15/2023 and 24:00:00 01/21/2023. The subset is available in CSV format as `data/nyc311_011523-012123_by022023.csv`. Read the data dictionary to understand the meaning of the variables,

    1. Clean the data: fill missing fields as much as possible; check for obvious data entry errors (e.g., can `Closed Date` be earlier than `Created Date`?); summarize your suggestions to the data curator in several bullet points.
    2. Remove requests that are not made to NYPD and create a new variable `duration`, which represents the time period from

the `Created Date` to `Closed Date`. Note that `duration` may be censored for some requests. Visualize the distribution of uncensored `duration` by weekdays/weekend and by borough, and test whether the distributions are the same across weekdays/weekends of their creation and across boroughs.

3. Define a binary variable `over3h` which is 1 if `duration` is greater than 3 hours. Note that it can be obtained even for censored `duration`. Build a model to predict `over3h`. If your model has tuning parameters, justify their choices. Apply this model to the 311 requests of NYPD in the week of 01/22/2023. Assess the performance of your model.
4. Now you know the data quite well. Come up with a research question of interest that can be answered by the data, which could be analytics or visualizations. Perform the needed analyses and answer your question.

12. **NYC Rodents** Rats in NYC are widespread, as they are in many densely populated areas (https://en.wikipedia.org/wiki/Rats_in_New_York_City). As of October 2023, NYC dropped from the 2nd to the 3rd places in the annual "rattiest city" list released by a pest control company. In the 311 Service Request Data, there is one complain type `Rodent`. Extract all the requests with complain type `Rodent`, created between January 1, 2022 and December 31, 2023. Save them into a csv file named `rodent_2022-2023.csv`.

    1. Are there any complains that are not closed yet?
    2. Are there any complains with a closed data before the created date?
    3. How many agencies were this complain type reported to?
    4. Summarize the missingess for each variable.
    5. Summarize a frequency table for the `descriptor` variable, and summarize a cross table by year.
    6. Which types of 'DESCRIPTOR' do you think should be included if our interest is rodent sighting?

7. Take a subset of the data with the descriptors you chose and summarize the response time by borough.

13. **NYC rodent sightings data cleaning** The data appears to need some cleaning before any further analysis. Some missing values could be filled based on other columns.

    1. Checking all 47 column names suggests that some columns might be redundant. Identify them and demonstrate the redundancy.
    2. Are zip code and borough always missing together? If geocodes are available, use reverse geocoding to fill the zip code.
    3. Export the cleaned data in both csv and feather format. Comment on the file sizes.

14. **SQL Practice on NYC rodent sightings** The NYC rodent sightings data that we prepared could be stored more efficiently using a database. Let us start from the csv file you exported from the last problem.

    1. Create a table called `rodent` from the csv file.
    2. The `agency` and `agency_name` columns are redundant in the table. Create a table called `agency`, which contains only these two columns, one agency a row.
    3. Drop the `agency_name` name from the `rodent` table. Justify why we do not need it here.
    4. Comment on the sizes of the table (or exported csv file) of `rodent` before and after dropping the `agency_name` column.
    5. Come up with a scheme for the two tables that allows even more efficient storage of the `agency` column in the `rodent` table. _Hint: use an integer to code the agencies.

15. **Logistic Modeling** The response time to 311 service requests is a measure of civic service quality. Let us model the response time to 311 requests with complain type `Rodent`.

1. Compute the response time in hours. Note that some response will be missing because of unavailable closed date.
2. Compute a binary variable `over3d`, which is one if the response time is greater than 3 days, and zero otherwise. Note that this variable should have no missing values.
3. Use the package `uszipcode` to obtain the zip code level covariates such as median house income and median home value. Merge these variables to the rodent data.
4. Split the data at random into training (80%) and testing (20%). Build a logistic model to predict `over3d` on the training data, and validate the performance on the testing data.
5. Build a lasso logistic model to predict `over3d`, and justify your choice of the tuning parameter. Validate on the testing data.

16. **Midterm Project: Rodents in NYC** Rodents in NYC are widespread, as they are in many densely populated areas. As of October 2023, NYC dropped from the 2nd to the 3rd places in the annual "rattiest city" list released by a pest control company. Rat sightings in NYC was analyzed by Dr. Michael Walsh in a 2014 PeerJ article. We investigate this problem from a different angle with the NYC Rodent Inspection data, provided by the Department of Health and Mental Hygiene (DOHMH). Download the 2022-2023 data by filtering the `INSPECTION_DATE` to between 11:59:59 pm of 12/31/2021 and 12:00:00 am of 01/01/2024 and `INSPECTION_TYPE` is either `Initial` or `Compliance` (which should be about 108 MB). Read the meta data information to understand the data.

    1. Data cleaning.
        - There are two zipcode columns: `ZIP_CODE` and `Zipcodes`. Which one represent the zipcode of the inspection site? Comment on the data dictionary.
        - Summarize the missing information. Are their missing values that can be filled using other columns? Fill them if yes.

- Are their redundant information in the data? Try storing the data using `arrow` and comment on the efficiency gain.
- Are there invalid zipcode or borough? Justify and clean them up if yes.

2. Data exploration.
    - Create binary variable `passing` indicating passing or not for the inspection result. Does `passing` depend on whether the inspection is initial or compliance? State your hypothesis and summarize your test result.
    - Are the passing pattern different across different boroughs for initial inspections? How about compliance inspections? State your hypothesis and summarize your test results.
    - If we suspect that the passing rate may depends on the time of a day of the inspection, we may compare the passting rates for inspections done in the mornings and inspections one in the afternoons. Visualize the comparison by borough and inspection type.
    - Perform a formal hypothesis test to confirm the observations from your visualization.
3. Data analytics.
    - Aggregate the inspections by zip code to create a dataset with five columns. The first three columns are `zipcode`; `n_initial`, the count of the initial inspections in that zipcode; and `n_initpass`, the number of initial inspections with a passing result in that zipcode. The other two variables are `n_compliance` and `n_comppass`, the counterpart for compliance inspections.
    - Add a variable to your dataset, `n_sighting`, which represent the number of rodent sightings from the 311 service request data in the same 2022-2023 period.
    - Merge your dataset with the simple zipcode table in package `uszipcode` by zipcode to obtain demographic and socioeconomic variables at the zipcode level.

- Build a binomial regression for the passing rate of initial inspections at the zipcode level. Assess the goodness-of-fit of your model. Summarize your results to a New Yorker who is not data science savvy.

4. Now you know the data quite well. Come up with a research question of interest that can be answered by the data, which could be analytics or visualizations. Perform the needed analyses and answer your question.

References

VanderPlas, Jake. 2016. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, Inc.