Project Id: 571261

WEBSITE SCRAPING

A PROJECT REPORT

Submitted By

Gadhia Aditi Darshan

221313132006

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

Information & Communication Technology

Adani Institute of Infrastructure Engineering

Gujarat Technological University, Ahmedabad

September, 2024

Project Id: 571261

adani
Institute of Infrastructure

ADANI INSTITUTE OF INFRASTRUCTURE ENGINEERING

Shantigram Township, Nr.Vaishnodevi Circle,

Sarkhej - Gandhinagar Hwy, Gujarat 382421

CERTIFICATE

This is to certify that the project report submitted along with the project entitled *WEBSITE SCRAPING* has been carried out by *GADHIA ADITI DARSHAN* under my guidance in partial fulfilment for the degree of Bachelor of Engineering in *INFORMATION & COMMUNICATION TECHNOLOGY*, 7th Semester of Gujarat Technological University, Ahmedabad during the academic year 2024-2025.

5/9/24

DR. ASHISH GOSWAMI
Internal guide

DR. AJAY KUMAR VYAS
Head Of Department

Project Id: 571261

VIRTUAL REALITY SYSTEMS

Date: 06-07-2024

CERTIFICATE

This is to certify that **Ms. Aditi Gadhia** has successfully completed project on **Web Scrapping** in our organization from 25 June 2024 to 05 July 2024 in our organization.

We found her honest, dedicated, hardworking and well behaved during her Internship period.

For Virtual Reality Systems,

Vimal Rughani

A-406 The Landmark, Opp. Spicy Street, Kudasan Gandhinagar
Email: info@virtualrealitysystems.net Phone No: +91 97260-67737

Project Id: 571261

ADANI INSTITUTE OF INFRASTRUCTURE ENGINEERING

Shantigram Township, Nr.Vaishnodevi Circle,

Sarkhej - Gandhinagar Hwy, Gujarat 382421

DECLARATION

We hereby declare that the Internship / Project report submitted along with the *WEBSITE SCRAPING* entitled submitted in partial fulfilment for the degree of Bachelor of Engineering in INFORMATION & COMMUNICATION TECHNOLOGY to Gujarat Technological University, Ahmedabad, is a bonafede record of original project work carried out by me / us at *VIRTUAL REALITY SYSTEMS* under the supervision of *VIMAL RUGHANI* and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

	Name of the Student	Sign of Student
1	_____	_____

TABLE OF CONTENTS

CHAPTER 1: OVERVIEW OF THE COMPANY

1.1 History

Virtual Reality Systems (VRS) in Gandhinagar was established by Vimal Rughani with a vision to advance the field of virtual reality (VR) and its applications in various industries. Founded in the early 2010s, VRS started as a small startup focused on leveraging emerging VR technologies to create immersive experiences for education, entertainment, and industrial applications.

Initially, Vimal Rughani and his team focused on developing VR content and applications tailored to local needs and markets. Over the years, the company has expanded its scope to include cutting-edge VR hardware and software solutions, becoming a key player in the VR industry in Gujarat and beyond. The company's growth reflects the increasing demand for VR solutions in various sectors, including education, healthcare, and entertainment.

1.2 Different Products / Scope of Work

1. VR Software Solutions:

- **Education and Training:** Interactive VR simulations for educational institutions and corporate training programs, offering immersive learning experiences in subjects such as science, engineering, and medical fields.

- **Entertainment:** Development of VR games and entertainment applications designed to provide engaging and immersive experiences for users.

- **Real Estate:** Virtual tours and property visualization tools that allow potential buyers to explore properties in a virtual environment.

2. VR Hardware Solutions:

- **VR Headsets:** Design and manufacture of high-quality VR headsets that offer superior visual and sensory experiences.

- **Motion Tracking Systems:** Advanced tracking systems to capture and interpret user movements within the virtual environment.

3. Customized VR Solutions:

- **Industrial Applications:** VR solutions tailored to specific industries, including manufacturing and healthcare, for tasks such as equipment training, process simulations, and patient treatment planning.

- **Consulting and Integration:** Providing consulting services to businesses and institutions to integrate VR technologies into their operations and enhance their capabilities.

4. Research and Development:

- **Innovation:** Continuous R&D to stay at the forefront of VR technology, developing new features and applications based on emerging trends and user feedback.

- **Collaboration:** Partnering with academic institutions and research organizations to advance VR technology and explore new use cases.

1.3 Organization Chart

The organization chart of Virtual Reality Systems (VRS) led by Vimal Rughani typically includes the following key positions:

- Founder & CEO (Vimal Rughani): Oversees the overall strategic direction and operations of the company, driving innovation and growth.

- Chief Technology Officer (CTO): Manages the development and implementation of VR technology, including hardware and software solutions.

- Chief Operating Officer (COO): Responsible for day-to-day operations, project management, and ensuring efficient business processes.

- Head of R&D: Leads research and development efforts, focusing on innovation and the creation of new VR technologies.

- Product Development Team: Engineers and designers working on the creation and refinement of VR products and applications.

- Sales and Marketing Team: Handles the promotion, sales, and customer support for VR products and solutions.

- Client Services and Support: Provides assistance and support to clients, including training, troubleshooting, and integration services.

1.4 Capacity of Plant

1. Facility Size:

- The VR Systems facility in Gandhinagar is equipped with state-of-the-art technology and infrastructure to support the development and manufacturing of VR products. The plant includes areas for hardware assembly, software development, and testing.

2. Production Capacity:

- **Hardware Production:** The plant has the capacity to produce a significant number of VR headsets and related hardware components annually, meeting the demands of both domestic and international markets.

- **Software Development:** The facility supports the development of multiple VR applications simultaneously, with dedicated teams for different types of software projects.

3. Research and Development:

- The R&D section is equipped with advanced VR equipment and tools for prototyping and testing new technologies. It supports ongoing innovation and ensures that the company remains competitive in the rapidly evolving VR industry.

4. Expansion Plans:

- The company has plans for future expansion, including increasing production capacity, enhancing R&D capabilities, and exploring new markets and applications for VR technology.

CHAPTER 2: INTRODUCTION TO PROJECT

2.1 Project / Internship summary – key to a good summary is the first sentence, which must contain the most essential information that you wish to convey.

The project involves developing a Python-based web scraping tool designed to automate the extraction of data from various websites, aiming to streamline data collection for research and analysis. By leveraging Python libraries such as Requests and BeautifulSoup, the tool efficiently retrieves and processes web content, handling diverse website structures and data formats. The project includes designing the scraper, testing its performance on multiple sites, and addressing ethical considerations to ensure compliance with web scraping best practices. This tool exemplifies the practical application of Python in data science and automation, providing a valuable resource for extracting and analyzing web data.

2.2 Purpose

The purpose of this project is to develop a web scraping tool using Python to automate the extraction of data from websites, making it easier to gather and analyze large volumes of information. By creating an efficient and reliable scraper, the project aims to streamline data collection processes for various applications, such as research, market analysis, and academic studies. This tool will enable users to quickly retrieve and process web data, overcoming the limitations of manual data gathering and providing a practical solution for accessing valuable information from the internet.

2.3 Objectives

1. Developing a Scraper: Build a Python-based tool using libraries like Requests and BeautifulSoup to fetch and parse web content efficiently.

2. Handling Diverse Content: Ensure the scraper can manage various website structures, including static and dynamically loaded data.

3. Data Storage and Processing: Implement methods to store and process the extracted data in a structured format suitable for analysis.

4. Performance Testing: Test the scraper on different websites to evaluate its accuracy and reliability.

5. Ethical Compliance: Adhere to best practices and legal guidelines for web scraping, including respecting website terms of service and protecting user data.

2.4 Scope (what it can do and can't do)

What It Can Do:

1. Automate Data Extraction: Efficiently retrieve and extract data from various websites using Python, including text, images, and links.

2. Handle Different Website Structures: Manage both static and dynamic content, accommodating sites with varying HTML layouts and JavaScript-rendered data.

3. Parse and Process Data: Use libraries like BeautifulSoup to parse HTML and extract relevant information, then store it in a structured format such as CSV or JSON for further analysis.

4. Support Multiple Sites: Work with a range of websites, adapting to different URL structures and content types.

5. Error Handling: Include basic error handling to manage common issues like missing elements or connection problems.

What It Can't Do:

1. Bypass Complex Anti-Scraping Measures: May not effectively circumvent advanced anti-scraping technologies like CAPTCHA, sophisticated bot detection, or IP blocking.

2. Handle Highly Dynamic Content Seamlessly: Limited ability to interact with highly dynamic or interactive content that requires real-time user interactions or complex JavaScript execution.

3. Provide Real-Time Updates: Does not support real-time data extraction or continuous monitoring of websites for live updates.

4. Manage User Authentication: Does not handle websites requiring complex user login or authentication processes beyond basic form submissions.

2.5 Technology and Literature Review

2.5.1. Technology Review

1. Web Scraping Technologies:

- Python Libraries:

- Requests: This library simplifies HTTP requests, enabling users to fetch web pages and handle responses. It's essential for retrieving raw HTML content.

- BeautifulSoup: A library for parsing HTML and XML documents, BeautifulSoup provides methods for navigating and searching the parse tree, making it easier to extract specific data from web pages.

2. Data Storage and Processing:

- CSV (Comma-Separated Values): A simple and widely used format for storing tabular data, CSV files are easy to create and import into spreadsheet applications and data analysis tools.

- JSON (JavaScript Object Notation): A lightweight data interchange format that is easy for both humans and machines to read and write. JSON is commonly used for structured data and API responses.

3. Anti-Scraping Technologies:

- CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart): A security measure that requires users to complete tasks that are easy for humans but difficult for bots, such as identifying distorted text or images.

- IP Blocking: Websites may monitor and block IP addresses that exhibit suspicious behavior or high scraping activity, requiring techniques like IP rotation or proxies to bypass.

2.5.2. Literature Review

1. Books and Articles:

- "Web Scraping with Python" by Ryan Mitchell: This book provides a comprehensive guide to web scraping techniques using Python, covering practical examples and advanced topics such as handling JavaScript content and managing large-scale scraping projects.

- "Python Web Scraping" by Katharine Jarmul and Richard Lawson: This book offers insights into web scraping using Python, including best practices, handling complex web pages, and working with different data formats.

2. Online Resources:

- **Official Documentation:** Python libraries such as Requests, BeautifulSoup, and Scrapy provide official documentation and tutorials that are valuable for understanding their capabilities and usage.

- **Online Tutorials and Blogs:** Numerous online resources, including blogs and video tutorials, offer practical guidance on implementing web scraping projects and troubleshooting common issues.

CHAPTER 3: SYSTEM ANALYSIS

3.1 Study of Current System

1. Overview of Existing Systems

Current web scraping systems range from simple scripts to complex frameworks, each designed to automate the extraction of data from websites. The choice of system depends on factors like the complexity of the target websites, the volume of data to be scraped, and the required data processing capabilities. Here's an overview of the common types of web scraping systems:

2. Manual Scraping:

- **Description:** Manual scraping involves copying and pasting data from web pages into spreadsheets or databases. It is often used for small-scale or one-time data collection tasks.

- **Limitations:** Time-consuming, error-prone, and impractical for large volumes of data or dynamic content.

3. Basic Web Scraping Scripts:

- **Description:** Simple scripts using languages like Python, often with libraries such as Requests and BeautifulSoup, to automate the extraction of data from static web pages.

- **Strengths:** Easy to implement and suitable for straightforward scraping tasks.

- **Limitations:** Struggles with dynamically loaded content, CAPTCHA, and websites with complex structures.

4. Advanced Web Scraping Frameworks:

- **Description:** More sophisticated frameworks like Scrapy provide a comprehensive set of tools for building complex scrapers and crawlers. They offer features for handling multiple pages, managing requests, and processing data.

- **Strengths:** Highly customizable, scalable, and capable of handling large-scale scraping tasks.

- **Limitations:** Requires more development effort and expertise to set up and maintain.

3.2 Problem and Weaknesses of Current System

1. Problems with Manual Scraping

- **Time-Consuming:** Manual data extraction is labor-intensive and impractical for large-scale or frequent data collection tasks.

- **Error-Prone:** Human errors can occur during data entry, leading to inaccuracies and inconsistencies in the collected data.

- **Not Scalable:** As the volume of data or number of websites increases, manual scraping becomes increasingly difficult and inefficient.

2. Limitations of Basic Web Scraping Scripts

- **Static Content Only:** Basic scripts using libraries like Requests and BeautifulSoup are limited to extracting static content. They struggle with pages that rely on JavaScript for content loading.

- **Limited Error Handling:** These scripts often lack sophisticated error handling and retry mechanisms, making them less reliable in the face of network issues or unexpected website changes.

- **Performance Issues:** For large-scale scraping tasks, basic scripts can be slow and may not handle large volumes of data efficiently.

3. Weaknesses of advanced web scraping frameworks

- **Complexity:** Advanced frameworks like Scrapy offer powerful features but require significant setup and configuration. They may have a steep learning curve for beginners.

- **Resource Intensive:** They can consume considerable system resources, particularly when handling large-scale data extraction and processing tasks.

- **Overhead:** The comprehensive features of these frameworks may introduce unnecessary complexity for simpler scraping tasks.

4. Challenges with Browser Automation Tools

- **Performance Overhead:** Browser automation tools like Selenium and Puppeteer are resource-intensive because they simulate user interactions with a web browser, making them slower compared to direct scraping methods.

- **Complex Configuration**: Setting up and managing browser automation for scraping can be complex and require more maintenance.

- **Limited Scalability:** Due to the overhead of running a full browser instance, scaling up scraping tasks to handle large volumes of data can be challenging and costly.

5. Common Issues Across All Systems

- **Anti-Scraping Measures:** Many websites employ anti-scraping technologies such as CAPTCHAs, IP blocking, and sophisticated bot detection, which can hinder the effectiveness of scraping tools.

- **Dynamic Content Handling:** Websites that use JavaScript to load content dynamically pose a challenge for traditional scraping methods, requiring additional tools or approaches to handle.

- **Legal and Ethical Concerns**: Scraping can raise legal and ethical issues, including compliance with website terms of service and data privacy regulations. Missteps in these areas can lead to legal consequences and damage to the scraper's reputation.

- **Data Integrity:** Ensuring the accuracy and consistency of scraped data can be difficult, especially when dealing with inconsistent website structures or frequent changes in web page layouts**.**

3.3 Requirements of New System

1. Functional Requirements

- **Data Extraction:** The system should be able to extract specific data from target websites. Define what data you need, such as text, images, links, etc.
- **Target Websites:** List the websites or types of websites you plan to scrape.
- **Scraping Frequency:** Determine how often the data needs to be scraped (e.g., one-time, daily, weekly).
- **Data Storage:** Decide how and where the scraped data will be stored (e.g., CSV, JSON, database).

2. Technical Requirements

- **Programming Language:** Python is your choice, but ensure you are comfortable with Python libraries relevant to web scraping.
- **Libraries and Tools:** Common libraries include:
    - **Requests:** For making HTTP requests.
    - **BeautifulSoup or lxml:** For parsing HTML and XML.
- **Error Handling:** Implement error handling for network issues, changes in website structure, or other unexpected problems.
- **Data Parsing:** Use appropriate parsers to handle different types of data and structures.

3. Performance Requirements

- **Efficiency:** Optimize your code to handle large volumes of data and reduce the load time.
- **Concurrency:** If necessary, implement asynchronous requests or multi-threading to speed up the scraping process.

4. Legal and Ethical Requirements

- **Terms of Service:** Ensure you're not violating any terms of service of the target websites.
- **Data Privacy:** Be mindful of any personal data you might be scraping and ensure compliance with data protection laws.

5. User Interface

- **Input Interface**: Create an interface for users to input URLs, select data fields, or set scraping options.
- **Output Interface:** Provide a way to view or download the scraped data in a user-friendly format.

6. Documentation and Reporting

- **Documentation:** Include clear documentation on how to use the system, including setup instructions and usage examples.
- **Reporting:** Generate reports or logs detailing the scraping process, including any issues encountered.

7. Security Considerations

- **Rate Limiting**: Implement rate limiting to avoid overwhelming target websites and getting blocked.
- **IP Rotation:** Use proxy servers or IP rotation if necessary to avoid detection and bans.

8. Testing and Maintenance

- **Testing:** Test your scraper thoroughly to ensure it works across different websites and handles various edge cases.
- **Maintenance:** Plan for maintenance in case the structure of the target websites changes or new features need to be added.

9. Scalability

- **Expandability:** Design your system to be easily expandable if you need to scrape additional websites or handle more data in the future.

10. Integration

- **APIs:** If applicable, integrate with APIs for more efficient data retrieval.
- **Other Systems:** Ensure compatibility with other systems or tools you might be using in your project.

CHAPTER 4: IMPLEMENTATION

4.1 Implementation Platform / Environment

1. Development Environment:

- **IDE/Text Editor:** Use an integrated development environment (IDE) such as PyCharm, VSCode, or Jupyter Notebook.
- **Python Version:** Ensure compatibility with Python 3.x (preferably the latest stable version).

2. Libraries and Tools:

- **Web Scraping Libraries:**
  - requests for making HTTP requests.
  - BeautifulSoup or lxml for parsing HTML.
  - Scrapy for more complex or large-scale scraping.
  - Selenium if dealing with websites requiring JavaScript rendering.
- **Data Storage:**
  - Local Files: CSV (pandas), JSON, or XML.
  - Databases: SQLite or PostgreSQL if a more robust solution is required.
- **Version Control:** Git for version control and GitHub for repository hosting.

3. Deployment Environment:

- **Operating System:** Development can be done on Windows, macOS, or Linux. Ensure cross-platform compatibility if necessary.

4.2 Process / Program / Technology / Modules Specification(s)

1. Process:

- **Requirement Analysis:** Identify and document the specific data you need and the target websites.
- **Design:** Plan the structure of your scraper, including how it will interact with web pages, handle data extraction, and store results.

- **Development:**
  - **Setup:** Install necessary libraries and tools.
  - **Implementation:**
    - Create HTTP requests to fetch web pages.
    - Parse HTML content to extract required data.
    - Handle pagination, form submissions, or dynamic content if needed.
  - **Error Handling:** Implement mechanisms to handle exceptions and retries.
- **Testing:** Test with various websites and edge cases to ensure robustness and reliability.
- **Deployment:** Set up any necessary deployment scripts or services if the scraper needs to be run on a schedule.

2. Program/Technology Modules:

- **Data Retrieval Module:**
  - Uses requests or Scrapy to fetch web pages.
- **Parsing Module:**
  - Uses BeautifulSoup or lxml to parse and extract data from HTML.
- **Data Storage Module:**
  - Handles saving data to files (CSV, JSON) or databases (SQLite, PostgreSQL).
- **Error Handling Module:**
  - Manages network errors, parsing errors, and retries.
- **Logging Module:**
  - Keeps track of operations, errors, and system performance.

3. Technology Stack:

- **Programming Language:** Python
- **Libraries:** requests, BeautifulSoup, lxml, Scrapy, Selenium, pandas (for data manipulation)
- **Data Storage:** SQLite, CSV, JSON
- **Version Control:** Git, GitHub

4.3 Finding / Results / Outcomes

1. Findings:

- **Accuracy:** Determine how accurately the scraper extracts data from different types of websites.
- **Performance:** Evaluate the performance, including the speed and efficiency of the scraper.
- **Error Handling:** Assess the effectiveness of error handling and recovery strategies.

2. Results:

- **Data Quality:** Review the quality and completeness of the data collected.
- **Scalability:** Determine how well the system scales with increasing data or target websites.
- **Compliance:** Verify that the scraper adheres to legal and ethical guidelines.

3. Outcomes:

- **Final Product:** A functional web scraping tool that meets the project requirements and objectives.
- **Documentation:** Comprehensive documentation detailing the setup, usage, and maintenance of the scraper.
- **Project Report:** A final report or presentation summarizing the process, results, and any challenges encountered.

4. Improvements and Future Work:

- **Enhancements:** Suggestions for improving the scraper, such as adding more features, handling additional websites, or optimizing performance.
- **Scalability:** Plans for scaling the project, such as deploying it in a cloud environment or integrating with other systems.

CHAPTER 5: CONCLUSION AND DISCUSSION

5.1 Overall Analysis of Internship / Project Viabilities

1. Objectives and Goals

- **Clear Objectives:** Determine if the project has clearly defined objectives. For instance, the goal might be to develop a web scraper that collects specific data from various websites for analysis.
- **Alignment with Goals**: Ensure that the project aligns with your academic or professional goals. For a college project, it should demonstrate your ability to solve real-world problems using technical skills.

2. Feasibility

- **Technical Feasibility:**
    - **Skills and Knowledge:** Assess whether you have or can acquire the necessary skills and knowledge to complete the project. For web scraping, this includes proficiency in Python, understanding of web technologies, and familiarity with relevant libraries and tools.
    - **Technology Stack:** Ensure the technology stack (libraries, tools, and frameworks) is appropriate for the task and you can effectively utilize it.
    - **Complexity:** Evaluate the complexity of the websites to be scraped and whether the project scope is manageable within the given timeframe.
- **Resource Feasibility:**
    - **Time:** Consider whether the project can be completed within the given timeframe, including all phases such as research, development, testing, and documentation.
    - **Tools and Infrastructure:** Ensure you have access to the necessary tools and infrastructure, such as a development environment, testing tools, and data storage solutions.
- **Legal and Ethical Feasibility:**
    - **Compliance:** Verify that the project complies with legal and ethical standards, including respecting website terms of service, privacy policies, and data protection regulations.

3. Impact

- **Academic Impact:**
    - **Learning Outcomes:** Evaluate how the project will contribute to your learning and understanding of web scraping, data extraction, and related concepts.
    - **Project Quality:** Assess the quality and depth of the project work, including the documentation, functionality, and presentation.
- **Practical Impact:**
    - **Usefulness:** Consider the practical usefulness of the data you'll collect and its potential applications. For instance, data from the project could be used in further **Usefulness:** research, analysis, or as part of a larger application.
    - **Scalability:** Evaluate whether the project can be scaled or adapted for different use cases or larger datasets in the future.

4. Risks and Challenges

- **Technical Risks:**
    - **Website Changes:** Websites frequently update their structure, which could impact the functionality of your scraper.
    - **Data Accuracy:** Challenges in ensuring the accuracy and consistency of the scraped data.
- **Operational Risks:**
    - **Resource Limitations:** Limited access to resources, such as development tools or cloud infrastructure, that may impact project execution.

5. Benefits

- **Skill Development:** The project will help you develop valuable technical skills, including programming, data extraction, and problem-solving.
- **Portfolio Enhancement:** Successfully completing the project will enhance your portfolio and demonstrate your capability to handle real-world challenges.
- **Networking Opportunities:** The project might open up opportunities for networking with professionals or academics in the field of data science and web scraping.

5.2 Problem Encountered and Possible Solutions

1. Technical Challenges

- **Website Structure Changes:**
    - **Problem:** Websites often update their HTML structure, which can break your scraper.
    - **Possible Solutions:** Implement robust parsing strategies using flexible selectors (like XPath or CSS selectors). Regularly update your scraper and consider using dynamic scraping tools like Selenium for sites with frequent changes.
- **Rate Limiting and IP Blocking:**
    - **Problem:** Websites may block your IP or throttle requests if they detect excessive scraping.
    - **Possible Solutions:** Implement rate limiting and respect the website's robots.txt. Use proxy servers or IP rotation services to distribute requests.
- **Data Parsing Errors:**
    - **Problem:** Inconsistent or malformed HTML can lead to parsing errors.
    - **Possible Solutions:** Use more robust parsing libraries and error handling. Consider implementing retry mechanisms and validation checks to handle unexpected data formats.

2. Legal and Ethical Issues

- **Terms of Service Violations:**
    - **Problem:** Scraping may violate the terms of service of some websites.
    - **Possible Solutions:** Always review and adhere to the terms of service of the target websites. Seek permission if necessary, and ensure compliance with data protection laws.
- **Data Privacy Concerns:**
    - **Problem:** Collecting personal data without consent can lead to legal issues.
    - **Possible Solutions:** Avoid scraping personal or sensitive data. Anonymize data where possible and ensure compliance with data protection regulations like GDPR.

3. Performance Issues

- **Scalability:**
    - **Problem:** As the volume of data grows, the scraper might become slower or less efficient.
    - **Possible Solutions:** Optimize code for efficiency, use asynchronous requests or multi-threading, and consider using distributed scraping frameworks like Scrapy for larger projects.
- **Data Storage and Management:**
    - **Problem:** Handling and storing large volumes of data can be challenging.
    - **Possible Solutions:** Use databases optimized for large datasets (e.g., SQLite, PostgreSQL) and implement efficient data processing and retrieval methods.

5.3 Summary of Internship / Project Work

1. Project Overview:

- **Objective:** The primary goal of the project was to develop a web scraping tool to extract specific data from target websites and store it for further analysis.
- **Scope:** The project involved designing and implementing a web scraper using Python, handling dynamic content, managing data storage, and ensuring compliance with legal and ethical standards.

2. Key Achievements:

- **Development:** Successfully built and tested a web scraper using libraries such as requests, BeautifulSoup, and Selenium.
- **Data Collection:** Extracted and stored data in a structured format, demonstrating the tool's functionality.
- **Documentation:** Created comprehensive documentation outlining the project setup, usage, and key findings.

3. Challenges Faced:

- **Technical:** Encountered issues with dynamic content and website structure changes.
- **Legal:** Navigated legal constraints and ensured compliance with website terms and data protection laws.

4. Solutions Implemented:

- **Technical Solutions:** Adapted the scraper to handle dynamic content and used flexible parsing techniques.
- **Legal Solutions:** Reviewed and adhered to terms of service and data protection guidelines.

5. Outcome:

- The project demonstrated the ability to build a functional web scraper, effectively manage and store data, and address common challenges in web scraping.

5.4 Limitation and Future Enhancement

1. Limitations:

- **Limited Scope:** The scraper was designed for specific websites and may not be adaptable to all types of sites or dynamic content.
- **Data Accuracy:** Variability in website structures and data formats can affect the accuracy of the extracted data.
- **Performance Constraints:** The scraper may face performance issues with very large datasets or highly dynamic sites.

2. Future Enhancements:

- **Generalization**: Enhance the scraper to handle a broader range of websites and dynamic content more effectively.
- **Advanced Features:** Implement advanced features such as machine learning algorithms **Advanced Features:** for data classification or sentiment analysis.
- **Scalability:** Optimize the scraper for better performance with larger datasets and consider implementing distributed scraping techniques.
- **User Interface:** Develop a user-friendly interface for easier configuration and management of the scraping process.
- **Monitoring and Alerts**: Add functionality for monitoring the scraper's performance and sending alerts in case of errors or issues.

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

`>>> x+2` → `7`	Sum of two variables
`>>> x-2` → `3`	Subtraction of two variables
`>>> x*2` → `10`	Multiplication of two variables
`>>> x**2` → `25`	Exponentiation of a variable
`>>> x%2` → `1`	Remainder of a variable
`>>> x/float(2)` → `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis

learn — Machine learning

NumPy — Scientific computing

matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset	
`>>> my_array[1]` → `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` → `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` → `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

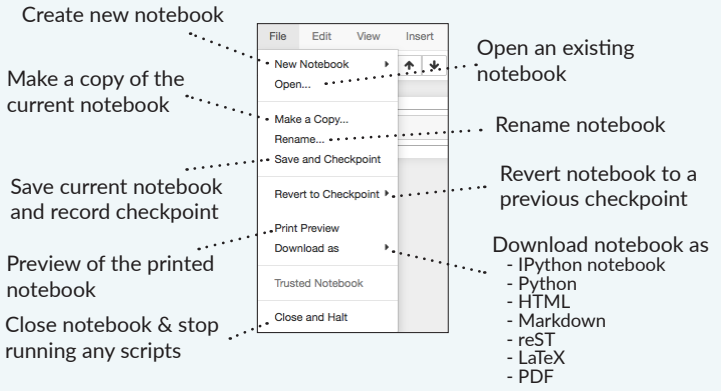
`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Python For Data Science *Cheat Sheet*

Jupyter Notebook

Learn More Python for Data Science Interactively at www.DataCamp.com

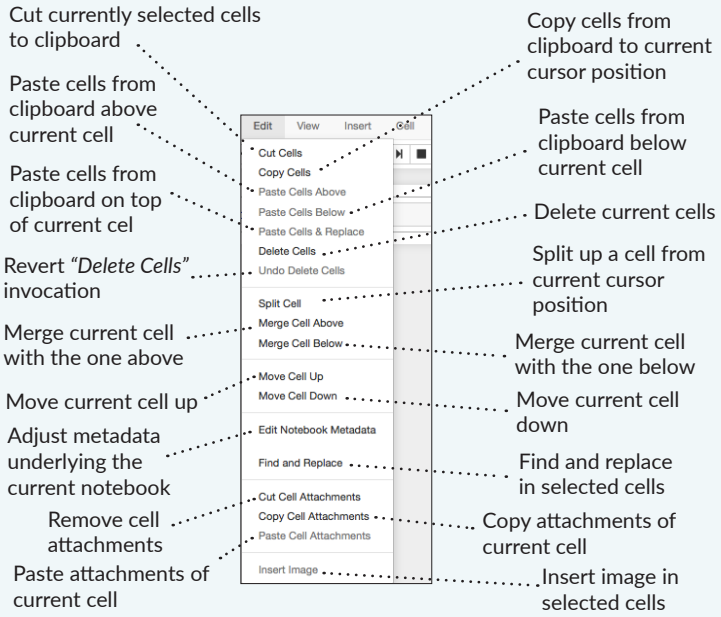
Saving/Loading Notebooks



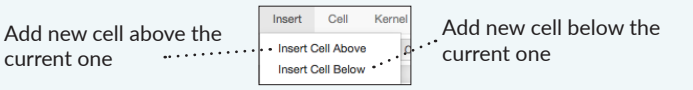
Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Edit Cells



Insert Cells

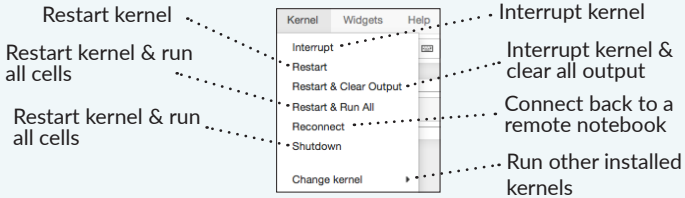


Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPython IRkernel IJulia

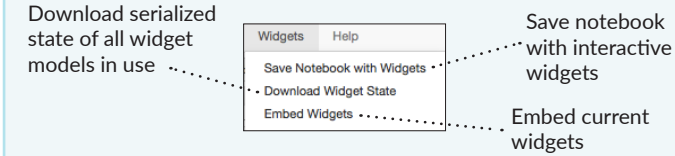
Installing Jupyter Notebook will automatically install the IPython kernel.



Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

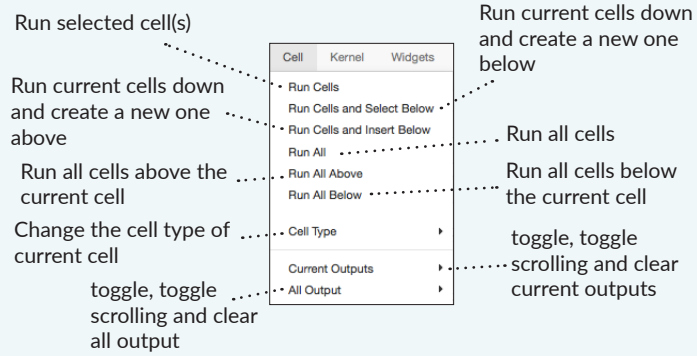


Command Mode:

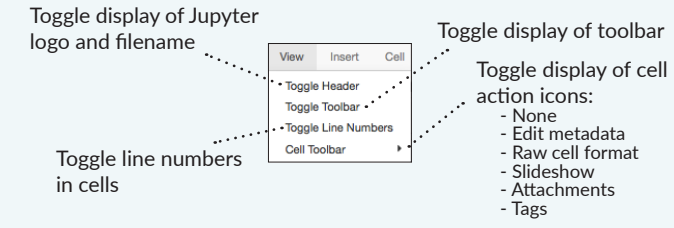
Edit Mode:

1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

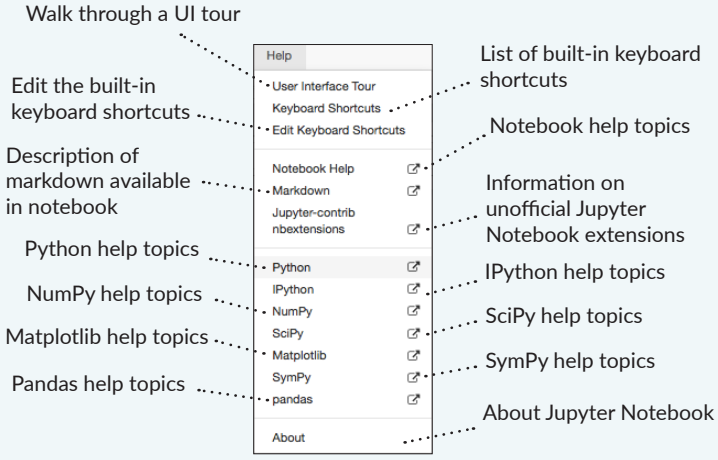
Executing Cells



View Cells



Asking For Help



Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

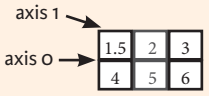
Use the following import convention:

```
>>> import numpy as np
```

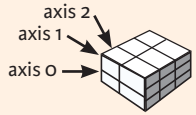
NumPy Arrays

1D array

2D array



3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                    Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)    Create an array of ones
>>> d = np.arange(10,25,5)             Create an array of evenly
                                       spaced values (step value)
>>> np.linspace(0,2,9)                 Create an array of evenly
                                       spaced values (number of samples)
>>> e = np.full((2,2),7)               Create a constant array
>>> f = np.eye(2)                      Create a 2X2 identity matrix
>>> np.random.random((2,2))            Create an array with random values
>>> np.empty((3,2))                    Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape             Array dimensions
>>> len(a)              Length of array
>>> b.ndim              Number of array dimensions
>>> e.size              Number of array elements
>>> b.dtype             Data type of array elements
>>> b.dtype.name        Name of data type
>>> b.astype(int)       Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                          Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                   Subtraction
>>> b + a                              Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                        Addition
>>> a / b                              Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                     Division
>>> a * b                              Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                   Multiplication
>>> np.exp(b)                          Exponentiation
>>> np.sqrt(b)                         Square root
>>> np.sin(a)                          Print sines of an array
>>> np.cos(b)                          Element-wise cosine
>>> np.log(a)                          Element-wise natural logarithm
>>> e.dot(f)                           Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                             Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                              Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)               Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                            Array-wise sum
>>> a.min()                            Array-wise minimum value
>>> b.max(axis=0)                      Maximum value of an array row
>>> b.cumsum(axis=1)                   Cumulative sum of the elements
>>> a.mean()                           Mean
>>> b.median()                         Median
>>> a.corrcoef()                       Correlation coefficient
>>> np.std(b)                          Standard deviation
```

Copying Arrays

```
>>> h = a.view()          Create a view of the array with the same data
>>> np.copy(a)            Create a copy of the array
>>> h = a.copy()          Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()              Sort an array
>>> c.sort(axis=0)        Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting


```
>>> a[2]
  3
```

Select the element at the 2nd index


```
>>> b[1,2]
  6.0
```

Select the element at row 0 column 2 (equivalent to `b[1][2]`)

Slicing


```
>>> a[0:2]
  array([1, 2])
```

Select items at index 0 and 1


```
>>> b[0:2,1]
  array([ 2.,  5.])
```

Select items at rows 0 and 1 in column 1


```
>>> b[:1]
  array([[1.5, 2., 3.]])
```

Select all items at row 0 (equivalent to `b[0:1, :]`)

```
>>> c[1,...]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
```

Same as `[1,:,:]`

```
>>> a[ : :-1]
  array([3, 2, 1])
```

Reversed array `a`

Boolean Indexing


```
>>> a[a<2]
  array([1])
```

Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
  array([ 4. , 2. , 6. , 1.5])
```

Select elements `(1,0)`, `(0,1)`, `(1,2)` and `(0,0)`

```
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
  array([[ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)        Permute array dimensions
>>> i.T                        Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                  Flatten the array
>>> g.reshape(3,-2)            Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))            Return a new array with shape (2,6)
>>> np.append(h,g)             Append items to an array
>>> np.insert(a, 1, 5)         Insert items in an array
>>> np.delete(a,[1])           Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)        Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                    Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                          Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                    Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))              Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                          Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                      Split the array horizontally at the 3rd index
  [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)                      Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5]          Create a dense meshgrid
>>> np.ogrid[0:2,0:2]          Create an open meshgrid
>>> np.r_[[3,[0]*5,-1:1:10j]   Stack arrays vertically (row-wise)
>>> np.c_[b,c]                 Create stacked column-wise arrays
```

Shape Manipulation

```
>>> np.transpose(b)            Permute array dimensions
>>> b.flatten()                Flatten the array
>>> np.hstack((b,c))           Stack arrays horizontally (column-wise)
>>> np.vstack((a,b))           Stack arrays vertically (row-wise)
>>> np.hsplit(c,2)             Split the array horizontally at the 2nd index
>>> np.vpslit(d,2)             Split the array vertically at the 2nd index
```

Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5])        Create a polynomial object
```

Vectorizing Functions

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)       Vectorize functions
```

Type Handling

```
>>> np.real(c)                 Return the real part of the array elements
>>> np.imag(c)                 Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000)  Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)        Cast object to a data type
```

Other Useful Functions

```
>>> np.angle(b,deg=True)       Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)  Create an array of evenly spaced values
                                    (number of samples)
>>> g [3:] += np.pi
>>> np.unwrap(g)               Unwrap
>>> np.logspace(0,10,3)        Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2])     Return values from a list of arrays depending on
                               conditions
>>> misc.factorial(a)          Factorial
>>> misc.comb(10,3,exact=True)  Combine N things taken at k time
>>> misc.central_diff_weights(3)  Weights for Np-point central derivative
>>> misc.derivative(myfunc,1.0)   Find the n-th derivative of a function at a point
```

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I                        Inverse
>>> linalg.inv(A)              Inverse
>>> A.T                        Tranpose matrix
>>> A.H                        Conjugate transposition
>>> np.trace(A)                Trace
```

Norm

```
>>> linalg.norm(A)             Frobenius norm
>>> linalg.norm(A,1)           L1 norm (max column sum)
>>> linalg.norm(A,np.inf)      L inf norm (max row sum)
```

Rank

```
>>> np.linalg.matrix_rank(C)   Matrix rank
```

Determinant

```
>>> linalg.det(A)              Determinant
```

Solving linear problems

```
>>> linalg.solve(A,b)          Solver for dense matrices
>>> E = np.mat(a).T            Solver for dense matrices
>>> linalg.lstsq(D,E)          Least-squares solution to linear matrix
                               equation
```

Generalized inverse

```
>>> linalg.pinv(C)             Compute the pseudo-inverse of a matrix
                               (least-squares solver)
>>> linalg.pinv2(C)            Compute the pseudo-inverse of a matrix
                               (SVD)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)         Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2)) Create a 2x2 identity matrix
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)   Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D)   Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A)   Dictionary Of Keys matrix
>>> E.todense()                Sparse matrix to full matrix
>>> sparse.isspmatrix_csc(A)   Identify sparse matrix
```

Sparse Matrix Routines

Inverse

```
>>> sparse.linalg.inv(I)       Inverse
```

Norm

```
>>> sparse.linalg.norm(I)      Norm
```

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I) Solver for sparse matrices
```

Sparse Matrix Functions

```
>>> sparse.linalg.expm(I)      Sparse matrix exponential
```

Matrix Functions

Addition

```
>>> np.add(A,D)                Addition
```

Subtraction

```
>>> np.subtract(A,D)           Subtraction
```

Division

```
>>> np.divide(A,D)             Division
```

Multiplication

```
>>> np.multiply(D,A)           Multiplication
>>> np.dot(A,D)                Dot product
>>> np.vdot(A,D)               Vector dot product
>>> np.inner(A,D)              Inner product
>>> np.outer(A,D)              Outer product
>>> np.tensordot(A,D)          Tensor dot product
>>> np.kron(A,D)               Kronecker product
```

Exponential Functions

```
>>> linalg.expm(A)             Matrix exponential
>>> linalg.expm2(A)            Matrix exponential (Taylor Series)
>>> linalg.expm3(D)            Matrix exponential (eigenvalue
                               decomposition)
```

Logarithm Function

```
>>> linalg.logm(A)             Matrix logarithm
```

Trigonometric Tunctions

```
>>> linalg.sinm(D)             Matrix sine
>>> linalg.cosm(D)             Matrix cosine
>>> linalg.tanm(A)             Matrix tangent
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)            Hypberbolic matrix sine
>>> linalg.coshm(D)            Hyperbolic matrix cosine
>>> linalg.tanhm(A)            Hyperbolic matrix tangent
```

Matrix Sign Function

```
>>> np.sigm(A)                 Matrix sign function
```

Matrix Square Root

```
>>> linalg.sqrtm(A)            Matrix square root
```

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)  Evaluate matrix function
```

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)      Solve ordinary or generalized
                               eigenvalue problem for square matrix
>>> l1, l2 = la                Unpack eigenvalues
>>> v[:,0]                     First eigenvector
>>> v[:,1]                     Second eigenvector
>>> linalg.eigvals(A)          Unpack eigenvalues
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)     Singular Value Decomposition (SVD)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)  Construct sigma matrix in SVD
```

LU Decomposition

```
>>> P,L,U = linalg.lu(C)       LU Decomposition
```

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)  Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2)         SVD
```

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

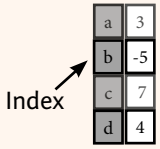
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

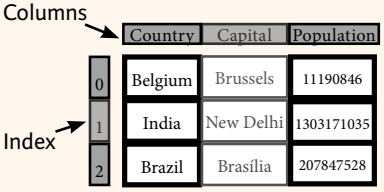
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5

>>> df[1:]
   Country     Capital   Population
 1   India   New Delhi   1303171035
 2  Brazil    Brasília    207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
 'Belgium'
>>> df.iat[[0],[0]]
 'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
 'Belgium'
>>> df.at[[0], ['Country']]
 'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
 Country      Brazil
 Capital    Brasília
 Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
 0     Brussels
 1    New Delhi
 2     Brasília
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
 'New Delhi'
```

Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series `s` where value is not >1
`s` where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index `a` of Series `s` to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a     10.0
  b      NaN
  c      5.0
  d      7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c    5.0
  d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

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

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

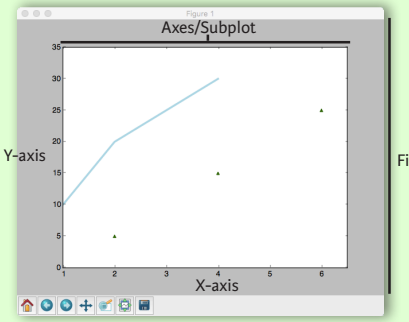
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6))
```
Create a figure and one subplot

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```
(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters

Return a dict of params or use with `with` to temporarily set the style

Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                    font_scale=1.5,
                    rc={"lines.linewidth":2.5})
```
Set context to `"talk"`
Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

```
>>> sns.set_palette("husl",3)
>>> sns.color_palette("husl")
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui)
```
Define the color palette
Use with `with` to temporarily set palette
Set your own color palette

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
```
Subplot grid for plotting conditional relationships

```
>>> sns.factorplot(x="pclass",
                   y="survived",
                   hue="sex",
                   data=titanic)
```
Draw a categorical plot onto a Facetgrid

```
>>> sns.lmplot(x="sepal_width",
               y="sepal_length",
               hue="species",
               data=iris)
```
Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
```
Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions

```
>>> i = sns.JointGrid(x="x",
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
```
Grid for bivariate plot with marginal univariate plots

```
>>> sns.jointplot("sepal_length",
                  "sepal_width",
                  data=iris,
                  kind='kde')
```
Plot bivariate distribution

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species",
                  y="petal_length",
                  data=iris)
```
Scatterplot with one categorical variable

```
>>> sns.swarmplot(x="species",
                  y="petal_length",
                  data=iris)
```
Categorical scatterplot with non-overlapping points

Bar Chart

```
>>> sns.barplot(x="sex",
                y="survived",
                hue="class",
                data=titanic)
```
Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

```
>>> sns.countplot(x="deck",
                  data=titanic,
                  palette="Greens_d")
```
Show count of observations

Point Plot

```
>>> sns.pointplot(x="class",
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                           "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```
Show point estimates and confidence intervals as rectangular bars

Boxplot

```
>>> sns.boxplot(x="alive",
                y="age",
                hue="adult_male",
                data=titanic)
```
Boxplot

```
>>> sns.boxplot(data=iris,orient="h")
```
Boxplot with wide-form data

Violinplot

```
>>> sns.violinplot(x="age",
                   y="sex",
                   hue="survived",
                   data=titanic)
```
Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width",
                y="sepal_length",
                data=iris,
                ax=ax)
```
Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,
                        kde=False,
                        color="b")
```
Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1)
```
Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
                      "Sex")
```
Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

```
>>> h.set(xlim=(0,5),
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```
Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax,yticks=[0,5])
>>> plt.tight_layout()
```
Add plot title
Adjust the label of the y-axis
Adjust the label of the x-axis
Adjust the limits of the y-axis
Adjust the limits of the x-axis
Adjust a plot property
Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

```
>>> plt.show()
>>> plt.savefig("foo.png")
>>> plt.savefig("foo.png",
                transparent=True)
```
Show the plot
Save the plot as a figure
Save transparent figure

Close & Clear

Also see Matplotlib

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```
Clear an axis
Clear an entire figure
Close a window

Python For Data Science *Cheat Sheet*

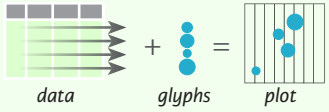
Bokeh

Learn Bokeh Interactively at www.DataCamp.com, taught by Bryan Van de Ven, core contributor

Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data:

Python lists, NumPy arrays, Pandas DataFrames and other sequences of values

2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]                                  Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",               Step 2
              x_axis_label='x',
              y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)           Step 3
>>> output_file("lines.html")                            Step 4
>>> show(p)                                              Step 5
```

1 Data

Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'],
                                [32.4,4,66, 'Asia'],
                                [21.4,4,109, 'Europe']]),
                      columns=['mpg','cyl', 'hp', 'origin'],
                      index=['Toyota', 'Fiat', 'Volvo'])
```

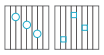
```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
                x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

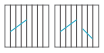
3 Renderers & Visual Customizations

Glyphs



Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
              color='blue', size=1)
```



Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Customized Glyphs

Also see Data

Selection and Non-Selection Glyphs

```
>>> p = figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

Hover Glyphs

```
>>> from bokeh.models import HoverTool
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```

Colormapping

```
>>> from bokeh.models import CategoricalColorMapper
>>> color_mapper = CategoricalColorMapper(
                   factors=['US', 'Asia', 'Europe'],
                   palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
             color=dict(field='origin',
                        transform=color_mapper),
             legend='Origin')
```

Legend Location

Inside Plot Area

```
>>> p.legend.location = 'bottom_left'
```

Outside Plot Area

```
>>> from bokeh.models import Legend
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" ,[p1, r1]),("Two",[r2])],
                    location=(0, -30))
>>> p.add_layout(legend, 'right')
```

Legend Orientation

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

Rows & Columns Layout

Rows

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2,p3)
```

Columns

```
>>> from bokeh.layouts import columns
>>> layout = column(p1,p2,p3)
```

Nesting Rows & Columns

```
>>>layout = row(column(p1,p2), p3)
```

Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Linked Plots

Linked Axes

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

Linked Brushing

```
>>> p4 = figure(plot_width = 100,
                tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200,
                tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

4 Output & Export

Notebook

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

HTML

Standalone HTML

```
>>> from bokeh.embed import file_html
>>> from bokeh.resources import CDN
>>> html = file_html(p, CDN, "my_plot")
```

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

PNG

```
>>> from bokeh.io import export_png
>>> export_png(p, filename="plot.png")
```

SVG

```
>>> from bokeh.io import export_svgs
>>> p.output_backend = "svg"
>>> export_svgs(p, filename="plot.svg")
```

5 Show or Save Your Plots

```
>>> show(p1)
>>> save(p1)
```

```
>>> show(layout)
>>> save(layout)
```