

Binary Brains: Misty Richardson, Ambalika Rajendran, Joseph Hiller, Favour Asu and Zaid Jamil

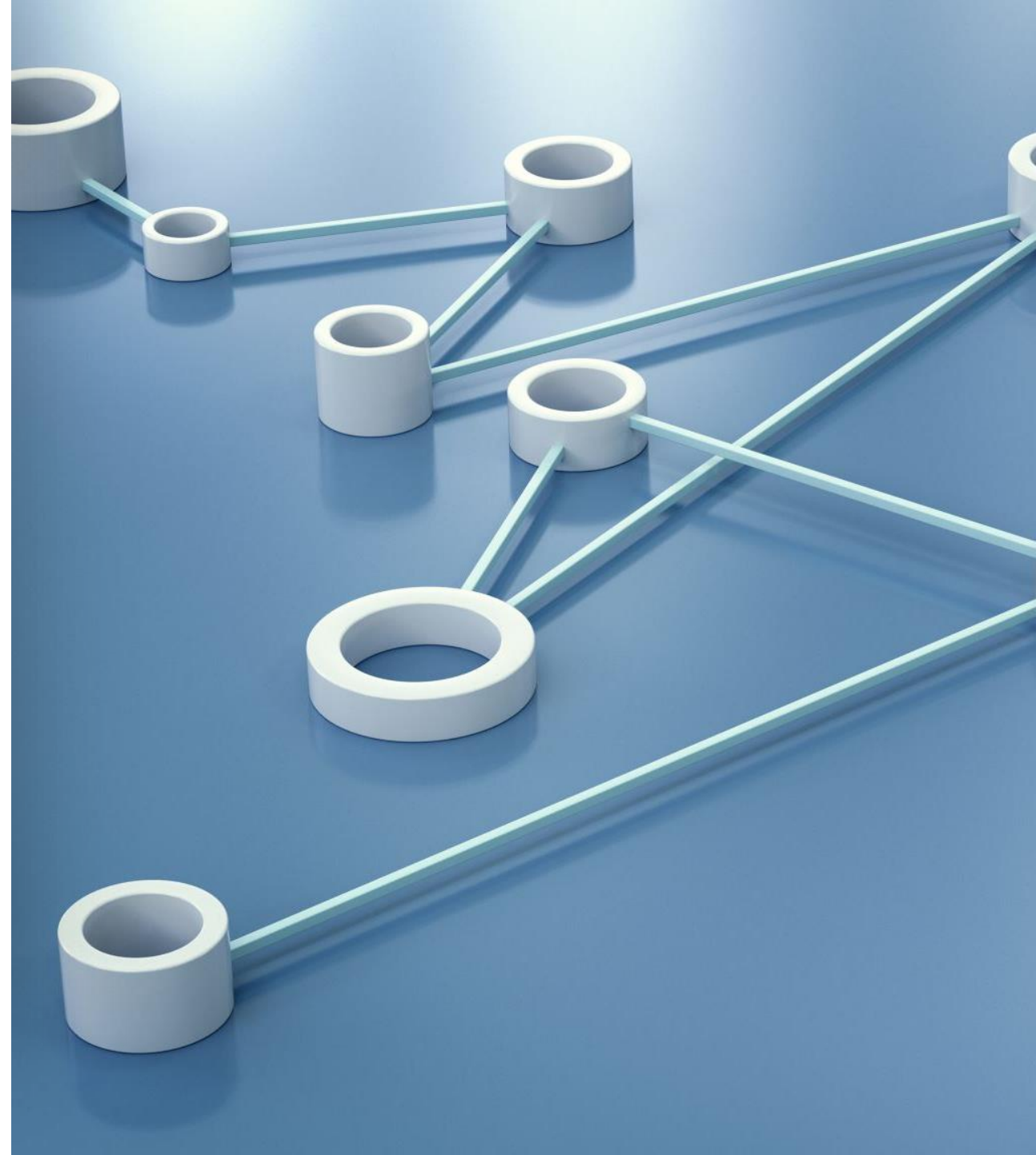
# What is GitHub?

---



# Understanding GitHub

- GitHub is a web-based platform primarily used for version control using Git. It allows developers to collaborate on projects, track changes to code, and manage software development projects. GitHub provides features such as bug tracking, feature requests, task management, and wikis for every project. It's widely used by individual developers, open-source projects, and companies alike to host their code repositories and manage their software development workflows.







# Understanding the GitHub

- The implementation of GitHub involves several key steps:
- **Account Creation:** To use GitHub, you need to create an account. You can sign up for free with a username, email address, and password.
- **Repository Creation:** After signing in, you can create a new repository by clicking the "New" button on your dashboard. You'll need to give your repository a name, description, and choose whether it's public or private.
- **Setting Up Git:** Install Git on your local machine if you haven't already. Git is a distributed version control system that GitHub is built on. You can download and install Git from the official website (<https://git-scm.com/>).
- **Cloning Repositories:** To work on a GitHub repository locally, you need to clone it to your machine. You can do this by using the `git clone` command followed by the URL of the repository.
- **Adding Files and Making Changes:** Once you have cloned a repository, you can add files, make changes to existing files, and create new ones using your preferred code editor.
- **Staging and Committing Changes:** After making changes to your files, you need to stage them for commit using the `git add` command. Once staged, you can commit the changes to your local repository using the `git commit` command.
- **Pushing Changes to GitHub:** To update the repository on GitHub with your local changes, you need to push your commits using the `git push` command.
- **Pulling Changes from GitHub:** If there have been changes made to the repository by other collaborators, you can pull those changes to your local repository using the `git pull` command.
- **Branching and Merging:** GitHub allows you to create branches to work on features or fixes independently. You can merge branches back into the main branch (usually `master` or `main`) using pull requests.
- **Collaboration and Pull Requests:** GitHub facilitates collaboration through pull requests. When you want to merge your changes into the main branch, you create a pull request, which allows other collaborators to review your changes before merging.
- **Issue Tracking and Project Management:** GitHub provides tools for issue tracking and project management. You can create issues to track bugs, feature requests, or other tasks, and organize them into projects and milestones.
- These are some of the fundamental steps involved in using GitHub for version control and collaborative software development.



# Real projects

---

- In real projects, developers utilize these features to organize code, manage versions, coordinate tasks, and review code contributions. This fosters efficient teamwork, reduces conflicts, and maintains code quality, enabling developers to build and deliver software collaboratively.



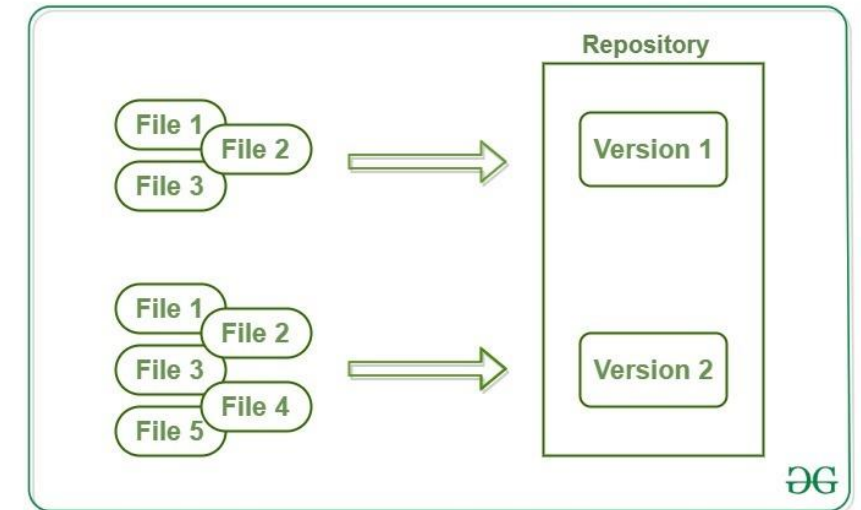
# Version Control

Version control is a system that tracks and manages changes to files over time, particularly crucial in software development. It allows developers to keep track of modifications made to code files, enabling them to revert to previous versions, collaborate seamlessly, and maintain code integrity. With version control systems like Git, developers can create branches to work on new features or bug fixes independently, merge changes, and resolve conflicts efficiently. Version control ensures accountability by attributing changes to specific users and provides a complete history of code alterations, aiding in debugging and auditing processes. It fosters collaboration by facilitating code review, enabling multiple team members to work on the same codebase concurrently, and streamlining the integration of changes into the main code repository. Ultimately, version control enhances productivity, code quality, and project management in software development endeavors.

# Repositories

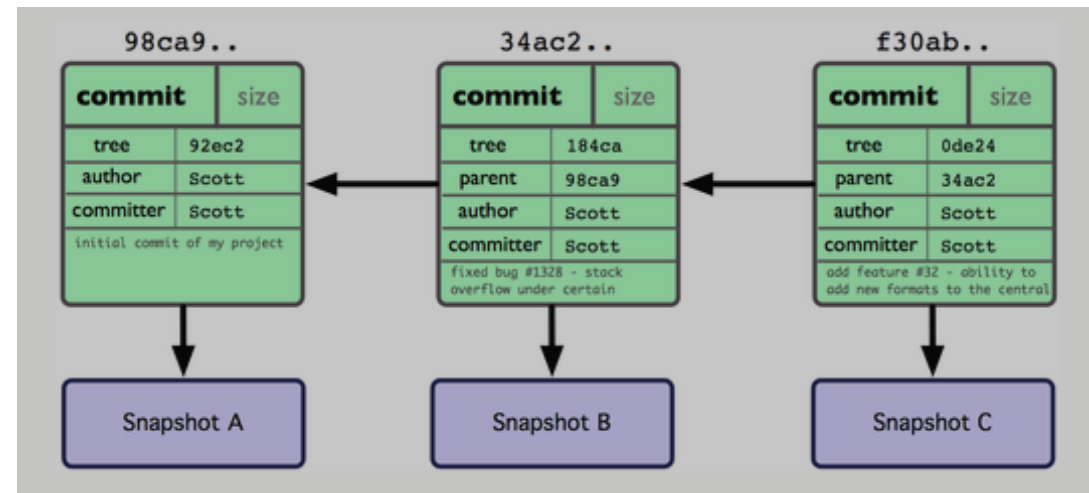
Repositories in GIT contain a collection of files of various different versions of a Project. It is used to store project files and history

- **Bare Repositories:** These repositories are used to share the changes that are done by different developers. A user is not allowed to modify this repository or create a new version for this repository based on the modifications done.
- **Non-bare Repositories:** Non-bare repositories are user-friendly and hence allow the user to create new modifications of files and also create new versions for the repositories. The cloning process by default creates a non-bare repository if any parameter is not specified during the clone operation.



# Branches

- Allow developers to work on different features simultaneously without disrupting the main codebase. Branches allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.
- You always create a branch from an existing branch. Typically, you might create a new branch from the default branch of your repository. You can then work on this new branch in isolation from changes that other people are making to the repository. A branch you create to build a feature is commonly referred to as a feature branch or topic branch





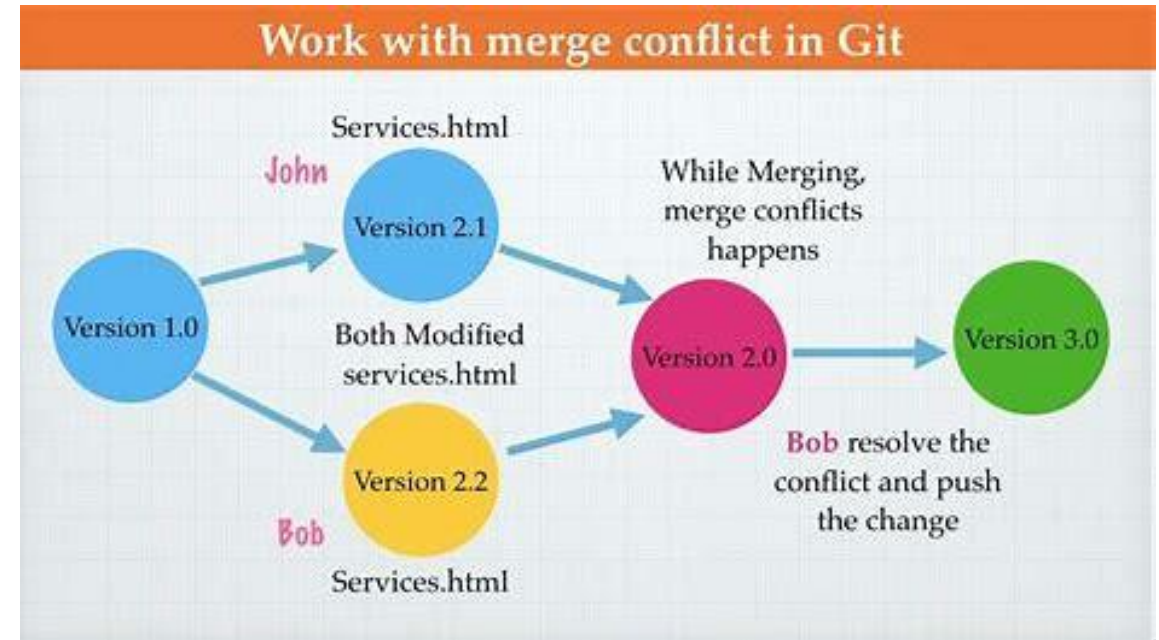
# Commits

- Capture changes made to files, helping teams track progress and understand code alterations
- Common usages and options for Git Commit
- Git commit: This starts the commit process, but since it doesn't include a -m flag for the message, your default text editor will be opened for you to create the commit message. If you haven't configured anything, there's a good chance this will be VI or Vim. (To get out, press esc, then :w, and then Enter. :wink:)
- Git commit -m "descriptive commit message": This starts the commit process, and allows you to include the commit message at the same time.
- Git commit -am "descriptive commit message": In addition to including the commit message, this option allows you to skip the staging phase. The addition of -a will automatically stage any files that are already being tracked by Git (changes to files that you've committed before).
- Git commit --amend: Replaces the most recent commit with a new commit.



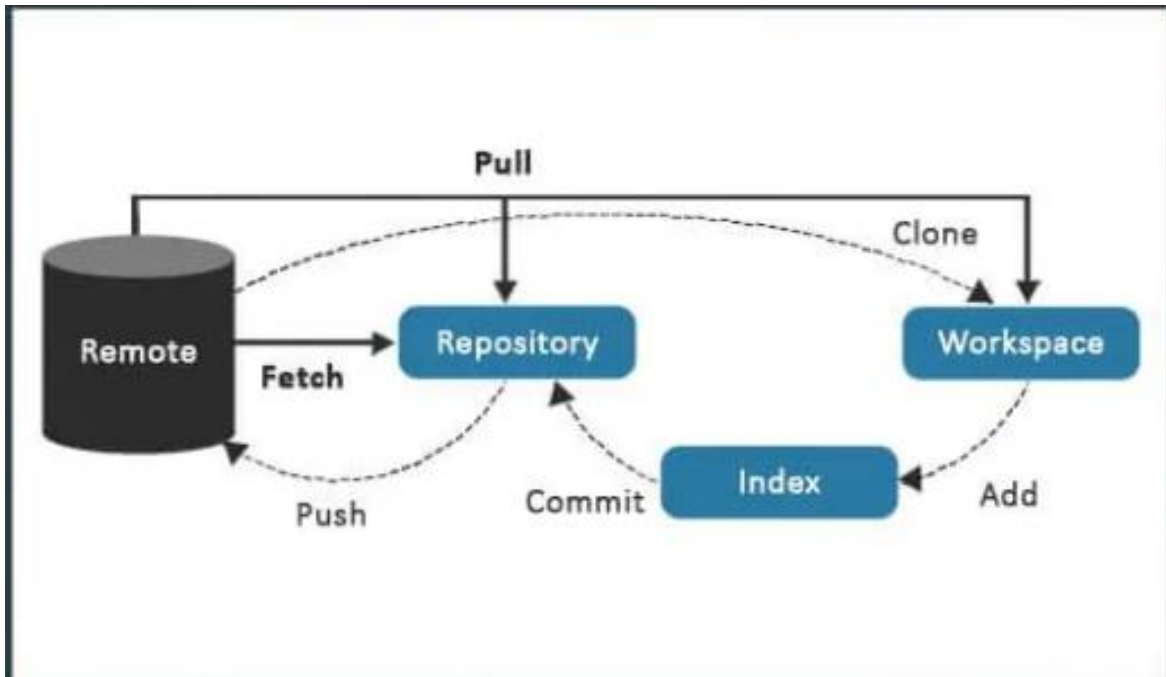
# Merges

- Merges integrate changes from one branch into another, ensuring smooth collaboration. Merging is the process of combining the recent changes from several branches into a single new commit that will be on all those branches.
- In a way, merging is the complement of branching in version control: a branch allows you to work simultaneously with others on a particular set of files, whereas a merge allows you to later combine separate work on branches that diverged from a common ancestor commit.




# Pull

Pull requests let developers propose changes, initiate discussions, and request code reviews before merging into the main branch. A pull request is a proposal to merge a set of changes from one branch into another. In a pull request, collaborators can review and discuss the proposed set of changes before they integrate the changes into the main codebase. Pull requests display the differences, or diffs, between the content in the source branch and the content in the target branch.





▶ M↓ 

```
import numpy as np
import pandas as pd
```

[2]

▶ M↓ 

```
myNparray = np.array([[ 'Bob', 1, 2, 3], [ 'Alice', 4, 5, 6], [ 'Gina', 7, 8, 9]])
myDataFrame = pd.DataFrame(myNparray, columns=[ 'name', 'b', 'c', 'd'])
myDataFrame
```

	name	b	c	d
0	Bob	1	2	3
1	Alice	4	5	6
2	Gina	7	8	9

[4]

▶ M↓ 

```
import qgrid
qgrid_widget = qgrid.show_grid(myDataFrame, show_toolbar=True)
qgrid_widget
```

Error: Module qgrid, semver range ^1.1.3 is not registered as a widget module

^ [-]

▶ M↓

# What is Jupyter Notebook?



# Understanding Jupyter Notebook

## Interactive Computing:

- Jupyter Notebooks offer a cell-based interface for writing and executing code, promoting iterative development and experimentation.
- Users can run code cells individually or sequentially, fostering an interactive workflow for exploration and prototyping.

## Summary:

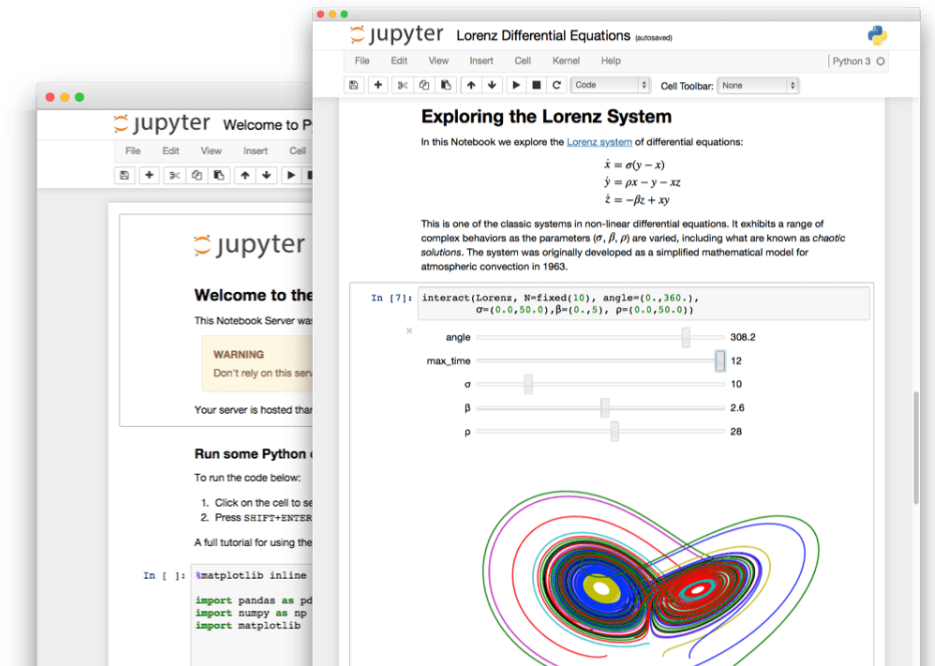
- Jupyter Notebooks play a pivotal role in data analysis, exploration, and presentation, enabling users to seamlessly integrate code, visualizations, and narrative text.
- Their versatility and ease of use make them indispensable tools for fostering collaboration, reproducibility, and innovation across various domains, including data science and machine learning.

### simple text input

```
In [ ]: name=input()
```

```
In [ ]:
```

# Documentation - Jupyter NoteBook



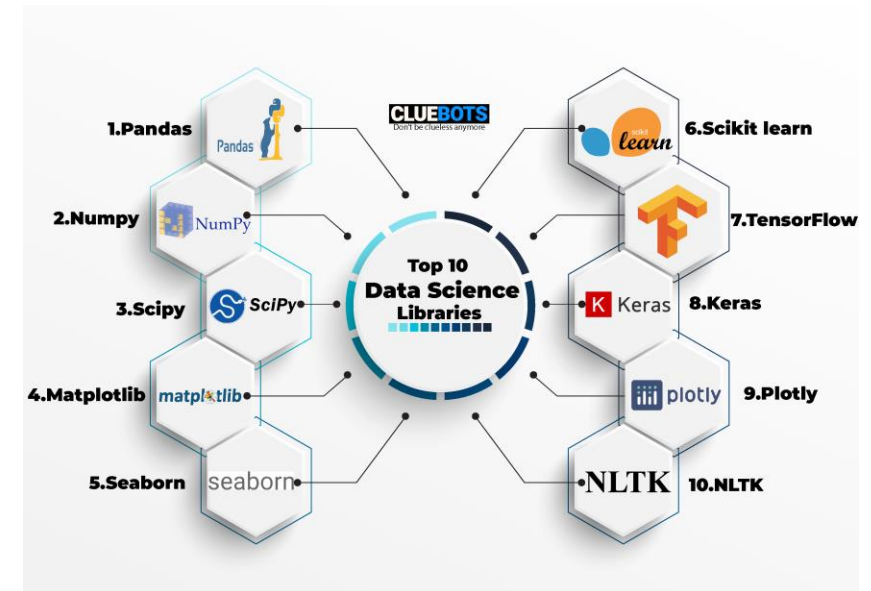
- Combining executable code with explanatory text and visualizations, Jupyter Notebooks serve as powerful documentation tools.
- Markdown cells allow users to add detailed explanations, instructions, and analysis interpretations within the same document, enhancing clarity and readability.
- Notebooks capture the entire computational process, from data preprocessing to analysis steps and results, ensuring a comprehensive record.
- Sites to find code documentation include [Project Jupyter Documentation](#), [Stack Overflow](#), and [Python.org](#)



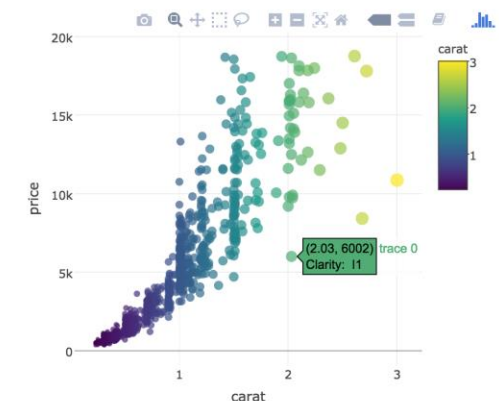
# Usage in Data Analysis Projects

## – Jupyter Notebook

- Widely used for data analysis projects, Jupyter Notebooks support flexible and interactive development processes.
- Analysts leverage Python libraries like Pandas and NumPy for data manipulation and exploration, supported by visualization tools like Matplotlib and Seaborn.
- Rich and interactive data visualizations aid in exploring patterns and relationships within datasets, facilitating insight generation and comprehension.
- Jupyter Notebooks also serve as ideal environments for developing, training, and evaluating machine learning models, with detailed documentation and commentary enhancing understanding and reproducibility.

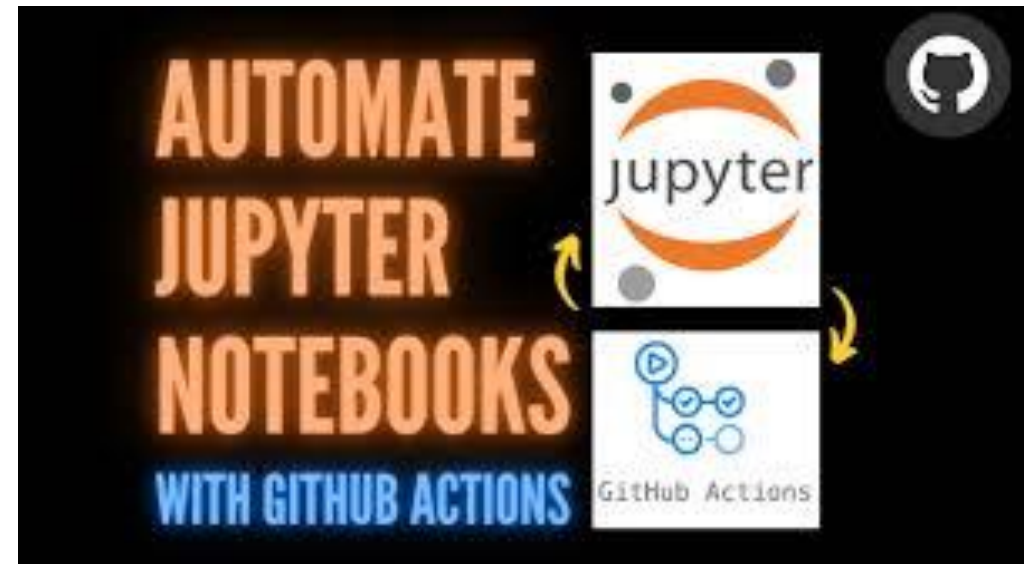


```
In [5]: library(plotly)
set.seed(100)
d <- diamonds[sample(nrow(diamonds), 1000), ]
plot_ly(d, type = 'scatter', mode = 'markers',
        x = ~carat, y = ~price,
        color = ~carat, size = ~carat,
        text = ~paste("Clarity: ", clarity))
```



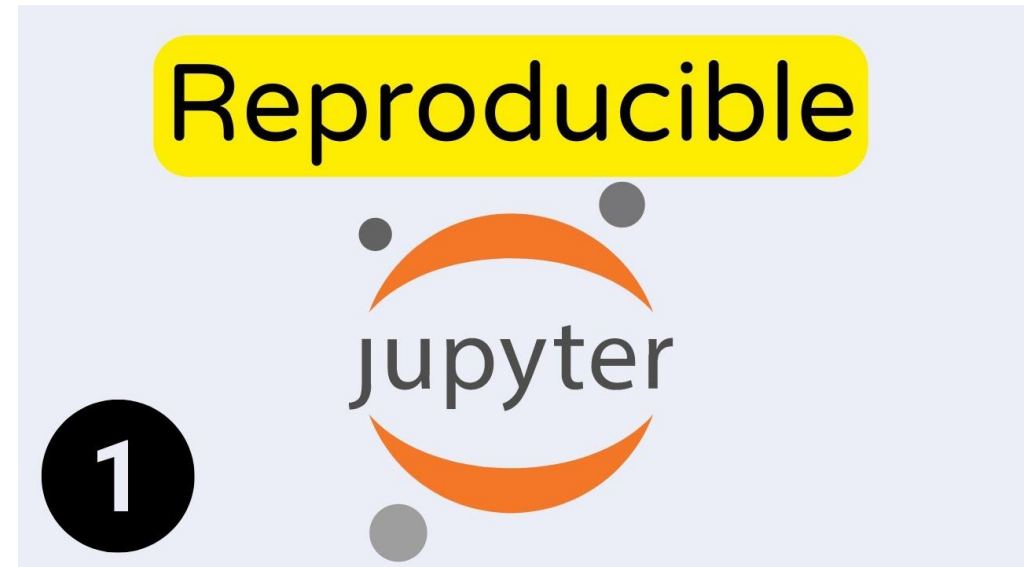
# Synergistically Operate in Real-World Scenarios.

- In real-world scenarios, GitHub, Python, and Jupyter synergistically operate to create a powerful and efficient workflow, especially in data science and software development projects.
- Developers initiate projects on GitHub, hosting their Python code and Jupyter Notebooks in repositories. its version control facilitates collaborative coding, allowing multiple contributors to work on the same project simultaneously without conflicts. Issues and project boards on GitHub are used for task tracking, bug reporting, and organizing development sprints.



# Reproducibility and Transparency:

- The integration of GitHub, Python, and Jupyter Notebooks ensures reproducibility, as the entire project, including code and documentation, is version-controlled.
- Jupyter Notebooks, embedded in GitHub, allow for transparent sharing of analyses, enabling others to reproduce and build upon the work.





# Python as the Core Language:

- It is the primary language for coding, scripting, and application development.
- GitHub integrates with Python, ensuring efficient version control and tracking of code changes made during development.
- Python's versatility allows it to serve as the main language for Jupyter Notebooks, providing an interactive and executable environment for data analysis and documentation within GitHub repositories.

