PORTFOLIO BUILDING AND VERSION CONTROL FOR DATA ANALYST

BY WALE OLAJUMOKE

Course Outline

1. Introduction to GitHub for Data Analysis

- Overview of GitHub and its relevance to data analysis projects.
- Understanding how version control benefits data analysts.

2. Setting Up GitHub

- Creating a GitHub account.
- Introduction to repositories and their role in managing data-related projects.

3. Basics of Git and GitHub

- Understanding Git commands commonly used in data analysis projects.
- Introduction to basic GitHub operations like cloning repositories and creating branches.

Course Outline Cntd..

4. Data Versioning with Git

- Using Git to version control data files

5. Collaborating on Data Projects

- Collaborating with team members on data analysis projects using GitHub.

6. Data Documentation and GitHub

- Creating clear and informative README files for data projects.
- Incorporating documentation within GitHub repositories to explain data analysis processes.

1. Introduction to github

- **GitHub** is a web-based platform that provides **version control** and **collaboration** for software development projects. While its primary focus is on code repositories, GitHub is also highly relevant to **data analysis projects**

- **Github alternatives**
- Gitlab
- Bitbucket

Benefits of Github

- **Version Control:** GitHub allows you to track changes made to files over time.
- **Collaboration:** Multiple team members or collaborators can work on the same project simultaneously. Each contributor can make changes independently in their branch, and then merge these changes into the main project when ready. It facilitates seamless teamwork, even in data analysis tasks.
- **Documentation and Project Showcase:** GitHub provides an opportunity to create detailed documentation for data analysis projects. By leveraging features like **README files**, wikis, and **GitHub Pages**, data analysts can communicate project objectives, methodologies, findings, and insights effectively.
- **Continuous Integration and Deployment:** GitHub integrates with various continuous integration (CI) and continuous deployment (CD) tools that can automate testing, build processes, and deployment of data analysis code.

2. Setting up account & repositories

- You can create a FREE account on **https://github.com**

- **Repositories** (repos) are **storage spaces where code, files, and related resources are stored and organized**. They serve as a central location for version control, collaboration, and project management. In the context of software development and data-related projects, repositories are commonly used to store source code, scripts, **documentation**, configuration files, and other project artifacts.

3. Basics of github

- **Remote Repository Hosting:** GitHub is a web-based platform that provides hosting for Git repositories.
- **Pull Requests**: Developers can propose changes to a repository by creating a pull request, which allows others to review, discuss, and suggest modifications before merging the changes into the main codebase.
- **Issue Tracking:** GitHub includes an issue tracking system that allows developers to create, track, and manage issues, bugs, and tasks related to a project.
- **Project Management:** GitHub provides tools for project management between teams.
- **Integration and Automation:** GitHub integrates with various tools and services, such as continuous integration (CI) systems, deployment services, code analysis tools, and project management platforms. This integration allows for automating tasks, improving code quality, and streamlining development workflows.

Git commands

git init: Initializes a new Git repository in the current directory, creating a hidden .git folder that tracks changes.

git clone [repository URL]: Creates a local copy of a remote Git repository, allowing you to work on the project locally.

git add [file(s)]: Adds specific file(s) or directories to the staging area, preparing them for the next commit. For example, git add script.py adds script.py to the staging area.

git commit -m "[commit message]": Commits the changes in the staging area, creating a new snapshot of the project with a descriptive commit message. For example, git commit -m "Add data preprocessing script".

git status: Shows the status of the repository, including modified files, files in the staging area, and untracked files.

Git Configuration

```
git config --global user.name "Your Name"

git config --global user.email "Your Email"
```

.gitignore:

Create a .gitignore file in your repository to specify files and directories that Git should ignore.

Git commands contd...

git pull: Fetches changes from a remote repository and merges them with the local branch. It is used to update your local repository with the latest changes from the remote repository.

git push: Pushes local commits to a remote repository, synchronizing your changes with the remote branch. It is used to share your work with collaborators.

git branch: Lists all the branches in the repository. The current branch is indicated with an asterisk.

git branch [branch name]: Creates a new branch with the specified name. For example, git branch feature-branch creates a new branch called "feature-branch."

git checkout [branch name]/[commit hash]: Switches to the specified branch or commit. It allows you to work on a different branch or check out a specific version of the project.

git merge [branch name]: Merges changes from the specified branch into the current branch. It combines the changes and resolves any conflicts that may arise.

Git commands contd...

git log: Displays the commit history of the repository, showing the author, timestamp, and commit message for each commit.

git diff [file]: Shows the differences between the current state of a file and the last committed version. It helps identify the changes made to a specific file.

git remote: Lists the remote repositories associated with the local repository.

git remote add [remote name] [repository URL]: Adds a new remote repository with the given name and URL to the local repository.

4. Data Versioning with Git

File Size and Storage: Consider the size of your data files and evaluate whether they should be stored directly in the repository or handled separately.

Data File Format: Whenever possible, use text-based data file formats that are human-readable like CSV (comma-separated values), JSON (JavaScript Object Notation), YAML (YAML Ain't Markup Language), or plain text files. These formats allow Git to track changes effectively and provide meaningful diffs.

.gitignore: Create a .gitignore file in your repository to specify files and directories that Git should ignore.

Documentation: Include documentation that explains the purpose and structure of your data files. It can be helpful to include a **README** file

Commit Messages: Provide clear and descriptive commit messages when making changes to data files.
Collaboration: Git enables collaboration on data files by allowing multiple team members to work on the same repository.

5. Collaborating on Data Projects

Fork and Clone: One team member can create a fork of the main project repository on GitHub. Each team member can then clone their own forked repository to their local machine. This allows everyone to work on their own copies of the project while maintaining a central repository for collaboration.

Branching: Encourage team members to create feature-specific branches when working on new tasks or making changes. This helps isolate individual work and makes it easier to manage and review changes. Branches can be created using the git branch command and switched to using git checkout.

Pull Requests: When a team member completes a feature or task on their branch, they can create a pull request (PR) to propose their changes for review and merging into the main project. PRs provide an opportunity for others to review the code, provide feedback, and discuss any necessary changes.

Code Reviews: Encourage team members to review each other's pull requests. Code reviews help ensure code quality, catch errors, share knowledge, and maintain consistency. GitHub provides features to add comments, suggest changes, and discuss specific lines of code within pull requests.

Collaborating on Data Projects contd…

README: Create a README file in your GitHub repository to provide an overview of the project and its data. Include information such as **project objectives, data sources, data format**, and any necessary setup instructions. This acts as a starting point for others to understand the project and its data.

Data Dictionary: Include a data dictionary that describes the structure, meaning, and characteristics of the data variables or fields. Document the data types, units of measurement, possible values, and any relevant metadata. This helps team members understand the data and ensures consistent interpretation.

Documentation Files: Create additional files to document specific aspects of the data, such as data preprocessing steps, transformation logic, or data cleaning procedures. These files can provide insights into how the data has been processed and manipulated, ensuring reproducibility and transparency.

Collaborating on Data Projects contd...

Issue Tracking: Use GitHub's issue tracking system to document and track data-related issues, such as data quality problems, missing values, or anomalies.

Version Control: With Git's version control capabilities, you can track changes made to data files over time.
Wiki Pages: GitHub provides a wiki feature that allows you to create additional documentation pages for your project. You can use wiki pages to provide more detailed information about the data, such as data collection methods, data sources, data schema, or data model diagrams. This can serve as a comprehensive knowledge base for the project.

Data Visualization: Leverage GitHub's support for data visualization tools and libraries. You can use tools like Jupyter Notebooks, R Markdown, or interactive data visualization libraries to create visualizations that enhance the understanding of the data. Embed these visualizations in relevant documentation files or README to provide a visual representation of the data.

6. Data Documentation and GitHub

Managing Issues: Utilize GitHub's issue tracking system to manage tasks, bugs, and feature requests.
Collaborative Discussion: GitHub provides features such as comments on code, pull requests, and issues, which facilitate discussion and collaboration among team members. Encourage open communication, discuss design decisions, clarify requirements, and address any questions or concerns.

Continuous Integration: Set up continuous integration (CI) workflows using tools like GitHub Actions or other CI services. CI workflows automatically build, test, and validate code changes, ensuring that the project remains in a stable state. This helps detect issues early and provides confidence when merging changes.

Pulling and Syncing: Regularly pull changes from the main repository to your local branch to keep it up to date with the latest developments.

Resolving Conflicts: Occasionally, conflicts may arise when merging changes. In such cases, communicate with your team members to resolve conflicts and ensure a smooth merge.

Thank You