**Credit Card Fraud Detection Model project to GitHub**

Here's a step-by-step guide on how to upload your **Credit Card Fraud Detection Model** project to GitHub.

**Step-by-Step Guide to Uploading to GitHub**

**1. Create a GitHub Repository**

1. Go to [GitHub](https://github.com/) and sign in to your account.
2. Click on the **"+"** icon at the top right corner and select **"New repository"**.
3. Enter the **Repository Name** (e.g., credit-card-fraud-detection).
4. Add a description (e.g., "A machine learning model for detecting credit card fraud using Python.").
5. Set it to **Public** or **Private** as needed.
6. Check **"Add a README file"** (optional).
7. Click **"Create repository"**.

**2. Upload Your Files to GitHub**

**Method 1: Using Git Commands (Recommended for Version Control)**

If you have Git installed on your computer, follow these steps:

**2.1 Open Terminal or Command Prompt**

Navigate to your project folder where your 01\_EDA.ipynb, 02\_Preprocessing.ipynb, and 03\_Modeling.ipynb files are located.

sh

CopyEdit

cd C:/Users/user/Documents/PROJECTS/DATA SCIENCE PROJECTS

**2.2 Initialize Git**

sh

CopyEdit

git init

**2.3 Link Your Local Folder to GitHub Repository**

sh

CopyEdit

git remote add origin https://github.com/YOUR\_GITHUB\_USERNAME/credit-card-fraud-detection.git

**2.4 Add Files to Git**

sh

CopyEdit

git add .

**2.5 Commit the Changes**

sh

CopyEdit

git commit -m "Initial commit - added EDA, preprocessing, and modeling notebooks"

**2.6 Push to GitHub**

sh

CopyEdit

git branch -M main

git push -u origin main

**Method 2: Uploading Files Manually via GitHub**

1. Open your repository on GitHub.
2. Click **"Add file"** > **"Upload files"**.
3. Drag and drop your .ipynb files, README.md, and dataset (if needed).
4. Scroll down and click **"Commit changes"**.

**3. Create a README.md File (Optional but Recommended)**

A README.md file helps others understand your project. You can create it using a text editor and add the following:

markdown

CopyEdit

# Credit Card Fraud Detection Model

This project applies machine learning to predict credit card fraud. It includes EDA, preprocessing, model training, and evaluation.

## 📂 Project Structure

- \*\*01\_EDA.ipynb\*\* - Exploratory Data Analysis

- \*\*02\_Preprocessing.ipynb\*\* - Data Preprocessing

- \*\*03\_Modeling.ipynb\*\* - Model Training & Evaluation

## 🚀 How to Use

1. Clone the repository:

```sh

git clone https://github.com/YOUR\_GITHUB\_USERNAME/credit-card-fraud-detection.git

1. Install dependencies:

sh

CopyEdit

pip install -r requirements.txt

1. Run the Jupyter notebooks.

**📊 Models Used**

* Logistic Regression
* Random Forest
* XGBoost

**📌 Features**

✅ Data visualization  
✅ Feature engineering  
✅ Model evaluation with AUC-ROC

**4. (Optional) Create a .gitignore File**

To avoid uploading unnecessary files, create a .gitignore file and add:

markdown

CopyEdit

\_\_pycache\_\_/

\*.csv

\*.xlsx

\*.log

.DS\_Store

venv/

**5. Keep Your Repository Updated**

Whenever you make changes to your project, follow these steps:

sh

CopyEdit

git add .

git commit -m "Updated model with hyperparameter tuning"

git push origin main

This ensures your project is properly stored and version-controlled on GitHub. 🚀 Let me know if you need any refinements!

**INCLUDING INTERACTIVE DASHBOARD IN GITHUB**

Yes, you can include your interactive dashboard and model evaluation results on GitHub. Here's how you can integrate it into your **GitHub repository** effectively:

**1. Where to Include This in Your Project?**

* **Best Location:** Add this code to **03\_Modeling.ipynb** (after "Evaluate models and save the best-performing one").
* **Reason:** This step saves the model evaluation results, which can later be used for visualization in the dashboard.

**2. How to Include This in GitHub?**

**A. Upload model\_metrics.csv to GitHub**

1. **Run the code** in 03\_Modeling.ipynb to generate model\_metrics.csv.
2. Navigate to your **GitHub repository**.
3. Click on **"Add File" > "Upload Files"**, then upload model\_metrics.csv.

**B. Create an Interactive Dashboard on GitHub**

You can use **GitHub Pages** or **Streamlit** to display the dashboard.

✅ **Option 1: Use GitHub Pages (for Static Dashboards)**

* If you're using **HTML/JavaScript** for visualization, push those files to a **docs/** folder.
* Enable **GitHub Pages** in repository settings.

✅ **Option 2: Use Streamlit (for a Python-Based Dashboard)**

* Create a dashboard.py file with **Streamlit** to load and display model\_metrics.csv.
* Users can **run it locally** or deploy it on **Streamlit Community Cloud**.

**3. Steps to Add a GitHub Action for Auto-Updating Metrics**

If you want **GitHub to update metrics automatically**, you can set up a **GitHub Action**:

1. Add a **workflow file** in .github/workflows/update\_metrics.yml.
2. Schedule it to **rerun the model and update model\_metrics.csv** automatically.

Would you like a **GitHub Action script** for automating this? 🚀

It depends on your goal for sharing the project on GitHub. Here’s a breakdown of the pros and cons of each approach:

**1️⃣ Uploading Files with Output (Recommended for Transparency)**

✅ **Pros:**

* Helps others **reproduce your work** and understand the results easily.
* Showcases your **EDA insights, model performance, and visualizations** without requiring users to rerun the code.
* Useful for **team collaboration**, as others can see what was done at each step.

❌ **Cons:**

* Larger file sizes, especially with **Jupyter Notebook outputs** (e.g., images, tables).
* Some outputs may contain **sensitive data** (e.g., model predictions, dataset statistics).

**2️⃣ Uploading Files with No Output (Recommended for Clean Code)**

✅ **Pros:**

* Keeps the repository **lightweight** and **clean**.
* Encourages users to **run the code themselves** to generate results.
* No risk of exposing **sensitive or unnecessary output**.

❌ **Cons:**

* Users must **rerun the notebooks** to see results, which might be inconvenient.
* No immediate visualization of **charts, tables, or model evaluations** on GitHub.

**Best Practice (Balanced Approach)**

👉 **Use a hybrid approach:**

1. **Keep outputs in key files** (like 03\_Modeling.ipynb & 04\_Deployment.ipynb) to showcase model results.
2. **Clear outputs from raw processing files** (like 02\_Preprocessing.ipynb) to keep the repo clean.
3. **Upload important outputs separately** (e.g., model\_metrics.csv, trained model .pkl files).

**Final Recommendation**

* If your **GitHub repo is for learning or transparency**, include outputs.
* If it’s for **professional work or deployment**, keep it clean and remove outputs.

HOW TO RUN EACH NOTEBOOK IN JUPYTER INDEPENDENTLY

Since you have created your project files separately (e.g., 01\_EDA.ipynb, 02\_Preprocessing.ipynb, etc.), you can run them independently in Jupyter Notebook by following these steps:

**📌 Running Each Notebook Independently in Jupyter**

**1️ Open Jupyter Notebook**

* If Jupyter Notebook is installed, launch it using:

bash

CopyEdit

jupyter notebook

* If using JupyterLab, launch with:

bash

CopyEdit

jupyter lab

**2️ Navigate to Your Project Folder**

* In the Jupyter interface, browse to your project directory where the .ipynb files are stored.
* Click on the notebook you want to run (e.g., 01\_EDA.ipynb).

**3️ Run the Notebook**

* Press **Shift + Enter** to run each cell.
* Or click **Kernel → Restart & Run All** to execute everything at once.

**📌 Handling Dependencies Between Notebooks**

Each notebook should work independently, but if one depends on another (e.g., 03\_Modeling.ipynb needs preprocessed data from 02\_Preprocessing.ipynb), you have two options:

**✅ Option 1: Save & Load Intermediate Data**

* Save processed data at the end of 02\_Preprocessing.ipynb:

python

CopyEdit

import pandas as pd

X\_train.to\_csv("X\_train.csv", index=False)

X\_test.to\_csv("X\_test.csv", index=False)

y\_train.to\_csv("y\_train.csv", index=False)

y\_test.to\_csv("y\_test.csv", index=False)

* Load it in 03\_Modeling.ipynb:

python

CopyEdit

import pandas as pd

X\_train = pd.read\_csv("X\_train.csv")

X\_test = pd.read\_csv("X\_test.csv")

y\_train = pd.read\_csv("y\_train.csv")

y\_test = pd.read\_csv("y\_test.csv")

**✅ Option 2: Use %run Magic Command**

* If you want to execute 02\_Preprocessing.ipynb inside 03\_Modeling.ipynb, use:

python

CopyEdit

%run 02\_Preprocessing.ipynb

**📌 Automating Execution (Run All Notebooks in Order)**

If you want to **run all notebooks sequentially** without opening them manually:

**Using Terminal (Command Line)**

Run the notebooks one after another with:

bash

CopyEdit

jupyter nbconvert --execute --inplace 01\_EDA.ipynb

jupyter nbconvert --execute --inplace 02\_Preprocessing.ipynb

jupyter nbconvert --execute --inplace 03\_Modeling.ipynb

This will **execute the notebooks in order** and save the output inside each .ipynb file.

**🎯 Final Takeaway**

* **Run notebooks independently** by manually executing each one.
* **Save/load intermediate results** to avoid re-running preprocessing steps.
* **Use %run or automation** if notebooks depend on each other.

Let me know if you need help setting up the automation! 🚀