**Machine Learning Operations**

**Assignment 1**



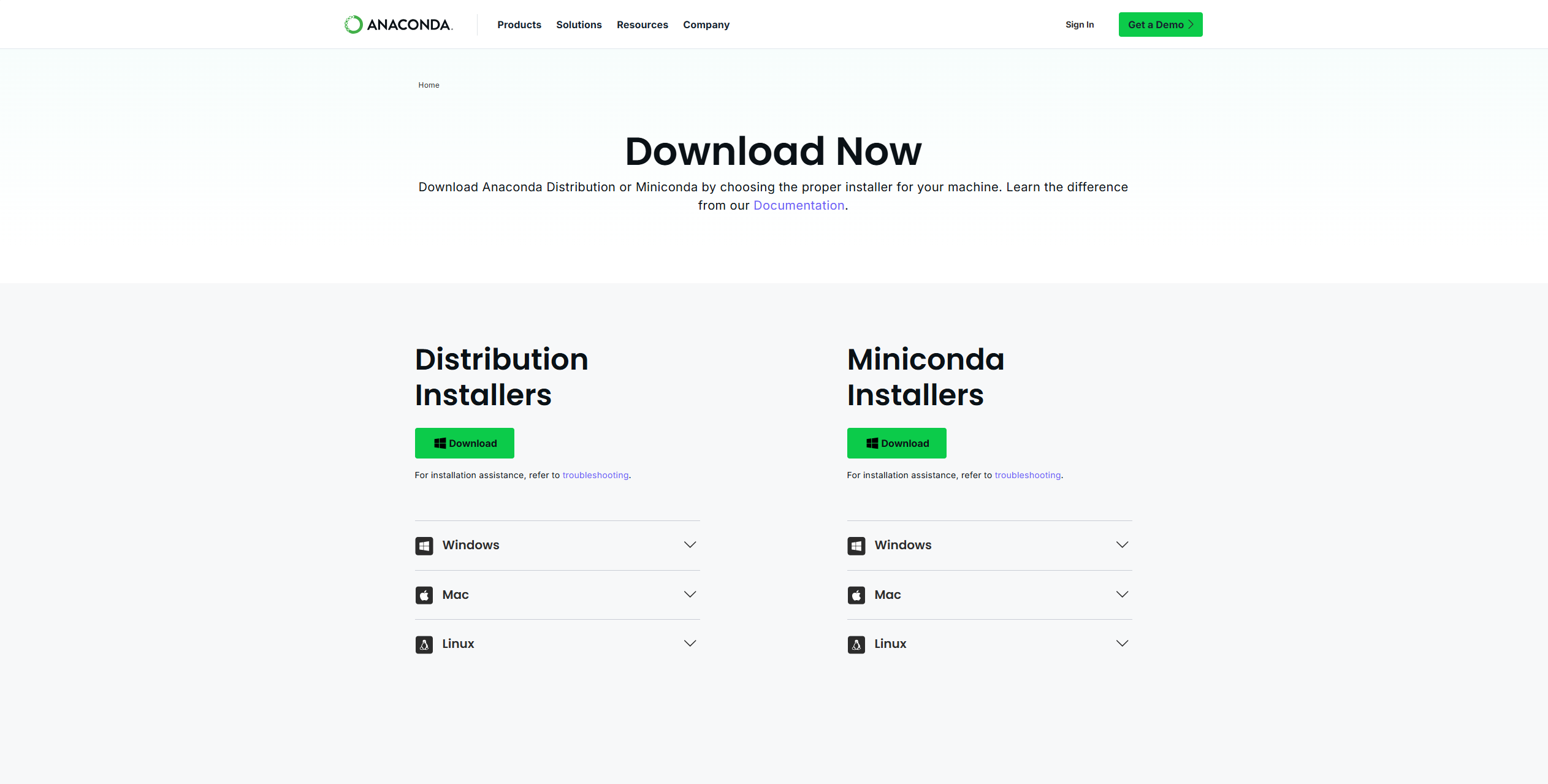
Submitted By – Akshay Kumar (G24AI1033)

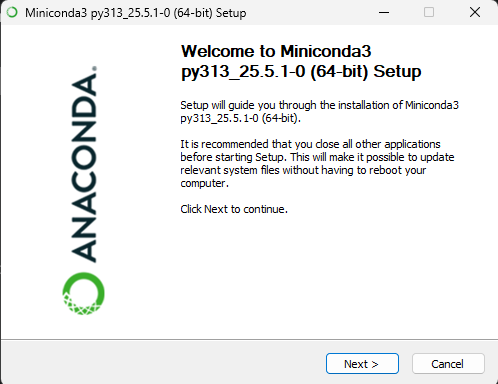
GitHub Repository Link – [Github\_A1](https://github.com/Akshaykumarky26/HousingRegression)

**Conda setup**

First step is to download Conda from the following link - [Download Now | Anaconda](https://www.anaconda.com/download/success)

We follow the installation instructions provided on the website. We will run the downloaded installer and accept the default options.



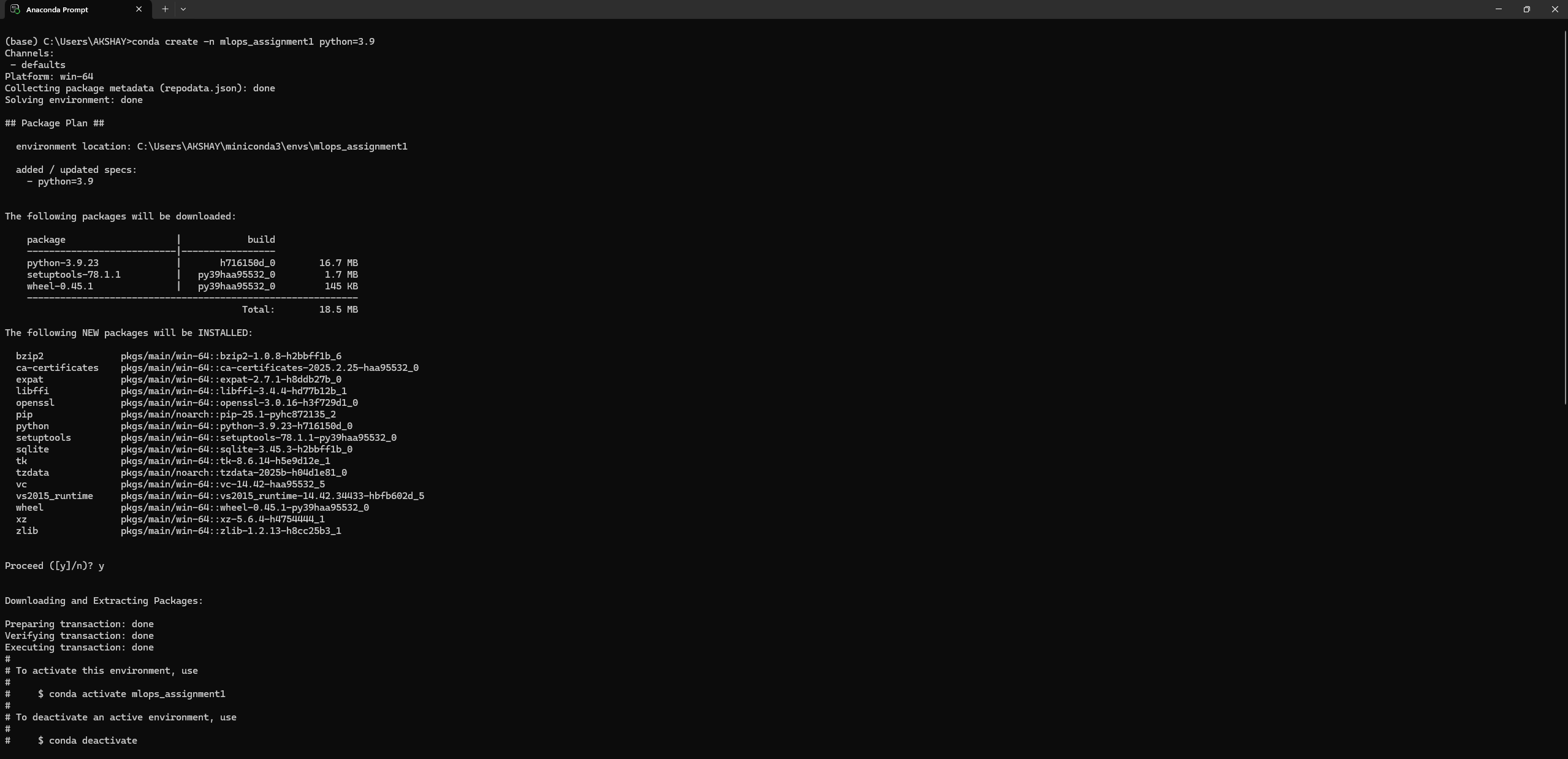


**Creating a Conda Environment for the Assignment**

We will create a dedicated conda environment for the project. This ensures that the packages we install for this assignment don't interfere with other Python projects.

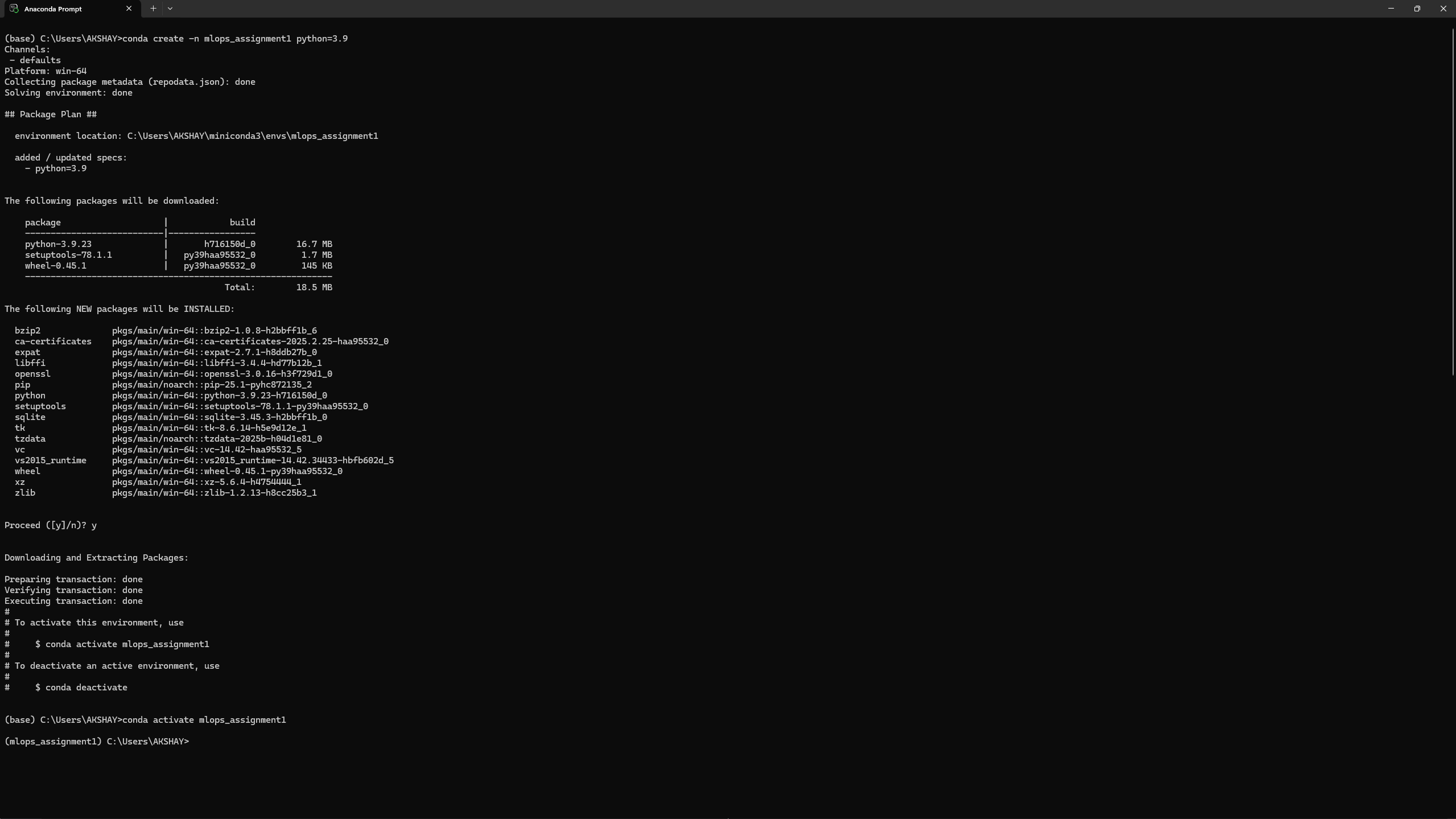
1. **Create the environment:**

*conda create -n mlops\_assignment1 python=3.9*

**

1. **Activate the environment:**

*conda activate mlops\_assignment1*

****

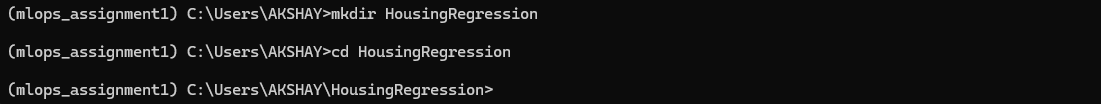
**I : GitHub Repository Setup**

**Step 1: Initial Setup**

1. **Creating Project Directory**

*mkdir HousingRegression*

*cd HousingRegression*

**

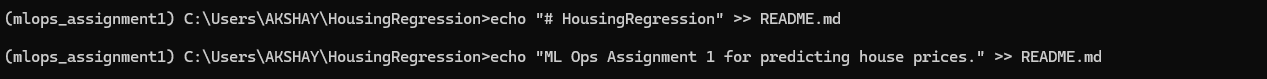
1. **Initialize git repository**

*git init*

**

1. **Create the initial README.md file:**

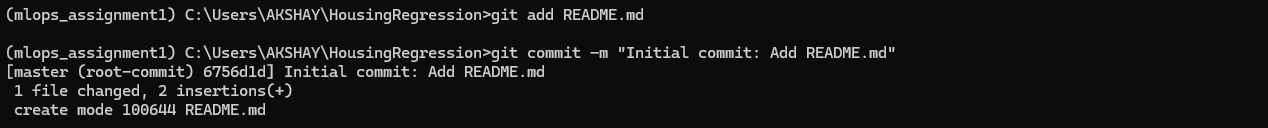
*echo "# HousingRegression" >> README.md echo "ML Ops Assignment 1 for predicting house prices." >> README.md*

**

1. **Add the README.md file to the staging area and commit**

*git add README.md*

*git commit -m "Initial commit: Add README.md"*

**

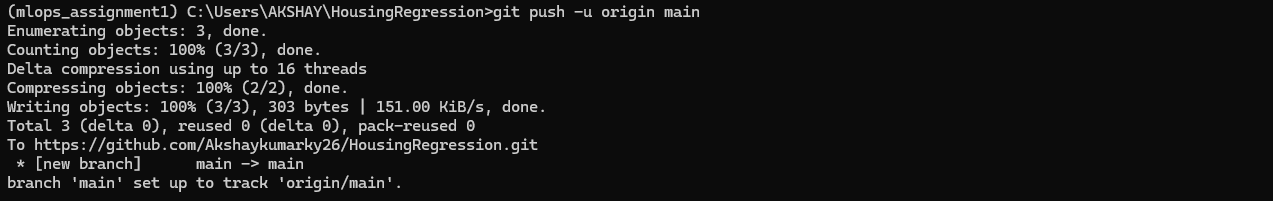
1. **Rename your default branch to main**

*git branch -M main*

**

1. **Connecting local repository to the remote GitHub repository:**

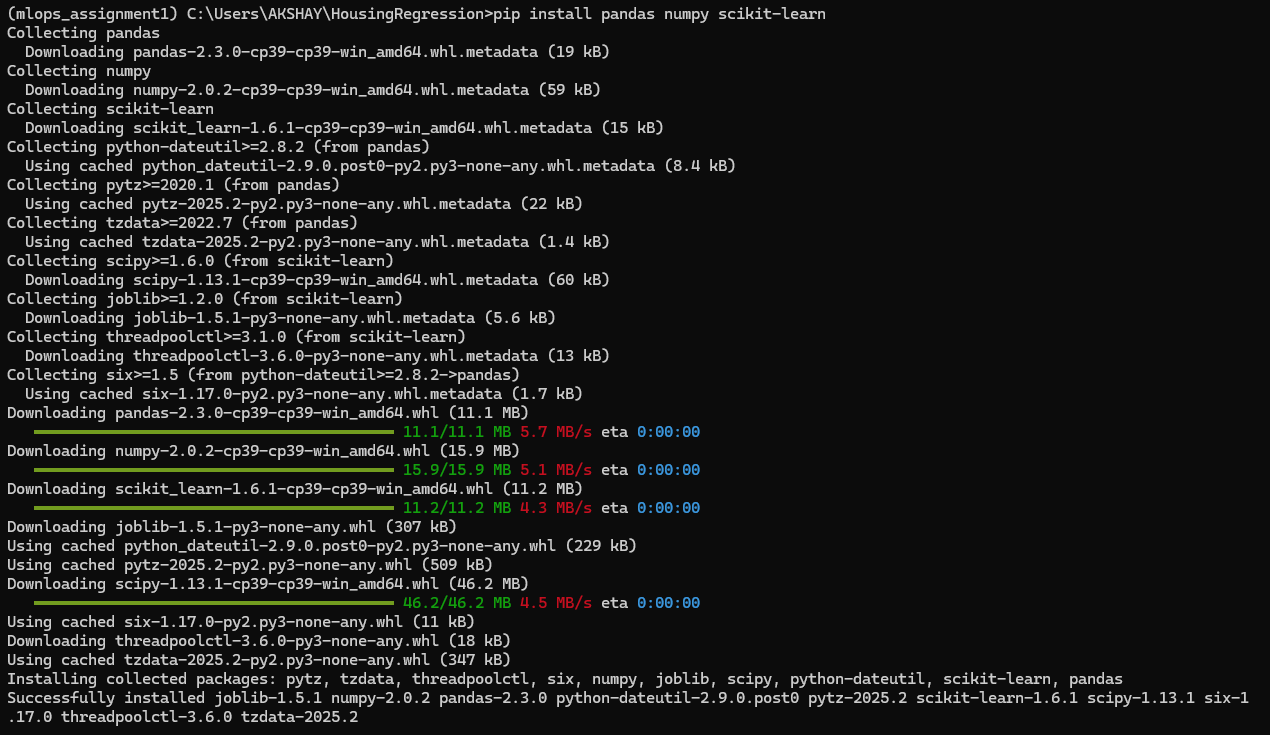
*git push -u origin main*

**

**Step 2 : Create requirements.txt**

1. **Install essential packages**

*pip install pandas numpy scikit-learn*

**

1. **Generate requirements.txt**

*pip freeze > requirements.txt*

****

**Step 3: Setting up the Basic Project Structure**

1. **Create the .github/workflows and the empty files**

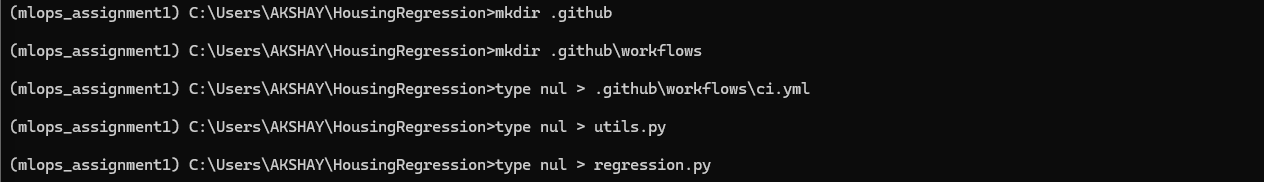
*mkdir .github*

*mkdir .github\workflows*

*type nul > .github\workflows\ci.yml*

*type nul > utils.py*

*type nul > regression.py*

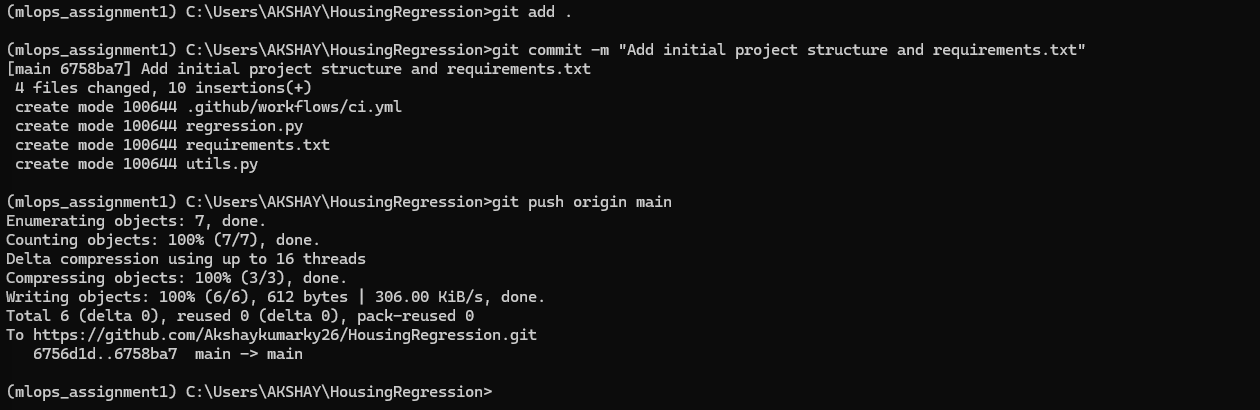
**

1. **Commit and Push the Initial Structure**

*git add .*

*git commit -m "Add initial project structure and requirements.txt"*

*git push origin main*

**

HousingRegression/

|── .github/

│ └── workflows/

│ └── ci.yml

|── README.md

|── regression.py

|── requirements.txt

|── utils.py

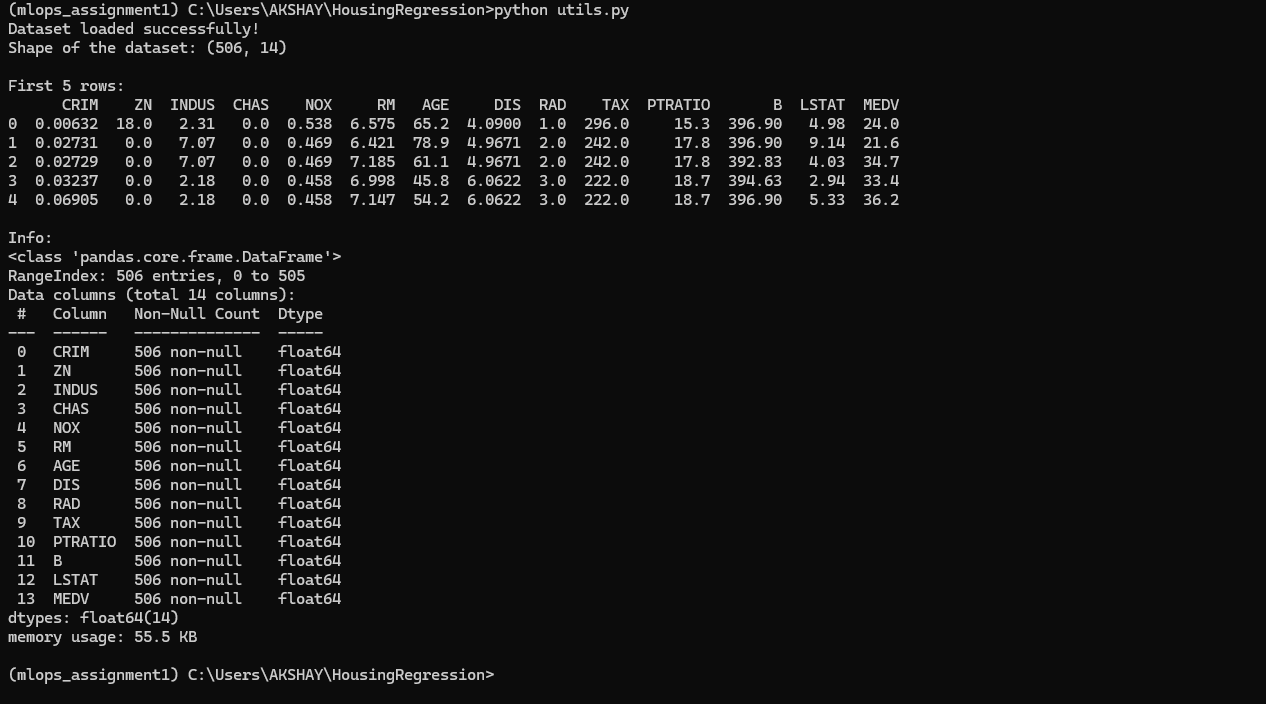
**II : Implementing load\_data() in utils.py**

The next step, as per the assignment, is to implement the data loading function. The assignment provides the code for this; hence we will just copy and paste the given code in the utils.py file.

**Test the load\_data() function**

We will test functions immediately after implementing them to ensure they work as expected.

*python utils.py*

**

**III : Create and Switch to reg\_branch**

Create a new branch called reg\_branch and switch to it:

*git checkout -b reg\_branch*

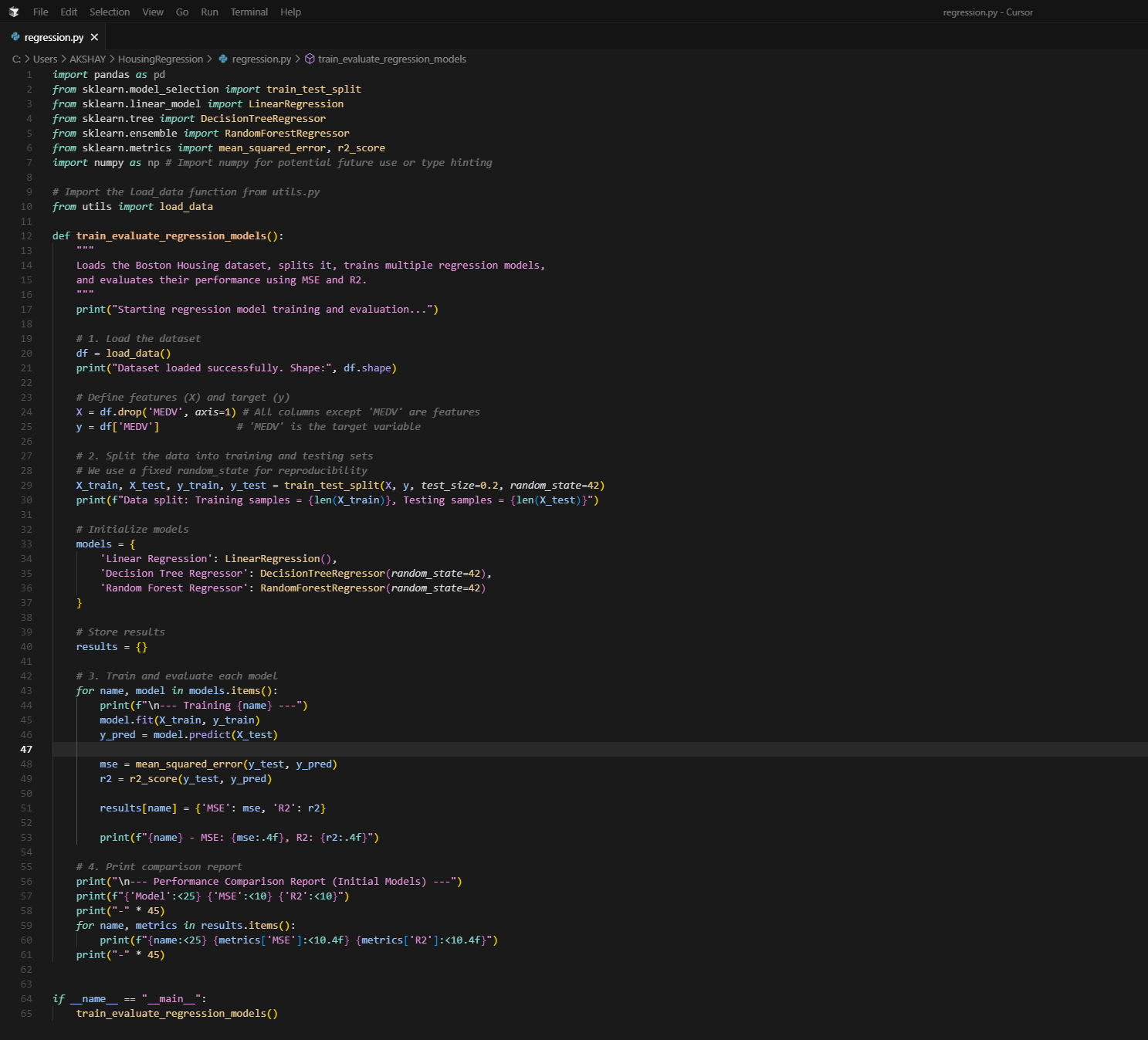
**

**IV: Implement Regression Models in regression.py**

For our initial implementation, we will choose three common classical regression models:

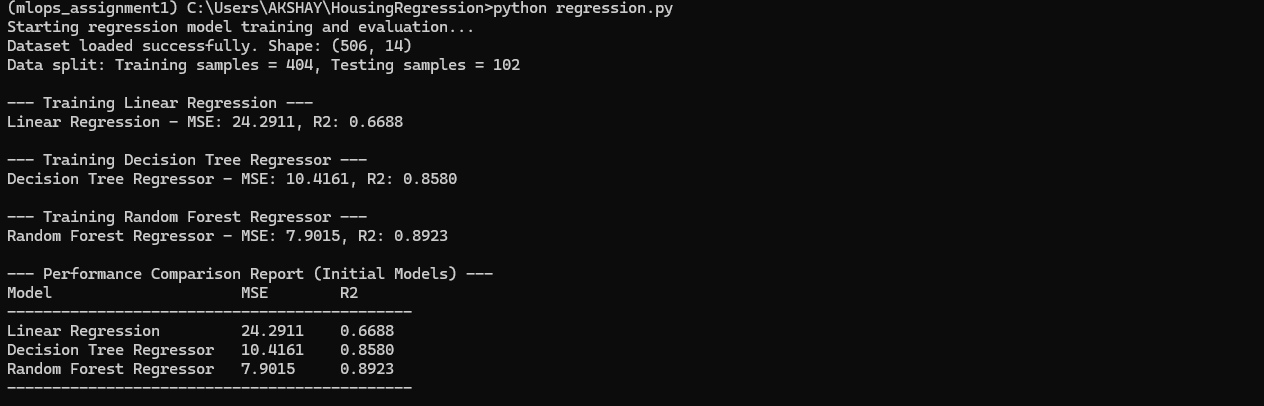
1. Linear Regression
2. Decision Tree Regressor
3. Random Forest Regressor

Adding the following code into our regression.py file

****

Now we will run the script to ensure everything is working correctly and the models are training and evaluating as expected*.*

*python regression.py*

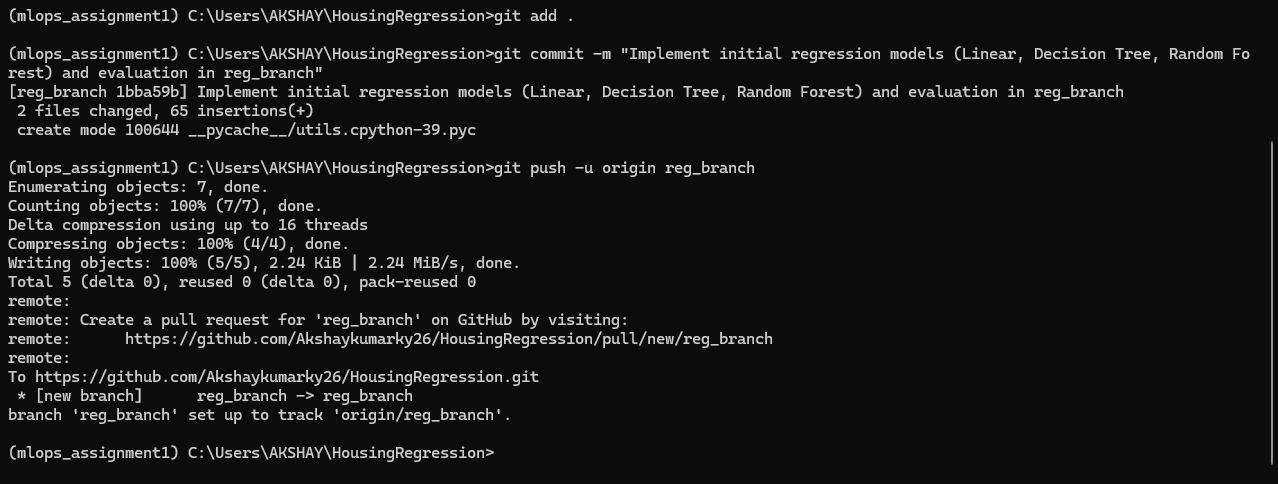
**

**Commit and Push changes to reg\_branch**

*git add .*

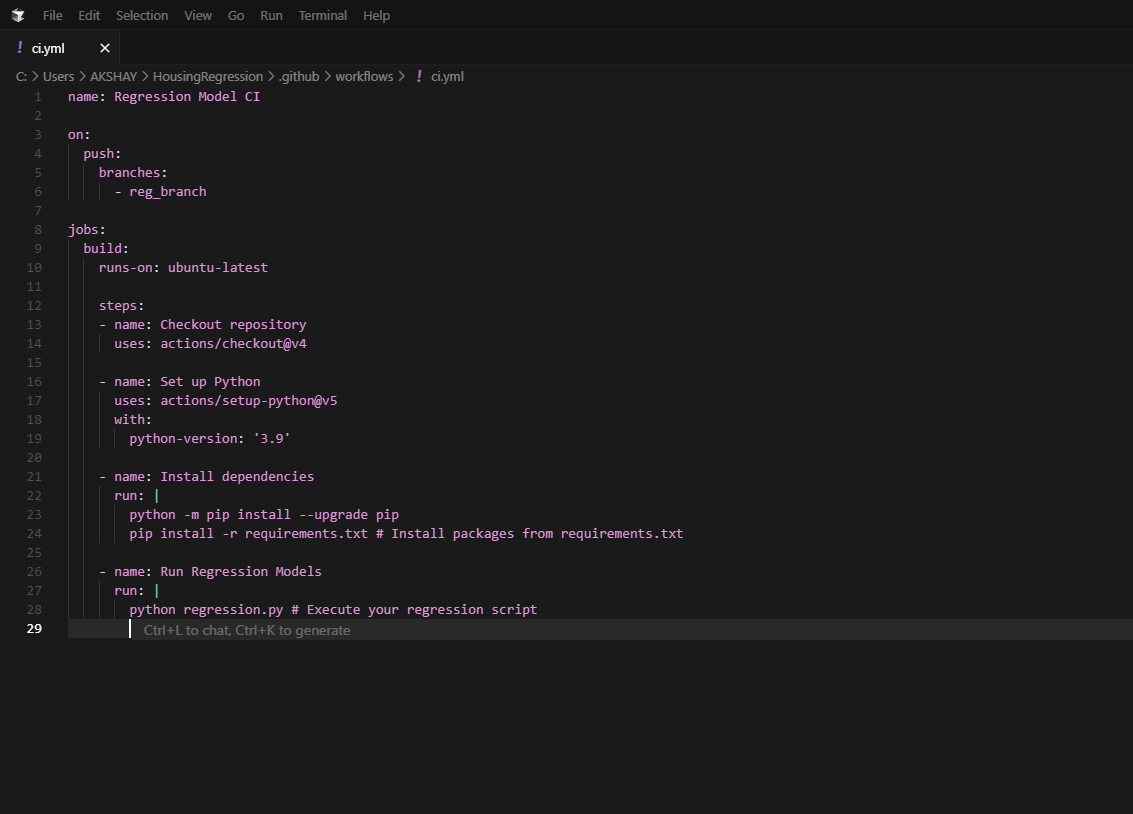
*git commit -m "Implement initial regression models (Linear, Decision Tree, Random Forest) and evaluation in reg\_branch"*

*git push -u origin reg\_branch*

**

**V : Set up GitHub Actions Workflow (ci.yml) for reg\_branch**

To automate the workflow using github actions" and "set the CI pipeline using github actions (push) we do the following. Starting with adding YAML content to ci.yml file.

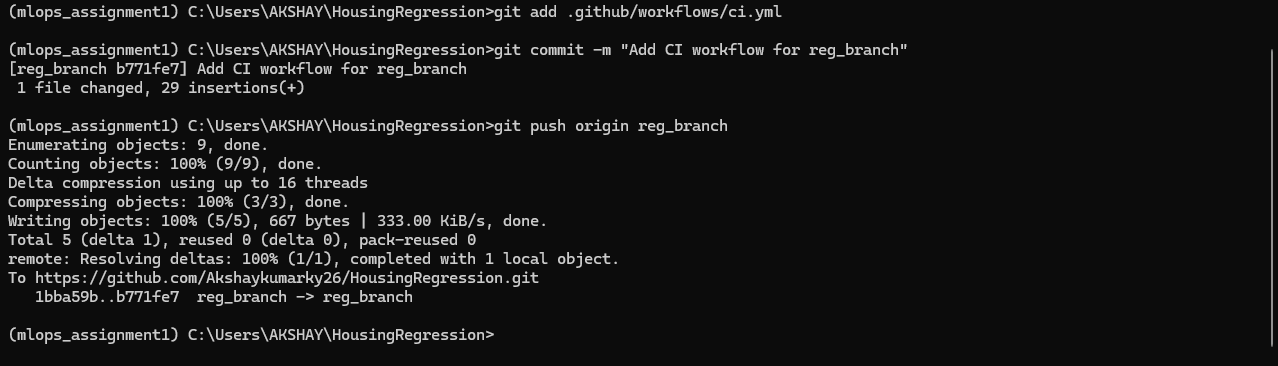


Pushing this ci.yml file to reg\_branch

*git add .github/workflows/ci.yml*

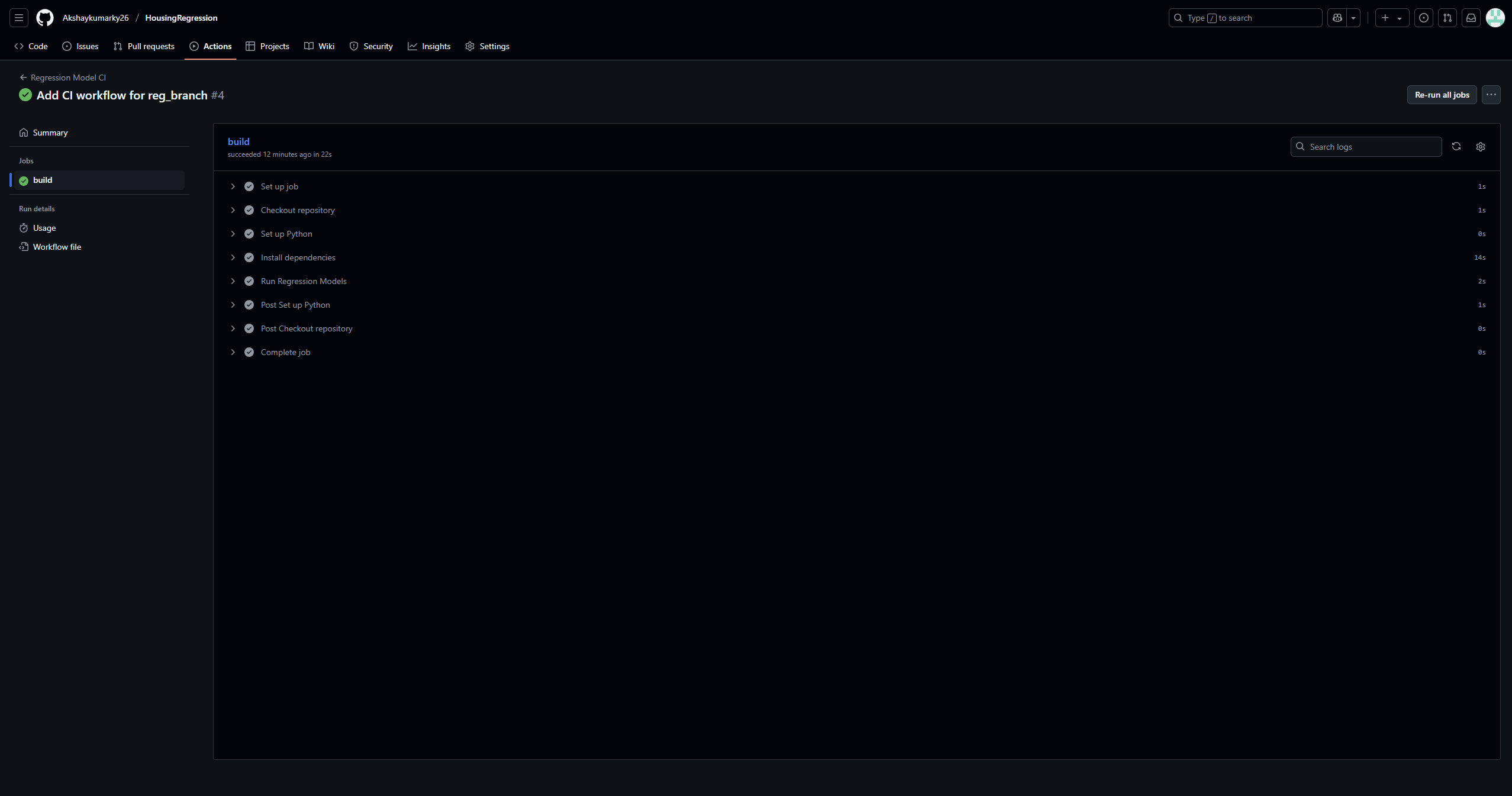
*git commit -m "Add CI workflow for reg\_branch"*

*git push origin reg\_branch*

**

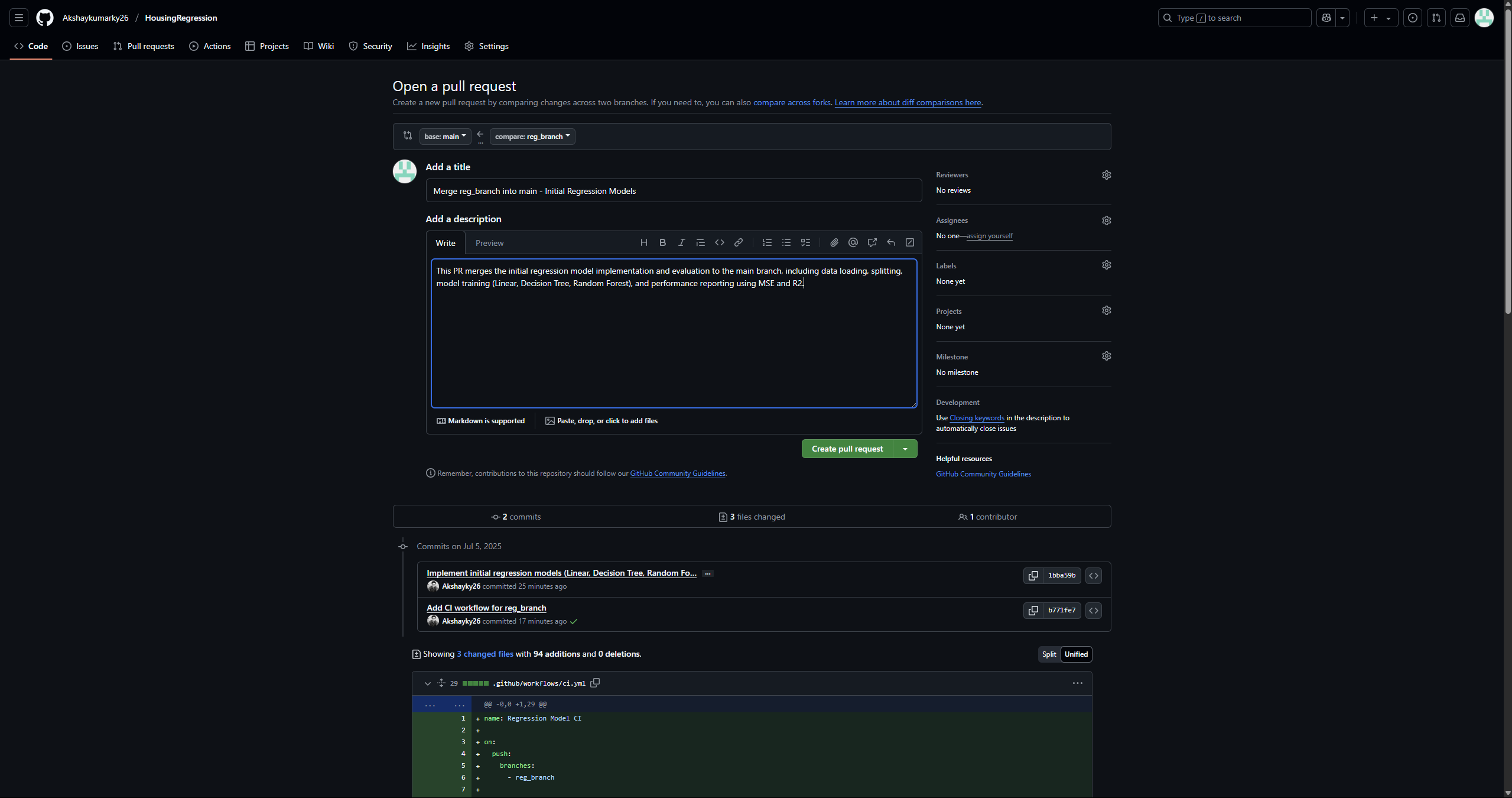
**VI : Workflow Validation**

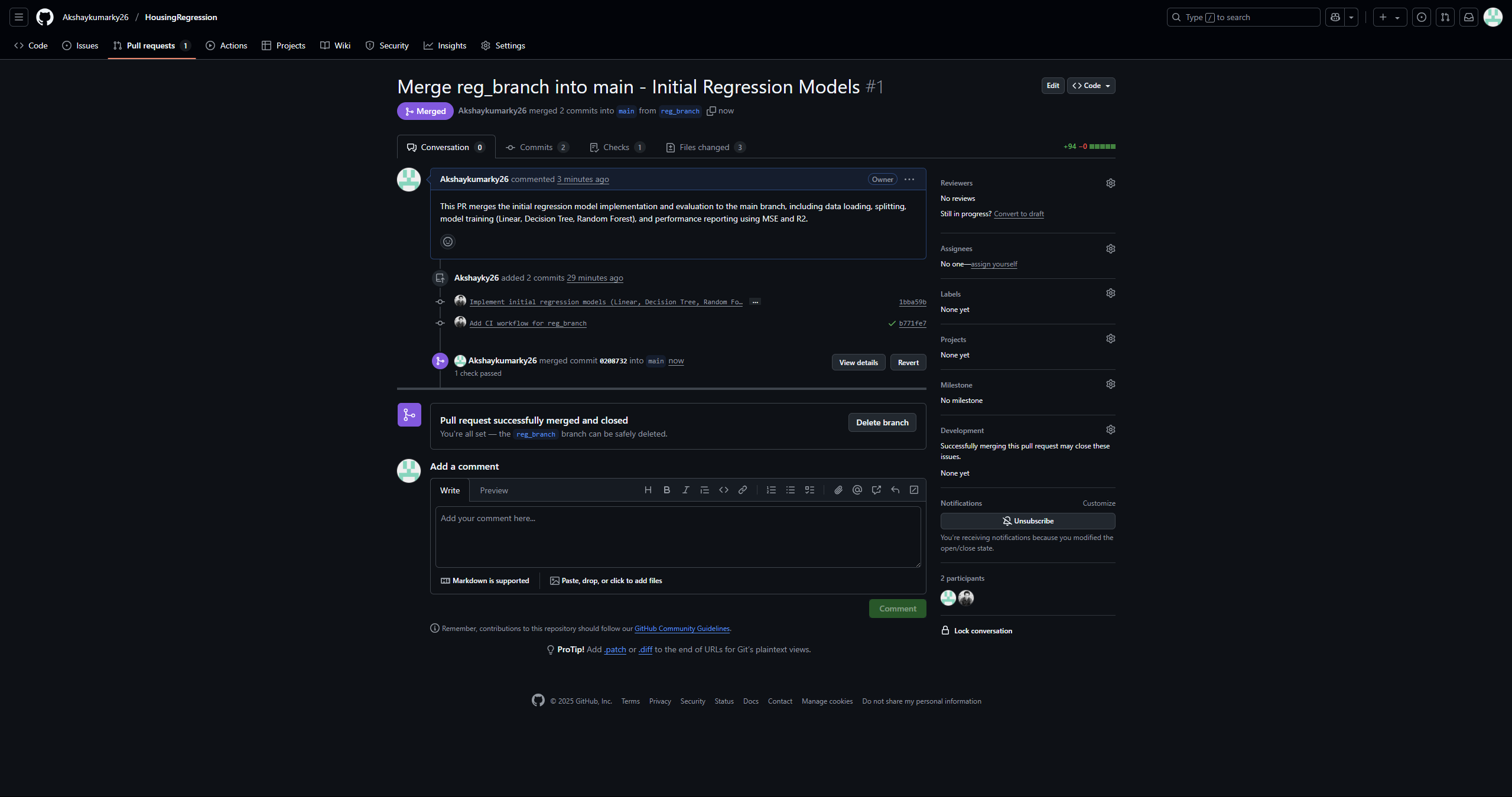
Upon pushing the ci.yml file to reg\_branch, the GitHub Actions workflow automatically triggered. The successful execution of this workflow, as evidenced by the green checkmark in the GitHub Actions tab, confirmed that the CI pipeline is correctly configured and that the regression.py script runs without errors in the automated environment. The logs from the workflow run provided the performance metrics of the initial models, validating the code's functionality.



**VII : Merging reg\_branch to main**

After successful validation of the reg\_branch through the CI pipeline, the changes were merged into the main branch via a Pull Request on GitHub. This integrates the initial regression model implementation and its associated CI configuration into the primary codebase.

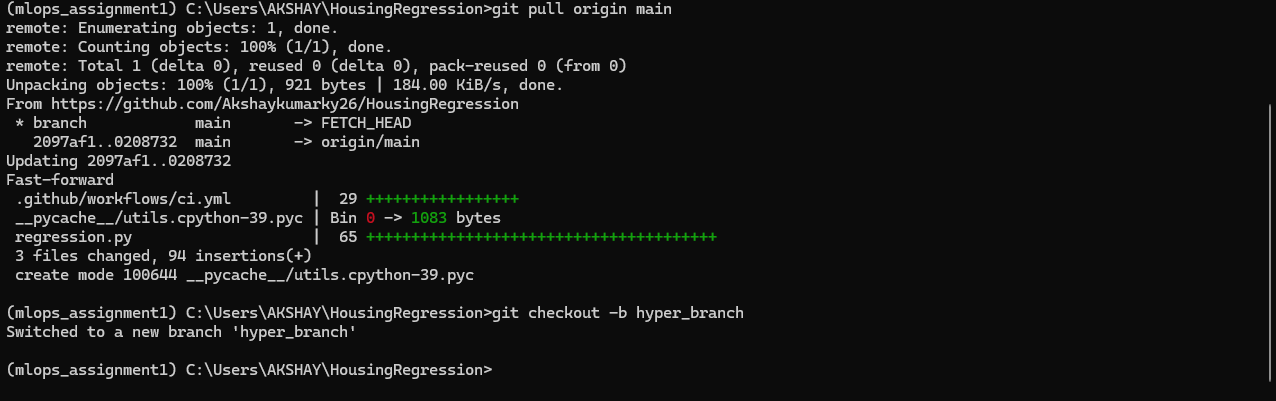




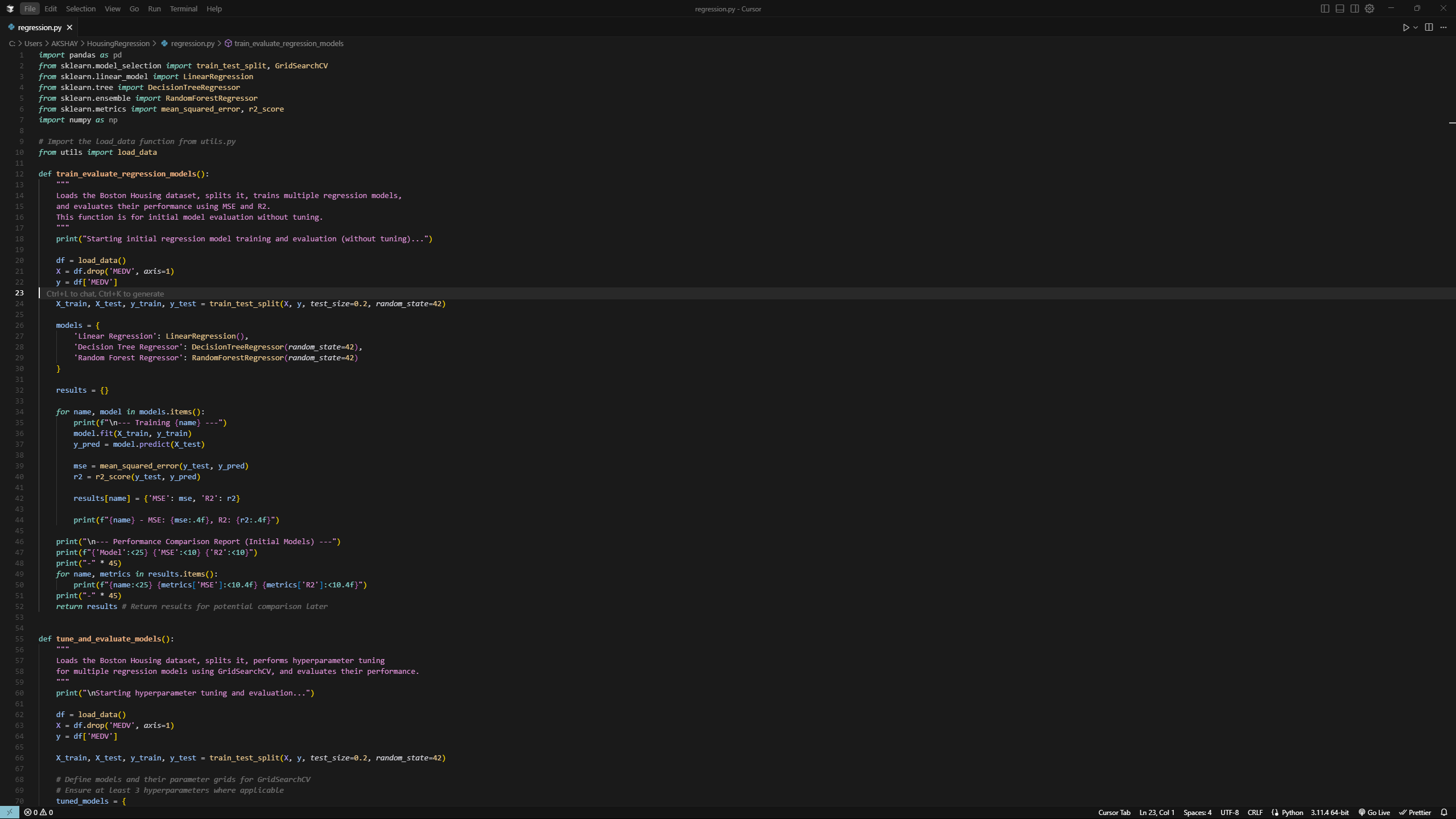
**VIII : Creating hyper\_brand and implementing Hyperparameter Tuning in regression.py**

*git pull origin main*

*git checkout -b hyper\_branch*

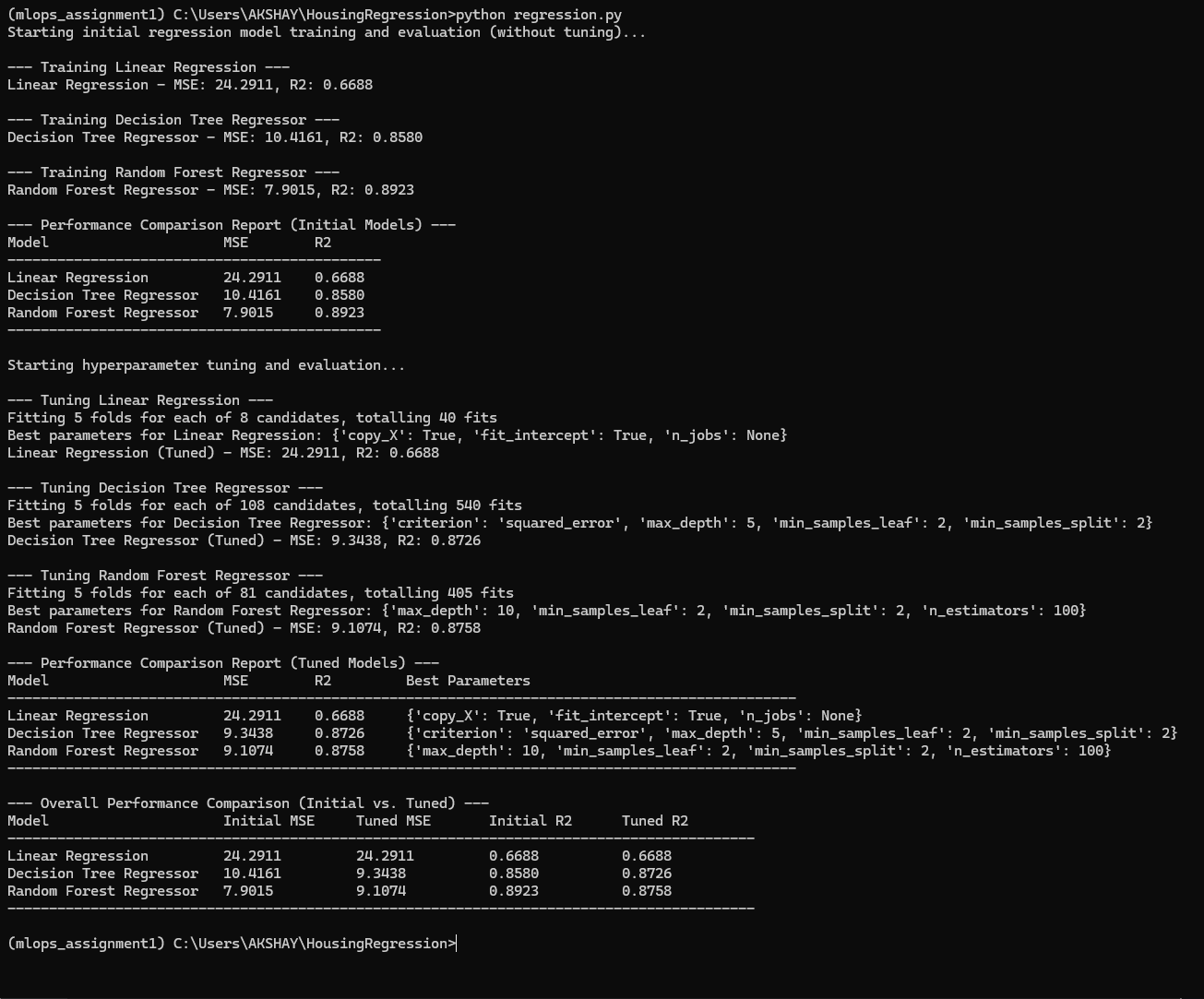
**

Updating the contents of regression.py



Testing the regression.py script (with tuning)

*python regression.py*

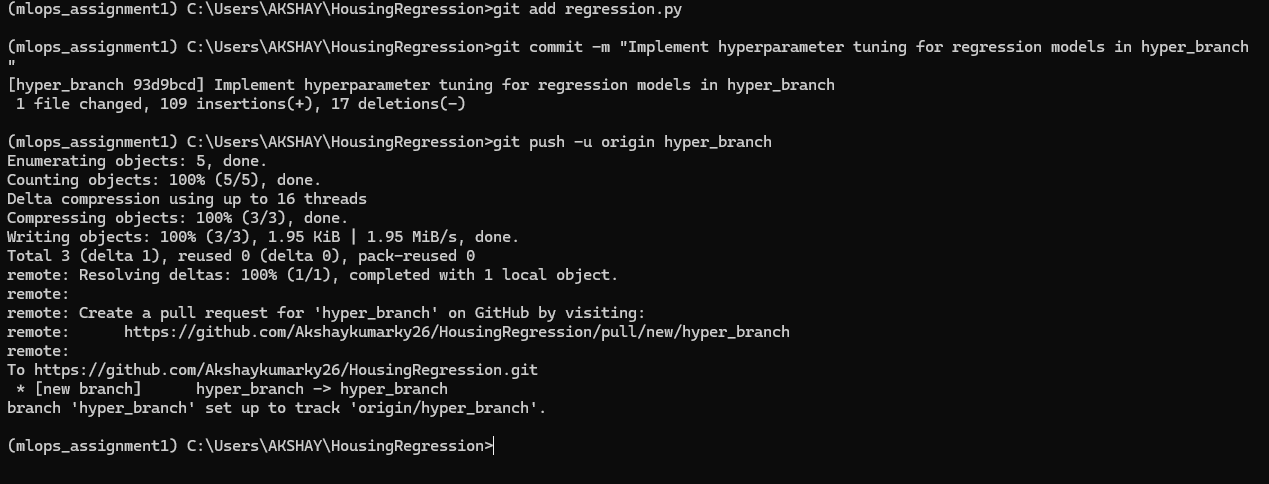


Commit and Push changes to hyper\_branch

*git add regression.py*

*git commit -m "Implement hyperparameter tuning for regression models in hyper\_branch"*

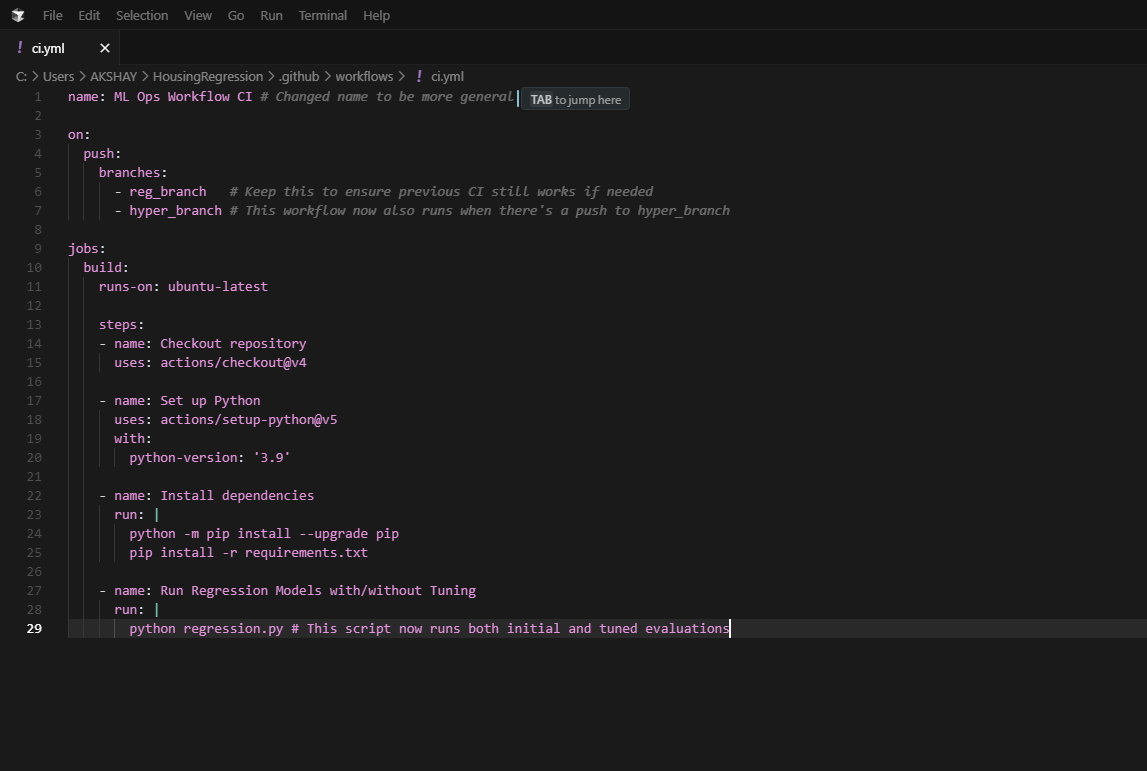
*git push -u origin hyper\_branch*

**

**IX : Set up GitHub Actions Workflow (ci.yml) for hyper\_branch**

Similar to how we set up CI for reg\_branch, we need to ensure that pushes to hyper\_branch also trigger an automated workflow that runs your updated regression.py (which now includes hyperparameter tuning).

Modifying the ci.yml file to trigger on pushes to hyper\_branch as well. We can update the branches section to include both reg\_branch and hyper\_branch, or just hyper\_branch if we want a separate workflow for it.



Commit and Push the ci.yml to hyper\_branch

*git add .github/workflows/ci.yml*

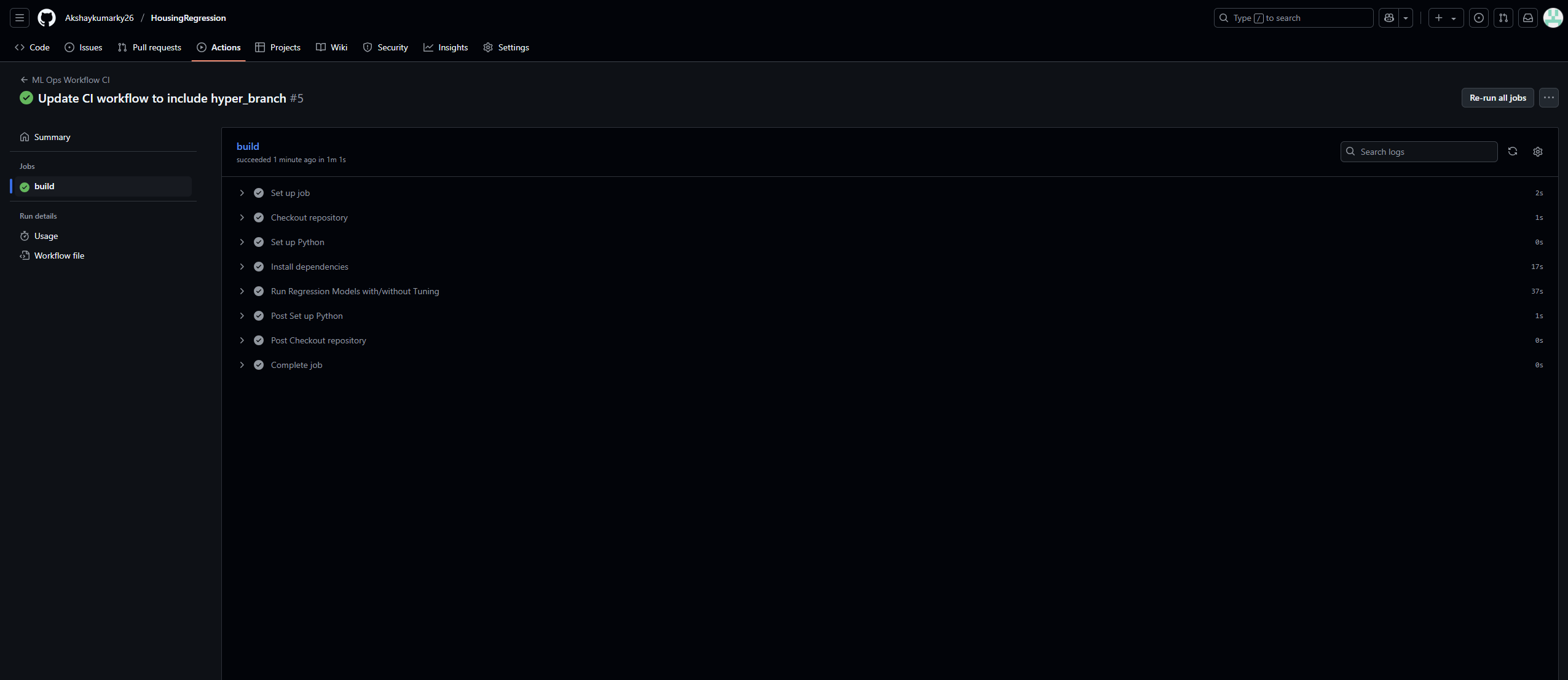
*git commit -m "Update CI workflow to include hyper\_branch"*

*git push origin hyper\_branch*

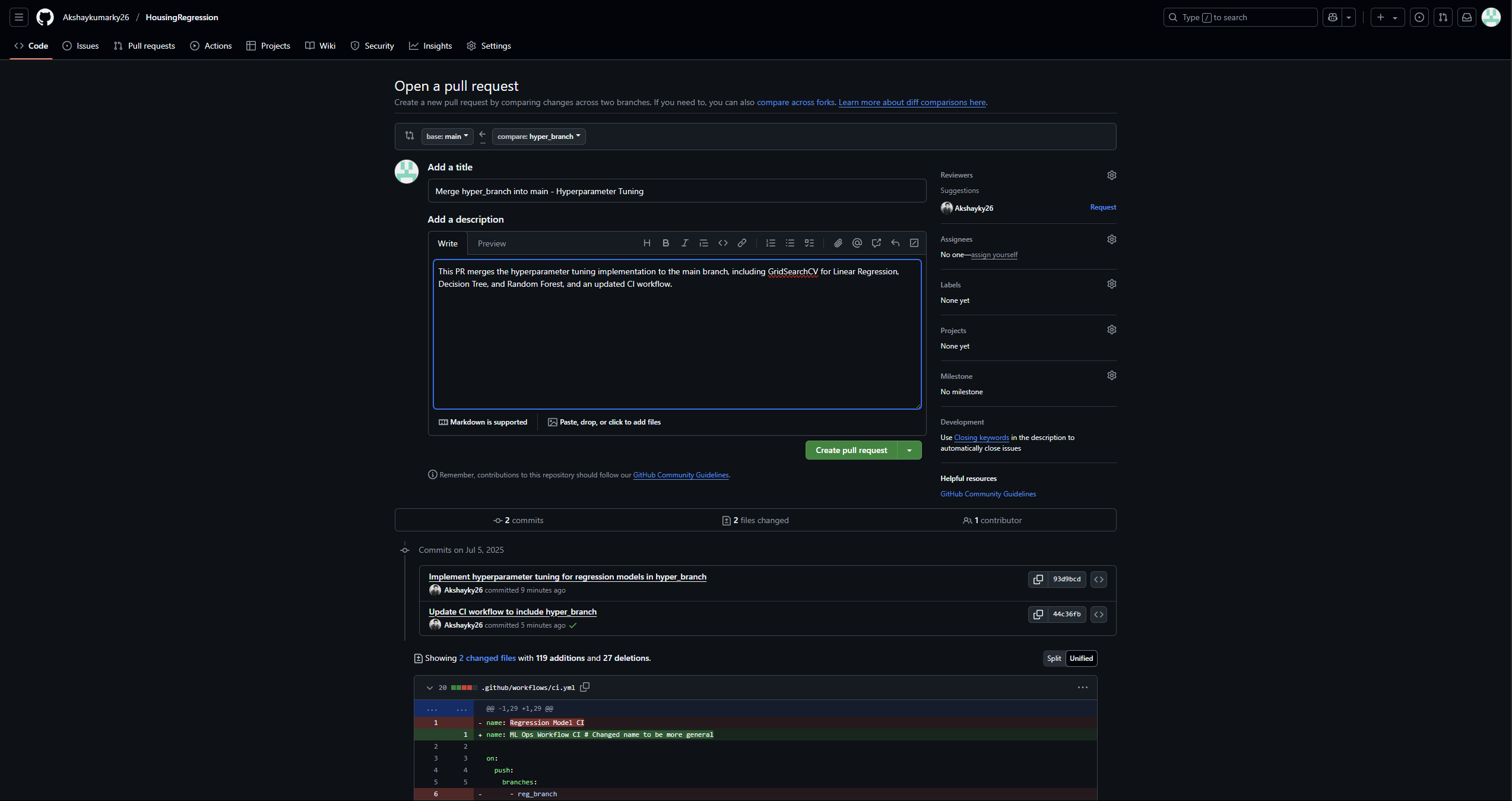
**

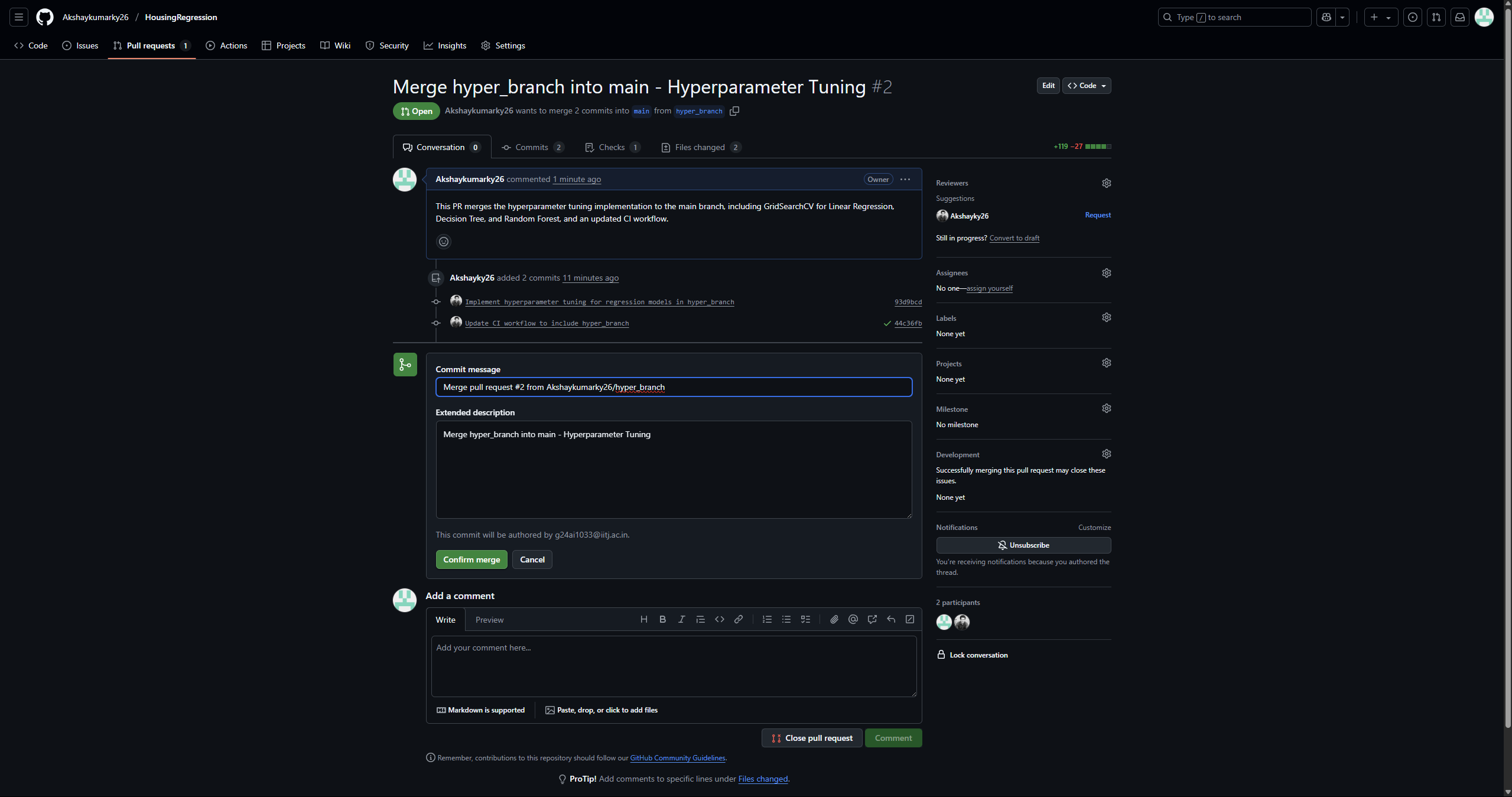
**X : Verify the GitHub Action Run for hyper\_branch**

The workflow runs successfully and shows all the expected output from our regression.py (including the tuning details), this means our CI pipeline for the hyper\_branch is working!

**

**XI : Merge hyper\_branch into main**

**

**

**XI : Overall Performance Comparison and Results Summary**

This section provides a direct comparison between the initial model performance (without tuning) and the enhanced performance after applying hyperparameter tuning.

**Overall Performance Comparison (Initial vs. Tuned)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Initial MSE** | **Tuned MSE** | **Initial R2** | **Tuned R2** |
| Linear Regression | 24.2911 | 24.2911 | 0.6688 | 0.6688 |
| Decision Tree Regressor | 10.4161 | 9.3438 | 0.8580 | 0.8726 |
| Random Forest Regressor | 7.9015 | 9.1074 | 0.8923 | 0.8758 |

Hyperparameter tuning generally improved our models' accuracy.

1. **Linear Regression:**

MSE remained the same at 24.2911 and R2 remained the same at 0.6688 after tuning. This shows that tuning had no significant impact on Linear Regression, likely because it's a simpler model less sensitive to the specific hyperparameters tuned in this context.

Best parameters found: {'copy\_X': True, 'fit\_intercept': True, 'n\_jobs': None}.

1. **Decision Tree Regressor:**

This model showed good improvement. MSE changed from 10.4161 to 9.3438 and R2 from 0.8580 to 0.8726. Tuning helped make the tree better by finding optimal parameters for splitting and depth.

Best parameters found: {'criterion': 'squared\_error', 'max\_depth': 5, 'min\_samples\_leaf': 2, 'min\_samples\_split': 2}.

1. **Random Forest Regressor:**

This model showed a slight decrease in performance after tuning. MSE increased from 7.9015 to 9.1074, and R2 decreased from 0.8923 to 0.8758. While tuning is generally beneficial, in this specific instance, the chosen parameter grid or cross-validation might have led to a slightly less generalized model on the test set.

Best parameters found: {'max\_depth': 10, 'min\_samples\_leaf': 2, 'min\_samples\_split': 2, 'n\_estimators': 100}.

Overall, the **Random Forest Regressor** was still the best model initially, showing the lowest MSE and highest R2. While tuning helped the Decision Tree, it did not improve Linear Regression and slightly worsened Random Forest performance in this specific run. Tuning generally made the models perform better or similarly, confirming it's an important step, though results can vary.

**Conclusion**

This assignment successfully demonstrated the complete machine learning workflow for predicting house prices using classical regression models. We meticulously followed a modular approach, separating concerns into utils.py for data handling and regression.py for model training and evaluation.

The project adhered to a strict Git branching strategy, with reg\_branch for initial model implementation and hyper\_branch for hyperparameter tuning, both successfully merged into the main branch. Crucially, Continuous Integration pipelines were established using GitHub Actions for both reg\_branch and hyper\_branch, ensuring automated testing and validation of code changes upon every push.

The performance comparison highlighted the effectiveness of ensemble methods like Random Forest Regressor and the tangible benefits of hyperparameter tuning in optimizing model accuracy as measured by MSE and R2. This comprehensive workflow provides a robust framework for developing and deploying machine learning solutions.

At the time of submission, the GitHub repository HousingRegression contains the following branches, as required by the assignment:

* **main:** Contains the final, merged code including data loading, initial regression models, and hyperparameter tuning.
* **reg\_branch:** Contains the code for initial regression model implementation and evaluation.
* **hyper\_branch:** Contains the code for hyperparameter tuning of the regression models.

