Ml OPS

House Price Prediction Assignment -1

Submitted By: Purushothaman S

Roll Number: G24AI1042

Email Id: [g24ai1042@iitj.ac.in](mailto:g24ai1042@iitj.ac.in)

Course: PGD and MTech – Trimester 3

Indian Institute of Technology Jodhpur – Rajasthan

Date: July 1st 2025

Submitted to: Dr. Prateek Mazumdar

Department: School of AIDE

Indian Institute of Technology Jodhpur

Git Link:

**Index**

**Introduction……………………………………………………………………………………………….02**

**Goal on this Assignment……………………………………………………………………………..02**

**Project Exploration……………………………………………………………………………………..02**

**Project Implementation – Step by Step……………………………………………………….03**

**File Structure………………………………………………………………………………………….03**

**Dataset Preparation……………………………………………………………………………….03**

**Data Preprocessing…………………………………………………………………………………03**

**Model Development………………………………………………………………………………03**

**Hyperparameter Tuning………………………………………………………………………….04**

**Automation with GitHub Actions……………………………………………………………04**

**Git Workflow………………………………………………………………………………………….04**

**Implementation and Execution…………………………………………………………………..04**

**Conclusion………………………………………………………………………………………………….14**

**Introduction**

Housing price prediction is one of the most popular applications of machine learning in the real estate and finance sectors. By analysing historical housing data such as location, number of rooms, age of the property, proximity to amenities, and crime rates machine learning models can learn patterns that help estimate the market value of a home.

This process involves using regression algorithms, where the goal is to predict a continuous value (the house price) based on several input features. Traditional statistical models like Linear Regression are often used for their simplicity and interpretability, while more complex models like Random Forests or Gradient Boosting can capture nonlinear relationships for better accuracy.

Machine learning brings scalability, automation, and improved precision to housing price estimation, which is valuable for buyers, sellers, banks, and policymakers. It reduces human bias, adapts to market changes, and supports data-driven decision-making in real estate

**Goal on this Assignment**

The main goal of this assignment was to experience what it is like to build a real-world machine learning workflow from scratch. Instead of just training a model, we focused on the entire process — loading and preparing data, building multiple regression models, evaluating their performance, and tuning them to improve results. We also organized our code properly, used version control with Git, and automated our workflow using GitHub Actions. This helped us understand how machine learning projects are managed in production environments, where collaboration, repeatability, and automation are just as important as the model itself

**Project Exploration**

This project was an opportunity to apply machine learning concepts to a practical problem predicting house prices. We began by manually loading the Boston Housing dataset, since it is no longer available directly in scikit-learn. This dataset includes various features that influence housing prices, such as crime rate, number of rooms, and distance to employment centres.

We explored two main stages in our project:

* In the **first stage** (reg branch), we implemented and compared three regression models — Linear Regression, Decision Tree Regressor, and Random Forest Regressor. We used metrics like Mean Squared Error (MSE) and R² score to evaluate how well each model predicted house prices.
* In the **second stage** (hyper branch), we took the same models and used Grid Search CV to tune at least three hyperparameters for each. This helped us identify the best configuration for each model and improve their performance.

We maintained a clean and modular codebase using separate files for utility functions and model training. Additionally, we automated the entire workflow using GitHub Actions, so that every code update was automatically tested. Throughout the project, we followed good ML Ops practices like version control, branching strategies, and CI/CD

**Project Implementation – Step by Step**

**File Structure**

***HousingRegression/  
├── .github/workflows/ci.yml # CI pipeline for automation  
├── utils.py # Data loading and preprocessing  
├── regression.py # Model training, evaluation, tuning  
├── requirements.txt  
└── README.md***

**Step 1: Dataset Preparation**

We began by manually loading the Boston Housing dataset from an external source, since it has been removed from scikit-learn due to ethical concerns. Using raw text data, we parsed the dataset into a clean Pandas DataFrame, defined 13 features, and set the target variable (MEDV) as the median house value.

**Step 2: Data Preprocessing**

The dataset was then split into training and testing sets. We used StandardScaler to normalize the feature values, which is especially important for models like Ridge Regression to perform optimally.

**Step 3: Model Development (Regression Branch)**

In the reg branch, we implemented three classical regression models:

* Linear Regression
* Decision Tree Regressor
* Random Forest Regressor

Each model was trained on the training data and evaluated on the test set using two metrics:

* **Mean Squared Error (MSE)** – to measure the average squared difference between actual and predicted values.
* **R² Score** – to measure how well the model explains the variance in the data.

**Step 4: Hyperparameter Tuning (Hyper Branch)**

In the hyper branch, we introduced hyperparameter tuning using GridSearchCV for each of the three models. We selected at least three key hyperparameters per model and searched for the best combinations using 5-fold cross-validation. The tuned models generally performed better than their default versions.

**Step 5: Code Organization**

We followed a modular approach to keep the codebase clean and maintainable:

* utils.py: for data loading and preprocessing
* regression.py: for training, evaluation, and hyperparameter tuning

This separation made the code reusable and easier to debug or extend.

**Step 6: Automation with GitHub Actions**

We set up a continuous integration (CI) pipeline using GitHub Actions. Every time we pushed changes to any branch, the workflow:

* Installed dependencies from requirements.txt
* Executed the regression.py script  
  This helped us automatically validate that the pipeline worked correctly with each update.

**Step 7: Git Workflow**

We strictly followed a Git-based branching model:

* Started with a clean main branch (only README)
* Implemented the base regression models in reg branch and merged into main
* Developed hyperparameter tuning in hyper branch and then merged that into main

This approach simulated a real-world software development workflow using version control.

**Implementation and Execution**

1. utils.py – Data Loading & Preprocessing

This file handles all the data-related tasks:

* **load\_data():** Fetches the Boston Housing dataset from an external URL, restructures it, and creates a DataFrame.
* **preprocess\_data():** Splits the dataset into training and testing sets and applies standard scaling to the features.

**Script**

**import pandas as pd**

**import numpy as np**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**def load\_data():**

**"""**

**Load the Boston Housing dataset from the URL manually.**

**Returns a pandas DataFrame with features and target.**

**"""**

**data\_url = "http://lib.stat.cmu.edu/datasets/boston"**

**raw\_df = pd.read\_csv(data\_url, sep="\s+", skiprows=22, header=None)**

**# Combine the features spread across two rows**

**data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])**

**target = raw\_df.values[1::2, 2]**

**feature\_names = [**

**'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',**

**'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'**

**]**

**df = pd.DataFrame(data, columns=feature\_names)**

**df['MEDV'] = target  # target variable**

**return df**

**def preprocess\_data(df, test\_size=0.2, random\_state=42):**

**"""**

**Splits the data into train/test and applies standard scaling.**

**Returns: X\_train, X\_test, y\_train, y\_test**

**"""**

**X = df.drop(columns='MEDV')**

**y = df['MEDV']**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,**

**test\_size=test\_size,**

**random\_state=random\_state)**

**scaler = StandardScaler()**

**X\_train\_scaled = scaler.fit\_transform(X\_train)**

**X\_test\_scaled = scaler.transform(X\_test)**

**return X\_train\_scaled, X\_test\_scaled, y\_train, y\_test**

**regression.py – Model Training and Evaluation**

This file is the main engine of the project. It changes slightly across branches:

* **In the reg branch**:
  + Initializes three regression models (Linear, Decision Tree, Random Forest).
  + Trains each model and prints its performance (MSE and R²).
* **In the hyper branch**:
  + Uses GridSearchCV to tune hyperparameters for each model.
  + Prints the best parameters along with the final evaluation metrics.

**Scrpit**

# regression.py (hyper branch)

from utils import load\_data, preprocess\_data

from sklearn.linear\_model import Ridge

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.model\_selection import GridSearchCV

def evaluate\_model(model, X\_test, y\_test):

    """

    Evaluate model on test data.

    """

    y\_pred = model.predict(X\_test)

    mse = mean\_squared\_error(y\_test, y\_pred)

    r2 = r2\_score(y\_test, y\_pred)

    return mse, r2

def main():

    # Load and preprocess data

    df = load\_data()

    X\_train, X\_test, y\_train, y\_test = preprocess\_data(df)

    # Model definitions with hyperparameter grids

    models = {

        "Ridge Regression": {

            "model": Ridge(),

            "params": {

                "alpha": [0.01, 0.1, 1.0, 10.0],

                "solver": ['auto', 'svd', 'cholesky'],

                "fit\_intercept": [True, False]

            }

        }, #Regression has done

        "Decision Tree": {

            "model": DecisionTreeRegressor(random\_state=42),

            "params": {

                "max\_depth": [None, 5, 10, 20],

                "min\_samples\_split": [2, 5, 10],

                "min\_samples\_leaf": [1, 2, 4]

            }

        }, # Decision Tree has done

        "Random Forest": {

            "model": RandomForestRegressor(random\_state=42),

            "params": {

                "n\_estimators": [50, 100],

                "max\_depth": [None, 10, 20],

                "min\_samples\_split": [2, 5]

            }

        }

    } # Random Forest has done

    # Run GridSearch for each model

    for name, mp in models.items():

        print(f"\n{name}")

        grid = GridSearchCV(mp["model"], mp["params"],

                            scoring="neg\_mean\_squared\_error", cv=5, n\_jobs=-1)

        grid.fit(X\_train, y\_train)

        best\_model = grid.best\_estimator\_

        mse, r2 = evaluate\_model(best\_model, X\_test, y\_test)

        print(f"Best Parameters: {grid.best\_params\_}")

        print(f"Test MSE = {mse:.4f}, R² = {r2:.4f}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Running Commands and Step by Step Execution

**Command 1:**

Before Execution of the command one, we have installed **Conda Env Activator** Extension in **VS Code**

Then Run this command on the terminal

**conda create -n housing-ml python=3.10 -y**

**The Out put**

**Retrieving notices: done**

**WARNING: A space was detected in your requested environment path:**

**'C:\Users\Purushothaman S\anaconda3\envs\housing-ml'**

**Spaces in paths can sometimes be problematic. To minimize issues,**

**make sure you activate your environment before running any executables!**

**Channels:**

**- defaults**

**Platform: win-64**

**Collecting package metadata (repodata.json): done**

**Solving environment: done**

**## Package Plan ##**

**environment location: C:\Users\Purushothaman S\anaconda3\envs\housing-ml**

**added / updated specs:**

**- python=3.10**

**The following packages will be downloaded:**

**package | build**

**---------------------------|-----------------**

**expat-2.7.1 | h8ddb27b\_0 259 KB**

**pip-25.1 | pyhc872135\_2 1.3 MB**

**python-3.10.18 | h981015d\_0 16.2 MB**

**setuptools-78.1.1 | py310haa95532\_0 1.7 MB**

**tk-8.6.14 | h5e9d12e\_1 3.5 MB**

**tzdata-2025b | h04d1e81\_0 116 KB**

**vc-14.42 | haa95532\_5 11 KB**

**vs2015\_runtime-14.42.34433 | hbfb602d\_5 1.2 MB**

**wheel-0.45.1 | py310haa95532\_0 145 KB**

**------------------------------------------------------------**

**Total: 24.4 MB**

**The following NEW packages will be INSTALLED:**

**bzip2 pkgs/main/win-64::bzip2-1.0.8-h2bbff1b\_6**

**ca-certificates pkgs/main/win-64::ca-certificates-2025.2.25-haa95532\_0**

**expat pkgs/main/win-64::expat-2.7.1-h8ddb27b\_0**

**libffi pkgs/main/win-64::libffi-3.4.4-hd77b12b\_1**

**openssl pkgs/main/win-64::openssl-3.0.16-h3f729d1\_0**

**pip pkgs/main/noarch::pip-25.1-pyhc872135\_2**

**python pkgs/main/win-64::python-3.10.18-h981015d\_0**

**setuptools pkgs/main/win-64::setuptools-78.1.1-py310haa95532\_0**

**sqlite pkgs/main/win-64::sqlite-3.45.3-h2bbff1b\_0**

**tk pkgs/main/win-64::tk-8.6.14-h5e9d12e\_1**

**tzdata pkgs/main/noarch::tzdata-2025b-h04d1e81\_0**

**vc pkgs/main/win-64::vc-14.42-haa95532\_5**

**vs2015\_runtime pkgs/main/win-64::vs2015\_runtime-14.42.34433-hbfb602d\_5**

**wheel pkgs/main/win-64::wheel-0.45.1-py310haa95532\_0**

**xz pkgs/main/win-64::xz-5.6.4-h4754444\_1**

**zlib pkgs/main/win-64::zlib-1.2.13-h8cc25b3\_1**

**Downloading and Extracting Packages:**

**Preparing transaction: done**

**Verifying transaction: done**

**Executing transaction: done**

**#**

**# To activate this environment, use**

**#**

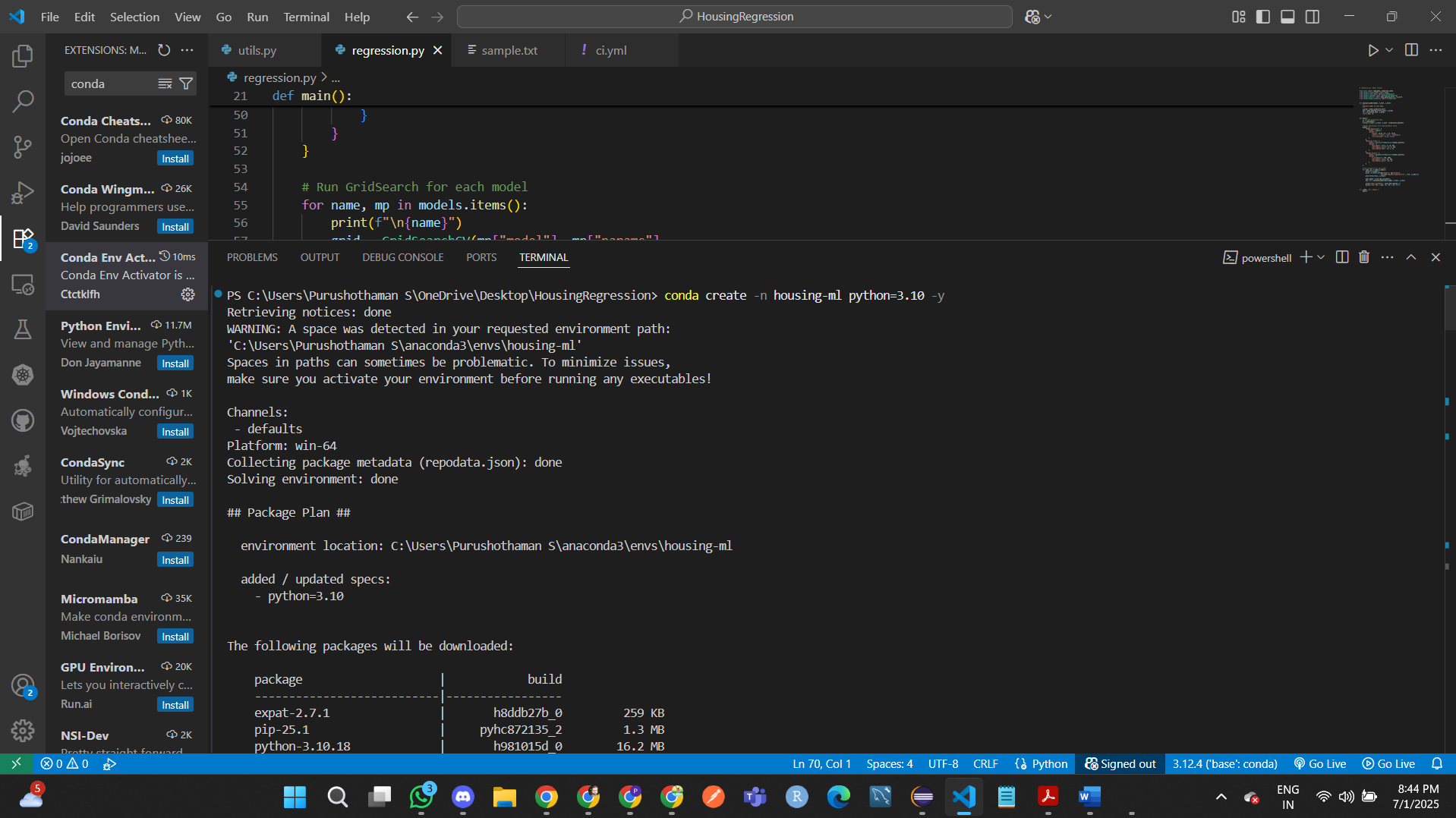
**# $ conda activate housing-ml**

**#**

**# To deactivate an active environment, use**

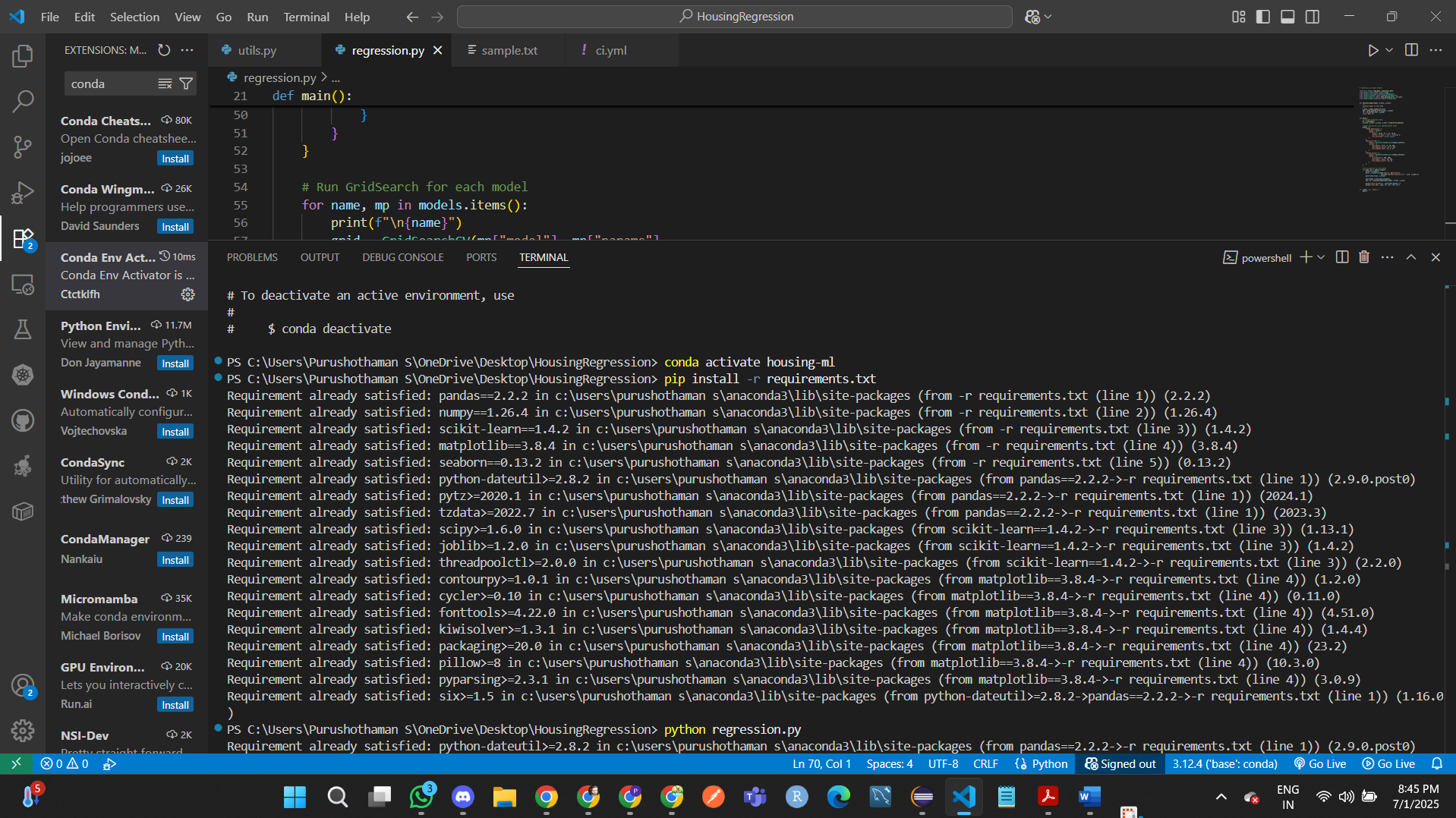
**#**

**# $ conda deactivate**

****

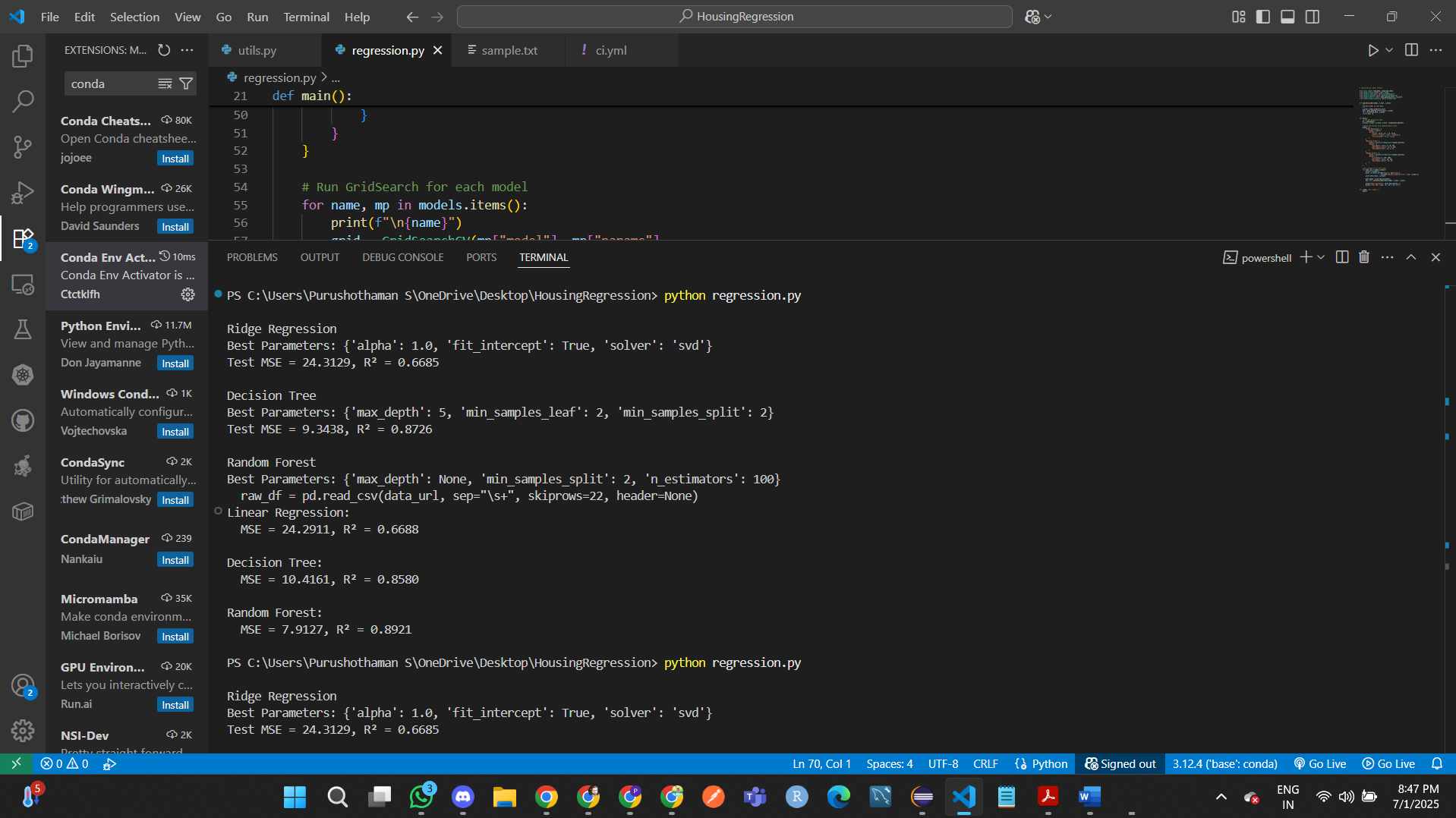
**Command 2: conda activate housing-ml**

**Command 3: pip install -r requirements.txt**

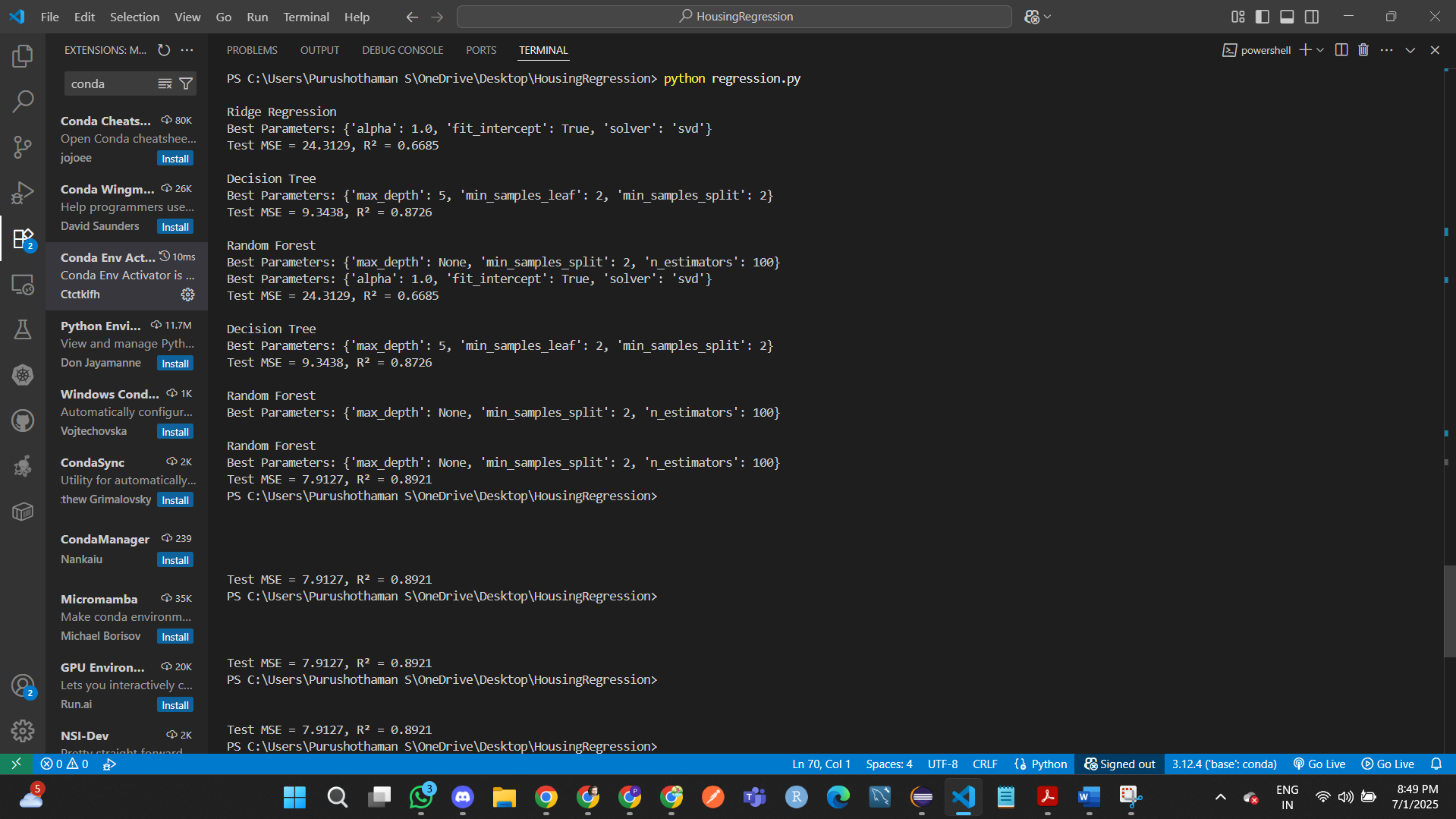
****

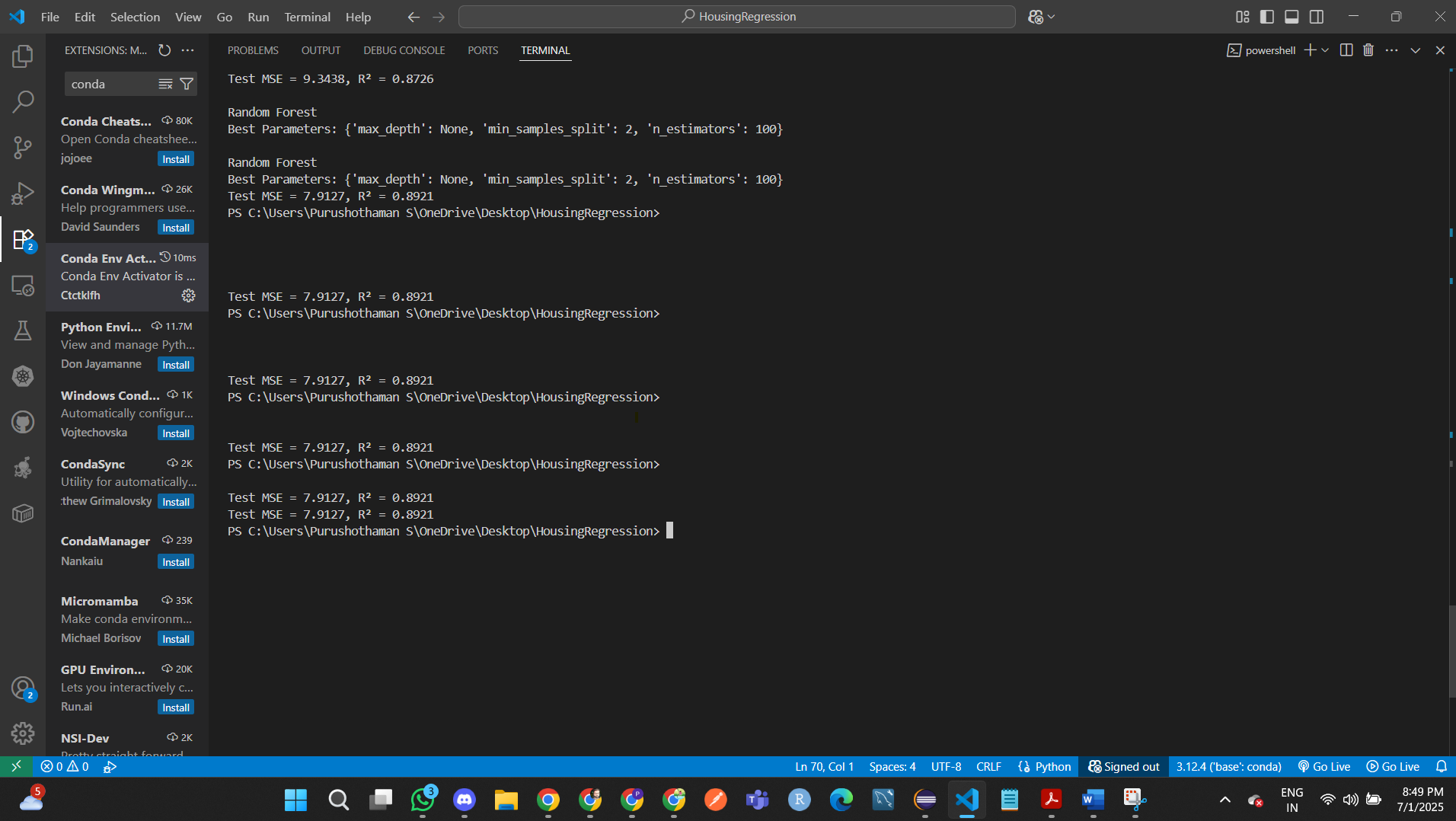
**Command 4: python regression.py**

**Out put before hyper parameter fine tune**

****

**Output after hyper parameter fine tune**

****



**Conclusion**

Machine learning is not just about building models is about building systems that work reliably, can be improved over time, and are easy to maintain. This assignment gave us a practical glimpse into that bigger picture. From loading and cleaning data to training and fine-tuning models, we saw how every step matters when you are trying to make accurate predictions in this case, estimating house prices.

Beyond the modelling itself, we also practiced important ML Ops concepts like modular coding, version control with Git, and automating workflows with GitHub Actions. These are the tools and habits that turn a simple machine learning script into something that could be part of a real-world application. Overall, this project helped bridge the gap between classroom theory and professional practice in a meaningful way