# **MLOPS Project Report**

# By: Samir JABBAR

# Introduction

In the dynamic landscape of today's real estate market, the ability to accurately predict housing prices is of paramount importance. Whether for property buyers, sellers, or real estate investors, having reliable insights into the market value of a property is crucial for making informed decisions. Machine learning, a branch of artificial intelligence, has emerged as a powerful tool in addressing this challenge by leveraging data-driven models to predict housing prices with a high degree of accuracy.

# Context

The real estate industry has witnessed a transformative shift in recent years, propelled by advancements in technology and the increasing availability of data. Traditional methods of pricing prediction, relying on historical trends and expert opinions, have proven insufficient in capturing the complex interplay of factors influencing property values. In this context, machine learning has emerged as a game-changer, providing a systematic and data-driven approach to predict housing prices.

# Machine Learning in Pricing Prediction

Machine learning algorithms excel in uncovering intricate patterns within vast datasets, enabling them to discern subtle correlations that may elude human analysis. In pricing prediction, these algorithms analyze a multitude of features, such as property characteristics, location, economic indicators, and market trends, to generate accurate and timely predictions. The ability to adapt to changing market dynamics and learn from new data makes machine learning particularly well-suited for real-time pricing predictions.

#### House Price Prediction

One of the notable applications of machine learning in the real estate domain is house price prediction. By harnessing historical sales data, demographic information, and various property attributes, machine learning models can provide reliable estimates of a house's market value. This not only benefits potential buyers and sellers but also aids real estate professionals in setting competitive listing prices and optimizing investment strategies.

In this project, we explore the intersection of machine learning and real estate by developing predictive models for house pricing. Leveraging a dataset of housing features and prices, we aim to showcase the efficacy of machine learning in predicting housing values accurately. The project involves the implementation of two applications—one using Flask and another using FastAPI—to demonstrate the practical integration of machine learning models into real-world applications.

# Project Outline: Using MLflow for Housing Price Prediction

#### I. Data Preprocessing

* Exploratory Data Analysis (EDA) to understand the distribution of features.
* Feature Engineering

#### II. Model Training and Tracking with MLflow

* Model Training
* Train five different models to compare their performance.
* Model Tracking with MLflow
* Save the best model in the ONNX format for portability and compatibility.
* Serialize preprocessing transformations using the Transformers API and save in pickle format.

#### III. FastAPI Integration

* Build a FastAPI application to expose the best-performing model as a REST API.
* Package the FastAPI application as a container using Docker.
* Consuming APIs with Postman

#### IV. Flask Application

* Develop a dedicated Flask application to consume the API created with FastAPI.
* Design an interface for users to interact with the model.
* Package the Flask application as a container using Docker.

# Data Preprocessing

#### Loading and Exploring the Dataset

The project initiates with loading the housing dataset and conducting an initial exploration. Using the Pandas library, the data is loaded into a DataFrame, and the first and last few rows are displayed to provide a snapshot of the dataset's structure. Additionally, essential information such as data types, missing values, and descriptive statistics are analyzed to gain insights into the dataset's characteristics.

#### Feature Selection and Correlation Analysis

Exploratory Data Analysis (EDA) involves assessing the correlation between features and the target variable (housing prices). A correlation matrix is visualized using a heatmap, aiding in the identification of features with significant impact. Features with weak correlations are pruned to enhance the model's efficiency.

# Model Training with MLflow

#### Model Building

Five distinct machine learning models are chosen for training:

* Linear Regression
* Ridge Regression
* Lasso Regression
* Decision Tree Regressor
* Random Forest Regressor

The Scikit-Learn library is employed for building these models. MLflow's experiment tracking is initiated, allowing seamless monitoring of metrics, parameters, and models during the training process.

#### Model Evaluation and Selection

Model performance is evaluated based on metrics such as accuracy and Mean Squared Error (MSE). The best-performing model, identified through MLflow's tracking capabilities, is selected for further deployment. In this project, the Random Forest Regressor emerges as the top performer in terms of MSE.

#### Model Training with MLflow

The MLflow framework is leveraged for seamless experimentation and tracking of model performance. Five diverse ML models, including Linear Regression, Ridge Regression, Lasso Regression, Decision Tree Regressor, and Random Forest Regressor, are trained on the preprocessed data. MLflow's capabilities for tracking metrics, parameters, and model versions play a pivotal role in the evaluation and selection of the best-performing model.

#### Best Model Serialization

The selected model (Random Forest Regressor) is serialized in the Open Neural Network Exchange (ONNX) format. ONNX provides a standardized format for representing machine learning models, ensuring interoperability across different frameworks.

#### Preprocessing Transformations Serialization

The preprocessing transformations, carried out using Scikit-Learn's StandardScaler, are serialized using the Pickle format. This ensures that the input data fed into the model during inference undergoes the same preprocessing steps as during training.

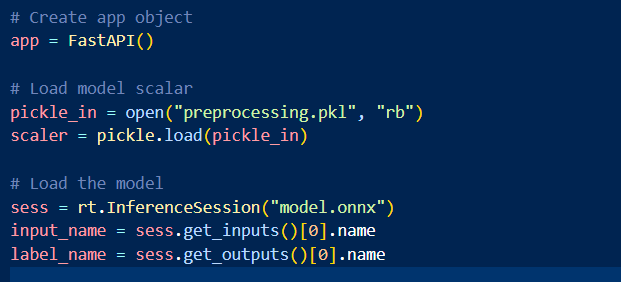
# Using FastAPI for Model Deployment

FastAPI is employed as a powerful and efficient framework for deploying machine learning models as REST APIs. Leveraging the serialized files from the model training phase, this section details the steps involved in creating a REST API for predicting house prices. The API is then packaged as a Docker container for easy deployment and testing.

#### FastAPI Implementation

The FastAPI implementation is encapsulated in the house\_api.py file. Key components include:

Initialization: The FastAPI app is created using FastAPI(). Serialized files, including the ONNX model and preprocessing scaler, are loaded.



**API Endpoints**: Two API endpoints are defined. The root endpoint (/) provides a welcome message, while the /predict endpoint handles house price predictions.

A blue screen with white text

Description automatically generated

Testing the API

A screenshot of a computer

Description automatically generated

We can see that the model prediction is 409038.5$, so the API is working.

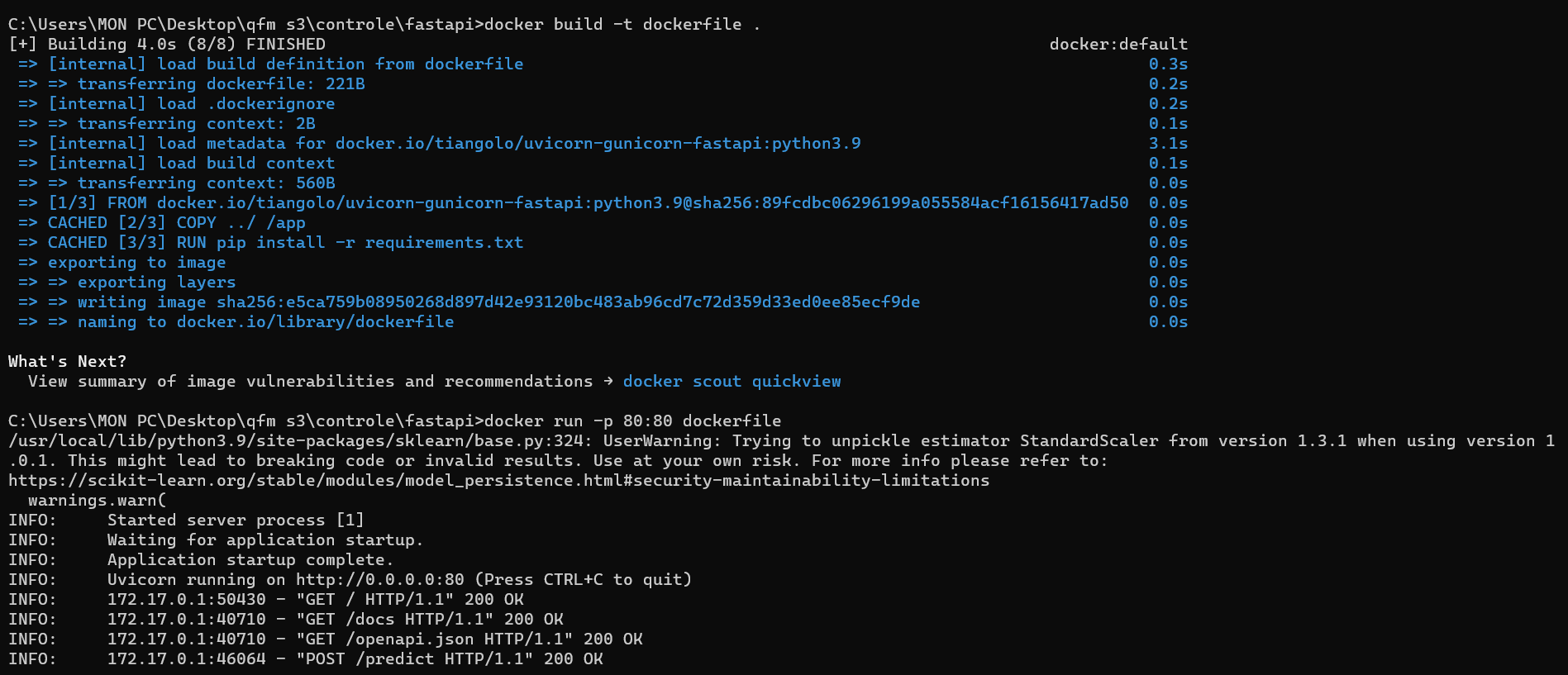
#### Docker Containerization

The Dockerfile defines the steps for packaging the FastAPI app and the model as a Docker container. It pulls the tiangolo/uvicorn-gunicorn-fastapi image, copies the project files into the container, installs dependencies, exposes port 80, and sets the command to run the app.

A blue screen with white text

Description automatically generated

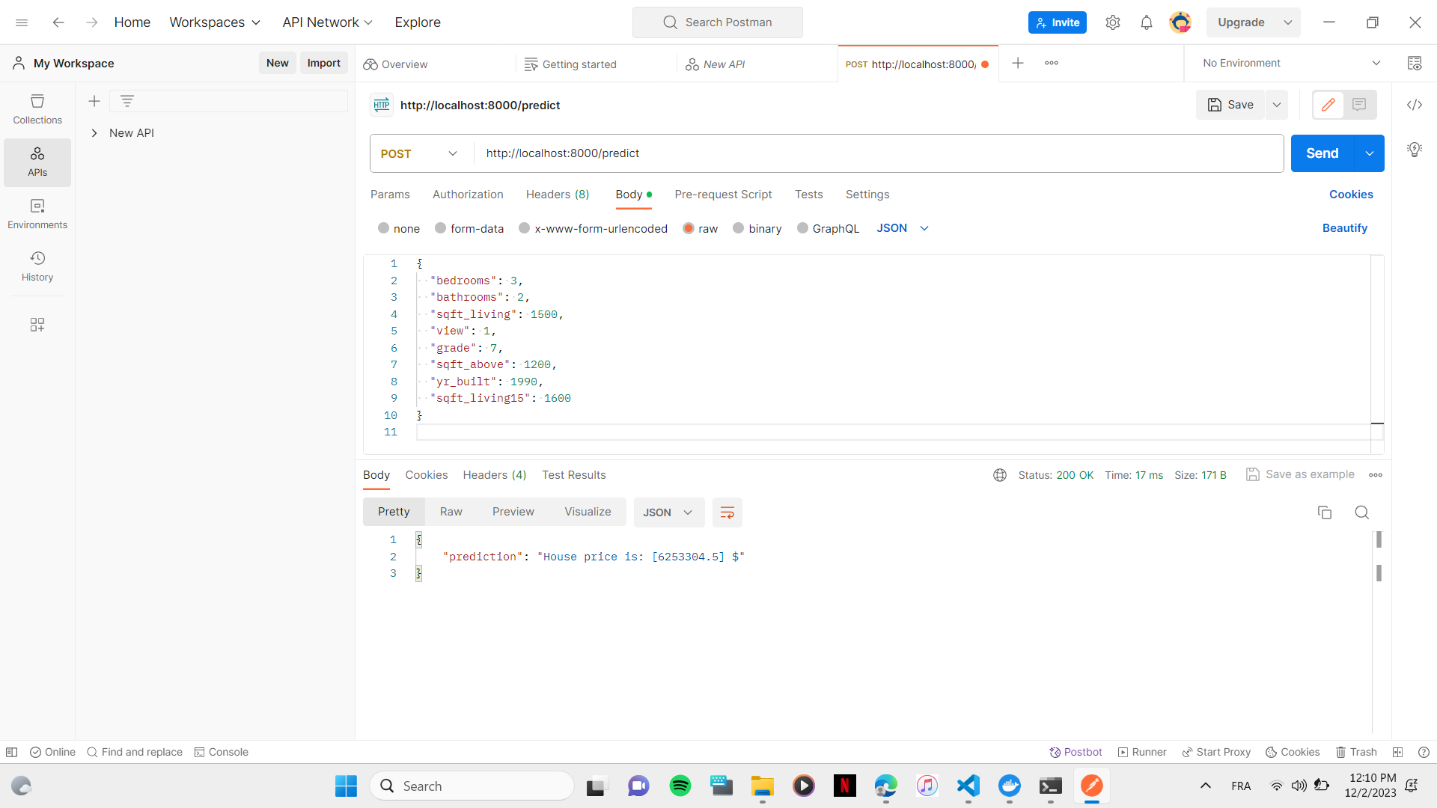
Testing the docker Image

A screenshot of a computer

Description automatically generated

#### Postman Testing

Postman, a popular API testing tool, can be used to interact with and test the FastAPI-based housing price prediction API. By sending POST requests to the /predict endpoint with appropriate input data, users can validate the functionality of the deployed model.



# Using Flask to Consume FastAPI Prediction Service

The integration of Flask with FastAPI further extends the project's capabilities by creating a dedicated application to consume the predictive API. Flask serves as a lightweight web framework, allowing the creation of user-friendly interfaces to interact with the underlying machine learning model deployed via FastAPI.

#### Flask Application

Within the flask\_app.py file, a Flask application is initiated. Two routes are defined - the root route (/) rendering an HTML file with a form, and the /predict route handling the prediction process. The latter extracts user input from the form, sends a POST request to the FastAPI server, and returns the prediction result.

The Flask application communicates with the FastAPI server using the requests library. Upon form submission, input data is collected, transformed into a JSON payload, and sent as a POST request to the FastAPI /predict endpoint. The received prediction result is then presented to the user.

#### Running the Application

The Flask application runs on port 5000, and users can access it through a web browser. By entering house feature details in the form and submitting it, users trigger a seamless interaction between Flask and FastAPI for on-the-fly predictions.

Testing the flask app

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

#### Flask Appp Dockerization with FastAPI Images and Docker Compose

To deploy the Flask application that consumes a FastAPI service for house price prediction we need to dockerizing both the Flask and FastAPI applications and orchestrating their deployment using Docker Compose.

1. Dockerizing FastAPI:

* A FastAPI application was created to serve a machine learning model for house price prediction.
* The FastAPI service exposed an endpoint /predict to receive house features and return price predictions.

1. Dockerizing Flask:

* Flask use the requests library to make POST requests to the FastAPI service for predictions.

1. Docker Compose Configuration:

* Docker Compose was employed to manage the deployment of both Flask and FastAPI applications.
* A custom bridge network named "mynetwork" was defined to enable communication between the two services.
* The FastAPI service was configured to run on port 8000, and the Flask service on port 5000.
* The docker-compose.yml file defined the services, networks, and dependencies for the applications.

Testing out the docker flask app

A black screen with white text

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated