**Case Study: Predicting Customer Churn at RetailGenius**

AI Project Methodology

*Report Submitted*

*by*

**Sivaprasad Puthumadathil Rameshan Nair**

# Sanjaya Deshapriya Gunawardena

# Yazid Ben

**EPITA FALL 2023**

**CONTENTS:**

1. Code structure
2. Standardization - Flake8
3. Documentation - Sphinx
4. API, Model tracking and local deployment - MLflow
5. Cloud deployment

**1.Code Structure:**

**1. 1 Introduction**

This project aims to predict customer churn for an E-Commerce platform. Using machine learning techniques, specifically Random Forest Classifier, I analyse customer behaviour and other relevant features to identify patterns that indicate the likelihood of churn.

**1.2 Data Overview**

This dataset is from Kaggle website. <https://www.kaggle.com/datasets/ankitverma2010/ecommerce-customer-churn-analysis-and-prediction/data>

As it is shown on the website, “The data set belongs to a leading online E-Commerce company. An online retail (E commerce) company wants to know the customers who are going to churn, so accordingly they can approach customer to offer some promos.”

The dataset has 5630 rows and 20 columns.

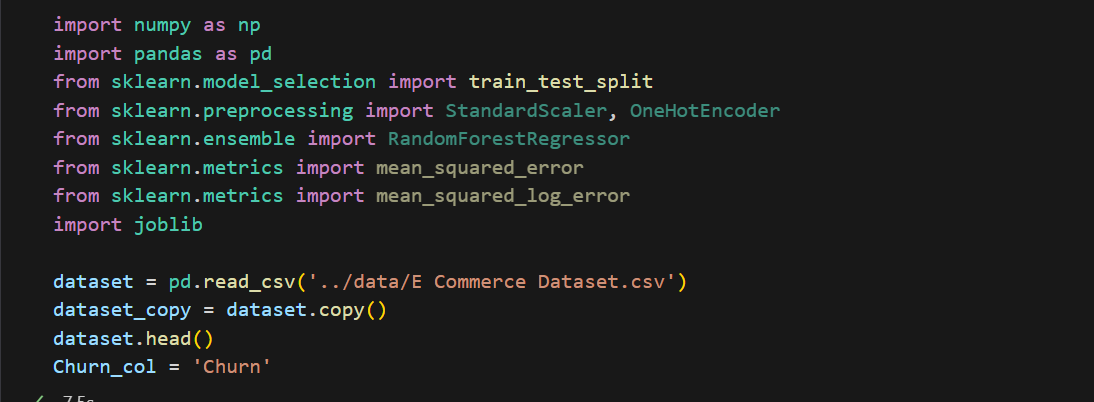
A screenshot of a computer

Description automatically generated

**Data Variable Discerption:**

E Comm CustomerID : Unique customer ID  
E Comm Churn : Churn Flag  
E Comm Tenure : Tenure of customer in organization  
E Comm PreferredLoginDevice : Preferred login device of customer  
E Comm CityTier : City tier  
E Comm WarehouseToHome : Distance in between warehouse to home of customer  
E Comm PreferredPaymentMode : Preferred payment method of customer  
E Comm Gender : Gender of customer  
E Comm HourSpendOnApp : Number of hours spend on mobile application or website  
E Comm NumberOfDeviceRegistered : Total number of deceives is registered on particular customer  
E Comm PreferedOrderCat : Preferred order category of customer in last month  
E Comm SatisfactionScore : Satisfactory score of customer on service  
E Comm MaritalStatus : Marital status of customer  
E Comm NumberOfAddress : Total number of added on particular customer  
E Comm Complain : Any complaint has been raised in last month  
E Comm OrderAmountHikeFromlastYear: Percentage increases in order from last year  
E Comm CouponUsed : Total number of coupon has been used in last month  
E Comm OrderCount : Total number of orders has been places in last month  
E Comm DaySinceLastOrder :Day Since last order by customer  
E Comm CashbackAmount :Average cashback in last month

* First, I imported the required libraries for the preprocessing, model training and evaluation processes.  
  Then I loaded the dataset, Created a copy of the dataset (This is often done to preserve the original dataset for reference or to avoid modifying it accidentally during data manipulation.), Defined the target variable.



**1.3 Feature Engineering**

First, I splitted the dataset into train and test sets. Then I checked for the sum of missing values for each feature.

Handling missing values :

A screenshot of a computer

Description automatically generated

* Then I dropped the columns with more than 50% null values.
* After that calculated the percentages of missing values for the remaining columns in X train and X test.

A screenshot of a computer

Description automatically generated

* Selected the features with the lowest missing values.
* selected\_features = ['CustomerID', 'PreferredLoginDevice', 'CityTier', 'PreferredPaymentMode', 'Gender', 'NumberOfDeviceRegistered']
* Convert the list of feature names into a data frame with the corresponding columns from the original dataset and then check the data types of the selected features data frame.

A screenshot of a computer program

Description automatically generated

* Defining categorical columns:
* Index(['PreferredLoginDevice', 'PreferredPaymentMode', 'Gender'], dtype='object')
* Defining continuous columns:
* Index(['CustomerID', 'CityTier', 'NumberOfDeviceRegistered'], dtype='object')

Encoding categorical variables

* **OneHotEncoder** is used to encode categorical integer features as one-hot numeric arrays. It creates a binary column for each category and returns a sparse matrix or dense array.

Scaling continuous variables

* **StandardScaler** standardizes features by removing the mean and scaling them to unit variance. In other words, it transforms the data distribution to have a mean of 0 and a standard deviation of 1.

Then for the final steps of feature engineering,

Transformed the data using preprocessing objects and then concatenated the transformed features into a single DataFrame.

**1.4 Model Selection and Training**

We used **RandomForestRegressor** as our model.

1. **Model Architecture or Algorithm**:
   * Random Forest Regressor: This is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mean prediction of the individual trees for regression tasks.
2. **Parameters Tuned or Optimized**:
   * Random Forest Regressor has several hyperparameters that can be tuned to improve model performance. Common parameters include the number of trees in the forest (**n\_estimators**), the maximum depth of the trees (**max\_depth**), the minimum number of samples required to split an internal node (**min\_samples\_split**), and the minimum number of samples required to be a leaf node (**min\_samples\_leaf**).

Overall, we used this model because it can potentially optimize its parameters, apply cross-validation for model evaluation, and potentially consider ensemble methods or stacking techniques to further enhance performance.

I have saved the preprocessing objects (Scaler and encoder) and the model using Joblib.

joblib.dump(model, '../models/model.joblib')

Then we can reuse them in model inference section.

**1.5. Model Evaluation**

After transforming the test set using preprocessing objects we used that data frame for the evaluation.

1. **Root Mean Squared Error (RMSE)**:
   * RMSE is a measure of the average deviation of predicted values from the actual values in the dataset. It's calculated by taking the square root of the average of the squared differences between predicted and actual values.
   * The RMSE value we've obtained is 0.39. This means, on average, the model's predictions are off by approximately 0.39 units from the actual values. Lower RMSE values indicate better model performance, as they represent smaller deviations from the true values.
2. **Mean Squared Logarithmic Error (MSLE)**:
   * MSLE is another measure of the difference between predicted and actual values, but it's computed using the natural logarithm of the predicted and actual values. This metric is particularly useful when the target variable spans several orders of magnitude.
   * The MSLE value we've obtained is 0.08. MSLE penalizes underestimates more heavily than overestimates. Like RMSE, lower MSLE values indicate better model performance.

**1.6. Model Inference**

For this part we used the test.csv file that we created using the main dataset. This section will help us to **Model Generalization** by evaluating the model on a separate test set helps assess its ability to generalize to unseen data.

A black screen with colorful text

Description automatically generated

For this section I loaded the previous preprocessing objects and the model to do the evaluation on the test data. For the result got the same value for the RMSE.

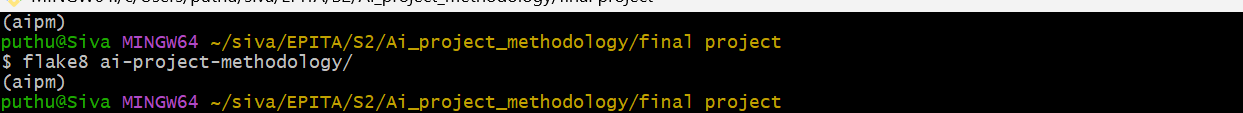
We have created four python scripts.

**init.py** : consists with the selected features.  
**preprocess.py** : which has a function preprocess\_data\_with\_objects, taking a data frame as input and returning the processed data.  
**model\_training.py** : taking a data frame as input and returning the performances(RMSE value).

**Inference.py**: make\_predictions function returning the predictions.

**2.Standardization – Flake8:**

We used flake8 to maintain PEP8 standards in the code base. Below is the flake8 report with no errors

****

**3.Documentation – Sphinx**:

Install Sphinx using :

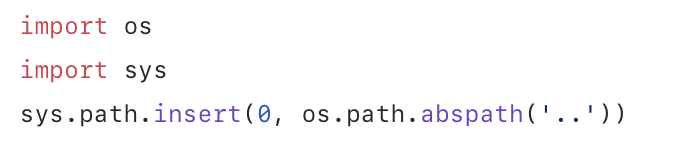
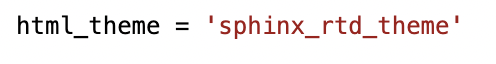
                     pip install sphinx sphinx\_rtd\_theme

To start, go to docs folder and run the command

               Sphinx- quickstart

And give the necessary details like project name, authors, etc Then it will create a template for the documentation.

The generated conf.py file can be modified to change the theme and adding additional confgurations.

Go to the root folder and specify the folder where the code to be documented is located

sphinx-apidoc -o docs  < your folder>

It will generate .rst file for all the .py scripts, index.rst and module.rst

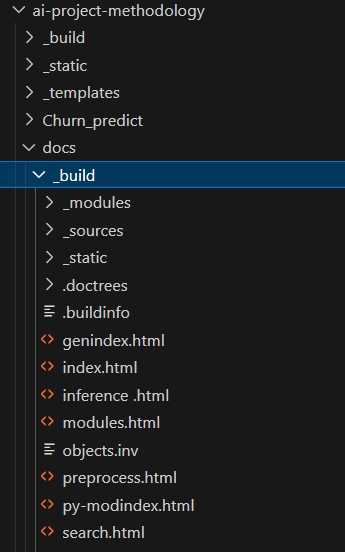
To generate HTML, First include module file inside index.rst

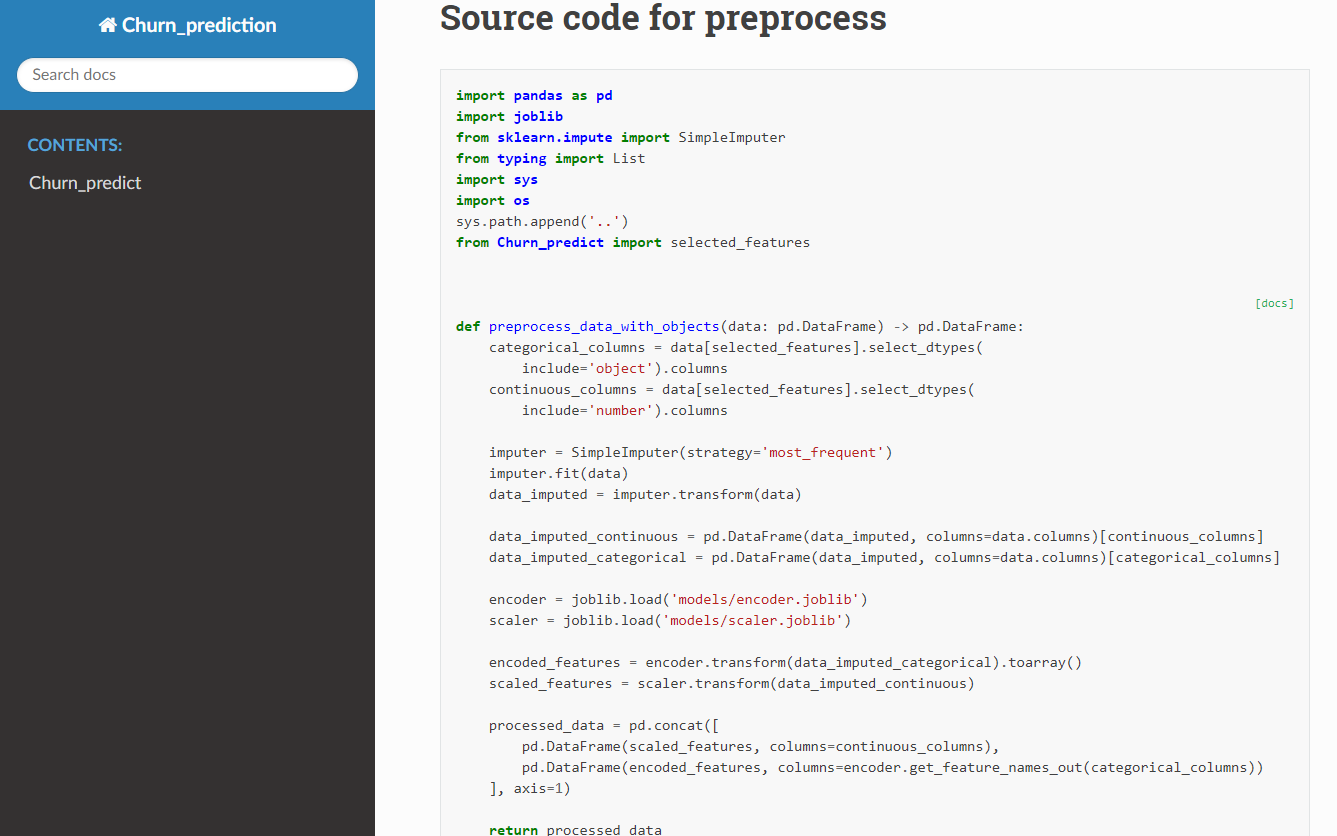
Go to the docs folder and excecute the following command

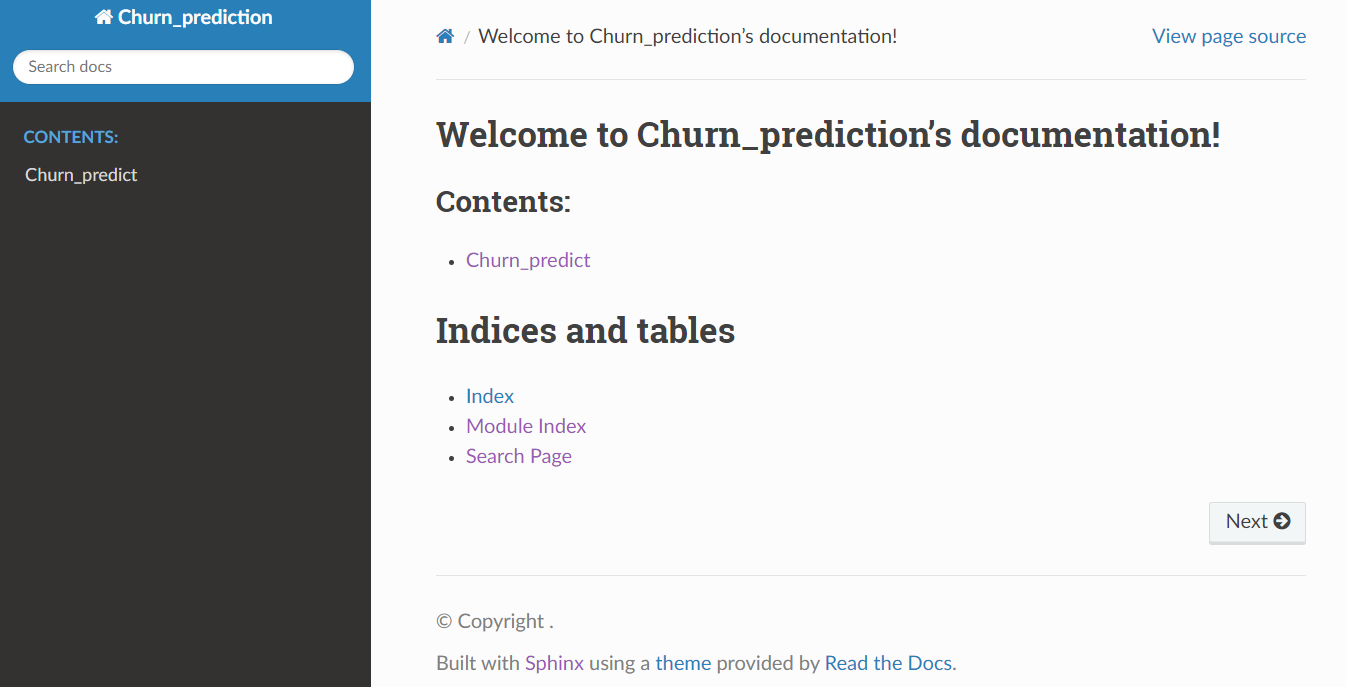
    $ cd docs

    $ make html

This will generate the html files inside build directory. 



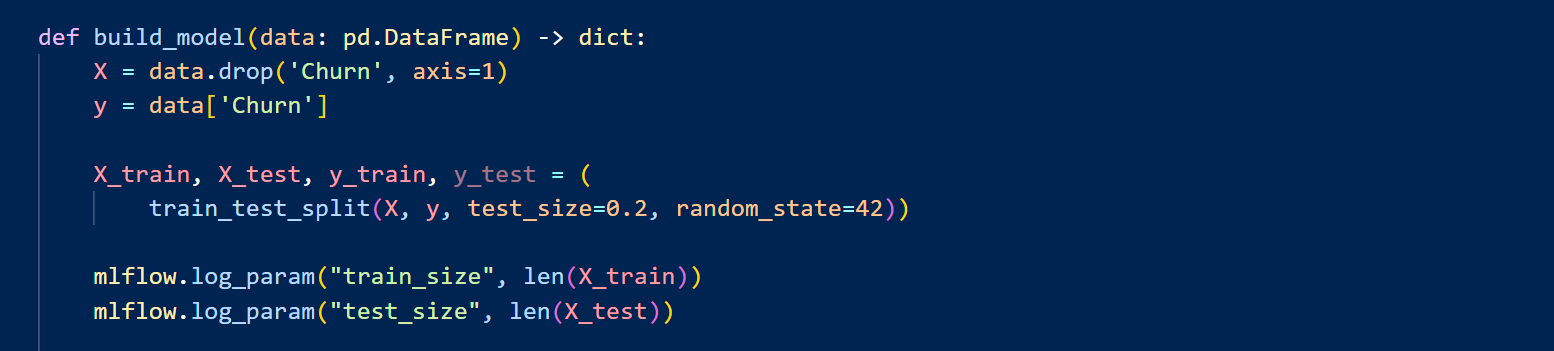




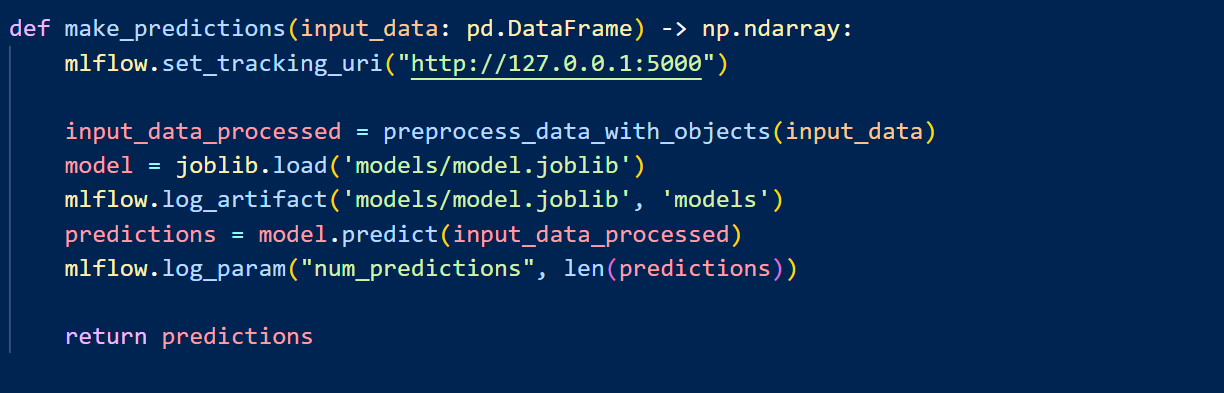
**4.API ,Model tracking and local deployment - MLflow:**

First install MLflow and integrate it to the code base.

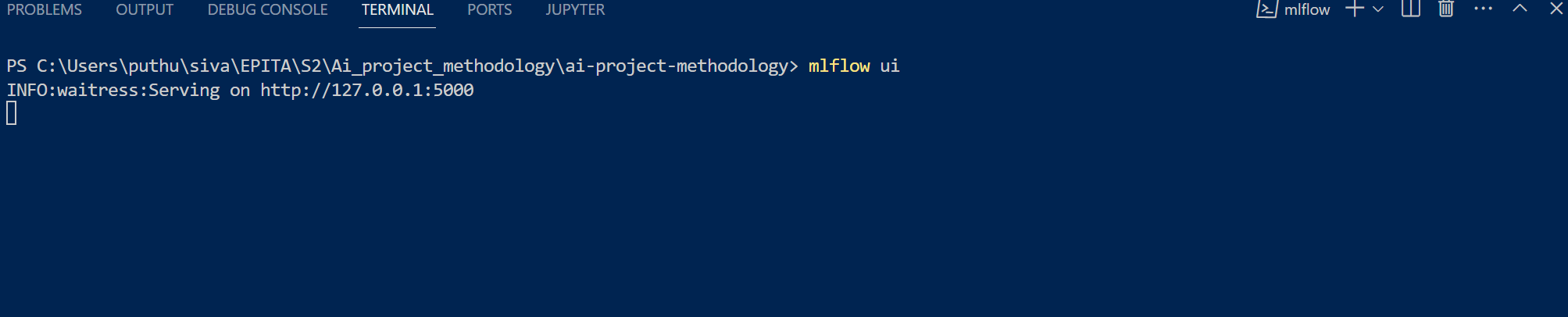
For example, the below code is used to log the parameters of the model during training. Like this through out the code base log all the necessary params in MLflow.



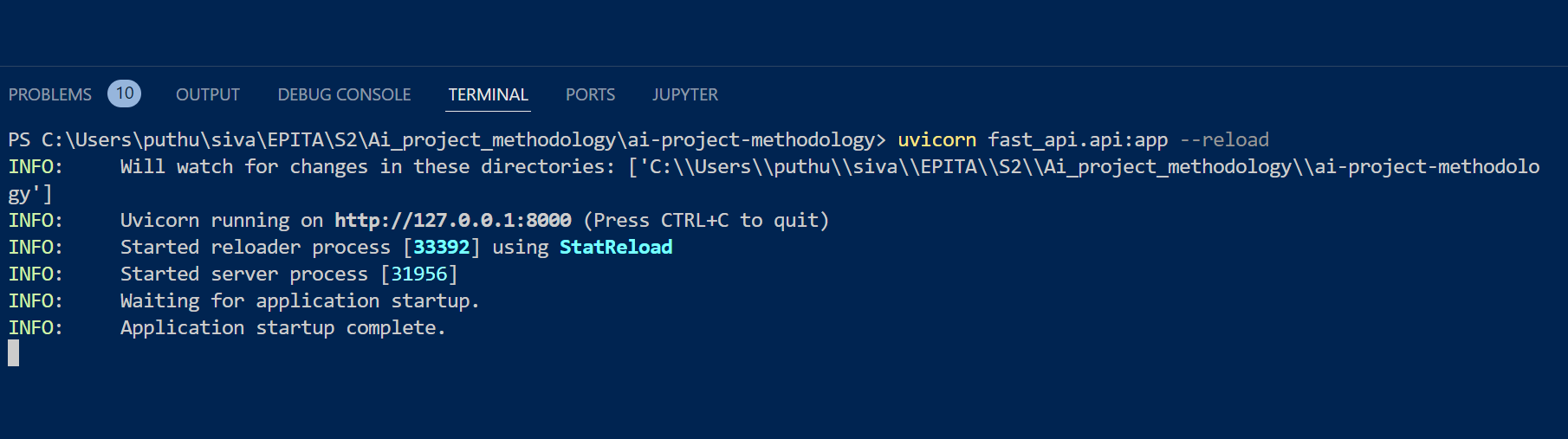
During inference, we are logging the model and predictions. MLflow tracking URI is set to http://127.0.0.1:5000 to direct experiment tracking to a local server.



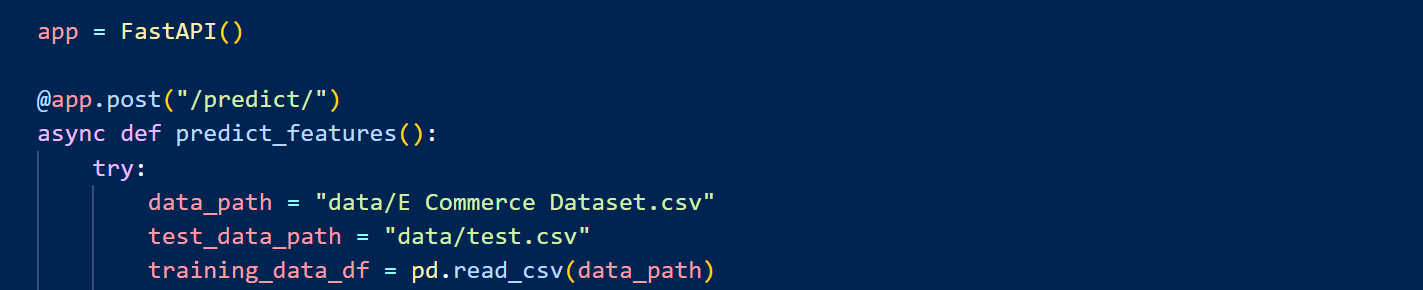
Then we invoke the MLflow using mlflow ui command :



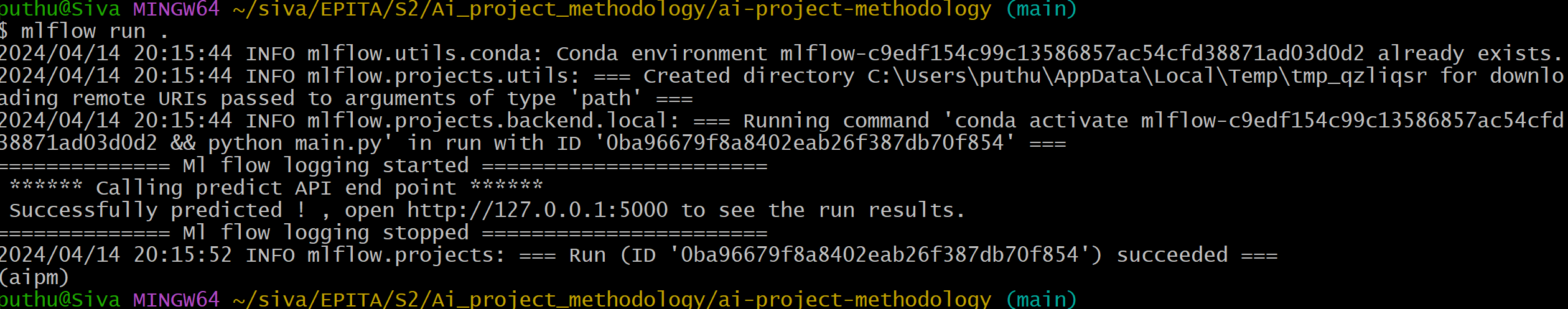
Once the server is up , next we have to make the FastAPI up. We are using fastAPI to deploy the model into API.



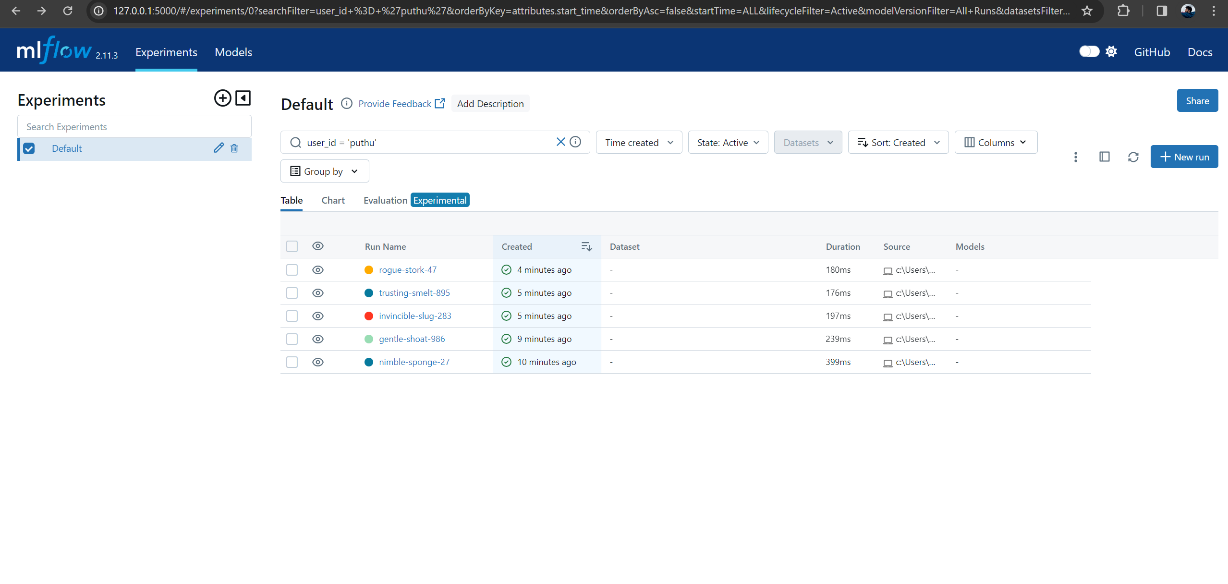
The model is available as below API point , so that whenever prediction is required we call call this API

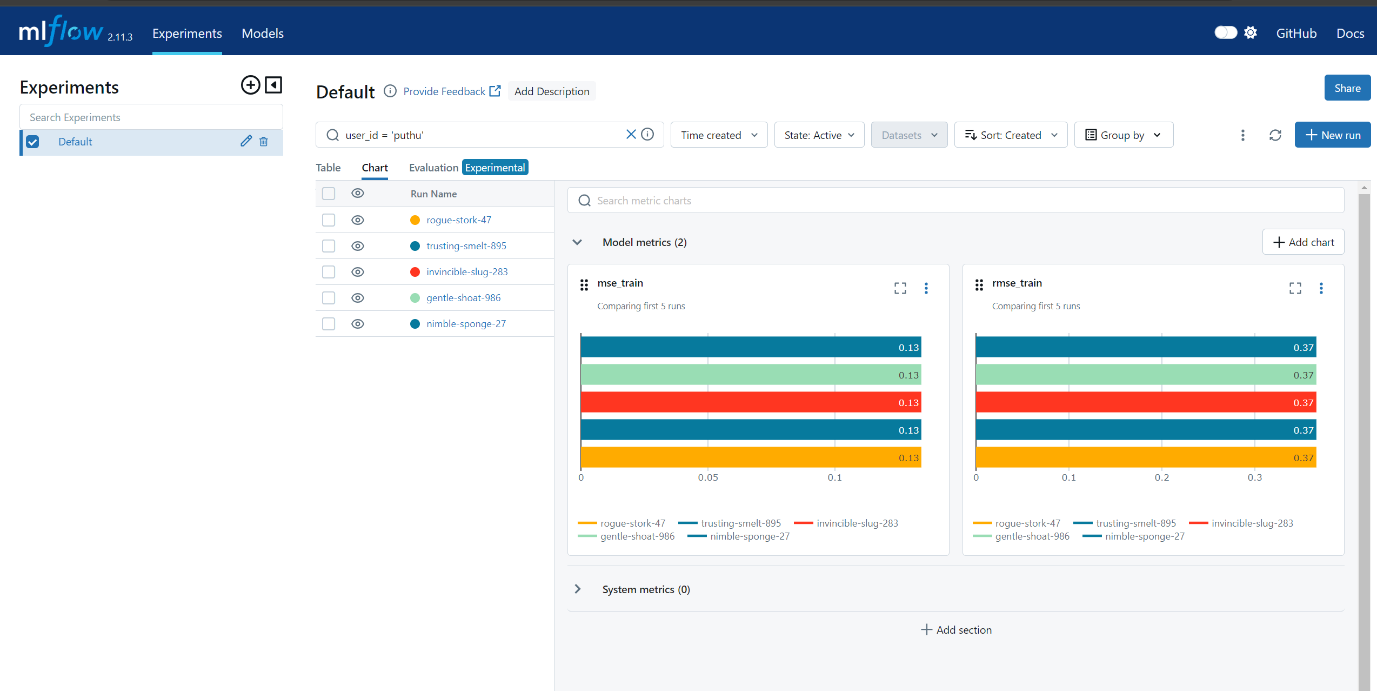


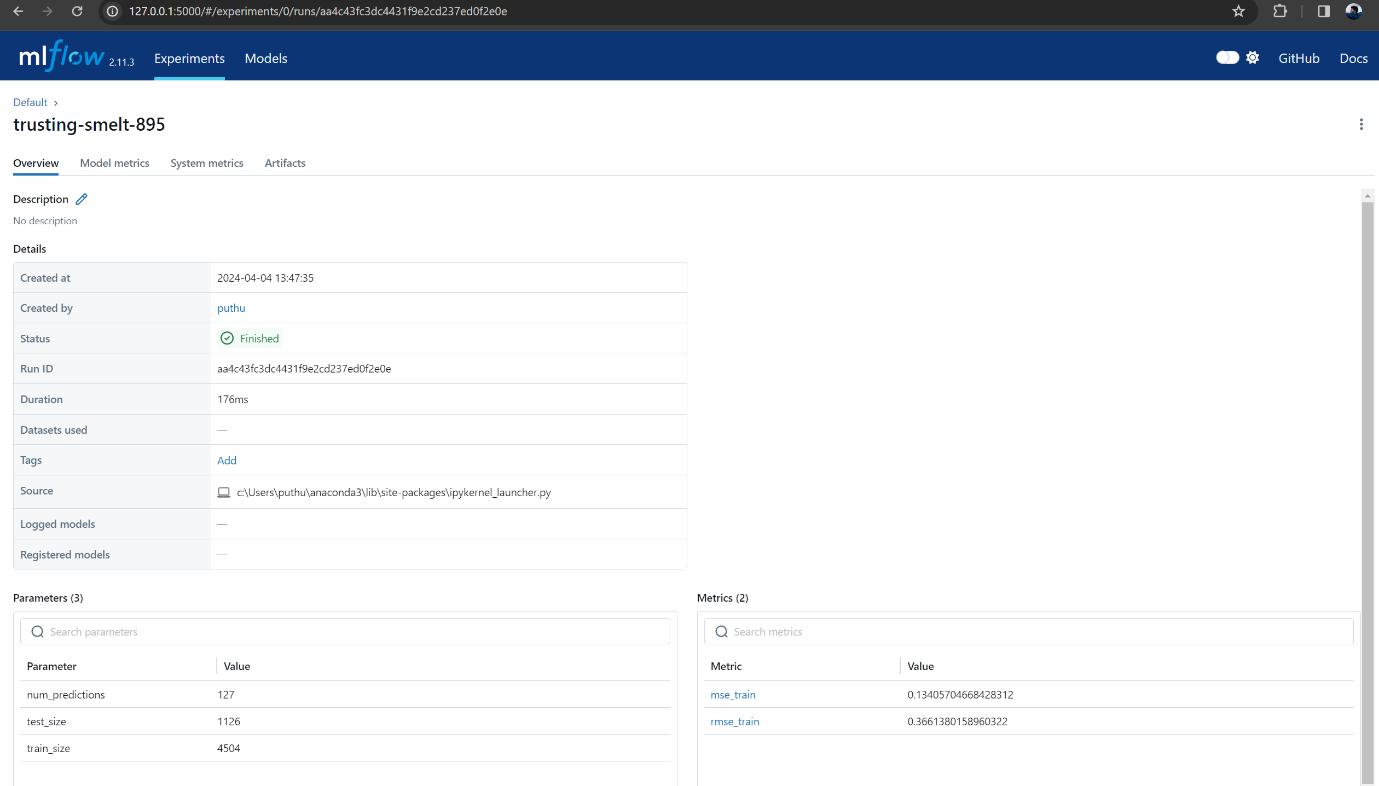
Once all are up , we can run the mlflow using the command mlflow run .



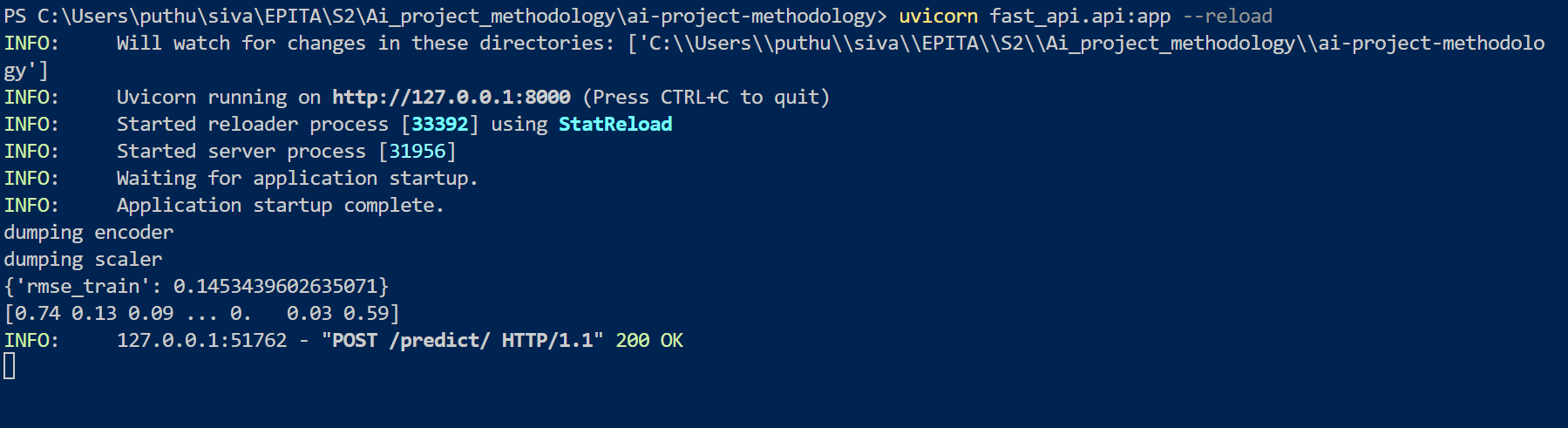
As we can see the run is completed and we can see the run details in the mlflow ui:







We can also see the terminal output from fastAPI , showing API request is successfully completed.



We have used:

● Track parameters & metrics of model using MLflow

● Packaging the code in a reusable and reproducible model format with MLFlow projects

● Deploying the model into an API that will enable to score predictions using ML Models

**5.Cloud deployment:**

*Source:* <https://chat.openai.com/>

A scalable and dependable cloud infrastructure is needed to implement the RetailGenius AI project, which aims to predict customer churn. An extensive range of tools and services are available from Google Cloud Platform (GCP) to help with the deployment, scaling, and management of AI applications.

**Proposed Architecture**

Data Storage:

Google Cloud Storage (GCS): Store the raw and processed datasets securely. GCS provides scalable, durable, and highly available object storage.

Data Processing and ETL:

Google Cloud Dataflow: Process and transform the data using Dataflow, a fully managed stream and batch data processing service. It supports Apache Beam, enabling seamless integration with various data sources and formats.

Machine Learning Model Training:

Google AI Platform (Unified): Train, tune, and deploy machine learning models on AI Platform Unified. It offers a unified platform to build, train, and deploy ML models, providing autoscaling and built-in distributed training.

Model Serving and Inference:

Google Kubernetes Engine (GKE): Deploy the trained models as microservices on GKE, a managed Kubernetes service. This allows for scalable and reliable model serving with auto-scaling capabilities.

Monitoring and Logging:

Google Cloud Monitoring and Logging: Monitor the deployed applications, track performance metrics, and diagnose issues using Cloud Monitoring and Logging. Set up alerts for any anomalies or errors.

User Interface:

Google App Engine: Develop a web-based dashboard or API endpoints using App Engine. This provides a platform to interact with the AI models, visualize insights, and integrate with other systems.

**Deployment Steps**

Data Preparation:

Upload the raw datasets to GCS.

Implement ETL processes using Dataflow to clean, transform, and prepare the data for model training.

Model Development and Training:

Develop machine learning models using Python and TensorFlow or PyTorch.

Train the models on AI Platform Unified using the processed data.

Model Deployment:

Deploy the trained models on GKE as containerized applications.

Expose the models through RESTful APIs for real-time predictions.

Monitoring and Logging Setup:

Configure monitoring and logging for deployed applications using Cloud Monitoring and Logging.

Set up dashboards to visualize key metrics and performance indicators.

User Interface Development:

Develop a user-friendly dashboard or API endpoints using App Engine.

Integrate the deployed models with the UI for interactive predictions and insights.