PROJECT REPORT

CS4085-MLOPS

Name: Abdul Rafay

ID: 21i-2582  
  
  
  
  


*FAST NUCES, Islamabad*

*Department of Computer Science*

Contents

[Data Collection Process 2](#_Toc185171677)

[Data Sources 2](#_Toc185171678)

[Collection Method 2](#_Toc185171679)

[Collected features 2](#_Toc185171680)

[Data Preprocessing 2](#_Toc185171681)

[Model Development 3](#_Toc185171682)

[Model Selection 3](#_Toc185171683)

[Hyperparameter Tuning Strategy 3](#_Toc185171684)

[Model Evaluation Metrics 3](#_Toc185171685)

[Model Deployment 3](#_Toc185171686)

[API Specification 3](#_Toc185171687)

[Deployment Configuration 4](#_Toc185171688)

[Monitoring and live testing 4](#_Toc185171689)

[Prometheus Integration 4](#_Toc185171690)

[Monitoring Components 4](#_Toc185171691)

[Key Metrics Tracked 5](#_Toc185171692)

[Grafana Integration 5](#_Toc185171693)

[Monitoring Workflow 5](#_Toc185171694)

# Data Collection Process

## Data Sources

Data is collected from the following APIs

* OpenWeatherMap API
* AirVisual API (IQAir)

## Collection Method

* Automated script using DVC for version control
* Scheduled data fetching using cron jobs after every hour

**\* 0 \* \* \* /usr/bin/python3 /home/abdulrafay/mlops/course-project-abdul-rafay-1/src/data\_collection.py >> /home/abdulrafay/mlops/course>**

## Collected features

* Temperature
* Humidity
* Wind Speed
* Pollutant Concentrations (CO, NO2, O3)
* Particulate Matter (PM2.5, PM10)

## Data Preprocessing

**Following are the data Preprocessing Steps applied**

1. Missing Value Handling
   * Median imputation for missing values
   * Intelligent feature engineering
2. Outlier Removal
   * Interquartile Range (IQR) method
   * Removes extreme values that could skew predictions
3. Feature Engineering
   * Created lag features for time series prediction
   * Temporal feature extraction

# Model Development

## Model Selection

* Time Series Forecasting Model: ARIMA
* Hyperparameter Tuning:
  + Explored multiple ARIMA orders
  + Best Model selection based on RMSE

## Hyperparameter Tuning Strategy

* **Search Space**:
  + p (Autoregressive terms): 0 to 2
  + d (Differencing): 0 to 1
  + q (Moving Average terms): 0 to 2
* **Total Configurations Explored**: 18 unique ARIMA model configurations
* **Tuning Method**: Grid Search
* **Selection Criteria**: Root Mean Square Error (RMSE)

## Model Evaluation Metrics

* Root Mean Square Error (RMSE)
* Mean Square Error (MSE)
* Mean Absolute Error (MAE)

## Model Deployment

### API Specification

* Flask-based prediction endpoint
* Input: No input data. It predicts based on the data on which the model is trained
* Output:
  + Predicted AQI
  + Prediction Confidence Interval
  + Model Parameters

## Deployment Configuration

* Host: 0.0.0.0
* Port: 8000
* Prediction Endpoint: /predict
  + It is used to load model and predict the aqi.
* Health Check Endpoint: /health
  + It is used to check that API is up and running.

# Monitoring and live testing

## Prometheus Integration

* Wrote a monitoring\_system.py script that calculates data ingestion and api performance.
* It starts Prometheus metrics sender on 9000 port.
* Configured Prometheus.xml in Prometheus directory

global:

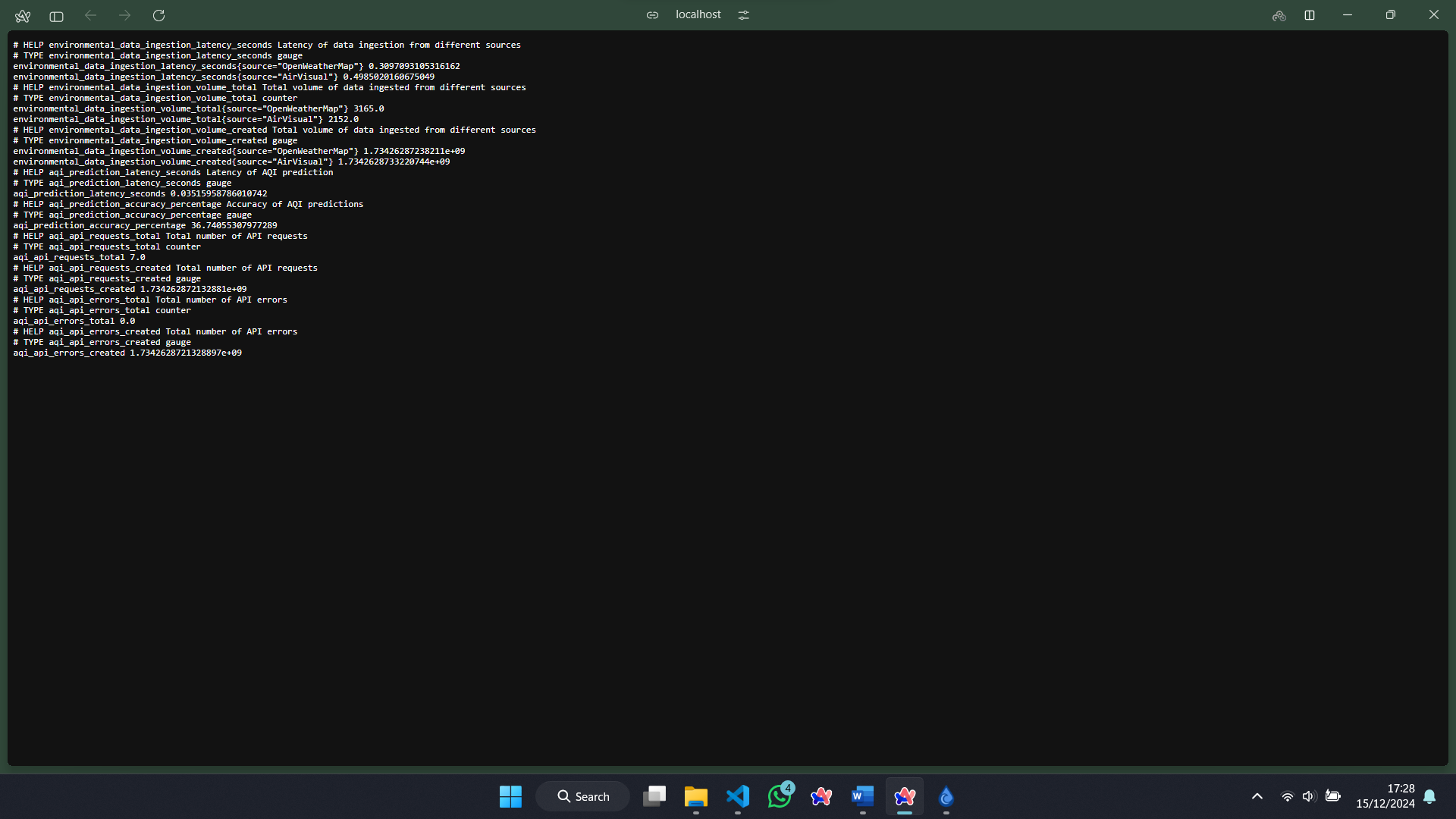
scrape\_interval: 15s

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:9090']



## Monitoring Components

* Prometheus Metrics Tracking
* Simulated Data Ingestion Monitoring
* API Performance Tracking
* Threaded Background Monitoring

## Key Metrics Tracked

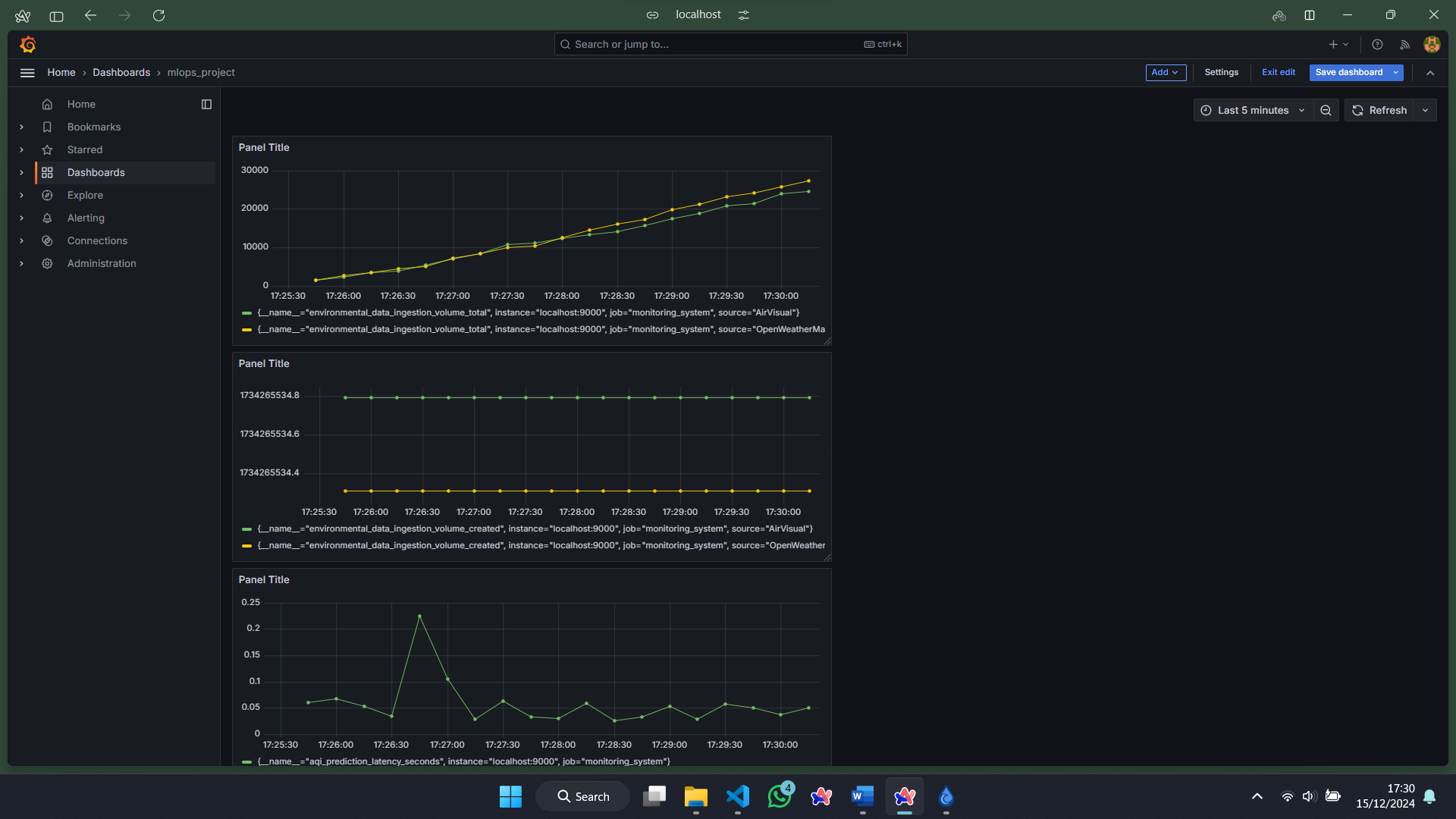
* Data Ingestion Latency
* Data Ingestion Volume
* Model Prediction Latency
* API Request Tracking
* API Error Counting
* Prediction Accuracy Estimation

## Grafana Integration

* Sample dashboard configuration
* Graphs for latency and data volume

## Monitoring Workflow

1. Simulates data ingestion from multiple sources
2. Tracks API prediction performance
3. Generates metrics for Prometheus
4. Displays metrics on Grafana



A screenshot of a computer

Description automatically generated