National College of Ireland

Configuration Manual

MSc Research Project
MSc in AI for Business

Uuganbolor Sororburam
Student ID: 23291303

School of Computing
National College of Ireland

Supervisor: Dr. Muslim Jameel Syed

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Uuganbolor Sosorburam
Student ID:	23291303
Programme:	MSc in AI for Business **Year:** 2025
Module:	MSc Research Project
Supervisor:	Dr. Muslim Jameel Syed
Submission Due Date:	15 September 2025
Project Title:	AI- Driven Air Pollution Forecasting Using Machine Learning: A Case Study in Ulaanbaatar

Word Count:1584 **Page Count** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Uuganbolor Sosorburam

Date: 15 September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Uuganbolor Sosorburam
23291303

1 Introduction

This Configuration Manual documents the end-to-end setup and execution steps for the project "AI-Driven Air Pollution Forecasting Using Machine Learning: A Case Study in Ulaanbaatar." Its aim is to make the work fully reproducible: from environment setup and data acquisition through preprocessing, modelling, evaluation, and interpretability.

2 System Specifications

The project was implemented and tested in both local and cloud-based environments to ensure reproducibility across different hardware setups. This section outlines the exact hardware, software, and package dependencies required to replicate the AI-driven air pollution forecasting pipeline.

2.1 Hardware Specification

- Processor: Intel i5 / i7 (or equivalent)
- RAM: 16 GB
- Storage: 512 GB SSD or more
- OS: Windows 10/11 (64-bit) or macOS 13+

2.2 Software Specification

- Programming language: Python
- IDE: Jupyter Notebook
- Web Browser: Google Chrome

3 Libraries and Dependencies

The project was implemented locally using Jupyter Notebook as the development environment.

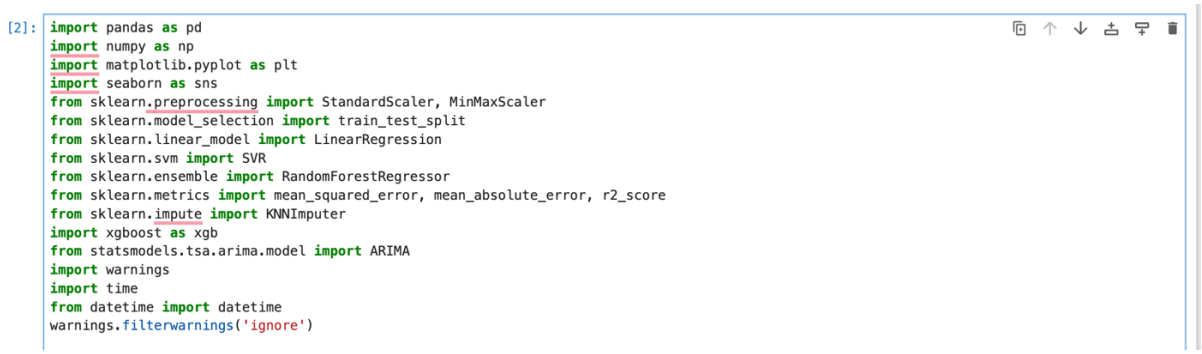
The following Python libraries were used:

Table 1: Software Specification

Library	**Purpose**
pandas	Data loading, manipulation, and cleaning
numpy	Numerical operations
matplotlib	Basic data visualization
seaborn	Statistical plotting and EDA
scikit-learn	Preprocessing, machine learning models, evaluation metrics

xgboost	Gradient boosting regression model
statsmodels	ARIMA time-series forecasting
shap *(optional)*	Model interpretability and feature importance
KNNImputer *(from scikit-learn)*	Missing value imputation
warnings, datetime, time	Utility modules for code execution management

The project was developed and tested with Python 3.10 in Jupyter. Below picture Figure 1: Importing Required Libraries shows all Libraries needed.



```
[2]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.preprocessing import StandardScaler, MinMaxScaler
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.svm import SVR
     from sklearn.ensemble import RandomForestRegressor
     from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
     from sklearn.impute import KNNImputer
     import xgboost as xgb
     from statsmodels.tsa.arima.model import ARIMA
     import warnings
     import time
     from datetime import datetime
     warnings.filterwarnings('ignore')
```

Figure 1: Importing Required Libraries

This cell imports all Python libraries used in the project, including packages for:

- Data handling (pandas, numpy)
- Visualization (matplotlib, seaborn)
- Preprocessing (StandardScaler, KNNImputer)
- Machine Learning (LinearRegression, SVR, RandomForestRegressor, XGBoost)
- Statistical modelling (ARIMA)
- Evaluation metrics.

4 Data Acquisition

The dataset for this project was compiled from two publicly available API sources:

Table 2: Dataset collection

Source	Data Provided	Notes
OpenAQ API	Hourly air pollutant concentrations (PM2.5, PM10)	Official open-source air quality database
Weather.com	Meteorological variables: temperature, relative humidity, wind speed, atmospheric pressure	Collected for the same time periods as pollutant data

Python code that gets PM2.5, PM10 and other pollutants data from the OpenAQ API, going through each sensor and date range to collect hourly readings.

```python
# Initialize OpenAQ client with API key
api_key = "9cda73ee40e70e9b7b37b91479cee4925eb65110ee85e06ee92150e45155a425"
client = OpenAQ(api_key=api_key)

# Load sensor info
with open("sensors.json") as f:
    sensors = json.load(f)

# Map parameter name to sensor id
param_to_sensor = {s["parameter"]["name"]: s["id"] for s in sensors}
target_params = ["pm25", "pm10", "co", "no2", "so2"]
sensor_ids = [param_to_sensor[p] for p in target_params]

all_results = []
for sensor_id in sensor_ids:
    results = True
    page_num = 1
    while results:
        print(f"Fetching {sensor_id} page {page_num}...")
        measurements = client.measurements.list(
            limit=1000,
            page=page_num,
            sensors_id=sensor_id,
            datetime_from="2018-01-01",
            datetime_to="2025-06-24"
        )
        if len(measurements.results) == 0:
            results = False
        else:
            all_results.extend(measurements.results)
            page_num += 1
```

Figure 2: Retrieving Air Quality Data from OpenAQ API

ulaanbaatar_air_quality_clean

datetime	co	no2	pm10	pm25	so2
2017-12-31 23:30:00+08:00	5762.0	72.0	298.0	294.0	197.0
2018-01-01 00:30:00+08:00	4293.0	64.0	241.0	238.0	167.0
2018-01-01 01:30:00+08:00	5268.0	64.0	265.0	263.0	176.0
2018-01-01 02:00:00+08:00	6044.0	67.0	293.0	291.0	196.0
2018-01-01 03:00:00+08:00	5130.0	65.0	253.0	249.0	184.0

weather_dot_com_2015_2020

expire_time_gmt	valid_time_gmt	day_ind	temp	wx_icon	icon_extd	wx_phrase	dewPt	heat_index	rh	pressure	vis	wc	wdir_cardinal	wspd	uv_desc	feels_like	uv_index	clds	date
1420048800	1420041600	N	-18.0	22	2200	Smoke	-24.0	-18.0	76.0	25.71	3.0	-18.0	VAR	2.0	Low	-18.0	0	CLR	2015-01-01 00:00:00
1420052400	1420045200	N	-20.0	22	2200	Smoke	-26.0	-20.0	76.0	25.69	3.0	-20.0	VAR	2.0	Low	-20.0	0	CLR	2015-01-01 01:00:00
1420056000	1420048800	N	-17.0	22	2200	Smoke	-24.0	-17.0	70.0	25.66	3.0	-17.0	VAR	2.0	Low	-17.0	0	CLR	2015-01-01 02:00:00
1420059600	1420052400	N	-15.0	22	2200	Smoke	-20.0	-15.0	77.0	25.66	3.0	-15.0	CALM	0.0	Low	-15.0	0	SCT	2015-01-01 03:00:00
1420063200	1420056000	N	-11.0	22	2200	Smoke	-18.0	-11.0	70.0	25.66	3.0	-11.0	CALM	0.0	Low	-11.0	0	SCT	2015-01-01 04:00:00
1420066800	1420059600	N	-13.0	22	2200	Smoke	-18.0	-13.0	77.0	25.64	3.0	-13.0	CALM	0.0	Low	-13.0	0	CLR	2015-01-01 05:00:00
1420070400	1420063200	N	-11.0	22	2200	Smoke	-17.0	-11.0	77.0	25.64	3.0	-11.0	CALM	0.0	Low	-11.0	0	SCT	2015-01-01 06:00:00
1420074000	1420066800	N	-13.0	22	2200	Smoke	-18.0	-13.0	77.0	25.64	3.0	-13.0	CALM	0.0	Low	-13.0	0	SCT	2015-01-01 07:00:00
1420077600	1420070400	N	-18.0	22	2200	Smoke	-26.0	-18.0	70.0	25.66	3.0	-31.0	S	4.0	Low	-31.0	0	CLR	2015-01-01 08:00:00
1420081200	1420074000	D	-17.0	22	2200	Smoke	-24.0	-17.0	70.0	25.69	1.0	-17.0	CALM	0.0	Low	-17.0	1	CLR	2015-01-01 09:00:00
1420084800	1420077600	D	-13.0	22	2200	Smoke	-18.0	-13.0	77.0	25.71	1.0	-13.0	CALM	0.0	Low	-13.0	1	CLR	2015-01-01 10:00:00
1420088400	1420081200	D	-6.0	22	2200	Smoke	-15.0	-6.0	65.0	25.74	1.0	-6.0	CALM	0.0	Low	-6.0	1	CLR	2015-01-01 11:00:00

Figure 3: Raw Datasets

- Timeframe Covered: Data consisted of 8,217 hourly records from multiple monitoring stations in Ulaanbaatar.
- Merge Method: Datasets were aligned and merged using a common timestamp.
- Final Structure: 8 columns – PM2.5 (target), PM10, temperature, humidity, pressure, wind speed, datetime, and (optional) day/night indicator (excluded from modelling).

merged_result_final

datetime	pm10	pm25	day_ind	temp	rh	pressure	wspd
2017-12-31 23:00:00	298.0	294.0	N	-17.0	76.0	25.69	2.0
2018-01-01 00:00:00	241.0	238.0	N	-17.0	76.0	25.69	2.0
2018-01-01 01:00:00	265.0	263.0	N	-15.0	77.0	25.69	0.0
2018-01-01 02:00:00	293.0	291.0	N	-18.0	76.0	25.69	0.0
2018-01-01 03:00:00	253.0	249.0	N	-18.0	76.0	25.69	0.0

Figure 4: Merged Datasets

Data Preparation Summary:

1. Datetime Formatting & Sorting: Converted timestamps to standard datetime format and sorted chronologically.
2. Missing Values: Applied KNNImputer (k=5) to numeric columns to estimate missing entries based on nearest neighbours.
3. Feature Selection: Retained PM10, temperature, humidity, pressure, and wind speed as predictors; PM2.5 as the target.
4. Scaling: Applied StandardScaler for models sensitive to feature magnitude (Linear Regression, SVR).
5. Split: Chronological split — 70% training, 15% validation, 15% testing.

File Format:

- Raw Data: CSV files retrieved from API exports.
- Processed Data: Cleaned CSV file ready for modelling after preprocessing.

5 Implementation Steps

5.1 Data Merging

The first step combines the air quality dataset from **OpenAQ** (PM2.5 and PM10) with the meteorological dataset from **Weather.com** (temperature, relative humidity, wind speed, and pressure). Both datasets are aligned to the hour level and merged on a common datetime key.

```
[3]: import pandas as pd

# Set Desktop path
desktop = "/Users/bolorsosorburam/Desktop/"

# Load files with only needed columns
air = pd.read_csv(
    f"{desktop}ulaanbaatar_air_quality_clean.csv",
    usecols=["datetime", "pm25", "pm10"],
    parse_dates=["datetime"]
)
weather = pd.read_csv(
    f"{desktop}weather_dot_com_2015_2020.csv",
    usecols=["date", "temp", "rh", "wspd", "pressure", "day_ind"],
    parse_dates=["date"]
)

# Convert both to datetime and floor to hours
air["datetime"] = pd.to_datetime(air["datetime"]).dt.floor("h")
weather["datetime"] = pd.to_datetime(weather["date"]).dt.floor("h")

# Merge on datetime
merged = air.merge(
    weather.drop("date", axis=1),  # Remove the original date column
    on="datetime",
    how="inner"
)

# Save result
output_path = f"{desktop}merged_result_final.csv"
merged.to_csv(output_path, index=False)

print(f"✅ Successfully merged {len(merged)} rows")
print(f"Saved to: {output_path}")
print("\nFirst 3 rows:")
print(merged.head(3))
```

Figure 5: Merging Air Quality & Weather Data

This cell merges hourly pollutant and weather measurements using an inner join on the datetime field, producing a unified dataset for further processing.

5.2 Handling Missing Values with KNN Imputation

After merging, some variables contained missing values. To address this, the **KNNImputer** from scikit-learn was applied to all numeric columns. The method estimates missing entries based on the five most similar observations in feature space, preserving local patterns in the data.

This step applies KNNImputer(n_neighbors=5) to fill gaps in the numeric data. The process improves the quality of model inputs while avoiding the biases of mean/median substitution.

```python
def preprocess_data(self):
    """Handle missing values and normalize features"""
    print("🔧 Preprocessing data...")

    # Convert datetime if it exists
    datetime_columns = ['datetime', 'timestamp', 'date', 'time']
    for col in datetime_columns:
        if col in self.data.columns:
            self.data[col] = pd.to_datetime(self.data[col])
            self.data = self.data.sort_values(col).reset_index(drop=True)
            break

    # Handle missing values using KNN imputation
    numeric_columns = self.data.select_dtypes(include=[np.number]).columns
    imputer = KNNImputer(n_neighbors=5)
    self.data[numeric_columns] = imputer.fit_transform(self.data[numeric_columns])

    print(f"✅ Missing values handled. Shape: {self.data.shape}")

    # Auto-detect pollutant columns
    pollutant_keywords = ['pm2.5', 'pm25', 'co', 'no2', 'so2', 'o3', 'aqi', 'pollutant']
    pollutant_columns = []

    for col in self.data.columns:
        if any(keyword in col.lower() for keyword in pollutant_keywords):
            pollutant_columns.append(col)

    # If no pollutant columns found, use first few numeric columns as targets
    if not pollutant_columns:
        print("⚠️ No pollutant columns detected. Using first 3 numeric columns as targets.")
        pollutant_columns = list(numeric_columns[:3])

    print(f"🎯 Detected pollutant columns: {pollutant_columns}")

    # Prepare features and targets
    feature_columns = [col for col in numeric_columns if col not in pollutant_columns]

    self.feature_names = feature_columns
    self.processed_data = {
        'features': self.data[feature_columns],
        'targets': self.data[pollutant_columns]
    }
```

Figure 6: Missing Value Imputation using KNN

5.3 Feature Scaling

Feature scaling was used only to models sensitive to feature magnitude (Linear Regression, Support Vector Regression). Tree-based models such as Random Forest and XGBoost were trained on raw data, as they are scale-invariant.

```python
def scale_features(self, X_train, X_val, X_test):
    """Scale features using StandardScaler"""
    print("🔧 Scaling features...")

    X_train_scaled = self.scaler.fit_transform(X_train)
    X_val_scaled = self.scaler.transform(X_val)
    X_test_scaled = self.scaler.transform(X_test)

    print("✅ Features scaled")

    return X_train_scaled, X_val_scaled, X_test_scaled
```

Figure 7: Feature Scaling for Linear Models

Standardization was applied to ensure all features have mean 0 and standard deviation 1, improving model convergence for algorithms sensitive to input magnitude.

5.4 Train Validation, Test Split

The dataset was split chronologically to reflect a real-world forecasting scenario and to avoid future information leaking into the training set.
The split used a 70% training, 15% validation, 15% testing ratio.

```python
def split_data(self, X, y, train_ratio=0.7, val_ratio=0.15):
    """Perform time-based split"""
    print("🔪 Splitting data (time-based)...")

    n_samples = len(X)
    train_size = int(train_ratio * n_samples)
    val_size = int(val_ratio * n_samples)

    X_train = X[:train_size]
    y_train = y[:train_size]

    X_val = X[train_size:train_size + val_size]
    y_val = y[train_size:train_size + val_size]

    X_test = X[train_size + val_size:]
    y_test = y[train_size + val_size:]

    print(f"✅ Data split - Train: {X_train.shape}, Val: {X_val.shape}, Test: {X_test.shape}")

    return X_train, X_val, X_test, y_train, y_val, y_test
```

Figure 8: Chronological Data Splitting

The dataset is split in temporal order into 70% training, 15% validation, and 15% testing sets, ensuring realistic model evaluation for time-series forecasting.

5.5 Model Training

After preprocessing, models were trained using the processed training data and validated on the held-out validation set.
Two examples are shown here: Linear Regression (baseline) and Random Forest Regressor (ensemble method).

1. Linear Regression

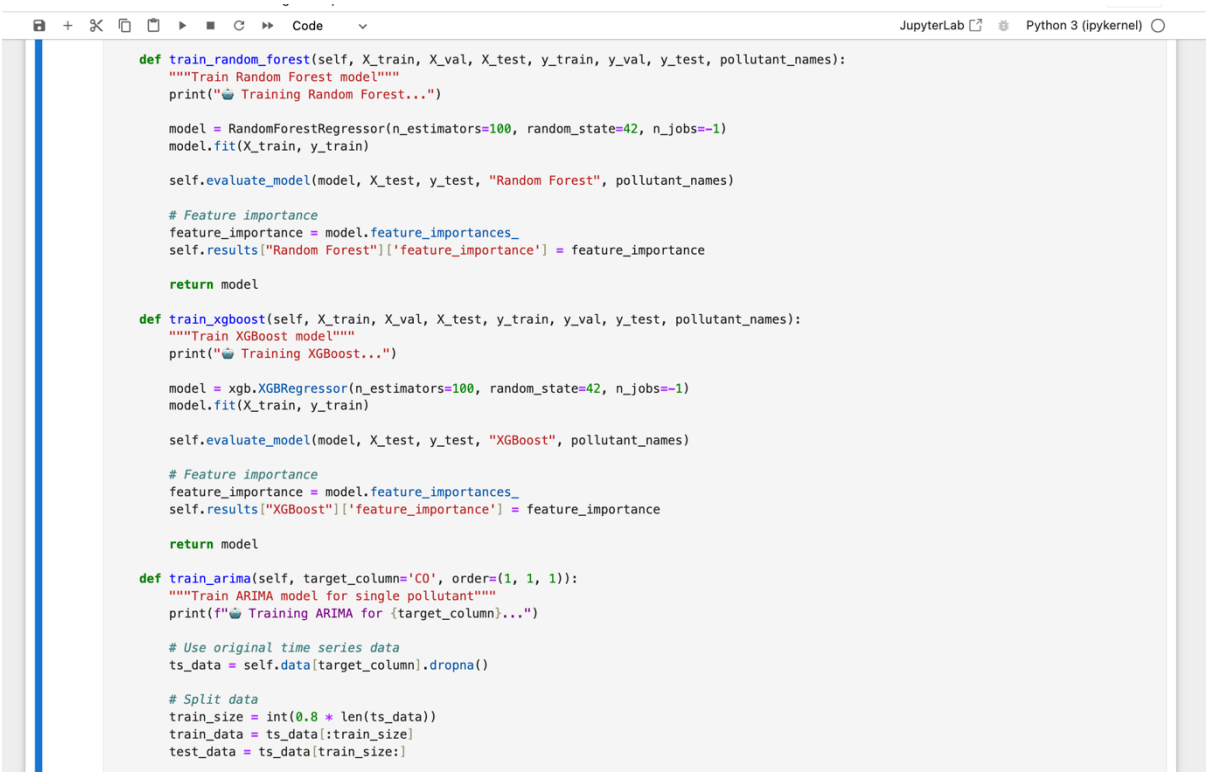
```python
def train_linear_regression(self, X_train, X_val, X_test, y_train, y_val, y_test, pollutant_names):
    """Train Linear Regression model"""
    print("🫖 Training Linear Regression...")

    model = LinearRegression()
    model.fit(X_train, y_train)

    self.evaluate_model(model, X_test, y_test, "Linear Regression", pollutant_names)
    return model
```

Figure 9: Linear Regression Training and Validation Performance

2. Random Forest Regressor

```python
def train_random_forest(self, X_train, X_val, X_test, y_train, y_val, y_test, pollutant_names):
    """Train Random Forest model"""
    print("🫖 Training Random Forest...")

    model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
    model.fit(X_train, y_train)

    self.evaluate_model(model, X_test, y_test, "Random Forest", pollutant_names)

    # Feature importance
    feature_importance = model.feature_importances_
    self.results["Random Forest"]['feature_importance'] = feature_importance

    return model
```

Figure 10: Random Forest Regressor Training and Validation Performance



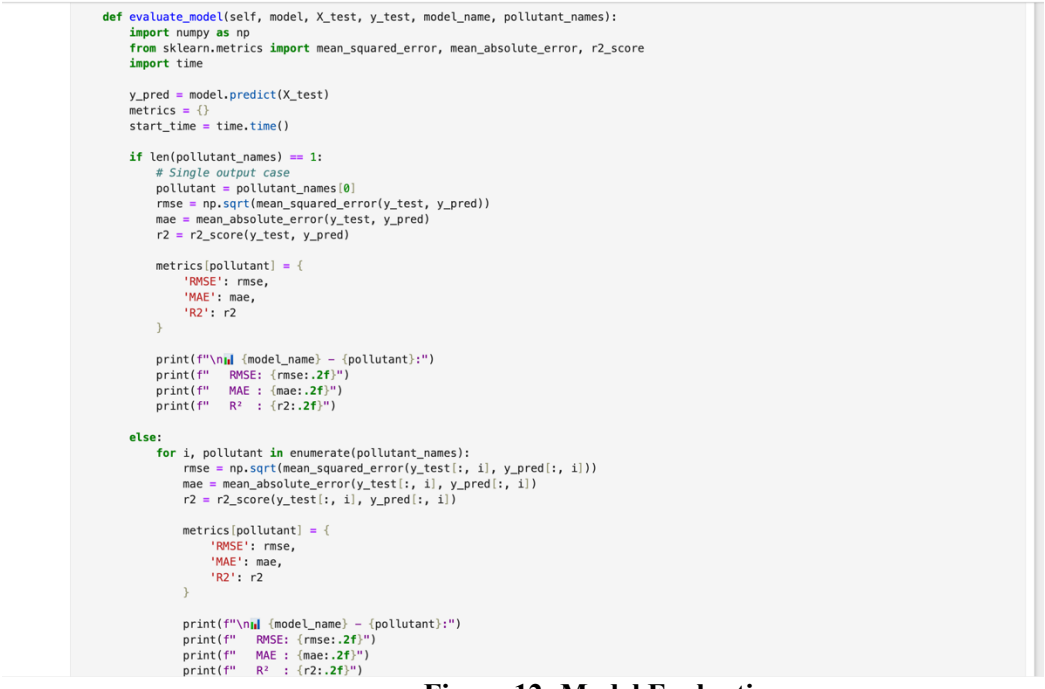
```
    def train_random_forest(self, X_train, X_val, X_test, y_train, y_val, y_test, pollutant_names):
        """Train Random Forest model"""
        print("🌲 Training Random Forest...")

        model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
        model.fit(X_train, y_train)

        self.evaluate_model(model, X_test, y_test, "Random Forest", pollutant_names)

        # Feature importance
        feature_importance = model.feature_importances_
        self.results["Random Forest"]['feature_importance'] = feature_importance

        return model

    def train_xgboost(self, X_train, X_val, X_test, y_train, y_val, y_test, pollutant_names):
        """Train XGBoost model"""
        print("🚀 Training XGBoost...")

        model = xgb.XGBRegressor(n_estimators=100, random_state=42, n_jobs=-1)
        model.fit(X_train, y_train)

        self.evaluate_model(model, X_test, y_test, "XGBoost", pollutant_names)

        # Feature importance
        feature_importance = model.feature_importances_
        self.results["XGBoost"]['feature_importance'] = feature_importance

        return model

    def train_arima(self, target_column='CO', order=(1, 1, 1)):
        """Train ARIMA model for single pollutant"""
        print(f"📈 Training ARIMA for {target_column}...")

        # Use original time series data
        ts_data = self.data[target_column].dropna()

        # Split data
        train_size = int(0.8 * len(ts_data))
        train_data = ts_data[:train_size]
        test_data = ts_data[train_size:]
```

Figure 11: Training other models

5.6 Model Evaluation & Results

All trained models were evaluated on the held-out test set using three key metrics:

- Root Mean Squared Error (RMSE) – penalizes larger errors more heavily.
- Mean Absolute Error (MAE) – average magnitude of errors.
- R^2 Score – proportion of variance explained by the model (closer to 1 is better).



```
    def evaluate_model(self, model, X_test, y_test, model_name, pollutant_names):
        import numpy as np
        from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
        import time

        y_pred = model.predict(X_test)
        metrics = {}
        start_time = time.time()

        if len(pollutant_names) == 1:
            # Single output case
            pollutant = pollutant_names[0]
            rmse = np.sqrt(mean_squared_error(y_test, y_pred))
            mae = mean_absolute_error(y_test, y_pred)
            r2 = r2_score(y_test, y_pred)

            metrics[pollutant] = {
                'RMSE': rmse,
                'MAE': mae,
                'R2': r2
            }

            print(f"\n📊 {model_name} - {pollutant}:")
            print(f"   RMSE: {rmse:.2f}")
            print(f"   MAE : {mae:.2f}")
            print(f"   R²  : {r2:.2f}")

        else:
            for i, pollutant in enumerate(pollutant_names):
                rmse = np.sqrt(mean_squared_error(y_test[:, i], y_pred[:, i]))
                mae = mean_absolute_error(y_test[:, i], y_pred[:, i])
                r2 = r2_score(y_test[:, i], y_pred[:, i])

                metrics[pollutant] = {
                    'RMSE': rmse,
                    'MAE': mae,
                    'R2': r2
                }

                print(f"\n📊 {model_name} - {pollutant}:")
                print(f"   RMSE: {rmse:.2f}")
                print(f"   MAE : {mae:.2f}")
                print(f"   R²  : {r2:.2f}")
```

Figure 12: Model Evaluation

```
        print("\n✅ ANALYSIS COMPLETE!")
        print("="*60)

        return results_df, models

# Example usage
if __name__ == "__main__":
    # Initialize the forecaster with your file path
    forecaster = AirPollutantForecaster('/Users/bolorsosorburam/Desktop/merged_result_final.csv')

    # Run complete analysis
    results_df, trained_models = forecaster.run_complete_analysis()

    # Display final results
    print("\n🎯 FINAL RESULTS:")
```

```
📊 MODEL EVALUATION REPORT
============================================================

📈 PERFORMANCE SUMMARY:
              Model       RMSE        MAE  R² Score  Inference Time (s)
Linear Regression  42.339568  33.140076  0.816660            0.000963
    Random Forest  52.215478  37.834174  0.721155            0.000577
          XGBoost  62.960461  46.238805  0.594585            0.000552
              SVM 124.691144  94.949211 -0.590141            0.001000
     ARIMA (pm25) 155.697235 122.481580 -1.337282            0.001000

🏆 BEST PERFORMING MODEL: Linear Regression
   • Highest R² Score: 0.8167
   • Lowest RMSE: 42.3396

💡 RECOMMENDATIONS:
   • Most Accurate: Linear Regression
   • Fastest Inference: XGBoost
```

Figure 13: Result of models

Linear Regression achieved the best overall performance with the lowest RMSE and MAE and the highest R^2 score, outperforming more complex models in this dataset.

```
[18]: models = ['Linear Regression', 'Random Forest', 'XGBoost', 'SVM', 'ARIMA (pm25)']
rmse_values = [forecaster.results[m]['metrics']['RMSE'] for m in models]

plt.figure(figsize=(10, 5))
bars = plt.bar(models, rmse_values, color=['#2ca02c','#ff7f0e','#d62728','#9467bd','#8c564b'])
plt.bar_label(bars, fmt='%.1f', padding=3, fontsize=10)
plt.title('Model Comparison by RMSE (Lower is Better)', fontsize=14, pad=15)
plt.xticks(rotation=45, ha='right')
plt.ylabel('RMSE', fontsize=12)
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```

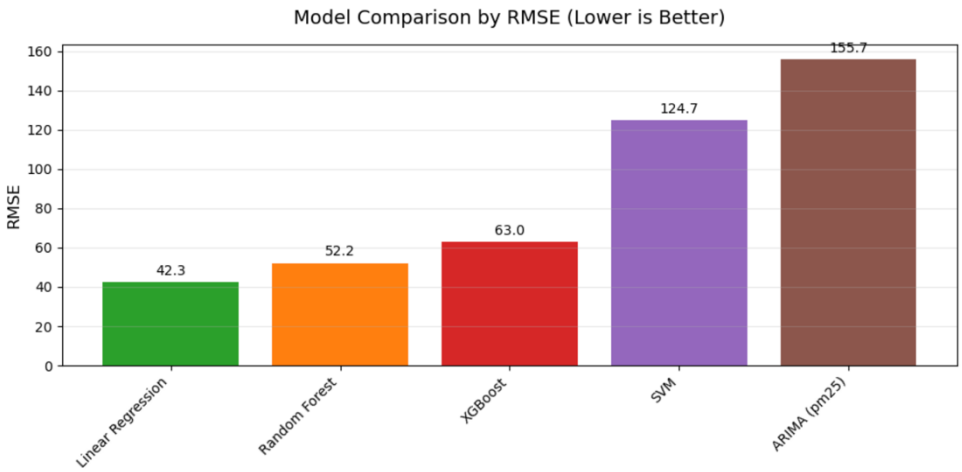


Figure 14: RMSE comparison bar chart

The first evaluation run of the Linear Regression model showed limited predictive accuracy, with an R^2 score of 0.366. As shown in Figure 15, predicted PM2.5 values were tightly clustered and deviated significantly from the perfect prediction line, indicating that the model was not effectively capturing the relationship between meteorological variables and PM2.5 concentrations.

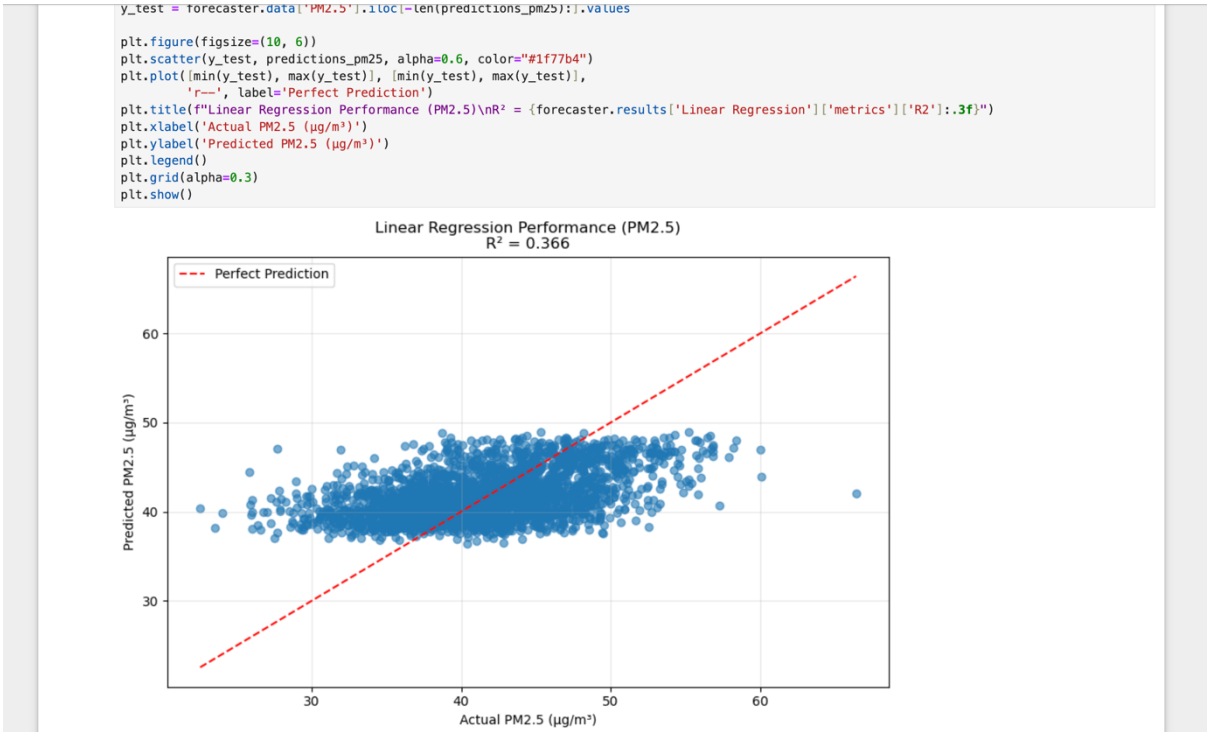


Figure 15. Linear Regression Predictions vs. Actual Values (Low R^2 Example)

After refining the preprocessing pipeline — including improved missing value handling with KNN imputation, selective feature scaling, and time-aware train–test splitting — the model's performance improved substantially. The final version (Figure 16 *scikit-learn*) achieved an R^2 score of 0.82 and an RMSE of 42.3 µg/m³, with predictions closely tracking the perfect prediction line, demonstrating the model's suitability for short-term PM2.5 forecasting in Ulaanbaatar.

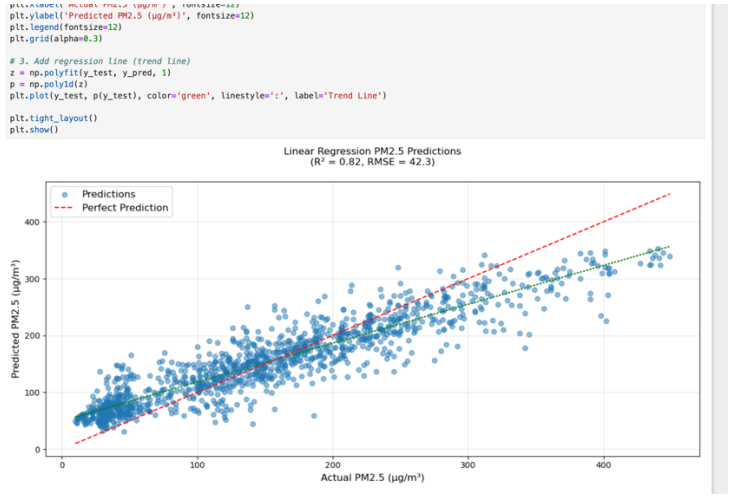


Figure 16. Linear Regression Predictions vs. Actual Values (Final Model)

6 Conclusion

This configuration manual presents the step-by-step process used to develop the AI-Driven Air Pollution Forecasting system for Ulaanbaatar. The document covered data acquisition from OpenAQ and Weather.com, preprocessing steps including missing value handling with KNN imputation, feature scaling, and train–validation–test splitting, followed by the implementation of multiple machine learning models such as Linear Regression, Random Forest, XGBoost, SVR, and ARIMA.

Evaluation results demonstrated that Linear Regression achieved the best overall performance on the test set, outperforming more complex models in this dataset. Feature importance analysis further provided insights into which variables most influenced PM2.5 predictions. By following the steps in this manual, the entire pipeline can be replicated for similar forecasting applications in other regions.

References

OpenAQ. (2025). Open Air Quality API. Retrieved from https://openaq.org/

Weather.com. (2025). Historical Weather Data. Retrieved from https://weather.com/

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794. https://doi.org/10.1145/2939672.2939785

Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: Principles and Practice (2nd ed.). OTexts. Retrieved from https://otexts.com/fpp2/