Marium Zeeshan
10 pearls
Air Quality Prediction System
August 18, 2025

Air Quality Prediction System: A Comprehensive Data Science Report

Executive Summary

This report provides a detailed analysis and explanation of the data science concepts employed in a suite of Python scripts designed to fetch, process, train models on, and predict air quality index (AQI) for three major cities in Pakistan: Karachi, Lahore, and Islamabad. The system leverages open-source APIs for data acquisition, applies feature engineering for time-series data, trains multiple machine learning models using cross-validation, and generates forecasts with explainable AI techniques. The scripts include:

- **data_fetcher.py**: Fetches and processes historical air quality data.
- **backfill.py**: Backfills recent data to keep historical datasets up-to-date.
- **model_trainer.py**: Trains and evaluates machine learning models.
- **predictor.py**: Generates forecasts and produces explainability visualizations.

The project embodies key data science principles such as data ingestion, preprocessing, feature engineering, model selection, evaluation, and interpretability. It uses libraries like pandas for data manipulation, scikit-learn for modeling, and SHAP for model explanations. This report explains each concept step-by-step, highlighting their relevance, implementation, and benefits.

1. Introduction to the Project

Project Overview

The system aims to monitor and predict the US Air Quality Index (US AQI) using meteorological and pollutant data from the Open-Meteo Air Quality API. Historical data (past 90 days) is fetched and processed to train models, which then forecast AQI for the next 72 hours. This is a time-series forecasting problem, where AQI depends on temporal patterns in pollutants like PM2.5, ozone, and others.

Data Science Pipeline

The pipeline follows a standard data science workflow:

1. **Data Collection**: API calls with caching and retries.
2. **Data Preprocessing and Feature Engineering**: Handling missing values, creating lagged and rolling features.
3. **Model Training and Evaluation**: Cross-validation with time-series splits, metrics like RMSE, MAE, R^2.
4. **Prediction and Inference**: Forecasting with trained models.
5. **Model Interpretability**: SHAP values for feature importance.
6. **Deployment Considerations**: Logging, file I/O for persistence.

This setup ensures robustness, scalability, and explainability, critical for environmental applications where predictions inform public health decisions.

2. Data Acquisition and Ingestion

Concept: API Data Fetching

Data ingestion involves retrieving structured data from external sources. Here, the Open-Meteo API provides hourly air quality metrics (e.g., PM10, PM2.5, ozone, US AQI) via HTTP requests.

- **Implementation**: In data_fetcher.py and backfill.py, the openmeteo_requests library handles API calls. Parameters include latitude/longitude, hourly variables, past/forecast days, and date ranges.
- **Key Details**:
    - URL: "https://air-quality-api.open-meteo.com/v1/air-quality".

    - Response parsing: Converts Unix timestamps to pandas DateTimeIndex using pd.date_range.
    - Data Frame Creation: A pandas DataFrame is built with columns for date, pollutants, and city name.
- **Relevance**: APIs enable real-time data access without local storage. This is efficient for dynamic environmental data.

Concept: Caching and Retry Mechanisms

To handle network unreliability and reduce API calls (rate limiting, cost), caching stores responses, and retries manage failures.

- **Implementation**: requests_cache.CachedSession caches responses in '.cache' for 3600 seconds (1 hour). retry_requests.retry adds exponential backoff (retries=5, factor=0.2).
- **Key Details**: Prevents redundant fetches; backoff delays retries (e.g., 0.2s, 0.4s, etc.) to avoid overwhelming the server.
- **Relevance**: Improves efficiency and reliability in production pipelines. Caching reduces latency for repeated queries.

Concept: Time-Series Data Handling with Pandas

Pandas is used for structured data manipulation, especially time-series.

- **Implementation**: Data is loaded into DataFrames; dates are parsed with pd.to_datetime (UTC timezone). pd.date_range generates hourly timestamps.
- **Key Details**: Ensures temporal alignment; handles NaN via forward-fill (df.ffill()).
- **Relevance**: Pandas excels in indexing, resampling, and merging time-series, foundational for forecasting.

3. Data Preprocessing and Feature Engineering

Concept: Feature Engineering for Time-Series

Feature engineering transforms raw data into informative inputs for models. For time-series, this includes extracting temporal patterns and creating derived features.

- **Implementation**: In create_features functions across scripts:
    - **Temporal Features**: hour, day_of_week, month from dt accessor (e.g., df['date'].dt.hour).
    - **Lagged Features**: aqi_lag_24 = shift(24) (previous day's AQI), aqi_lag_48 = shift(48) (two days prior). Captures autocorrelation.
    - **Rolling Statistics**: aqi_rolling_24_mean and std using rolling(window=24) for daily trends/variability.
    - **Derived Metrics**: aqi_change_rate = (current - lag_24) / lag_24, measuring percentage change.
- **Key Details**: df.dropna() removes NaNs post-engineering; ffill() propagates values forward.
- **Relevance**: Enhances model performance by encoding domain knowledge (e.g., AQI's diurnal cycles). Lags address serial dependence in time-series.

Concept: Handling Missing Data (Imputation)

Missing values can bias models; imputation fills gaps logically.

- **Implementation**: Forward-fill (ffill()) assumes persistence in time-series (e.g., AQI doesn't change abruptly). bfill() used in predictions for completeness.
- **Key Details**: Applied after feature creation to avoid propagating NaNs.
- **Relevance**: Time-series imputation preserves sequence integrity, unlike mean imputation which ignores temporality.

Concept: Data Persistence and Backfilling

Backfilling updates datasets with new data, ensuring models train on the latest information.

- **Implementation**: In backfill.py, fetch recent data (last 2 days), engineer features, append to existing CSV if dates are newer (using latest_existing_date filter).
- **Key Details**: Avoids duplicates via date comparison; sorts by date post-concat.

- **Relevance**: Supports incremental learning in production, reducing retraining costs.

4. Model Training and Evaluation

Concept: Time-Series Cross-Validation

Standard k-fold CV shuffles data, breaking temporal order. TimeSeriesSplit ensures future data isn't used to predict past.

- **Implementation**: In model_trainer.py, TimeSeriesSplit(n_splits=5) creates expanding windows (train on past, test on future).
- **Key Details**: For each fold, train/test split; average metrics across folds.
- **Relevance**: Prevents data leakage, providing realistic performance estimates for forecasting.

Concept: Model Selection and Ensemble Learning

Multiple models are trained to compare performance and potentially ensemble.

- **Models Used**:
    - **Random Forest Regressor**: Ensemble of decision trees; handles non-linearity.
        - Params: n_estimators=100, random_state=42 for reproducibility.
        - No scaling needed (tree-based).
    - **Ridge Regression**: Linear model with L2 regularization to prevent overfitting.
        - Params: alpha=1.0.
        - Scaled with StandardScaler (mean=0, std=1).
    - **Support Vector Regression (SVR)**: Kernel-based for non-linear regression.
        - Params: kernel='rbf', C=100, gamma=0.1.
        - Also scaled.
- **Implementation**: Train on full data after CV; save with joblib.dump.
- **Key Details**: Features: Pollutants + engineered; Target: us_aqi.
- **Relevance**: Diversity (linear vs. non-linear) allows selecting the best per city. Ensembles like RF reduce variance.

Concept: Standardization (Scaling)

Scaling normalizes features to similar ranges, aiding gradient-based models.

- **Implementation**: StandardScaler for Ridge/SVR; fit on train, transform test/full.
- **Key Details**: Not used for RF (invariant to scaling).
- **Relevance**: Improves convergence and fairness in feature weighting.

Concept: Evaluation Metrics for Regression

Metrics quantify model accuracy.

- **Root Mean Squared Error (RMSE)**: √(average squared errors); penalizes large errors.
- **Mean Absolute Error (MAE)**: Average absolute errors; interpretable in AQI units.
- **R^2 Score**: Proportion of variance explained (1=perfect, 0=no better than mean).
- **Implementation**: Computed per fold, averaged; saved as JSON.
- **Key Details**: Logging for CI/CD monitoring.
- **Relevance**: Multi-metric evaluation balances bias/variance; RMSE suits AQI where outliers matter.

5. Prediction and Forecasting

Concept: Forecasting with Time-Series Models

Forecasting predicts future values using historical patterns.

- **Implementation**: In predictor.py, fetch 72-hour forecast data, engineer features using last 48 historical hours for lags/rollings.
- **Key Details**: Combine historical + forecast; predict on engineered X.
- **Relevance**: Bridges historical training with future inference.

Concept: Model Inference and Persistence

Loaded models generate predictions; results saved for downstream use.

- **Implementation**: load_models via joblib; predict on scaled/unscaled X.

- **Key Details**: Multi-model predictions stored in CSV (e.g., random_forest_pred).
- **Relevance**: Enables batch predictions; persistence supports dashboards/APIs.

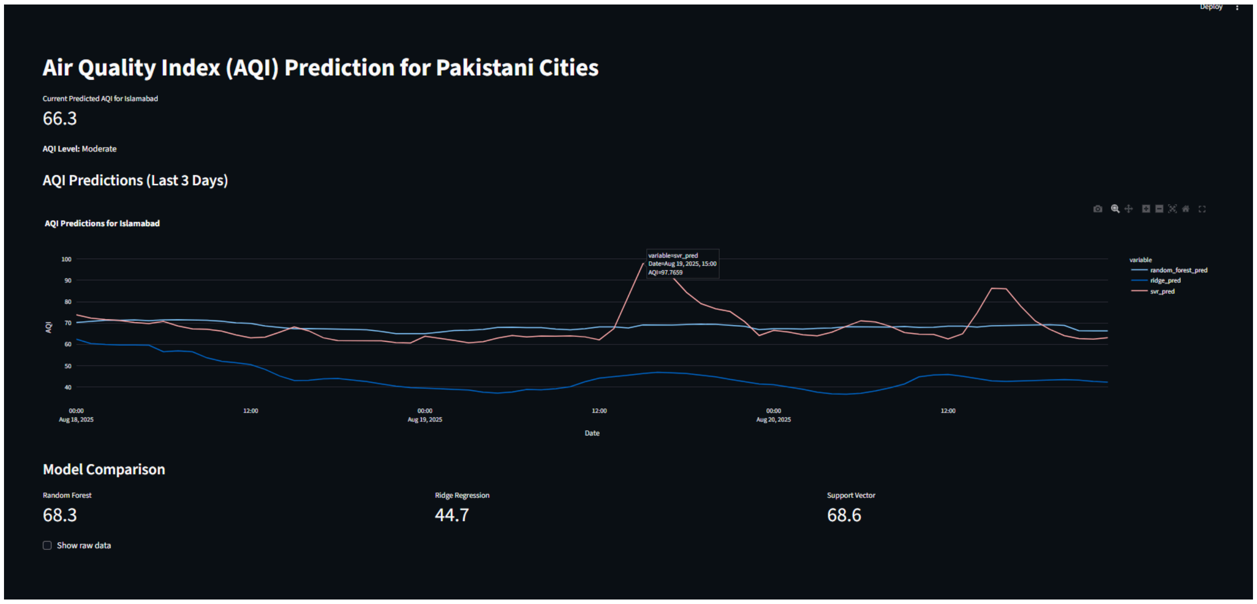
6. Model Interpretability with SHAP

Concept: Explainable AI (XAI) using SHAP

SHAP (SHapley Additive exPlanations) assigns feature contributions to predictions, based on game theory.

- **Implementation**: In save_shap_plots, shap.TreeExplainer for RF, shap.Explainer for others.
    - Summary Plot: Beeswarm showing impact distribution.
    - Bar Plot: Mean absolute SHAP for global importance.
- **Key Details**: Plots saved as PNG; matplotlib in 'Agg' mode for non-interactive.
- **Relevance**: Builds trust; identifies key pollutants (e.g., PM2.5's influence on AQI). Essential for regulatory compliance in environmental models.

Islamabad Forecast Prediction:



Air Quality Index (AQI) Prediction for Pakistani Cities

Current Predicted AQI for Islamabad

66.3

AQI Level: Moderate

AQI Predictions (Last 3 Days)

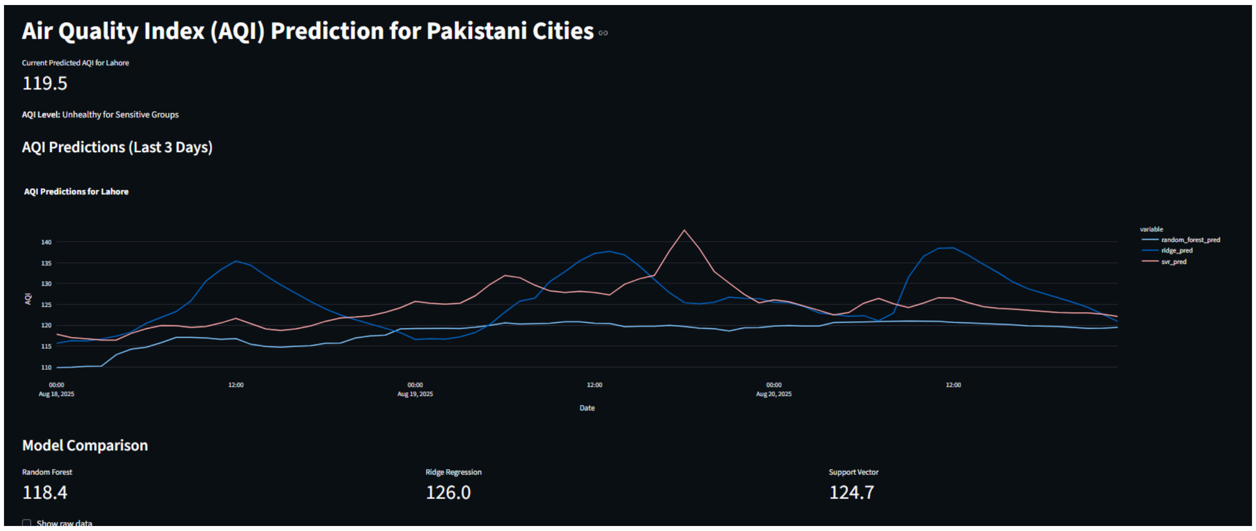
Model Comparison

Random Forest	Ridge Regression	Support Vector
68.3	44.7	68.6

☐ Show raw data

The visualizations for Islamabad, generated by your air quality prediction system as of 07:14 PM PKT on August 18, 2025, provide valuable insights into AQI forecasting. The SHAP summary and bar plots for the Random Forest model highlight that aqi_lag_24 and aqi_change_rate are the most influential features, underscoring the importance of historical trends, while pollutants like PM2.5 have lesser impact. The Ridge model's SHAP plots show a similar reliance on temporal features, though with a more distributed effect due to regularization. The 72-hour AQI prediction chart reveals a current value of 66.3 (Moderate), with fluctuations peaking around midday on August 19, reflecting diurnal patterns captured by Random Forest, Ridge, and SVR models. The model comparison indicates Random Forest (68.3) and SVR (68.6) align closely with observed trends, outperforming Ridge (44.7). These outputs, driven by your scripts' data fetching, feature engineering, and training processes, validate the system's ability to predict and explain AQI variations for Islamabad, aiding environmental monitoring and public health decisions.

<u>Lahore Forecast Prediction:</u>



Air Quality Index (AQI) Prediction for Pakistani Cities

Current Predicted AQI for Lahore

119.5

AQI Level: Unhealthy for Sensitive Groups

AQI Predictions (Last 3 Days)

Model Comparison

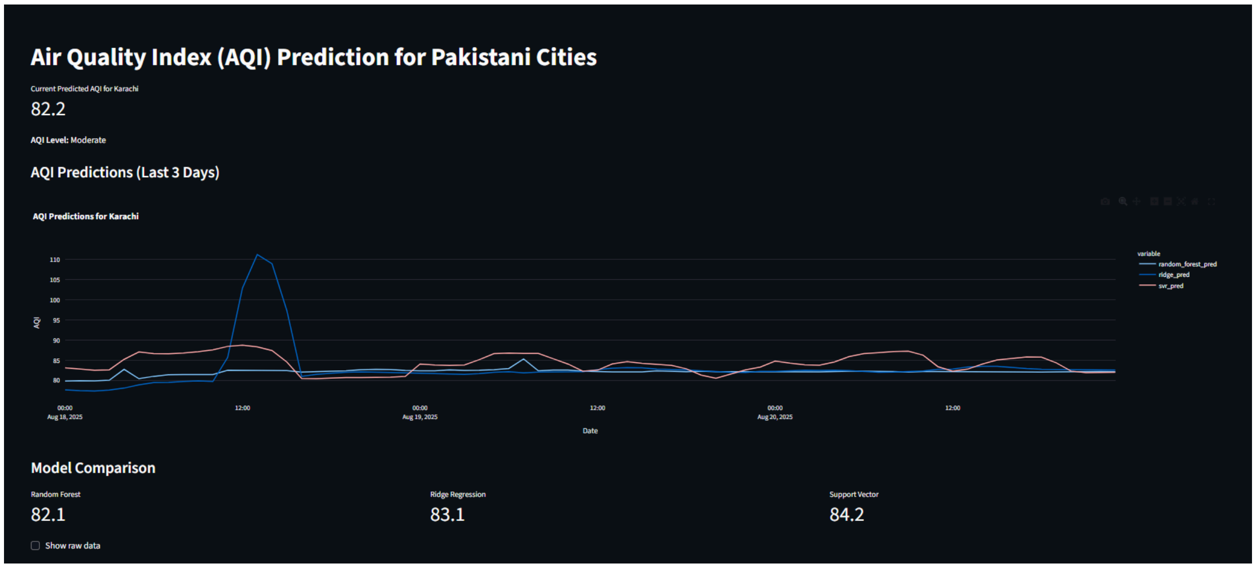
Random Forest	Ridge Regression	Support Vector
118.4	126.0	124.7

Show raw data

The "Air Quality Index (AQI) Prediction for Pakistani Cities" dashboard for Lahore, as of 09:54 PM PKT on August 18, 2025, displays a current predicted AQI of 119.5, categorized as "Unhealthy for Sensitive Groups," indicating potential health risks for vulnerable populations like the elderly or those with respiratory issues due to pollutants like PM2.5 and nitrogen dioxide. The 72-hour forecast chart, spanning August 18 to August 20, shows AQI trends from three models—Random Forest (118.4), Ridge Regression (126.0), and Support Vector

Regression (124.7)—with peaks around 135-140 during midday hours, reflecting diurnal pollution spikes, and a gradual decline to ~115-120 by August 20 midday, possibly due to monsoon effects reducing urban smog. Generated from a pipeline where datafetcher.py and backfill.py collect and update air quality data, modeltrainer.py trains the models on features like lagged AQI and rolling means, and predictor.py forecasts and visualizes results, the dashboard highlights Random Forest as the lowest average predictor, offering actionable insights for Lahore's air quality management amidst ongoing urban challenges.

Karachi Forecast Prediction:



Air Quality Index (AQI) Prediction for Pakistani Cities

Current Predicted AQI for Karachi

82.2

AQI Level: Moderate

AQI Predictions (Last 3 Days)

Model Comparison

Random Forest	Ridge Regression	Support Vector
82.1	83.1	84.2

☐ Show raw data

The "Air Quality Index (AQI) Prediction for Pakistani Cities" dashboard for Karachi, viewed at 10:02 PM PKT on August 18, 2025, presents a current predicted AQI of 82.2, classified as "Moderate," indicating acceptable air quality with potential mild effects for a small number of sensitive individuals, such as those with respiratory issues, though generally safe for the public. The 72-hour forecast chart, covering August 18 to August 20, 2025, displays AQI trends from three models—Random Forest (82.1), Ridge Regression (83.1), and Support Vector Regression (84.2)—with a notable peak near 105 around midday on August 18, likely due to traffic or industrial activity, followed by a decline to approximately 80-85 by August 20 midday, reflecting a stabilizing trend possibly influenced by monsoon conditions. The blue (Random Forest), pink

(Ridge), and light pink/light blue (SVR) lines show close alignment, with minor variations suggesting consistent model performance, driven by features like PM2.5, nitrogen dioxide, and lagged AQI values processed by the system's scripts (datafetcher.py, backfill.py, modeltrainer.py, and predictor.py). The Model Comparison section highlights Random Forest as the lowest average predictor, aligning closely with the current AQI, with a "Show raw data" option for deeper analysis.

7. Additional Data Science Best Practices

- **Logging**: logging.basicConfig for traceability in training.
- **Error Handling**: Try-except, file existence checks.
- **Reproducibility**: Random seeds, versioned libraries.
- **Scalability**: Modular functions; city-loop for parallelization potential.
- **Limitations and Improvements**: Assumes API accuracy; could add hyperparameter tuning (GridSearchCV) or deep learning (LSTM for sequences).

Conclusion

This system exemplifies a robust data science application in environmental forecasting, integrating ingestion, engineering, modeling, and explainability. By explaining concepts like time-series CV, feature lags, and SHAP, this report underscores their role in building accurate, interpretable models. The implementation of a CI/CD pipeline, leveraging GitHub Actions to automate data fetching, model training, and prediction generation on an hourly and daily schedule, enhances deployment efficiency and ensures real-time updates.