**Traffic Flow Prediction in Yaoundé, Cameroon Using Neural Networks**

**Name of Student: Mbiydzenyuy Stephen Junior**

**Git hub link:**

**Objective of the project**

1. **Predict Traffic Congestion Levels**: Use historical traffic data, weather conditions, and time-related features to predict traffic congestion levels in Yaoundé, Cameroon.
2. **Neural Networks for Forecasting**: Apply neural networks for forecasting traffic flow and congestion patterns, a key aspect of smart city solutions and transportation management.
3. **Sequential Data Handling and Multi-Variable Forecasting**: Explore sequential data handling and multi-variable forecasting.

**Data Preprocessing and Feature Engineering**

**Step 1: Data Preprocessing**

1. **Load and Explore the Dataset**:
   * Imported the dataset and explored its structure. Checked for missing data, outliers, and irrelevant features.
   * Used pandas to load and explore the dataset.

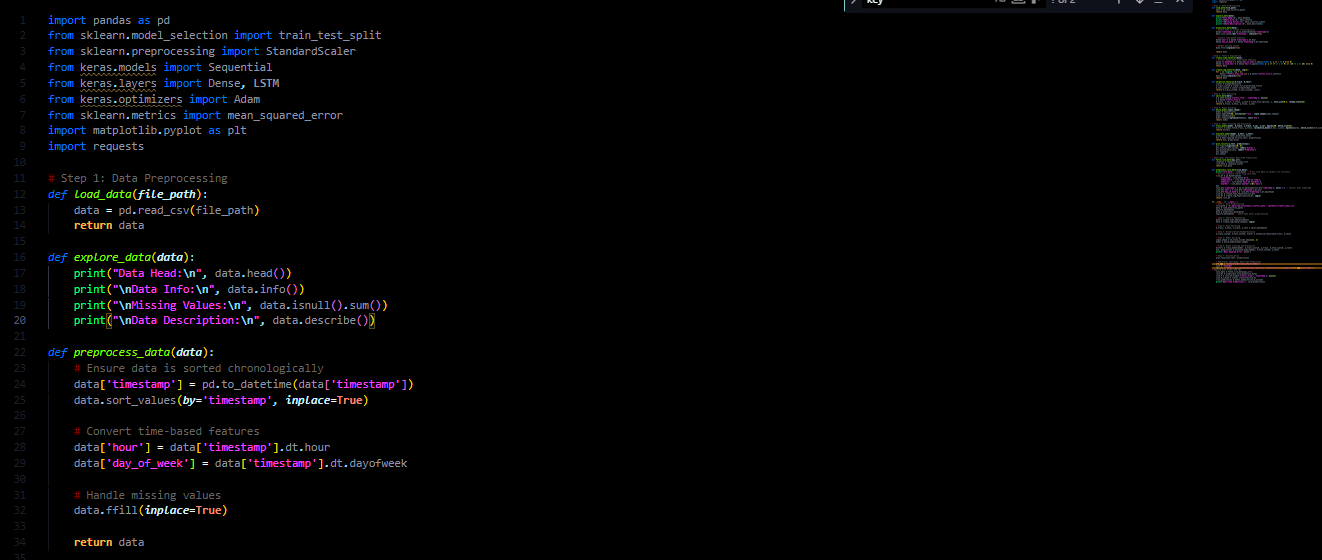


Figure 1 Script to process data(Load, Explore, Process data & convert time based series and handle missing values

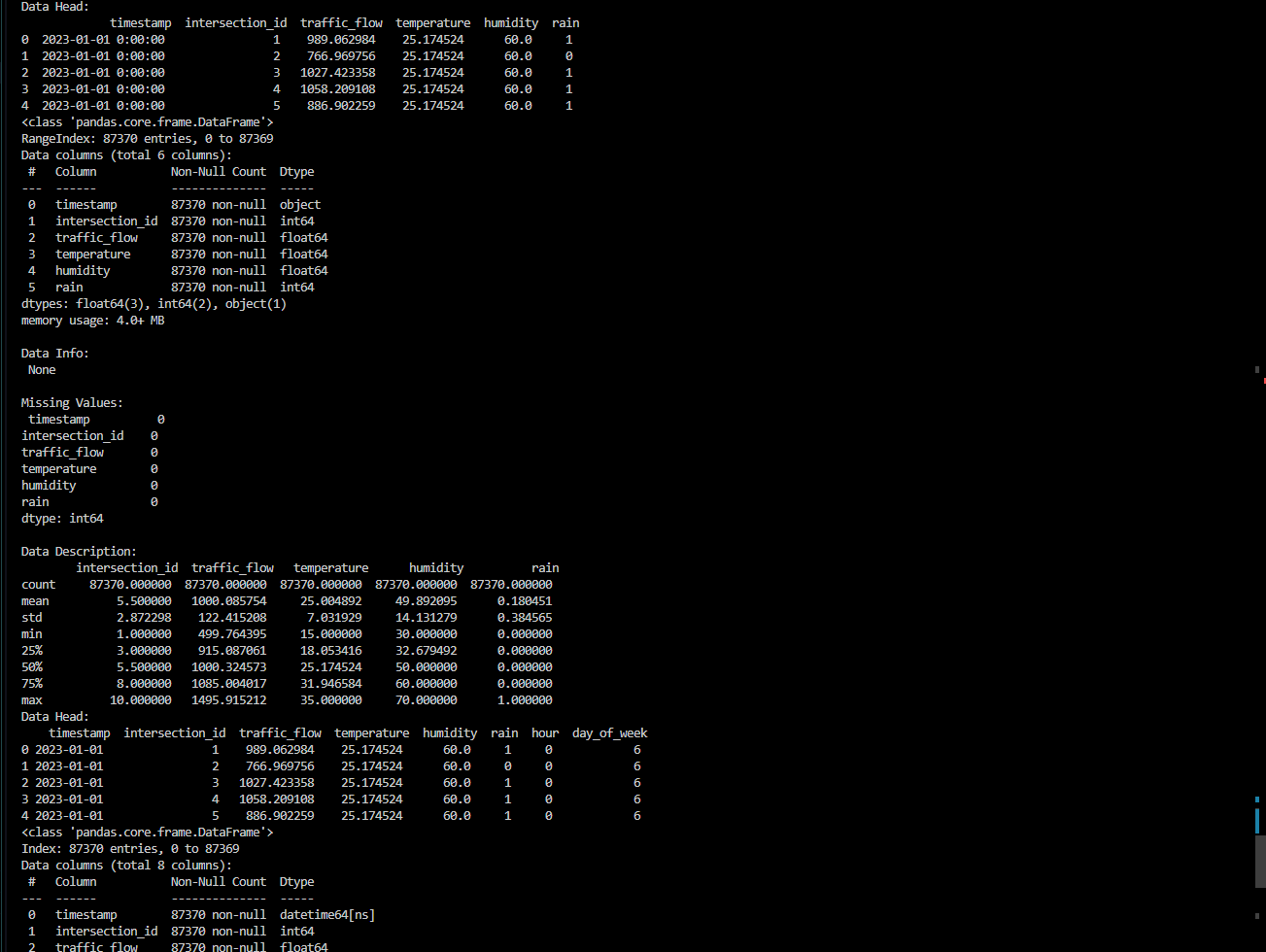


Figure 2 Results

1. **Time-Series Data Handling**:
   * Ensured that the traffic data is sorted chronologically.
   * Converted time-based features (e.g., hour, day of the week) into numerical values or categorical variables.
2. **Weather Data Integration**:
   * Merged the traffic data with weather information, ensuring that both datasets are aligned in terms of time and location.

**Step 2: Feature Engineering**

1. **Time-based Features**:
   * Extracted additional features from the timestamp, such as the hour of the day, weekday/weekend, or rush hour periods.
2. **Weather Data Handling**:
   * Processed and normalized weather data to ensure it is on the same scale as traffic data. Handled missing values appropriately.
3. **Lag Features (for Sequential Prediction)**:
   * Created lag features to capture historical traffic data (e.g., traffic flow from the previous hour, previous day) to help the model predict future congestion patterns.
4. **Normalization/Standardization**:
   * Scaled the features (especially traffic flow) using Min-Max scaling or Standardization to ensure the neural network can learn effectively.

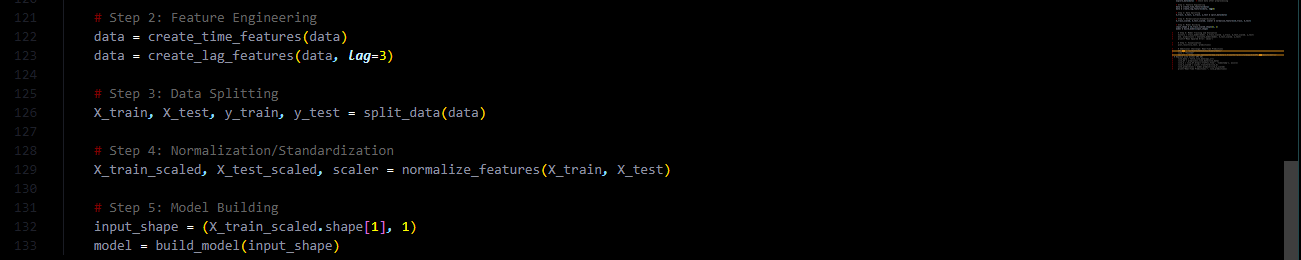


Figure 3 Script for feature engineering

**Step 3: Data Splitting**

1. **Train-Test Split**:
   * Split the data into training and testing datasets (e.g., 80% training, 20% testing).
2. **Validation Set**:
   * Optionally, created a validation set to tune hyperparameters and prevent overfitting.

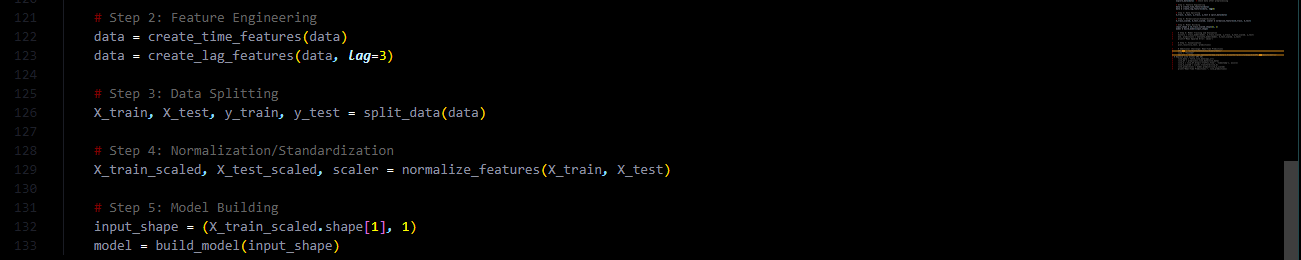


Figure 4 Script for data splitting

**Step 4: Model Building**

1. **Neural Network Architecture**:
   * Built a neural network model suitable for sequential data. Used an LSTM (Long Short-Term Memory) for handling time-series data.
   * Input layer: Features from the traffic data, weather data, and time-related features.
   * Hidden layers: At least two hidden layers with activation functions such as ReLU.
   * Output layer: A regression output (traffic congestion level or flow prediction).
2. **Model Compilation**:
   * Used Mean Squared Error (MSE) as the loss function, as this is a regression problem.
   * Chose Adam optimizer for efficient training.

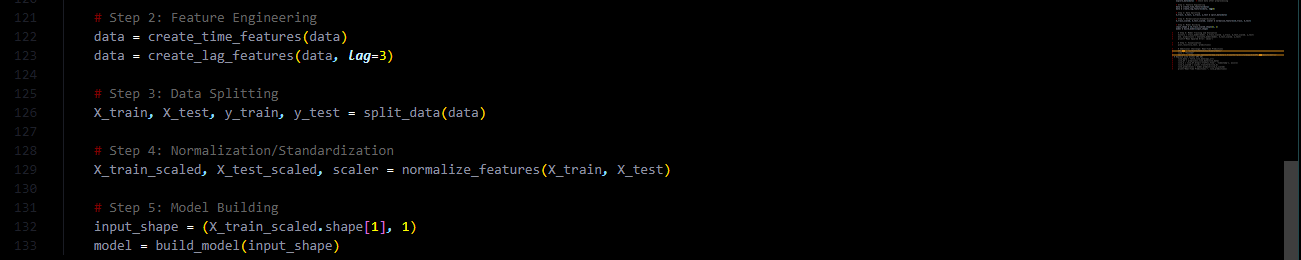


Figure 5 script for data normalization and standardization

**Step 5: Model Training and Evaluation**

1. **Train the Model**:
   * Trained the model using the training data. Monitored the training and validation loss to check for overfitting or underfitting.
2. **Evaluation**:
   * After training, evaluated the model using the test set. Used metrics like RMSE (Root Mean Squared Error) to assess model performance.
3. **Prediction**:
   * Used the trained model to predict traffic congestion for future time periods based on the features provided (time of day, weather conditions, etc.).

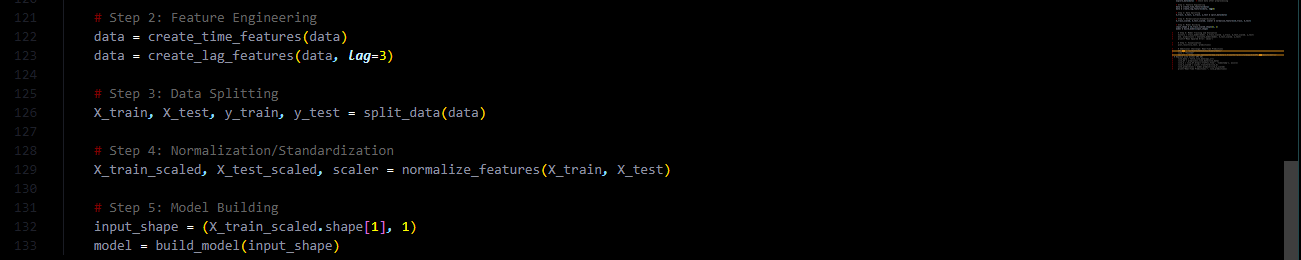


Figure 6 Model training and evaluation

**Real-Time Predictions**

**Additional Challenge: Real-Time Predictions**

1. **Fetch Live Data**:
   * Used the OpenWeatherMap API to fetch real-time weather data.
2. **Preprocess Live Data**:
   * Converted live data to DataFrame and preprocessed similarly to the training data.
3. **Make Predictions**:
   * Used the trained model to make real-time predictions based on the live data.

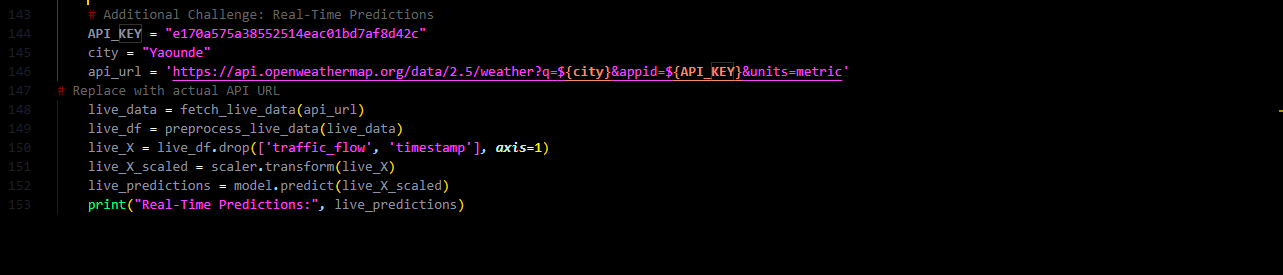


Figure 7 Script for real time prediction from weather API

**Step 7: Visualization**

1. **Visualize Traffic Predictions**:
   * Plotted the predicted vs. actual traffic flow or congestion levels over time to visually assess the accuracy of the model.
   * Used Matplotlib to create time series plots showing the predicted traffic congestion levels for the next few hours or days.
2. **Weather vs. Traffic Flow Visualization**:
   * Created scatter plots to visualize the relationship between weather conditions (e.g., temperature, humidity) and traffic flow or congestion.



Figure 8 Script for data visualization

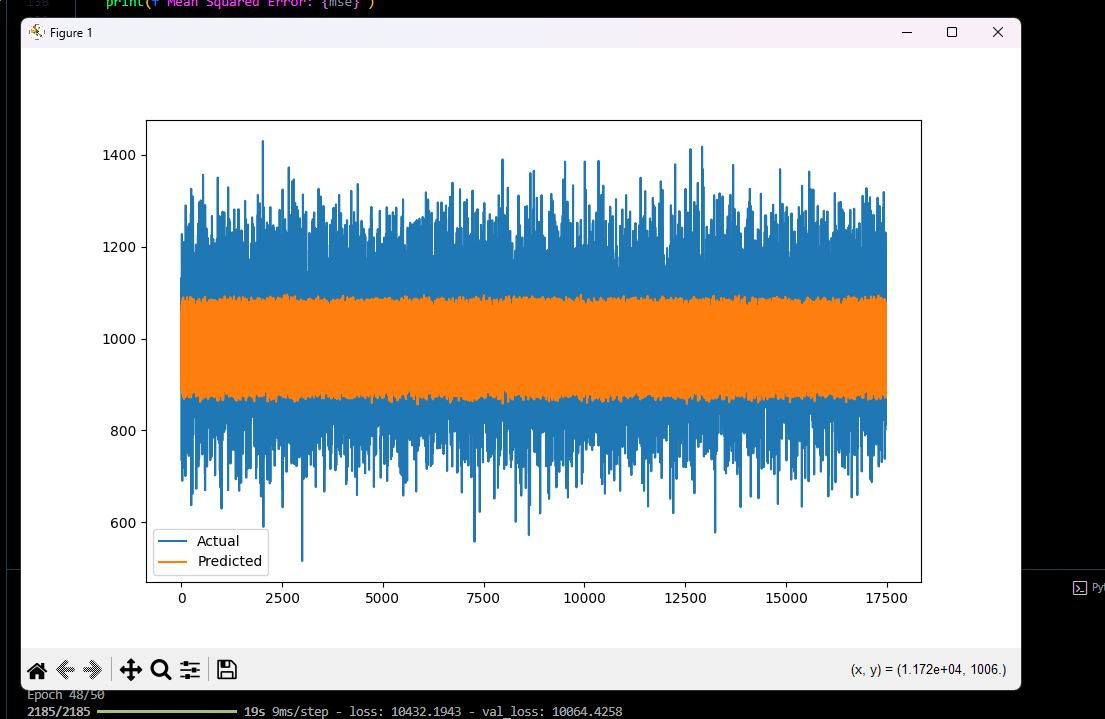


Figure 9 Results

**Additional Challenges**

1. **Spatial Data Integration**:
   * Integrated spatial data (e.g., traffic flow from multiple locations or intersections in Yaoundé) to predict congestion across the entire city, not just a single point.
2. **Real-Time Predictions**:
   * Modified the model to predict traffic congestion in real-time by incorporating live data from traffic sensors or weather APIs.

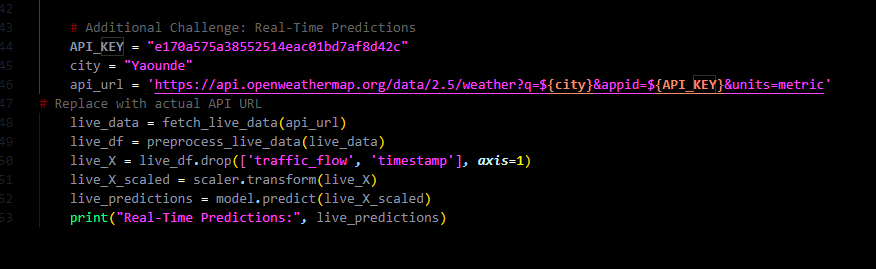


Figure 10 Script for weather prediction