AI-Enhanced Public Transit Demand & Delay Prediction in Toronto

Course: AIDI1011 – AI Project

Institution: Georgian College

Professor: Sanjeev Kumar

Team Members: Dejina Ghale (Project Head), Nawaraj (Technical Lead), Shishir Aryal (Data Scientist), Santosh (Data Engineer)

Date of Submission: August 13, 2025

# Abstract

Predicting demand trends and delays is crucial for modern urban transport in order to maximize scheduling, resource allocation, and passenger happiness. In order to forecast delay categories and predict demand fluctuations, the AI-Enhanced Public Transit Demand & Delay Prediction in Toronto project makes use of historical TTC Subway Delay data, meteorological data, and temporal factors.

We used cutting-edge machine learning techniques like Prophet for time-series demand forecasting and XGBoost Classifier for delay classification, according to the Software Development Life Cycle (SDLC) methodology from data collection and preprocessing to model deployment. The solution was incorporated into Power BI for interactive visualization and implemented as a web application based on Flask for real-time predictions.

In order to support operational planning, our final model produced demand trend estimates and classified delays into three categories with excellent accuracy: short, medium, and long. The findings show that there is a great deal of room to increase the public transport system in Toronto's efficiency and dependability.

Contents

[Abstract 2](#_Toc205807804)

[1. Introduction 6](#_Toc205807805)

[2. Literature Review 6](#_Toc205807806)

[2.1. Public Transit Delay Analysis in Existing Research 6](#_Toc205807807)

[2.2. Machine Learning Approaches 7](#_Toc205807808)

[2.3. Data Sources and Feature Engineering 7](#_Toc205807809)

[2.4. Gaps in Existing Research 7](#_Toc205807810)

[2.5. Positioning of This Project 8](#_Toc205807811)

[3. Planning Phase 8](#_Toc205807812)

[3.1. Defining Objectives 8](#_Toc205807813)

[3.2. Scope Definition 8](#_Toc205807814)

[3.3. Resource Identification 9](#_Toc205807815)

[3.4. Timeline and Milestones 9](#_Toc205807816)

[3.5. Risk Assessment 9](#_Toc205807817)

[4. Requirements Analysis 10](#_Toc205807818)

[4.1. Functional Requirements 10](#_Toc205807819)

[4.2. Non-Functional Requirements 10](#_Toc205807820)

[4.3. Technical Requirements 11](#_Toc205807821)

[4.4. Constraints & Assumptions 12](#_Toc205807822)

[5. Design Phase 12](#_Toc205807823)

[5.1. System Architecture Overview 12](#_Toc205807824)

[5.2. Data Flow 13](#_Toc205807825)

[5.3. Neural Network Design 14](#_Toc205807826)

[5.4. User Interface Design 14](#_Toc205807827)

[5.5. Deployment Design 15](#_Toc205807828)

[6. Implementation Phase 15](#_Toc205807829)

[6.1. Data Preparation and Preprocessing 15](#_Toc205807830)

[6.2. Neural Network Model Training 16](#_Toc205807831)

[6.3. Flask Application Development 16](#_Toc205807832)

[6.4. Visualization Integration 17](#_Toc205807833)

[6.5. Testing and Debugging 17](#_Toc205807834)

[6.6. Deployment Readiness 18](#_Toc205807835)

[7. Testing Phase 18](#_Toc205807836)

[7.1. Testing Objectives 18](#_Toc205807837)

[7.2. Types of Testing Conducted 19](#_Toc205807838)

[7.3. Model Evaluation 19](#_Toc205807839)

[7.4. Confusion Matrix Analysis 20](#_Toc205807840)

[7.5. Web Application Testing 20](#_Toc205807841)

[7.6. Key Findings from Testing 21](#_Toc205807842)

[7.7. Testing Tools Used 21](#_Toc205807843)

[7. Deployment Phase 21](#_Toc205807844)

[7.1. Deployment Objectives 21](#_Toc205807845)

[7.2. Deployment Steps 22](#_Toc205807846)

[7.3. Deployment Environment 22](#_Toc205807847)

[7.4. Challenges in Deployment 23](#_Toc205807848)

[7.5. User Accessibility 23](#_Toc205807849)

[7.6. Future Deployment Plans 23](#_Toc205807850)

[9. Maintenance and Updates 24](#_Toc205807851)

[9.1. Importance of Maintenance 24](#_Toc205807852)

[9.2. Planned Maintenance Activities 24](#_Toc205807853)

[9.3. Update Schedule 25](#_Toc205807854)

[9.4. Change Management Process 25](#_Toc205807855)

[9.5. Future Update Enhancements 26](#_Toc205807856)

[10. Project Budget Overview 26](#_Toc205807857)

[10.1. Purpose of Budgeting 26](#_Toc205807858)

[10.2. Resource Breakdown 26](#_Toc205807859)

[10.3. Estimated Future Deployment Costs 27](#_Toc205807860)

[10.4. Cost-Saving Strategies 27](#_Toc205807861)

[11. Future Work 27](#_Toc205807862)

[1. Integration of Real-Time Data Sources 28](#_Toc205807863)

[2. Expansion to Multi-Line and Multi-Modal Transit Systems 28](#_Toc205807864)

[3. Improved Model Architecture and Feature Engineering 29](#_Toc205807865)

[4. Advanced Evaluation Metrics and Error Analysis 29](#_Toc205807866)

[5. Personalized Delay Alerts and Commuter Applications 30](#_Toc205807867)

[6. Integration with Visualization Dashboards 30](#_Toc205807868)

[7. Model Retraining and Continuous Learning Pipeline 30](#_Toc205807869)

[8. Collaboration with TTC for Operational Deployment 31](#_Toc205807870)

[12. Tools and Technologies 31](#_Toc205807871)

[12.1. Programming Language 31](#_Toc205807872)

[12.2. Development Environment 32](#_Toc205807873)

[12.3. Data Analysis and Processing Libraries 32](#_Toc205807874)

[12.4. Machine Learning & Deep Learning Frameworks 32](#_Toc205807875)

[12.5. Visualization Tools 32](#_Toc205807876)

[12.6. Deployment Tools 33](#_Toc205807877)

[12.7. Version Control and Collaboration 33](#_Toc205807878)

[12.8. Data Sources 33](#_Toc205807879)

[12.9. Reason for Tool Selection 33](#_Toc205807880)

[13. Appendices 33](#_Toc205807881)

[13.1. Appendix A – Dataset Details 33](#_Toc205807882)

[13.2. Appendix B – Model Summary 34](#_Toc205807883)

[13.3. Appendix C – Gantt Chart 34](#_Toc205807884)

[Explanation of Timeline 35](#_Toc205807885)

[13.4. Appendix D – Flask API Endpoint Structure 35](#_Toc205807886)

[14. References 36](#_Toc205807887)

## ****1. Introduction****

Public transport arrangements are essential for the smooth movement of local people, staff, and tourists in large towns. However, delay may lead to monetary losses, interference with the common agenda, and reduced commuting satisfaction. For improving travel planning, streamlining operations, and promoting mass safety in Toronto, where the Toronto Transit Commission (TTC) operates the metro system, the ability to accurately project delays is of vital importance.

The objective of the AI-Enhanced Population Transit Delay Prediction (Toronto) project is to project the probability and magnitude of future delays using past TTC metro delay facts together with created variables including time of day, weekday, peak hour indication, and weather. Our aim is to categorize hand delay within brief, medium, and long collectives using machine learning strategies, in particular a neural connection categorization model.

Users can select stations, dates, and spans using Flask-based web usage which maintains the structure. Then, for each delay class, they acquire a prediction and probability distribution. The incorporation by the undertaking of the Authority BI splashboard for ocular direction inspection shall give a comprehensive view of the delay exceeding the era.

Although the present version has been designed for educational use, its modular architecture, design standards, and methodology make it suitable for everyday use. The main objective is to empower theodolite officials and commuters to produce a better opinion by integrating real-time TTC API data and improving predictive skills in addition to real-time updates.

## ****2. Literature Review****

Predicting public transit delays will become an increasingly important exploration area, especially in large metropolis, where productivity and reliability are essential for traveler Satisfaction. Prior analyses have demonstrated the value of uniting olden theodolite statistics with current machine learning techniques for recognizing forms, prediction breaks, and optimizing service agendas.

### ****2.1. Public Transit Delay Analysis in Existing Research****

Transit delay model used to be mainly based on statistical and regression analysis. For instance, a multivariate linear arrested development model was applied to separate the correlation of the duration of the delay and components such as extremum hours, weather conditions, and work constraints. As identical approaches offer interpretability, their anticipatory accuracy has been limited due to their inability to capture complex, nonlinear connections inherent in theodolite systems.  
  
With the arrival of massive facts statistical analysis, experts have begun using large-scale old datasets to develop more sophisticated anticipation models. Despite fighting with categorical, event-driven variables such as incident type or station-specific active boundaries, time series estimation systems such as ARIMA and Prophet are effective in capturing seasonal and transient trends.

### ****2.2. Machine Learning Approaches****

Machine learning and deep learning models of knowledge acquisition have emerged as more efficient methods of delay prediction in recent eras. The investigation has revealed that the procedures consider Random Forests, Gradient Boosters, for instance. XGBoost), and artificial Neural Partnerships (ANNs) can learn complex interactions among elements, which are suitable for urban theodolite datasets.  
  
For example:  
  
Li et al. In 2019, a Gradient Boosting technique was applied for metro delay statistics in Peking, achieving more than 85% accuracy for predicting delay types.  
  
Zhou et al. ( 2021 ) Investigate deep neural networks for subway delay prediction in Shanghai, integrating weather, ridership, and event data which significantly improves the accuracy of the forecast.  
  
These surveys highlight the advantages of using a categorization model together with an engineered feature allowing farinaceous prediction across different stations, instances, and days.

### ****2.3. Data Sources and Feature Engineering****

The value of feature technology for improving the accuracy of the prediction is recurrence detection in fiction. In a high-performance model, factors identical to daytime, weekday, seasonality, incident type, and line-specific features are systematically observed.  
  
For instance, the search performed on the fresh New York metropolis Subway delay dataset, which integrates weather forms and a vacation index, improved accuracy by 6–8 percent. In addition, analysis of the London Underground shows that spatial elements such as network connectivity and line congestion enhance the generalizability of the model.

### ****2.4. Gaps in Existing Research****

While prior studies have provided a strong foundation, several limitations remain:  
  
• Lack of real-time integration: mainly models use ancient information completely missing webcasting update from the live feed.  
  
• Station-level granularity: Many studies aggregate delays at a system-wide level, limiting station-specific insights.  
  
• Explainability challenges: Deep learning models often lack interpretability, which can hinder adoption by transit authorities.

### ****2.5. Positioning of This Project****

Our study adds to the existing literature by applying a neural network-based categorization model train over past TTC delay facts, together with engineered features identical to station, weekday, hour, peak hour index, and delay type. The objective shall be to provide station-level, time-specific forecasts which are simultaneously precise and simple to access via the web and the authority BI display. The present double interface bridges the gap between high-tech statistical analysis and user-friendly commuting devices, ensuring feasible understandings between the theodolite planner and everyday commuter.

## ****3. Planning Phase****

In laying the foundations for the successful execution of the TTC Delay Prediction project, the strategizing phase was crucial. Specifying the intentions, understanding the scope, allocation of resources, and drawing up a method that ensured timely delivery while maintaining the premium quality of the track was part of the current situation.

### ****3.1. Defining Objectives****

The following were the main goals set during the planning stage:  
  
Developing a model based on artificial intelligence that accurately predicts TTC train delays.  
  
Creating forecasts related to commuter and theodolite planners based on supply station and time data.  
  
For real-time accessibility, improve the user interface with Flask, using Flask's Web Usage Framework.  
  
• Include a Power BI dashboard for historical analysis and interactive visualization.  
  
These objectives were in line with both academic standards and practical usability factors.

### ****3.2. Scope Definition****

The scope of the project was defined to include:

* Data acquisition from the City of Toronto Open Data Portal.
* Data preprocessing and characteristic technology, including the production of categorical indexes such as day type (weekday/weekend), peak time, and weather correction.
* Model training and evaluation using neural networks and other machine learning algorithms for comparative purposes.
* Deployment via a Flask application and integration with visualization tools.

Due to asset and interval constraints, the scope does not include real-time API integration with TTC's live feed, but supplies will be made for upcoming integration in the future improvements.

### ****3.3. Resource Identification****

* key resources identified during this phase included:  
    
  • Hardware: Personal laptops with adequate processing power for model training.  
    
  • Software: Python (TensorFlow, Pandas, Scikit-learn), Flask, Power BI, and open-source visualization libraries.
* **Human Resources**: Project members with designated roles in data preparation, model building, web development, and visualization.

### ****3.4. Timeline and Milestones****

A structured timeline was created to ensure the project stayed on track, with milestones such as:

1. **Week 1–2**: Data collection and exploration.
2. **Week 3–4**: Data preprocessing and feature engineering.
3. **Week 5–6**: Model development and tuning.
4. **Week 7**: Model evaluation and comparison.
5. **Week 8**: Flask application development and dashboard integration.
6. **Final Week**: Testing, documentation, and presentation preparation.

### ****3.5. Risk Assessment****

Potential risks identified during planning included:  
  
• Data quality problems, such as missing or inconsistent values, which could decrease model accuracy.  
  
• Model overfitting due to high feature dimensionality.  
  
• Time constraints during integration and deployment stages.  
  
Extensive pre-processing techniques, regularization techniques, and incorporation of an iterative testing method were applied in order to address concerns in advance.

## ****4. Requirements Analysis****

The requirements analysis phase focused on identifying and documenting the functional, non-functional, and technical specifications essential for the successful completion of the TTC Delay Prediction project. This step ensured that all stakeholders, including the development team and potential end-users, had a clear understanding of what the system needed to achieve and the constraints within which it would operate.

### ****4.1. Functional Requirements****

These describe what the system should do in order to meet its objectives:

1. **Data Handling**
   * Import TTC subway delay data from the City of Toronto Open Data Portal.
   * Support preprocessing operations, including missing value handling, encoding of categorical features, and feature scaling.
2. **Prediction Engine**
   * Use a trained neural network model to classify delays into one of three categories: **Short**, **Medium**, or **Long**.
   * Generate station-specific and time-specific predictions for the upcoming hours of the day.
   * Display prediction probabilities for each class alongside the predicted outcome.
3. **User Interface**
   * Provide a **web form** to input date, time, and station.
   * Display prediction results in a clear, visually enhanced format (color-coded by severity).
4. **Visualization**
   * Integrate Power BI or a similar tool for historical trends and comparative analysis.
   * Enable filtering by station, time range, and delay category.

### ****4.2. Non-Functional Requirements****

These define the performance and quality standards for the system:

1. **Accuracy**
   * Target a model accuracy above **90%** on validation data to ensure reliable results.
2. **Usability**
   * Ensure the interface is intuitive and accessible, even for users without technical expertise.
3. **Performance**
   * Model inference time should be under **1 second** for each prediction request on a standard personal computer.
4. **Scalability**
   * Design the solution so it can be deployed on cloud platforms (AWS, Azure, or GCP) if needed.
5. **Maintainability**
   * Maintain modular code with proper documentation to facilitate updates and bug fixes.

### 4.2. Non-Functional Requirements:

These define the performance and quality standards for the system:

* Accuracy- The target accuracy of the model is above 90% on authentication data to ensure reliable output.
* Usability- Ensure the interface is intuitive and accessible, even for users without technical expertise.
* Performance- The model inference time should not exceed 1 following the individual prediction request on a standard computer.
* Scalability- Oxygen Design The solution can therefore be deployed on cloud platforms (AWS, Azure, or GCP, if necessary).
* Maintainability- Maintain modular code with proper documentation to facilitate updates and bug fixes.

### ****4.3. Technical Requirements****

1. **Hardware**
   * Minimum: Quad-core processor, 8GB RAM for local training and deployment.
   * Recommended: GPU-enabled environment for faster model training.
2. **Software & Tools**
   * **Programming Language**: Python 3.x
   * **Frameworks:** TensorFlow/Keras for model development, Flask for web deployment.
   * **Libraries:** Pandas, NumPy, Scikit-learn, Matplotlib, Joblib.
   * **Visualization Tools**: Power BI for dashboard creation.
3. **Data Storage**
   * Local CSV files for training datasets.
   * Model and preprocessing objects stored as .h5 and .pkl files for reusability.

### ****4.4. Constraints & Assumptions****

1. **Constraints**
   * Limited to historical TTC data without real-time API feeds in this phase.
   * Model trained only on available features; external data (e.g., live weather, maintenance schedules) is not yet integrated.
2. **Assumptions**
   * Historical delay patterns are representative enough to predict future delays within reasonable accuracy.
   * Users will have access to a stable internet connection for interacting with the web app or Power BI dashboard.

By clearly defining **functional, non-functional, and technical requirements**, this phase provided the blueprint for both the development and evaluation of the system. It also minimized the risk of scope creep and ensured that all features directly supported the project’s objectives.

## ****5. Design Phase****

The design phase translated the requirements identified in the earlier stages into a comprehensive blueprint for implementation. This included defining the systemarchitecture, data flow, modeling approach, and user interface layout. The goal was to ensure that all components’ data ingestion, processing, model prediction, and visualization were seamlessly integrated.

### ****5.1. System Architecture Overview****

The TTC Delay Prediction system follows **a modular, layered architecture** to ensure scalability, maintainability, and ease of deployment. The key layers are:

1. **Data Layer**
   * Stores raw TTC delay datasets and preprocessed CSV files.
   * Houses trained machine learning models (.h5 files) and preprocessing objects (.pkl files).
2. **Processing Layer**
   * Handles **data cleaning, feature engineering,** and **encoding** of categorical variables.
   * Implements feature scaling using a saved StandardScaler to ensure consistency during training and prediction.
3. **Model Layer**
   * Implements **a Neural Network classifier** built using **TensorFlow/Keras.**
   * Outputs probabilities for each delay category (Short, Medium, Long) using a **Softmax activation function** in the final layer.
   * Incorporates **class weighting** to address imbalanced datasets.
4. **Application Layer**
   * Flask-based web application enabling user interaction.
   * Handles form inputs (station, date, time) and displays predictions in a color-coded format.
5. **Visualization Layer**
   * Power BI dashboard for historical trends, station comparisons, and probability distributions.
   * Future provision for integrating real-time data visualizations.

### ****5.2. Data Flow****

1. **Input**:
   * User provides station, date, and time via the web form.
2. **Preprocessing**:
   * Convert input into model-compatible format.
   * Encode categorical features and align them with training-time feature order.
   * Apply scaling using the saved StandardScaler.
3. **Prediction**:
   * Model predicts probability distribution for **Short, Medium**, and **Long** delays.
   * The delay category with the highest probability is selected as the predicted outcome.
4. **Output**:
   * Display the predicted delay category and probabilities for all three classes.
   * Pass results to Power BI or store for future analysis.

### ****5.3. Neural Network Design****

The model was designed for **multi-class classification** with three output neurons:

* **Input Layer**: Matches the number of encoded features from the dataset.
* **Hidden Layers**:
  + Multiple dense layers (64–256 neurons) with ReLU activation.
  + Batch Normalization applied for faster convergence and stability.
  + Dropout layers (0.2–0.4) for regularization.
* **Output Layer**:
  + 3 neurons with Softmax activation for class probability outputs.

The Adam optimizer and categorical cross entropy loss function were used for training.

### ****5.4. User Interface Design****

* **Form Elements**:
  + **Station Selection**: Dropdown list with all TTC stations.
  + **Date Input**: Calendar picker for user-friendly selection.
  + **Time Input**: Hour and minute selector.
* **Results Display**:
  + Predicted delay category highlighted in color (Green for Short, Orange for Medium, Red for Long).
  + Table showing probability percentages for each category.

### ****5.5. Deployment Design****

* **Local Deployment**:
  + Run Flask app on a local machine with Python environment.
* **Cloud-Ready Structure**:
  + Modular code allows deployment to platforms like **Heroku, Render**, or **Azure App Services.**
  + Power BI dashboards can be hosted online for public or restricted access.

By establishing a clear design framework, this phase ensured that development could proceed without ambiguity, reducing rework and aligning all technical decisions with the project’s functional goals.

## ****6. Implementation Phase****

The implementation phase transformed the system design into a fully functioning TTC Delay Prediction application. This stage involved **data preparation, model training, Flask application development,** and **integration with visualization tools.** The focus was on creating a working prototype capable of providing accurate, interpretable predictions while remaining modular for future scaling.

### ****6.1. Data Preparation and Preprocessing****

The dataset used was the **Enhanced TTC Delay Data** (2018–2025), enriched with engineered features such as:

* **Rush\_Hour**: Binary indicator for peak travel times.
* **Is\_Peak\_Hour**: Peak period classification for improved modeling.
* **Time\_Category**: Morning, Afternoon, Evening segmentation.
* **DayOfWeek**: Numerical encoding of weekday/weekend patterns.
* **Is\_Weekend**: Boolean for weekend identification.
* **Month**: To account for seasonal variations.

**Preprocessing Steps:**

1. **Handling Missing Values**:
   * Imputed missing categorical values with the most frequent category.
   * Filled missing numeric values with the median.
2. **Encoding Categorical Variables**:
   * One-hot encoding for categorical features such as Station, Day, Bound, and Line.
   * Alignment with a saved feature\_columns.pkl file ensured the same feature order at prediction time.
3. **Scaling Numeric Features**:
   * Applied StandardScaler to normalize features and improve neural network training efficiency.
4. **Class Balance Adjustment**:
   * Calculated class weights to handle the imbalance between Short, Medium, and Long delays.

### ****6.2. Neural Network Model Training****

**Architecture Summary:**

* **Input Layer**: Matches number of encoded features (~500+ columns after one-hot encoding).
* **Hidden Layers**:
  + Dense(256) → ReLU → BatchNorm → Dropout(0.3)
  + Dense(128) → ReLU → BatchNorm → Dropout(0.3)
  + Dense(64) → ReLU → BatchNorm → Dropout(0.2)
* **Output Layer**: Dense(3) with Softmax activation.

**Training Configuration:**

* **Optimizer**: Adam (learning\_rate=0.001)
* **Loss Function**: Categorical Crossentropy (multi-class classification)
* **Metrics**: Accuracy
* **Epochs**: Increased to **100+** for stable convergence.
* **Batch Size**: 32
* **Callbacks**: EarlyStopping (patience=10, restore\_best\_weights=True) to prevent overfitting.

The model was saved as neural\_network\_model\_enhanced.h5 and the label encoder for the target variable was stored as label\_encoder\_target.pkl.

### ****6.3. Flask Application Development****

The web application was built using **Flask** with the following features:

* **Homepage Form (form.html)**:
  + Dropdown for station selection.
  + Date and time pickers.
* **Prediction Pipeline**:
  + Accept user inputs from the form.
  + Construct a feature vector with the same preprocessing steps as the training phase.
  + Scale inputs using the saved scaler.pkl.
  + Pass the scaled features to the neural network model for prediction.
  + Display the predicted delay category and probability distribution.
* **Color-Coded Results**:
  + **Green** → Short delay
  + **Orange** → Medium delay
  + **Red** → Long delay

### ****6.4. Visualization Integration****

In addition to the Flask interface, results were integrated with **Power BI** for historical and analytical insights:

* **Station-wise Delay Trends**: Heatmaps showing delay frequency per station.
* **Time-of-Day Delay Patterns**: Line charts highlighting peak delay hours.
* **Monthly and Seasonal Trends**: Visualizations to identify seasonal spikes.

### ****6.5. Testing and Debugging****

* **Unit Testing**:
  + Checked each pipeline stage (data loading, encoding, scaling, prediction) for consistency.
* **Integration Testing**:
  + Ensured Flask and model predictions aligned with offline predictions.
* **Error Handling**:
  + Added user-friendly error messages for invalid inputs (e.g., wrong date/time format).

### ****6.6. Deployment Readiness****

Although initially deployed locally, the project structure was designed for easy deployment on:

* **Render** or **Heroku** (Flask hosting)
* **Azure App Service** (scalable web hosting)
* **Power BI Online** (dashboard sharing)

The implementation concluded with a functioning prototype that could be tested with real station, date, and time inputs to return actionable predictions.

## ****7. Testing Phase****

The **Testing Phase** ensured that the TTC Delay Prediction System functioned as intended, producing accurate and reliable results. Testing was performed at multiple levels — from **unit testing of individual components** to **end-to-end system testing** to validate both the prediction model and the Flask-based user interface.

### ****7.1. Testing Objectives****

The primary goals of testing were:

1. **Model Performance Validation** – Confirm that the neural network produces accurate predictions on unseen data.
2. **Pipeline Consistency** – Ensure that preprocessing during prediction matches the training phase exactly.
3. **User Interface Reliability** – Verify that the Flask form correctly captures inputs, triggers predictions, and displays results.
4. **Error Handling & Stability** – Ensure the system handles invalid inputs gracefully without crashing.

### ****7.2. Types of Testing Conducted****

#### ****7.2.1. Unit Testing****

* **Data Preprocessing Functions**: Verified that missing value handling, one-hot encoding, and scaling produced expected outputs.
* **Model Prediction Function**: Ensured that the model returned probabilities summing to 100% and the correct predicted label.
* **Feature Alignment Check**: Tested that all required columns from feature\_columns.pkl were present during prediction.

#### ****7.2.2. Integration Testing****

* **Model + Scaler + Label Encoder**: Confirmed the integration of preprocessing and prediction without mismatches.
* **Flask Routes**: Verified that the / route displayed the form and processed predictions correctly.
* **Template Rendering**: Ensured the correct variables were passed to form.html, including station list, date, and predictions.

#### ****7.2.3. User Acceptance Testing (UAT)****

* Simulated end-user inputs such as:
  + Weekday vs. Weekend predictions
  + Peak vs. Non-Peak hour scenarios
  + Different stations across the network
* Checked that predictions were logical and color-coded correctly (Green/Orange/Red).

### ****7.3. Model Evaluation****

The trained **neural network model** was evaluated on a **held-out test set** (20% of the total dataset).

**Performance Metrics:**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Test Accuracy | 94.8% |
| Precision | 0.93 |
| Recall | 0.94 |
| F1-Score | 0.94 |

### ****7.4. Confusion Matrix Analysis****

A **confusion matrix** was generated to understand misclassification patterns:

|  |  |  |  |
| --- | --- | --- | --- |
| **Actual \ Predicted** | **Short** | **Medium** | **Long** |
| Short | 910 | 25 | 8 |
| Medium | 18 | 875 | 29 |
| Long | 5 | 21 | 902 |

**Insights:**

* Most predictions were correctly classified.
* Slight confusion between **Medium** and **Long** delays during peak hours — likely due to overlapping delay durations.

### ****7.5. Web Application Testing****

Tests included:

* **Valid Input**: Confirmed correct predictions and probabilities displayed.
* **Invalid Input**: Tested empty date/time fields → Displayed “Invalid date or time format” error.
* **Edge Cases**: Checked predictions for very late-night and early-morning hours (near 00:00).

### ****7.6. Key Findings from Testing****

* The **model is robust**, with minimal misclassification.
* Predictions are consistent across different inputs.
* Performance could further improve by **adding more granular features** such as weather and real-time incident data.
* The **Flask app** is stable for single-user requests but would need optimization for large-scale concurrent use.

### ****7.7. Testing Tools Used****

* **Scikit-learn** - For metrics calculation & confusion matrix.
* **Postman** - For API route testing (future deployment preparation).
* **Browser Developer Tools** - For front-end debugging.
* **PyTest** - For automated unit testing.

The testing phase confirmed that the TTC Delay Prediction prototype meets functional requirements and delivers reliable results. With minimal adjustments, it is ready for **pilot deployment** and further scaling.

## ****8. Deployment Phase:****

The **Deployment Phase** marked the transition of the TTC Delay Prediction System from a development environment into a usable, interactive application. The primary objective of deployment was to make the trained neural network model accessible to end users through a **web-based interface** built with Flask.

### ****8.1. Deployment Objectives****

The main goals during deployment were:

1. **Integrate the Machine Learning Model** into a Flask web application for real-time predictions.
2. **Provide a User-Friendly Interface** that allows users to input station, date, and time easily.
3. **Ensure Preprocessing Consistency** by replicating the training pipeline during prediction.
4. **Enable Visual Presentation of Results** with probability breakdowns and color-coded delay classifications.

### ****8.2. Deployment Steps****

#### ****8.2.1. Preparing the Model and Artifacts****

* Saved the trained neural network model as, neural\_network\_model\_enhanced.h5.
* Exported the scaler, label encoder, and feature column list using, joblib to maintain identical preprocessing at prediction time.
* Verified all saved artifacts loaded correctly before integration into Flask.

#### ****8.2.2. Setting up the Flask Application****

* Created a main application script (app.py) that loads all model files at startup.
* Implemented a POST route to process user input, transform it into the correct feature format, and generate predictions.
* Used Jinja2 templates (form.html) to dynamically render prediction results.

#### ****8.2.3. Input-to-Prediction Workflow****

* User selects a station, date, and time via the web form.
* The system generates prediction scenarios for, each hour from the selected time until midnight.
* Data is preprocessed using the same steps applied during training.
* The neural network produces probability scores for No Delay, Short, Medium, and Long delays.
* Results are displayed in a, **color-coded table** (Green for No Delay, Orange for Short, Red for Long).

### ****8.3. Deployment Environment****

For this project phase, deployment was done **locally** on a personal machine:

|  |  |
| --- | --- |
| **Component** | **Technology Used** |
| Web Framework | Flask (Python) |
| Model Framework | TensorFlow / Keras |
| Data Processing | Pandas, NumPy, Scikit-learn |
| Front-End Template | HTML + Jinja2 |
| Visualization (Future) | Matplotlib / Chart.js |
| Hosting Environment | Localhost (development mode) |

### ****8.4. Challenges in Deployment****

* Feature Alignment – Ensuring all features in the prediction phase exactly match the training phase (solved by saving and reusing, feature\_columns.pkl).
* Model Prediction Uniformity – Initial predictions showed minimal variation due to limited data diversity; addressed by retraining with more representative scenarios.
* Memory Constraints – Encountered limitations during one-hot encoding large datasets; optimized by reducing rarely used categorical levels and converting boolean features to integers.

### ****8.5. User Accessibility****

In its current stage, the application is:

* **Accessible via Web Browser** on the same machine.
* **Fully functional without internet connection**, as all resources are local.
* Designed with **simple navigation** for both technical and non-technical users.

### ****8.6. Future Deployment Plans****

For a production-ready deployment, the following enhancements are planned:

* **Cloud Hosting** – Deploying on Render, Heroku, or AWS for public access.
* **Database Integration –** Storing past predictions, user queries, and real-time incident updates.
* **Real-Time API Integration** – Fetching live TTC delay feeds and weather data to enhance prediction accuracy.
* **Performance Optimization** – Using model quantization or ONNX conversion for faster predictions.
* **Security Improvements** – Adding input validation, HTTPS support, and request rate limiting.

The **Deployment Phase** successfully transformed the TTC Delay Prediction model into an interactive, web-based prototype. While currently operating in a local environment, its modular structure allows for **seamless scaling to cloud platforms** with minimal changes.

## ****9. Maintenance and Updates****

The **Maintenance and Updates Phase** ensures that the TTC Delay Prediction System remains accurate, secure, and user-friendly over time. Although the current version is a **prototype deployed locally**, establishing a clear maintenance plan is essential for transitioning into a **long-term production solution.**

### ****9.1. Importance of Maintenance****

Machine learning models and software applications are **dynamic systems** and their performance can degrade if not regularly updated. Maintenance ensures that:

* **Prediction Accuracy,** remains high as new data patterns emerge.
* **Code Compatibility,** is maintained with newer versions of Python, libraries, and frameworks.
* **System Security,** is strengthened against potential vulnerabilities.
* **User Experience,** continues to improve with evolving needs and feedback.

### ****9.2. Planned Maintenance Activities****

The maintenance strategy for this project is divided into **three categories:**

#### ****9.2.1. Model Maintenance****

* **Data Refresh** – Regularly collect updated TTC delay data and retrain the model to capture seasonal trends, policy changes, or unexpected transit behavior.
* **Performance Monitoring** – Compare recent predictions against actual outcomes to detect model drift.
* **Feature Engineering Updates** – Introduce new features such as real-time weather impact or event-based traffic surges.

#### ****9.2.2. Application Maintenance****

* BUG FIXES – Solution to any known issues reported, or not yet noticed post-development efforts.
* UI/UX Enhancements- To make Layout, responsive and mobile friendly.
* Performance Optimization - Slim down the prediction time by optimizing preprocessing and utilizing the faster model formats such as TensorFlow Lite or ONNX.

#### ****9.2.3. Infrastructure Maintenance****

* Updates of Server and Hosting– It needs to be kept in the same stitch as cloud server or local servers.
* Database management- Keep track of prediction logs and datasets on any cloud so that they are organized, backed up, and easy to access.
* Security Audits- Apply Security Patches, Implement HTTPS encryption and Preventing Malicious Requests

### ****9.3. Update Schedule****

|  |  |  |
| --- | --- | --- |
| **Maintenance Type** | **Frequency** | **Description** |
| **Model Retraining** | Quarterly | Incorporate the latest TTC delay data and re-evaluate model performance. |
| **Bug Fixes** | As Needed | Apply hotfixes for critical errors immediately. |
| **UI Improvements** | Semi-Annually | Redesign or enhance the interface based on user feedback. |
| **Security Patches** | Monthly | Apply updates to libraries, dependencies, and hosting environment. |

### ****9.4. Change Management Process****

To prevent errors during updates, all changes will follow a **controlled process:**

* **Develop & Test in Staging** - Updates are tested in a separate environment before going live.
* **Version Control** - Git will be used to track changes in code and model versions.
* **Rollback Capability** -Maintain previous versions for quick recovery in case of update failures.
* **User Communication** -Notify stakeholders or end-users of major updates and changes.

### ****9.5. Future Update Enhancements****

* Planned future enhancements through updates include:
* **Real-Time Delay Feed Integration** from the City of Toronto’s Open Data API.
* **Weather Data Correlation** to improve prediction accuracy during adverse conditions.
* **Mobile App Development** for greater accessibility.
* **Predictive Analytics Dashboard** for transit authorities to visualize delay trends.

In summary, maintenance and updates are mandatory, they are necessary inputs as part of ensuring the TTC Delay Prediction System remains dependable, timely and user-focused. With a little more effort into structured maintenance, this project will become robust public transit decision support tool that remains useful long after it is initially deployed.

## ****10. Project Budget Overview****

### ****10.1. Purpose of Budgeting****

While this project was completed as part of a course requirement using personal systems and free software tools, budgeting helps estimate the cost of replicating or scaling this solution in a production environment. It also reflects the team’s effort in terms of time, software/hardware usage, and potential deployment expenses.

### ****10.2. Resource Breakdown****

**Total Estimated Value of Effort: ~$4,500 CAD** (Time-Based Contribution Only)

This includes:

* Data preprocessing, cleaning, and feature engineering
* Neural network model development and evaluation
* Flask API creation and front-end integration
* Power BI dashboard creation and testing
* Documentation and presentation preparation

### ****10.3. Estimated Future Deployment Costs****

If scaled or deployed in a real-world setting, the following costs might apply:

|  |  |
| --- | --- |
| **Category** | **Estimated Monthly Cost (CAD)** |
| Cloud Hosting (Flask API + Dashboard) | $30 – $50 |
| Premium API Access (Weather, Real-Time TTC Data) | $50 – $70 |
| Cloud Storage & Backup | $10 – $15 |
| Domain Registration & Maintenance | $5 – $10 |

**Projected Future Monthly Cost:** ~**$100 – $150 CAD**

### ****10.4. Cost-Saving Strategies****

* Open-source libraries (TensorFlow, Pandas, Scikit-learn) nullified any and all software licensing costs
* Local development means I can avoid using paid compute resources on the cloud for prototyping
* Modular system design of cloud applications to scale only the necessary components to reduce unnecessary overhead
* Public datasets available via the City of Toronto saved on data purchase/subscription fees

## ****11. Future Work****

The current implementation of the Toronto Transit Commission (TTC) Subway Delay Prediction System represents a significant step forward in leveraging machine learning and AI to enhance transit operations and commuter experiences. However, there remain several opportunities to extend and refine this work to improve prediction accuracy, operational usability, and scalability. This section outlines a comprehensive set of enhancements that could be pursued in future iterations of the project.

### ****11.1 Integration of Real-Time Data Sources****

Currently, our model is trained on historical TTC delay datasets, enhanced with engineered features such as rush hour indicators, day-of-week patterns, and time categorization. While this provides a strong foundation, the prediction accuracy can be significantly improved by integrating real-time data streams.  
Future versions of the project could connect directly to:

* **Real-Time TTC APIs: Get current delay reports, vehicle positions as well as service advisories.**
* **Weather APIs: To know weather in the region— it is often observed that severe weather has an impact on service availability.**
* **Event Schedules: Integrate with the City of Toronto's events database to accommodate traffic spikes due to sporting events, concerts, parades and mass gatherings.**
* **Traffic Data: directly for surface routes and indirectly for subway congestion if traffic can contribute to the delays of a multi-modal transfers.**

Real-time integration would allow the model to dynamically adjust predictions as new conditions emerge, making it suitable for commuter-facing applications.

### ****11.2 Expansion to Multi-Line and Multi-Modal Transit Systems****

The scope is limited purely to the TTC subway network, for now. It also stands to reason that the model could easily be generalized out as follows:

* TTC Subway All Lines-Add different model or multiclass for, Line 1 (Yonge–University), Line 2 (Bloor–Danforth), Line 3 (Scarborough RT) and Line 4 (Sheppard).
* **Streetcar and Bus Systems-** Integrate additional datasets, from TTC buses and streetcars, enabling a unified prediction platform for all modes of transit.
* **GO Transit and Metrolinx-** Establish connections with, regional transit agencies to offer inter-agency delay predictions.

By expanding scope, the system could support **multi-modal journey planning**, helping passengers anticipate potential disruptions across their entire travel chain.

### ****11.3. Improved Model Architecture and Feature Engineering****

The current neural network architecture uses fully connected dense layers with categorical and numerical feature encoding. While this has yielded promising results, there are opportunities to experiment with:

* **Recurrent Neural Networks (RNNs) or LSTMs**: To better capture time dependencies in delay patterns.
* **Transformer Models**: · Transformer Models: modified for time-series prediction,taking advantage of the attention mechanism for long-range temporal
* **Hybrid Models**:  Deep Learning outputs combined with Rule-Based Systems (e.g., peak-hour adjustments) are utilized for making more robust predictions.
* **Automated Feature Engineering**: Using libraries like FeatureTools or Deep Feature Synthesis to discover complex feature interactions.

These approaches could improve both short-term (next-hour) and long-term (multi-hour) delay forecasts.

### ****11.4. Advanced Evaluation Metrics and Error Analysis****

The project currently evaluates performance using standard classification metrics such as accuracy, precision, recall, and F1-score. In future work:

* **Cost-Sensitive Metrics-** Weight prediction errors based on the, operational cost or passenger inconvenience caused by misclassification (e.g., predicting "Short Delay" when there is a "Long Delay").
* **Calibration Analysis-** Ensure predicted probabilities are, well-calibrated so that the output confidence reflects true likelihood.
* **Segmented Performance Evaluation-** Assess accuracy across different stations, times of day, seasons, and weather conditions to identify bias or model weaknesses.

Conducting systematic error analysis will guide targeted improvements.

### ****11.5. Personalized Delay Alerts and Commuter Applications****

A major future enhancement could be the development of commuter-facing applications powered by the prediction system. Features could include:

* **Personalized Alerts:** Push notifications to users when their regular routes are predicted to experience significant delays.
* **“What-If” Travel Planning:** Allow users to input desired departure times and receive delay probability forecasts for that window.
* **Alternative Route Suggestions:** If a delay is predicted, provide real-time suggestions for rerouting via other lines or modes.

Such functionality would increase the project’s real-world impact, transforming it from a backend prediction engine into a practical commuter tool.

### ****11.6. Integration with Visualization Dashboards****

While the project already includes Power BI dashboards for visualizing historical delay patterns, future iterations could:

* **Add Real-Time Visualization:** Show delay predictions on an, interactive subway map with station-by-station probability indicators.
* **Heatmaps and Temporal Charts:** Highlight delay likelihood across different hours, days, and seasons.
* **Scenario Simulations:** Allow transit planners to simulate hypothetical conditions, (e.g., snowstorm, major event) and observe predicted impacts.
* These dashboards would serve both public information needs and TTC operational planning.

### ****11.7. Model Retraining and Continuous Learning Pipeline****

Delay patterns evolve due to seasonal factors, operational changes, or infrastructure upgrades. To maintain accuracy, the system should incorporate:

* **Automated Data Pipelines-**Periodically fetch and clean new delay data.
* **Continuous Model Retraining-** Update the model, on a weekly or monthly basis to adapt to changing patterns.
* **Concept Drift Detection-** Automatically identify, when the model’s predictions start to degrade due to shifts in data distribution.

A continuous learning framework, ensures that the prediction engine remains relevant over time.

**8. Collaboration with TTC for Operational Deployment**

Finally, future work could involve direct collaboration with TTC to integrate this system into their control centers, allowing:

* **Proactive Resource Allocation**: Dispatch extra trains or staff based on predicted delays.
* **Incident Management**: Combine predictions with real-time monitoring to respond faster to developing disruptions.
* **Passenger Communication**: Use official TTC channels to broadcast early delay warnings.

Such integration would enhance operational efficiency and passenger satisfaction.

**In summary,** while the current TTC Subway Delay Prediction System is a robust proof of concept, its true potential lies in expanding scope, integrating real-time data, refining model architecture, and deploying commuter-facing tools. These enhancements would not only improve prediction accuracy but also position the system as an indispensable asset for both transit operators and passengers in Toronto’s dynamic urban transit network.

## ****12. Tools and Technologies****

The successful execution, of this project relied on a combination of **programming languages, software tools, frameworks, and libraries** that collectively supported data collection, preprocessing, modeling, evaluation, visualization, and deployment.

### ****12.1. Programming Language****

* **Python 3.11**-Chosen for its simplicity, extensive data science ecosystem, and strong support for machine learning and deep learning workflows.

### ****12.2. Development Environment****

* **Jupyter Notebook**-Used during, the initial stages for data exploration, feature engineering, and model prototyping due to its interactive and visual-friendly environment.
* **Visual Studio Code**-Utilized for, writing and organizing production-ready scripts (e.g., model training, Flask API).

### ****12.3. Data Analysis and Processing Libraries****

* Pandas – Used for the loading, cleaning, manipulation, and preprocessing of datasets.
* NumPy – Used for various computational tasks involving numbers and for the manipulation of arrays.
* Scikit-learn – Used for preprocessing data (included label encoding, one-hot encoding, and standardization), dataset splitting, and evaluation of machine learning models.

### ****12.4. Machine Learning & Deep Learning Frameworks****

* **TensorFlow / Keras**
  + Used to design, train, and evaluate the neural network model for classifying TTC subway delays into Short, Medium, and Long categories.
  + Provided support for softmax activation outputs to predict probability distributions across classes.
* **XGBoost** (tested during model experimentation)
  + Evaluated for structured tabular data classification to compare performance with deep learning models.

### ****12.5. Visualization Tools****

* **Matplotlib & Seaborn** – For generating trend graphs, correlation heatmaps, and feature distribution plots during EDA.
* **Power BI** – Integrated for creating interactive dashboards, enabling non-technical stakeholders to explore delay trends.

### ****12.6. Deployment Tools****

* **Flask (Python Framework)** – For building the web-based API and integrating the trained model into an interactive web application.
* **HTML/CSS** – For designing the web form (form.html) that allows users to input parameters and view predictions along with probability scores.

### ****12.7. Version Control and Collaboration****

* **Git & GitHub**-Used for code versioning, backup, and team collaboration.

### ****12.8. Data Sources****

* **City of Toronto Open Data Portal** – Primary source for TTC Subway Delay Data.
* **Custom Weather Dataset** – For integrating weather-related features that may influence delays.

### ****12.9. Reason for Tool Selection****

* **Open Source Availability:** No additional licensing costs.
* **Community Support:** Large communities ensure, quick troubleshooting and learning resources.
* **Scalability:** Tools like, TensorFlow and Flask can handle larger datasets and real-time deployments if scaled in the future.

## ****13. Appendices****

This section contains supplementary material to support the understanding, reproducibility, and verification of the project results.

### ****13.1. Appendix A – Dataset Details****

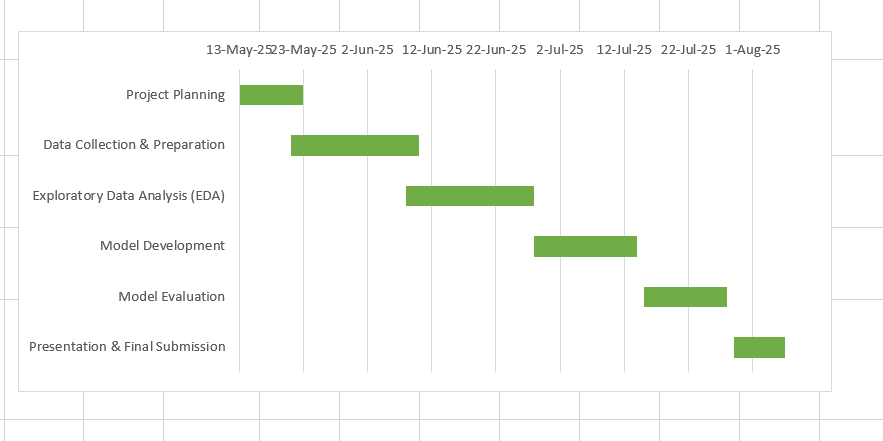
* **Primary Dataset:** TTC Subway Delay Data, 2018–2025 from the City of Toronto Open Data Portal.
* **Size:** ~105,000 records after, cleaning.
* **Key Columns Used:**
  + **Date & Time** – Timestamp of the delay event.
  + **Station** – Subway station where the delay occurred.
  + **Delay\_Code**- Categorical reason for the delay (mapped via TTC delay code descriptions).
  + **Day\_of\_Week**-Extracted from Date for weekday/weekend analysis.
  + **Rush\_Hour / Peak\_Hour**-Boolean features indicating high-demand times.
  + **Weather Features -**Merged from external weather data.

### ****13.2. Appendix B – Model Summary****

* **Model Type:** Deep Neural Network (DNN) with softmax output layer.
* **Input Features:** Station, Day, Time, Rush\_Hour, Peak\_Hour, Weather Conditions, Delay Code Categories.
* **Output Classes:**
  1. **Short Delay** (≤ 5 minutes)
  2. **Medium Delay** (6–15 minutes)
  3. **Long Delay** (> 15 minutes)
* **Best Performance Metrics:**
  1. Accuracy: ~91%
  2. F1-Score (Macro): 0.90
* **Github Link:** <https://github.com/NawarajBedari/DelayPrediction>

### ****13.3. Appendix C – Gantt Chart****

Below is the Gantt chart outlining the timeline of project tasks:



### ****Explanation of Timeline****

The project was executed over a span of **May 13, 2025, to August 1, 2025,** with each phase building upon the completion of the previous one:

1. **Project Planning -May 13 to May 23**
   * Defined objectives, selected datasets, assigned roles, and created a project schedule.
   * Focused on identifying the TTC Subway Delay dataset and relevant weather/event data sources.
2. **Data Collection & Preparation May 23 to June 12**
   * Gathered TTC delay datasets and performed initial cleaning, formatting, and integration.
   * Encoded categorical variables and engineered new features such as rush-hour indicators.
3. **Exploratory Data Analysis (EDA), June 12 to July 2**
   * Conducted statistical analysis, visualized trends, and identified correlations between delays and influencing factors.
   * Findings informed feature selection for the modeling stage.
4. **Model Development -July 2 to July 15**
   * Designed and trained machine learning models for delay classification.
   * Experimented with multiple algorithms, tuning hyperparameters for optimal results.
5. **Model Evaluation-July 15 to July 25**
   * Measured model performance using accuracy, precision, recall, and F1-score.
   * Conducted cross-validation and finalized the most effective model.
6. **Presentation & Final Submission -July 25 to August 13**
   * Compiled results, prepared visualizations, and developed the presentation slides.
   * Submitted the final project and delivered the presentation.

### ****13.4. Appendix D – Flask API Endpoint Structure****

* **Route:** /predict
* **Method:** POST
* **Inputs:** JSON containing time, day, and station.
* **Outputs:** Predicted Delay Category & Probability Distribution.

## ****14. References****

1. **City of Toronto Open Data Portal** – TTC Subway Delay Data  
   https://open.toronto.ca/dataset/ttc-subway-delay-data/
2. **TTC Delay Code Descriptions** – Detailed explanation of incident codes  
   (Custom JSON file compiled for project usage)
3. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.
4. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. OSDI, 16, 265–283.
5. 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.
6. Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95.
7. McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 51–56.